

Massimo Cossentino · Vincent Hilaire
Ambra Molesini · Valeria Seidita *Editors*

Handbook on Agent-Oriented Design Processes

 Springer

Handbook on Agent-Oriented Design Processes

Massimo Cossentino • Vincent Hilaire •
Ambra Molesini • Valeria Seidita
Editors

Handbook on Agent-Oriented Design Processes

 Springer

Editors

Massimo Cossentino
ICAR-CNR
Palermo
Italy

Vincent Hilaire
University of Technology
of Belfort Montbéliard
Belfort cedex
France

Ambra Molesini
Alma Mater Studiorum
Università di Bologna
Bologna
Italy

Valeria Seidita
DICGIM
University of Palermo
Palermo
Italy

ISBN 978-3-642-39974-9

ISBN 978-3-642-39975-6 (eBook)

DOI 10.1007/978-3-642-39975-6

Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014932973

© Springer-Verlag Berlin Heidelberg 2014

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

Designing a multi-agent system (MAS) is not an easy task: creating agents, environments, norms, organizations, and make them cooperate in order to solve a collective task is both an art and a science.

MAS have to deal with two important and difficult issues: autonomy and interaction. Autonomy is the ability for an agent to solve local and individual problems using its own memory and ability to decide what to do next in accordance with what kind of information it perceives. Of course, agents do not have free will: everything is programmed. It is generally possible at this individual level to have a perfect control of the agent's behavior and to specify its functioning through standard computer design tools and methods. Interaction, on the other hand, may be defined as the mutual influence agents—and the environment—exert to one another. Interaction is a kind of glue that makes agents behave in a loosely coupled way, coordinating their behavior in order to achieve a task, leaving all the little details of the adjustments of actions to their ability to adapt themselves to the dynamic's minor alterations of the whole system.

Autonomy and interaction lead to something that usually computer scientists don't like: the inability to have a perfect control on the whole system. Unpredictability is not something you like when you design software systems. Another amazing thing that happens when you program MAS is that your program does not bug . . . Usually, when programming standard computer programs, your program fails until you have what you want, and programmers spend their time in the debugger, understanding what is wrong, and why the program breaks down . . . But this is not true with MAS. Your program does work nearly all the time, but what you see is not what you expect . . . So debuggers and breakpoints are not as useful as in standard programming, and the MAS programmer has to rely on other tools to understand what is programmed and why the results do not correspond to what is expected.

Why do MAS show such strange behaviors? The main reason is that, in MAS, programming is done at the agent level while results are observed at the system level. And this is both the marvel and the curse of MAS programming. Taken aside, each agent is carefully crafted and programmed to achieve what is expected. But when a group of agents interact in a more or less coordinated way, new properties arise as the result of those interactions, which are difficult to foresee. This collective behavior can solve a lot of tricky issues that would have required hours and hours of delicate design and programming without MAS, if we had to plan everything

ahead. And even though, the system would not adapt to new situations as neatly as a MAS. From interaction and autonomy comes magic! Actually this is not magic. It is called emergence, i.e. the ability to produce new results and solve complex problems as a side effect of all the particular actions that agents perform through their coordination of action with other agents. In a MAS, everything is dynamic by nature. Nothing remains the same, and this is this dynamicity which is both required and feared.

But this fantastic feature of MAS has a terrible drawback: the difficulty to control and constrain the global behavior of the system in order for the system to do what you want. And this is precisely why standard methods and tools cannot cope with the specificity of MAS, and why it has been necessary to propose new ones that are collected in this volume.

This book is an invaluable guide, providing both students and advanced practitioners with a thorough compilation of the main methodologies used in MAS. Contributions come from international authors well renowned in the field, who have dedicated their time and effort to the development and application of MAS. This book is the result of decades of intense work while analyzing, designing, and testing multi-agent systems in both research and application domains.

For the first time, a wide panel of MAS methodologies is presented. They range from the reactive point of view, where a solution is obtained as a side effect of an adaptive process, to the more cognitive approach, where agents have beliefs and goals and are defined as members of organizations. Thus, there are both bottom-up and top-down methodologies, allowing MAS designers to get a practical overview of the many approaches available.

Therefore, it is my pleasure to recommend this book that will stand as a reference which long-term users could consult to improve their knowledge and know-how in developing MAS.

Montpellier, France
May 2013

Jacques Ferber

Contents

Introduction	1
Massimo Cossentino, Ambra Molesini, Vincent Hilaire, and Valeria Seidita	
The IEEE-FIPA Standard on the Design Process Documentation Template	7
Massimo Cossentino, Vincent Hilaire, Ambra Molesini, and Valeria Seidita	
ADELFE 2.0	19
N. Bonjean, W. Mefteh, M.P. Gleizes, C. Maurel, and F. Migeon	
The ASPECS Process	65
Massimo Cossentino, Vincent Hilaire, Nicolas Gaud, Stephane Galland, and Abderrafiaa Koukam	
ELDAMeth Design Process	115
Giancarlo Fortino, Francesco Rango, and Wilma Russo	
The Gaia Methodology Process	141
Luca Cernuzzi, Ambra Molesini, and Andrea Omicini	
GORMAS: A Methodological Guideline for Organizational-Oriented Open MAS	173
Sergio Esparcia, Estefanía Argente, Vicente Julián, and Vicente Botti	
INGENIAS-Scrum	219
Juan C. González-Moreno, Alma Gómez-Rodríguez, Rubén Fuentes-Fernández, and David Ramos-Valcárcel	
The O-MASE Methodology	253
Scott A. DeLoach and Juan C. Garcia-Ojeda	
PASSI: Process for Agent Societies Specification and Implementation	287
Massimo Cossentino and Valeria Seidita	

ROMAS Methodology	331
Emilia Garcia, Adriana Giret, and Vicente Botti	
INGENIAS with the Unified Development Process	371
Alma Gómez-Rodríguez, Rubén Fuentes-Fernández, Juan C. González-Moreno, and Francisco J. Rodríguez-Martínez	
The SODA Methodology: Meta-model and Process Documentation	407
Ambra Molesini and Andrea Omicini	
The Tropos Software Engineering Methodology	463
Mirko Morandini, Fabiano Dalpiaz, Cu Duy Nguyen, and Alberto Siena	
The OpenUp Process	491
Massimo Cossentino, Vincent Hilaire, and Valeria Seidita	
Index	567

Introduction

Massimo Cossentino, Ambra Molesini, Vincent Hilaire,
and Valeria Seidita

Nowadays, software engineers face a wide range of particular application domains such as electronic commerce, enterprise resource planning, mobile computing, self-organisation, pervasive and adaptive computing, etc. The resulting heterogeneity and required functionalities call for complex systems and open architectures that may evolve dynamically over time so as to accommodate new components and meet new requirements. This is probably one of the main reasons why the agent metaphor and agent-based computing are gaining momentum in these areas.

As far as software engineering is concerned, the key implication is that the design and development of software systems according to a (new) paradigm can by no means rely on conceptual tools and methodologies conceived for a totally different (old) paradigm [19]. In this book we do not want to enter the debate about the differences in the definition of *methodology* and *design process* or simply *process*. For the aim of this book we may consider them as synonymous.

Even if it is still possible to develop a complex distributed system in terms of objects and client–server interactions, for example, such a choice appears odd and ill-adapted when the system is a Multi-agent System (MAS) or can be assimilated to an MAS. Rather, a brand new set of conceptual and practical tools—

M. Cossentino

Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, Palermo, Italy
e-mail: cossentino@pa.icar.cnr.it

A. Molesini

Alma Mater Studiorum – Università di Bologna Viale Risorgimento 2, 40136 Bologna, Italy
e-mail: ambra.molesini@unibo.it

V. Seidita (✉)

Dipartimento di Ingegneria Chimica, Gestionale, Informatica, Meccanica, University of Palermo, Palermo, Italy
e-mail: valeria.seidita@unipa.it

V. Hilaire

IRTES-SET, UTBM, UPR EA 7274 90 010 Belfort cedex, France
vincent.hilaire@utbm.fr

specifically suited to the abstractions of agent-based computing—is needed to facilitate, promote and support the analysis, design and development of MASs, and to fulfil the great general-purpose potential of agent-based computing.

The definition of agent-specific methodologies is definitely one of the most explored topics in Agent-Oriented Software Engineering (AOSE). A great number of special-purpose methodologies were developed in the last few years to tackle the challenges of specific application domains.

A methodology has several constituent parts, including a full life cycle process, a comprehensive set of concepts, a set of rules, heuristics and guidelines, a set of metrics, information on quality assurance, a set of coding and other organisational standards, techniques for reuse and project management procedures [5, 10, 12]. Among the different methodologies developed [2, 6, 11, 14–16, 18, 20] there is a great consent to organise the process life cycle according to two classical phases: Analysis and Design. However, apart from this consensus, the cited methodologies exhibit many differences. Indeed, and it is not the only difference, regarding the abstractions adopted for modelling the systems there is no agreement among the different methodologies: each methodology proposes an own set of abstractions for modelling the systems. Due to this missing agreement, AOSE methodologies typically define their abstractions through a metamodel [1, 7, 8] that details all the abstractions adopted by the methodology and their relationships. Even if a lot of abstractions are labelled in the same way in the different methodologies, these abstractions frequently present different meaning and may belong to different process phases. For example, the “agent” concept is always present in the AOSE methodologies, however in some methodologies such as [6, 15, 18] the agent appears since Analysis phase, while in other methodologies such as [2, 16, 20] “agent” is only a design concept.

A key concept both in traditional software engineering and in agent-oriented one is a common agreement that the “ideal methodology” [17] for any system does not exist. This means that the methodology must be adapted to the particular characteristics of the adopted usage scenarios. There are two major ways of adapting methodologies: tailoring (particularisation or customisation of a pre-existing methodology) or Situational Method Engineering (SME)[3,4]. SME is the discipline that studies the composition of new, ad hoc software engineering processes for each specific need. This is based on the assumption that the “situation” is the information leading to the identification of the right design approach. This paradigm provides tools for the construction of ad hoc processes by means of reuse of the existing portion of methodologies (called method fragments) stored in a repository.

In order to be effective, this approach requires the design process and its fragments to be suitably documented and modelled.

In this context, the work of the IEEE-FIPA Design Process Documentation and Fragmentation (FIPA-DPDF) Working Group [9] aims at providing the possibility of representing design processes and method fragments through the use of standardised templates, thus allowing the creation of easily sharable repositories and enabling an easier composition of new design processes. At the beginning of 2012 the Process Documentation Template became an IEEE FIPA standard specification with number SC00097B [13].

Following this standardisation work, this book gathers the documentations of some of the most known AOSE methodologies. Chapters describing the methodologies have been produced, in several cases, by the original authors of the methodology itself. In all cases, authors are FIPA-DPDF WG Members who with their effort also aimed at validating the effectiveness of the Process Documentation Template specification. In particular, the methodologies documented in this book are as follows:

- **ADELFE**—it is dedicated to applications characterised by openness and the need of system adaptation to an environment. Its main goal is to help and guide any designer during the development of an Adaptive Multi-agent System.
- **ASPECS**—it is a step-by-step requirement-to-code iterative development process for engineering Complex Systems using Multi-agent Systems and Holonic Multi-agent Systems.
- **ELDAMeth**—it is an agent-oriented methodology specifically designed for the simulation-based prototyping of distributed agent systems. It is centred on the ELDA agent model and on an iterative development process covering distributed agent system modelling, simulation and implementation for a target agent platform.
- **Gaia**—it focuses on the use of organisational abstractions to drive the analysis and design of MAS. Gaia models both the macro (social) aspects and the micro (agent internals) aspects of MAS, and it devotes a specific effort to model the organisational structure and the organisational rules that govern the global behaviour of the agents in the organisation.
- **GORMAS**—it is focused on designing large scale, open, service-oriented Organisation-Centred Multi-agent Systems, where organisations are able to accept external agents into them. GORMAS is based on a specific method for designing human organisations.
- **INGENIAS**—it covers the full development cycle of Multi-agent Systems. It includes support for requirements elicitation (basic), analysis, design, coding and testing. It is intended for general use, that is, with no restrictions on application domains.
- **INGENIAS-Agile**—it introduces the definition of an agile process for the INGENIAS methodology according to a well-known development process: Scrum. The process adopts the iterative and fast plan presented originally by the methodology and uses some of the activities and most of the work products of the INGENIAS proposal with the Unified Development Process.
- **O-MaSE**—it is a new approach in the analysis and design of agent-based systems, being designed from the start as a set of method fragments to be used in a method engineering framework. The goal of O-MaSE is to allow designers to create customised agent-oriented software development processes.
- **OpenUp**—it is an open source process framework developed within the Eclipse Foundation. It aims at enabling an easy adoption of the core of the RUP/Unified Process. It proposes an iterative and incremental approach within a structured life cycle. OpenUp embraces a pragmatic, agile philosophy that focuses on the collaborative nature of software development; this is the unique not agent-oriented

methodology reported in this book and it demonstrates the feasibility of the adopted Process Documentation Template even in the OO context.

- PASSI—it is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies integrating design models and concepts from both OO software engineering and artificial intelligence approaches using the UML notation.
- ROMAS—it defines a set of activities for the analysis and design of regulated open multi-agent systems. The defining characteristics of systems of this kind are that they are social, open and regulated.
- SODA—it deals with MAS analysis and design, and focuses on critical issues such as agent coordination and MAS–environment interaction. SODA adopts the Agents & Artifacts meta-model, and it introduces a *layering principle* as an effective tool for scaling with system complexity.
- Tropos—it is a comprehensive, agent-oriented methodology for developing socio-technical systems. Such systems explicitly recognise the existence of and interplay between technical systems (software) and social actors (humans and organisations).

The book is organised as follows:

- Chapter “The IEEE-FIPA Standard on Design Process Document Template” presents the motivations and some basic concepts of the Process Documentation Template.
- Chapters “ADELFE 2.0” to “The OpenUp Process” present (with the exception of report) the documentation of the AOSE methodologies.
- Chapter “ROMAS Methodology” presents the documentation of a non-AOSE methodology (OpenUP).

Finally, to conclude this short introduction, we would like to highlight that this book represents a collective effort of the IEEE-FIPA DPDF Working Group, which would not have been possible to realise without the contributions of a number of individuals, to whom we are deeply grateful. In particular, we thank all the authors who accepted to contribute to this book and who took effort to contribute original chapters presenting their methodologies and design processes according to the Process Documentation Template. We would also like to thank all the other WG members for their contribution to the definition of the IEEE FIPA SC00097B specification. Finally, we would like to thank Prof. J. Ferber for having kindly written his interesting preface to this book.

We all hope you will enjoy reading this book.

References

1. Bernon, C., Cossentino, M., Gleizes, M.P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In: Odell, J., Giorgini, P., Müller, J.P. (eds.) *Agent Oriented Software Engineering V. Lecture Notes in Computer Science*, vol. 3382, pp. 62–77. Springer, New York (2004)
2. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: an agent-oriented software development methodology. *Auton. Agent Multi Agent Syst.* **8**(3), 203–236 (2004). <http://www.springerlink.com/content/g757056736223u65/>

3. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. *Inf. Softw. Technol.* **38**(4), 275–280 (1996)
4. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-modelling based assembly techniques for situational method engineering. *Inf. Syst.* **24**(3), 209–228 (1999)
5. Cernuzzi, L., Cossentino, M., Zambonelli, F.: Process models for agent-based development. *Eng. Appl. Artif. Intell.* **18**(2), 205–222 (2005)
6. Cossentino, M.: From requirements to code with the PASSI methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.): *Agent Oriented Methodologies*, Chap. IV, pp. 79–106. Idea Group Publishing, Hershey (2005). <http://www.idea-group.com/books/details.asp?id=4931>
7. Cossentino, M., Gaglio, S., Sabatucci, L., Seidita, V.: The passi and agile passi mas meta-models compared with a unifying proposal. In: *Proceedings of the CEEMAS'05 Conference*, Budapest, Hungary, pp. 183–192 (2005)
8. Cossentino, M., Gaglio, S., Galland, S., Gaud, N., Hilaire, V., Koukam, A., Seidita, V.: A MAS metamodel-driven approach to process fragments selection. In: Luck, M., Gómez-Sanz, J.J. (eds.) *Agent-Oriented Software Engineering IX*. *Lecture Notes in Computer Science*, vol. 5386, pp. 86–100. Springer, Berlin (2009)
9. Cossentino, M., Hilaire, V., Molesini, A.: Fipa design process documentation and fragmentation working group. <http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/> (2010)
10. Fuggetta, A.: Software process: a roadmap. In: *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pp. 25–34. ACM Press, New York (2000). <http://doi.acm.org/10.1145/336512.336521>
11. Garijo, F.J., Gómez-Sanz, J.J., Massonet, P.: The MESSAGE methodology for agent-oriented analysis and design. In: Henderson-Sellers, B., Giorgini, P. (eds.): *Agent Oriented Methodologies*, Chap. 8, pp. 203–235. Idea Group Publishing, Hershey (2005). <http://www.idea-group.com/books/details.asp?id=4931>
12. Ghezzi, C., Jazayeri, M., Mandrioli, D.: *Fundamental of Software Engineering*, 2nd edn. Prentice Hall, Englewood Cliffs (2002)
13. IEEE-FIPA: Design Process Documentation Template. <http://fipa.org/specs/fipa00097/SC00097B.pdf> (2012)
14. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. In: Henderson-Sellers, B., Giorgini, P. (eds.): *Agent Oriented Methodologies*, Chap. IX, pp. 236–276. Idea Group Publishing, Hershey (2005). <http://www.idea-group.com/books/details.asp?id=4931>
15. Picard, G., Bernon, C., Gleizes, M.P.: Cooperative agent model within ADELFE framework: an application to a timetabling problem. In: Jennings, N.R., Sierra, C., Sonenberg, L., Tambe, M. (eds.) *Autonomous Agents and Multi Agent Systems*, vol. 3, pp. 1506–1507. ACM Press, New York (2004). <http://doi.ieeecomputersociety.org/10.1109/AAMAS.2004.10200>
16. SODA: Home page. <http://soda.apice.unibo.it>. <http://soda.alice.unibo.it> (2012)
17. Sommerville, I.: *Software Engineering*, 8th edn. Addison-Wesley, Reading (2007)
18. Wood, M.F., DeLoach, S.A.: An overview of the multiagent systems engineering methodology. In: Ciancarini, P., Wooldridge, M.J. (eds.) *Agent-Oriented Software Engineering*. *Lecture Notes in Computer Science*, vol. 1957, pp. 207–221. Springer, Berlin (2001). <http://www.springerlink.com/link.asp?id=0pw8rqj5kpbdnflx>. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers
19. Zambonelli, F., Omicini, A.: Challenges and research directions in agent-oriented software engineering. *Auton. Agent Multi Agent Syst.* **9**(3), 253–283 (2004). doi:10.1023/B:AGNT.0000038028.66672.1e. <http://journals.kluweronline.com/article.asp?PIPS=5276775>. Special Issue: Challenges for Agent-Based Computing
20. Zambonelli, F., Jennings, N., Wooldridge, M.: Multiagent systems as computational organizations: the Gaia methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.): *Agent Oriented Methodologies*, Chap. 6, pp. 136–171. Idea Group Publishing, Hershey (2005). <http://www.idea-group.com/books/details.asp?id=4931>

The IEEE-FIPA Standard on the Design Process Documentation Template

Massimo Cossentino, Vincent Hilaire, Ambra Molesini,
and Valeria Seidita

Abstract

Nowadays, it is a matter of fact that a “one-size-fit-all” methodology or design process useful and fitting every kind of problem, situation, or design context does not exist. (Situational) Method Engineering (SME) discipline aims at determining techniques and tools for developing ad hoc design methodologies. SME mainly and highly focuses on the reuse of portion of existing design processes or methodologies (*the method fragments*). In order to have means for creating SME techniques and tools and for creating new design processes, some key elements are needed: a unique process metamodel for representing design processes and fragments, a proper template for the description of AO design processes and for the description of method fragments. The FIPA Design Process Documentation and Fragmentation Working Group gave an important contribution to the SME research area in terms of the IEEE-FIPA standard Design Process Documentation Template (DPDT) that provides a standardized template for the description of design processes.

M. Cossentino

Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, Palermo, Italy
e-mail: cossentino@pa.icar.cnr.it

V. Hilaire

IRTES-SET, UTBM, UPR EA 7274, 90 010 Belfort cedex, France
e-mail: vincent.hilaire@utbm.fr

A. Molesini

DISI - Alma Mater Studiorum – Università di Bologna, Viale Risorgimento 2,
40136 Bologna, Italy
e-mail: ambra.molesini@unibo.it

V. Seidita (✉)

Dipartimento di Ingegneria Chimica, Gestionale, Informatica, Meccanica, University of Palermo,
Palermo, Italy
e-mail: valeria.seidita@unipa.it

1 Introduction

The Design Process Documentation Template (DPDT) is the result of the work done within the FIPA Design Process Documentation and Fragmentation Working Group.¹ This Working Group was created and is inserted in the research context of Method Engineering and Situational Method Engineering for the creation of ad hoc agent-oriented methodologies.

It is currently admitted in both mainstream software engineering and agent-oriented software engineering that there is not a “one-size-fits-all” methodology or design process for designing and developing systems able to solve any kind of problems. It is to date a factual data that software systems are strictly dependent on the environment they will work, on the people who will use them, and above all on the class of problems they will solve; for instance software systems for high-risk military applications are intrinsically different from e-commerce applications, or from health care, etc. The different features that software systems present greatly affect the designer choice about the right design process (or methodology) to use.

Designers often spend a lot of time in studying, adapting, and customizing, when possible, existing design processes, thus also increasing software development costs. It would be useful to have the possibility (techniques, tools, and so on) for the designer to create the design process best fitting his/her needs in a quick and useful way.

In order to overcome this problem, in the 1990s the Method Engineering discipline rose. Method Engineering is the “engineering discipline to design, construct and adapt methods, techniques and tools for the development of systems”; this is the definition generally accepted since 1996 [1]. Then several researchers coined the term Situational Method Engineering (SME) for indicating the part of ME dealing with the creation of method(ologies) (or design process) for specific situations [7, 14, 15]. SME provides means for constructing ad hoc software engineering design processes following an approach based on the reuse of portions of existing design processes, the so-called *method fragments*, stored in a repository called *method base*.

Method fragment is the main concept in an SME approach and several different definitions exist in the literature [1, 5, 9, 13]. All the SME approaches in the literature are based on the assumption that a generic SME process is composed of three main phases: method requirements engineering, method design, and method construction [4]. During the method requirements engineering the method engineer analyzes all the elements useful for identifying the right method fragments to be retrieved from the repository. During method design and method construction phases, the method fragments are, respectively, selected and modified, if necessary, and then assembled in the new method(ology).

One need, raised by this type of approaches, is having fragments description or documentation available in a way that aids a method engineer in easily choosing

¹<http://www.fipa.org/subgroups/DPDF-WG.html>

among a pertinent subset of fragments among existing ones in order to build a new process. To date, several method fragments exist but they are not described in a homogeneous fashion, thus severely affecting the selection and assembly. A unique, versatile, and standard description and documentation of method fragments would inevitably derive from a standard description of design processes.

The FIPA Design Process Documentation and Fragmentation Working Group has three main goals:

- identifying the most suitable process metamodel and notation for representing existing design processes and for representing fragments themselves.
- defining a proper template for the description of agent-oriented design processes.
- defining a proper template for the description of method fragment structure.

An important contribution to the first objective has been identified in the OMG specification, the Software Process Engineering Metamodel 2.0 (SPEM) [11] that meets the definition of design process the WG adopts and provides means for easily representing the elements it is composed of. Roughly speaking, a design process represents *the work* to be done by *roles* in order to deliver *products*. The work to be done is supposed to be divided into three main levels: phases, activities, and tasks. SPEM provides means for representing design processes using the following elements: phase, activity, task, role, and work product. Moreover, in order to have a complete set of concepts for representing design processes and above all because of agent-oriented specific needs, some extensions were needed and were identified in the work of Cossentino and Seidita [2, 12]. Here the notion of MAS metamodel, MAS metamodel concepts, work product kind, and work product content diagram have been introduced.

The IEEE-FIPA standard DPDT [6]—shown below—is the contribution to the second objective of the WG. Indeed, it provides a standardized template that allows the description of design processes. A standardized description is a good feasible way to help method engineers in fragmenting existing design processes and thus choosing pertinent fragments; it is also particularly relevant to researchers and practitioners working on SME approaches. Even if this template addresses the documentation of full processes—it is not intended to document portions of processes—the definition of substantial fragments of a process is facilitated if the whole process has been previously described in a standard way that makes identification and definition of fragments easier. Hence, this phase is preparatory and preliminary to the third WG objective.

The DPDT intends to solve the process description problem by proposing a standard way to describe design processes in order to study and reuse them as they are, or for fragmentation and extraction of method fragments. Even if the idea of this specification was born in the context of multi-agent system design processes, this template has been conceived without considering any particular process or methodology, and this should guarantee that all processes can be documented using the DPDT. Moreover, the DPDT is also neutral in regard to: (1) the MAS (or system) metamodel and/or the modeling notation adopted in the process deliverables as well as in describing the process itself; (2) the content of the process since it may (or not) include activities such as testing, maintenance, and so on; (3) the process life cycle,

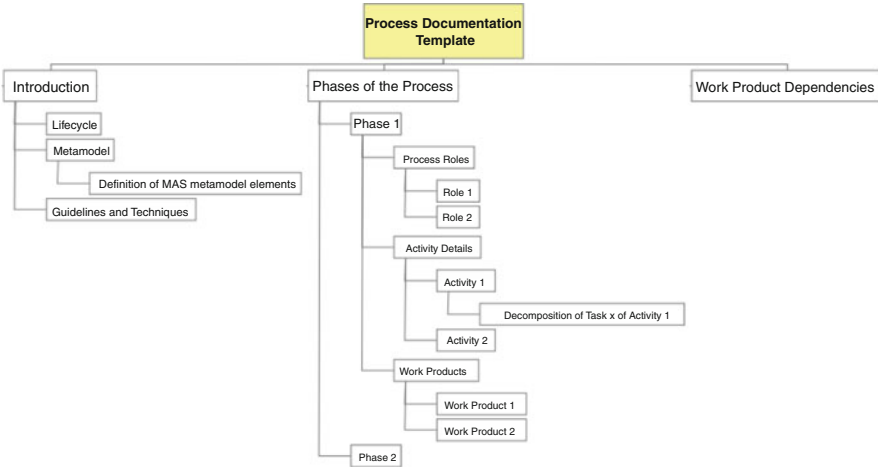


Fig. 1 The design process documentation template structure [6]

since waterfall, iterative-incremental, spiral, and many other kinds of life cycle are fully supported.

The template has a simple structure resembling a tree (see Fig. 1). This implies that the documentation is built in a natural and progressive way, initially addressing the process general description and the metamodel definition, which constitute the root elements of the process itself. After that, the process phases are described as branches of the tree, and also thinner branches like activities or sub-activities are documented. This means the template can support complex processes and very different situations.

Finally, in the DPDT the notation used for modeling some aspects of the process is not considered fundamental. Nevertheless, the use of standards is important. In particular, as already said, the OMG's standard Software Process Engineering Metamodel 2.0 (SPEM) is suggested for modeling such parts of the template. In any case, this does not mean that other standards cannot be used with the template as far as the concepts implied and the underlying view of the system proposed by the work product is reflected in the notation used.

To sum up, the goal of the DPDT is twofold: providing method designers with templates describing design processes in a standard and homogeneous fashion thus highlighting the main concepts they are composed of, supporting the fragmentation process, and the choice of fragments.

In the rest of this chapter we first introduce some basic notions that are useful for understanding the document structure (Sect. 2); then we describe the DPDT outline, and how each part of the document has to be created (Sect. 3).

2 Documentation Template Basic Notions

Before going on with the description of the IEEE FIPA SC00097B specification it is necessary to introduce the reader with some important notions related to the Multi-agent Systems (MAS) metamodel (Sect. 2.1), the set of work product kind used to detail the outcome (work product) of a specific portion of design process, and the work product content diagram that gives the possibility to model all the concepts a specific outcome is devoted to manage/produce (Sect. 2.2).

First of all it is worth noting the notion of design process on which the whole WG's work is based. According to Fuggetta [3], a design process is “*the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy and maintain (evolve) a software product*”. In such a view, a design process includes components devoted to describe the work—the set of activities and/or tasks—to be done for pursuing an objective, hence delivering some kind of work products, and the stakeholder responsible for doing this work.

The following subsections will introduce the multi-agent system metamodel, some SPEM extensions adopted (better say suggested) to model the process, and the design process documentation template structure suggested by specification SC00097B.

2.1 The Multi-agent System Metamodel

Metamodeling is an essential foundation of Model Driven Engineering (MDE) proposed by OMG² where the language for describing metamodels is the Meta Object Facility (MOF) [8,10] standard. Metamodeling consists in identifying a set of concepts (or elements) to use for describing the properties of models. A model is an instance of a metamodel and it is an abstraction of real world's elements, phenomena, etc. represented in the outcome of the design process phases/activities/tasks. In plain terms, “metamodel is a model of models” that is composed of concepts and relationships among them used for ruling the creation of models.

Some agent-oriented methodologists represent the metamodel of their own methodology by separating the elements into three logical areas: the *Problem domain*, the *Agency domain*, and the *Solution domain*.

The *Problem domain* includes concepts coming from the world where the multi-agent system has to operate; here concepts like requirements, scenario, and so on may be found.

The *Agency domain* collects concepts used to define an agent-based solution for the class of problems the methodology is devoted to address; concepts like agent, role, group, society and communication can be found.

²<http://www.omg.org/index.htm>

The *Solution domain* is composed of concepts representing a well specific mapping among agency domain concepts and the chosen implementation platform, hence the code of the solution.

The metamodel of a methodology gives information about all the concepts that the designer has to manage during his/her work and has to instantiate in, at least one work product. Moreover, the metamodel relationships represent the intra and inter work products dependencies among metamodel concepts, thus establishing logical connections among different parts of a methodology (or of different methodologies).

2.2 The SPEM 2.0 Extensions

The SPEM 2.0 covers almost all the work essentials needed by this working group for representing design processes. Only few extensions to SPEM 2.0 proved to be necessary and were identified in [2, 12]; they extend the elements of the *Process with Method Package* (see [11] for a detailed description of all the SPEM packages). Some details about these extensions will be provided in the following paragraphs.

Work Product Kind

Different processes adopt different modeling styles, thus using different kinds of work products in their flow of work. It is obviously very useful to clearly identify whether a work product follows a specific notation and it points out the structural or the behavioral aspects of the system; or whether it is “intangible” or does not present a specific defined form or more if it aggregates different kinds of work product. What is important to note is that the approach adopted by this working group is hardly grounded on the hypothesis that a work product is something that is produced, consumed or modified during the execution of a portion of work and that each work product serves to model at least one metamodel concept. Moreover, for each metamodel concept drawn in a work product, a set of possible *design actions* may be identified, giving information about the kind of work to be done during a specific portion of a design process.

The set of work product kinds used to detail the outcome of a specific portion of process is:

- *Behavioral*, it is a graphical kind of work product and it is used to represent a dynamic view of the system (for instance an UML sequence diagram representing the flow of messages among agents along time).
- *Structural*, it is a graphical kind of work product too, and it is used for representing a structural view of the system (for instance a UML class diagram).
- *Structured*, it is a text document ruled by a particular template or grammar, for instance a table or a programming code document.
- *Free*, it is a document freely written in natural language.
- *Composite*, this work product can be made by composing the previous work product kinds, for instance a diagram with a portion of text used for its description.

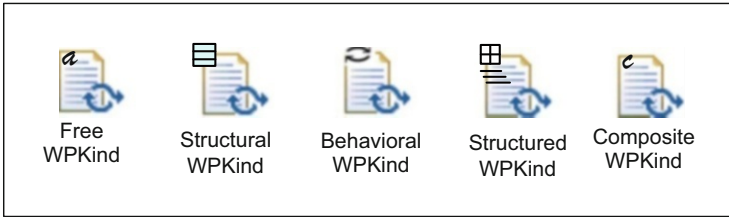


Fig. 2 SPEM 2.0 extensions icons

To complete the presented SPEM 2.0 extensions, the icons shown in Fig. 2 were created.

Work Product Content Diagram

For each work product of the methodology, the work product content diagram gives an overview of the metamodel elements there reported; for each metamodel element the actions to be done are also specified. This diagram is complementary to the textual guidelines useful for drawing the work product.

Design actions that can be made on a metamodel element while composing a work product are:

- *Define*: instantiating an element in the work product. This corresponds to create a new model element. This modeling action is labelled with a **D**.
- *Relate*: instantiating a relationship between two metamodel elements.
- *Quote*: reporting an already defined metamodel element in the work product. Sometimes a newly defined element in a work product has to be related with a previously defined one, this latter element is therefore quoted and related to the defined one. The same action may be done in the metamodel relationships.
- *Refine*: this happens when an already defined element in the work product is in some way refined. For instance, consider to have already defined the A metamodel element, refining that might mean adding an attribute to it in its representation in a class diagram.

Figure 3 shows an example of a work product content diagram; the adopted notation implies a *package* for representing the work product, a *class* for representing the metamodel element, a *label* for representing the action made on the element and *lines* for representing relationships among elements.

The work product content diagrams collect only the elements managed during a portion of design work that are also reported in the work product. During the design process enactment, there can be elements of the metamodel used for reasoning on the output to be produced or used as an input for defining other elements; these elements are not reported in the work product content diagram since they do not appear in the work product.

The DPDT standard does not prescribe the description of one work product content diagram for each activity of the methodology but a unique work product content diagram per phase.

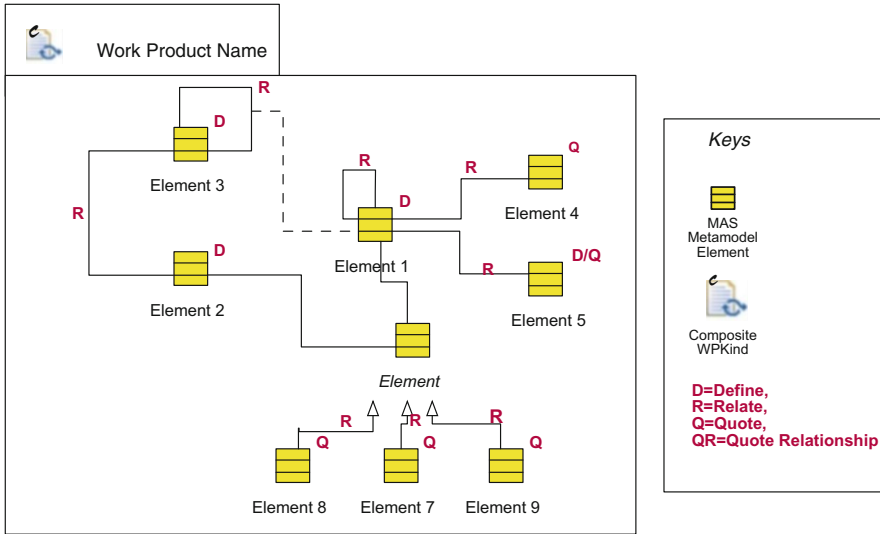


Fig. 3 An example of a work product content diagram

Further refinements and discussions about the notion of metamodel, design actions, and work product content diagram can be found in [2].

3 The Design Process Documentation Template Structure

As it can be seen in Fig. 1 the description of a design process by using this IEEE FIPA standard goes through three main sections: the introduction (see Sect. 3.1), the phases of the process (see Sect. 3.2) and the work products dependency section (see Sect. 3.3). As already said, the work to be done in the process is supposed to be divided into three main levels: phases, activities, and tasks. Phases are composed of activities, which in turn are composed of other activities or tasks. In addition, from a work product point of view, phases are supposed to deliver a major artefact (for instance a requirement analysis document or a detailed design document). Activities are supposed to produce finer grained artifacts (like a diagram often complemented by a text description of the elements reported within it). Tasks are supposed to concur to the definition of activity-level artifacts. The first and the second section of the template contain several subsections, each one structured according to a specific format:

Goal: describing the goal addressed in this part of the documentation. Examples of goals include the documentation of the general philosophy that is behind a design process or the description of the involved stakeholders.

Structure: describing what is to be reported in this part of the process documentation. This may include diagrams as well as the textual description of specific process elements.

Guidelines: describing best practices suggested for a good application of the process documentation template or techniques about how to perform the prescribed work.

Example: addressing an example of outcome produced by the process.

3.1 The Introduction Section

The DPDT *Introduction* section aims at introducing the philosophy, basic ideas, scope, and limits of the process. It is composed of three subsections:

- **Global process overview (Life cycle):** the aim of this subsection is to organize the process phases according to the selected life cycle (or process model) in order to provide a bird-eye view of the whole process at the highest level of details.
- **Metamodel:** the aim of this subsection is to provide a complete description of the MAS metamodel adopted in the process with the definition of its composing elements.
- **Guidelines and techniques:** the goal of this subsection is to provide some guidelines on how to apply the process or to clarify techniques to be used in process enactment whose validity spreads throughout the whole process.

It is very important that the Introduction gives an overview of the methodology from the creators point of view in order to maintain a feeling with the original methodology, thus possibly including original figures, reference materials, and documents.

3.2 The Phases of the Process Section

The *Phases of the Process* section aims at pointing out the process view of the methodology by describing, in a top-down fashion, the activities and comprised tasks each phase is composed of. Moreover, for each phase the list of the involved process roles with the related responsibilities and the list of work products is given.

One different section should be devoted to the discussion of each phase. Indeed, each phase should be studied from a process-oriented point of view: workflow, work products, and process roles are the center of the discussion. The workflow is detailed by means of activities and one subsection for each activity is devoted to describe activities/tasks composing that. The description includes a SPEM activity diagram reporting involved roles (as swim lanes). Further details about each activity can be provided in additional sections. In particular, the subsection discussing each phase should:

- introduce the phase workflow by using a SPEM activity diagram. It reports activities that compose the phase, and it includes a quick description of work products and roles.
- introduce a SPEM diagram reporting the structural decomposition of the activities in terms of the involved elements: tasks, roles, and work products.

The subsection describing each activity should include details about tasks and sub-activities that may be illustrated by a stereotyped UML Activity Diagram. This kind of diagram explains the execution of complicated Tasks by denoting the possible sequences of Steps, which are identified by the «steps» stereotype. Moreover, each diagram is discussed in a separated paragraph that explains the illustrated steps and their relations. Finally a table is required for summarizing:

- Activity name
- Tasks/Sub-activity
- Task/Sub-activity description
- Roles involved

As regards the subsection for the description of work products, it has a twofold aim: (1) detailing the information content of each work product by representing which MAS metamodel elements are reported in it and which design actions are performed on them through one work product content diagram for each phase; (2) describing the notation adopted by the process in the specific work product, also using an example to show it.

Moreover, a table is used to further detail each work product through three columns:

- name of the work product
- description of the content
- classification of the work product according to the already cited paper (categories: Free/Structured Text, Behavioral, Structural, and Composite)

3.3 The Work Products Dependencies Section

The goal of this section is highlighting the dependencies among all the work products produced during the enactment of the design process. Dependencies among work products reflect the dependencies among the portions of work delivering them. Indeed, if the information presented in one work product is used for producing another one, then the two portions of work are obviously related to each other. Work Products Dependencies are represented by a classic diagram (specified by SPEM) called *Work Product Dependencies diagram*.

This diagram can prove to be very important for different reasons, for instance for project managers who have to reschedule project activities according to new needs occurring at design time or for designers who want to keep track of changes in the development process and in the system model documents. For the sake of Situational Method Engineering approaches and for the objectives of this working group the Work Product Dependencies diagram helps the method engineer in identifying dependencies among the fragments that he/she might extract from a design process and above all in identifying the set of MAS metamodel elements each fragment needs.

In fact, it is to be noted that according to the importance that is paid to the MAS metamodel in the Agent-Oriented Software Engineering (AOSE) field, the real input of each portion of a process is a subset of model elements (instances of the MAS metamodel) that constitute the portion of design reported in the input documents.

References

1. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. *Inf. Softw. Technol.* **38**(4), 275–280 (1996)
2. Cossentino, M., Seidita, V.: Metamodeling: representing and modeling system knowledge in design processes. In: Proceedings of the 10th European Workshop on Multi-agent Systems, EUMAS 2012, pp. 103–117, 17–19 December 2012
3. Fuggetta, A.: Software process: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland, 4–11 June 2000, pp. 25–34. ACM Press, New York (2000)
4. Gupta, D., Prakash, N.: Engineering methods from method requirements specifications. *Requir. Eng.* **6**(3), 135–160 (2001)
5. Henderson-Sellers, B.: Method engineering: theory and practice. In: Karagiannis, D., Mayr, H.C. (eds.) *Information Systems Technology and its Applications*. 5th International Conference ISTA'2006, 30–31 May 2006, Klagenfurt. LNI, vol. 84, pp. 13–23. GI (2006)
6. IEEE-FIPA. Design Process Documentation Template. <http://fipa.org/specs/fipa00097/SC00097B.pdf>, January 2012
7. Kumar, K., Welke, R.J.: Methodology engineering: a proposal for situation-specific methodology construction. In: Cotterman, W.W., Senn, J.A. (eds.) *Challenges and Strategies for Research in Systems Development*, pp. 257–269. Wiley, New York (1992)
8. Miller, J., Mukerji, J.: Mda guide version 1.0.1. Technical Report omg/2003-06-01, Object Management Group (2003)
9. Mirbel, I., Ralyté, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requir. Eng.* **11**(1), 58–78 (2006)
10. MOF. OMG meta object facility home page. <http://www.omg.org/technology/documents/formal/mof.htm>
11. Object Management Group. Software & Systems Process Engineering Meta-model Specification 2.0. <http://www.omg.org/spec/SPEM/2.0/PDF>, April 2008
12. Seidita, V., Cossentino, M., Gaglio, S.: Using and extending the spem specifications to represent agent oriented methodologies. In: Luck, M., Gomez-Sanz, J.J. (eds.) *Agent-Oriented Software Engineering IX*, pp. 46–59. Springer, Berlin (2009)
13. Seidita, V., Cossentino, M., Hilaire, V., Gaud, N., Galland, S., Koukam, A., Gaglio, S.: The metamodel: a starting point for design processes construction. *Int. J. Softw. Eng. Knowl. Eng.* **20**(4), 575–608 (2010)
14. Slooten, K., Brinkkemper, S.: A method engineering approach to information systems development. In: Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process, pp. 167–186. North-Holland Publishing, Como (1993)
15. ter Hofstede, A.H.M., Verhoef, T.F.: On the feasibility of situational method engineering. *Inf. Syst.* **22**(6/7), 401–422 (1997)

ADELFE 2.0

N. Bonjean, W. Mefteh, M.P. Gleizes, C. Maurel, and F. Migeon

Abstract

ADELFE is a French acronym that means “Toolkit for Designing Software with Emergent Functionalities” (“Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente” in French). ADELFE methodology is dedicated to applications characterized by openness and the need of the system adaptation to an environment. Its main goal is to help and guide any designer during the development of an Adaptive Multi-agent System (AMAS). An AMAS is characterized by the following points: it is plunged into an environment and composed of interdependent agents, each agent carries out a partial function and the agents, organization during runtime makes the system realize an emergent function. Actually, an agent is locally cooperative, i.e. it is able to recognize cooperation failures called Non-cooperative Situations (NCS, which could be related to exceptions in classical programs) and treat them.

ADELFE includes five Work Definitions that were initially inspired from the Rational Unified Process (RUP) and gathers 21 activities, producing or refining 12 work products. These products are aggregating modelling diagrams or structured or free text. ADELFE, which is a Model-Driven (model-centred) development method, is not hardly dependent on Domain Specific Modelling Languages (DSML) but currently the recommendation is to use UML2 for general activities and to use AMASML (AMAS Modelling Language) and SpeADL (Species-based Modelling Language) for specific activities appearing in Analysis, Design or Implementation phases.

N. Bonjean (✉) • W. Mefteh • M.P. Gleizes • C. Maurel • F. Migeon
Institut de Recherche en Informatique de Toulouse (IRIT), Toulouse, France
e-mail: bonjean@irit.fr

1 Introduction

ADELFE is a French acronym that means “Toolkit for Designing Software with Emergent Functionalities” (“Atelier de Développement de Logiciels à Fonctionnalité Emergente” in French). The main goal of ADELFE is to help and guide any designer during the development of an Adaptive Multi-agent System (AMAS). Adaptive software is used in situations in which the requirements are incompletely expressed, the environment is unpredictable or the system is open. In these cases, designers cannot implement a global control on the system and cannot list all situations that the system encounters. In these situations, ADELFE guarantees that the software is developed according to the AMAS theory. This theory, based on self-organizing multi-agent systems, enables one to build systems in which agents only pursue a local goal while trying to keep cooperative relations with their neighbouring agents. An AMAS is characterized by the following points: it is plunged into an environment and composed of interdependent agents, each agent carries out a partial function and the agents organization during runtime makes the system realize an emergent function.

In the following, the ADELFE process is described by initially considering its whole process and then its five phases, which gather 21 activities, producing or refining 12 work products.

Since 2003, ADELFE has been used in many academic and industrial projects, for a total of more than 20 AMASs produced. Recently, ADELFE has been slightly modified in order to integrate last research results on Multi-agent Oriented Software Engineering and practical usages related by the industrial ADELFE partners. This chapter presents this up-to-date version of the method.

As it can be seen in the following sections, ADELFE is composed of several activities dedicated to Adaptive Multi-agent Systems. Considering this, it contains junction activities where the designer has to control if their problem requires an MAS solution or, even more, an AMAS solution. In the negative case, the design should be continued with a traditional process or a process dedicated to the problem characteristics. This documentation does not describe such activities and focuses only on the process parts dedicated to AMAS development.

Finally, it is important to notice that ADELFE is a Model-Driven (model-centred) Development method which is still in progress. Works on fragmentation, simulation, formal methods and AMAS patterns are feeding the method every year in order to improve the development of complex systems based on the AMAS. As the maturity of these topics is not sufficient to be included in this chapter, we left their presentation in research papers.

The following are the relevant references for the ADELFE process and the ADELFE extensions:

- Bernon, C.; Gleizes, M.-P.; Peyruqueou, S.; Picard, G.; ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering International Workshop on Engineering Societies in the Agents World (ESAW), Madrid, Spain, 16/09/2003–17/09/2003, Springer-Verlag, 2003, 156–169

- Bernon, C.; Camps, V.; Gleizes, M.-P.; Picard, G. Designing Agents' Behaviours within the Framework of ADELFE Methodology International Workshop on Engineering Societies in the Agents World (ESAW), Imperial College London, 29/10/2003–31/10/2003, Springer-Verlag, 2003, 311–327
- Rougemaille, S.; Arcangeli, J.-P.; Gleizes, M.-P.; Migeon, F. ADELFE Design, AMAS-ML in Action International Workshop on Engineering Societies in the Agents World (ESAW), Saint-Etienne, 24/09/2008–26/09/2008, Springer-Verlag, 2008
- Bernon, C.; Gleizes, M.-P.; Picard, G., Enhancing Self-Organising Emergent Systems Design with Simulation International Workshop on Engineering Societies in the Agents World (ESAW), Dublin, 06/09/2006–08/09/2006, Springer-Verlag, 2007, 4457, 284–299
- Lemouzy, S.; Bernon, C.; Gleizes, M.-P. Living Design: Simulation for Self-Designing Agents European Workshop on Multi-Agent Systems (EUMAS), Hammamet, 13/12/07–14/12/07, Ecole Nationale des Sciences de l'Informatique (ENSI, Tunisie), 2007
- Mefteh, W.; Migeon, F.; Gleizes, M.-P.; Gargouri, F. Simulation Based Design International Conference on Information Technology and e-Services, Sousse, Tunisie, 2012
- Bonjean, N.; Gleizes, M.-P.; Maurel, C.; Migeon, F., Forward Self-Combined Method Fragments. Workshop on Agent Oriented Software Engineering (AOSE 2012), Valencia, Spain, 04/06/2012–08/06/2012, Jorg Muller, Massimo Cosentino (Eds.), IFAAMAS, p. 65–74, June 2012

1.1 The ADELFE Process Life Cycle

ADELFE includes five Work Definitions (WD1–5) (see Fig. 1):

- Preliminary Requirements (WD1): This phase represents a consensus description of specifications between customers, users and designers on what must be and what must give the system its limitations and constraints.
- Final Requirements (WD2): In this work definition, the system achieved with the preliminary requirements is transformed into a use cases model, and the requirements (functional or not) and their priorities are organized and managed.
- Analysis (WD3): The analysis begins with a study or analysis of the domain. Then, identification and definition of agents are processed. The Analysis Phase defines an understanding view of the system, its structure in terms of components, and identifies if the AMAS theory is required.
- Design (WD4): This phase details the system architecture in terms of modules, subsystems, objects and agents. These activities are important from a multi-agent point of view in that a recursive characterization of a multi-agent system is achieved at this point.
- Implementation (WD5): Implementation of the framework and agent behaviours is produced in this work definition.



Fig. 1 The ADELFE process phases

It is important to notice that ADELFE is not a simple waterfall process but includes loops and increments that are not depicted in the graphical representation shown in Fig. 1. For example, at the end of each phase, a validation step is executed, which requires a possible transition towards an activity previously passed.

Each phase produces at least one document that is aggregated from modelling diagrams or from structured or free text. ADELFE is not hardly dependent on Domain Specific Modelling Languages (DSML) but currently the recommendation is to use UML2 for general activities and to use AMAS Modelling Language (AMASML) and Species-bAsed moDelling Language (SpeADL) for specific activities appearing in Analysis, Design or Implementation phases.¹

1.2 The ADELFE MAS Metamodel

1.2.1 Definition of MAS Metamodel Elements

The ADELFE method is composed of various tools based on model-driven development. To support model transformation and DSML editors an ADELFE metamodel has been defined. However, this metamodel is too much based on natural language (precise and complex) to support a designer guidance.

The ADELFE MMM is organized according to the five phases comprising the process. We give in the following a short description of the main elements of this metamodel that is presented in five diagrams to simplify the layout and the discussion.

The Preliminary Requirements Phase (see Fig. 2) focuses on acquiring information about the client and their needs. Therefore, a consensual description of the problem is made in terms of functional and non-functional requirements, keywords, limits and constraints of the system. A business model with business concepts and business activities is defined to complement the documentation.

In the Final Requirements Phase (see Fig. 3), the objective is to validate the requirements and to detail the need through a description of the actors, a use case model and scenarios. The end of the phase is dedicated to the characterization of the system environment and the identification of cooperation failures among the system interactions. This leads to a conclusion on the MAS adequacy to treat the problem of the client.

¹See the SMAC Team website (<http://www.irit.fr/-Equipe-SMAC>) for more information on DSML.

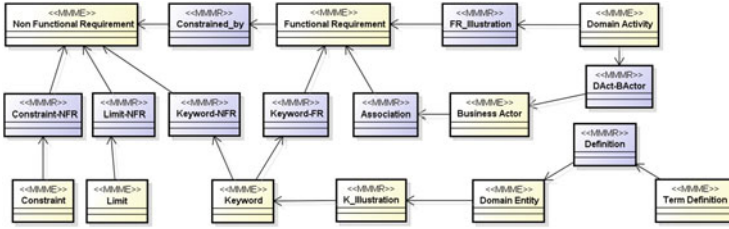


Fig. 2 The preliminary requirements phase MAS metamodel

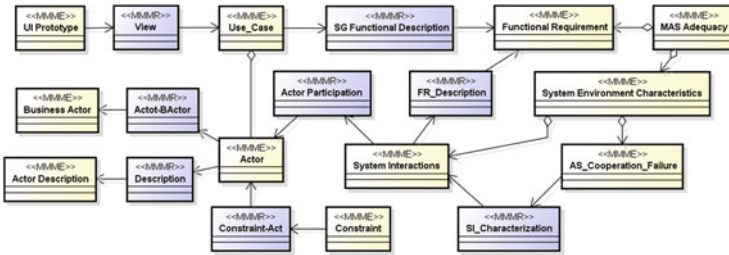


Fig. 3 The final requirements phase MAS metamodel

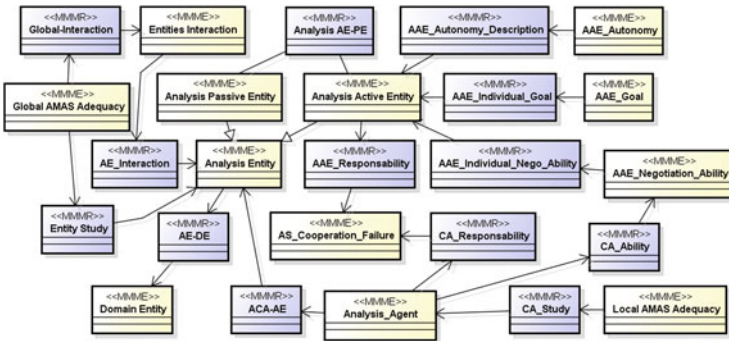


Fig. 4 The analysis phase MAS metamodel

During the Analysis Phase (see Fig. 4), the entities are characterized as passive or active and their interactions are described. The work product obtained enables an AMAS analyst to conclude on the adequacy (or not) of the AMAS to deal with the problem. If the result is positive, all the interactions between the entities are described and cooperation failures are identified. From this information, agents (according to the definition of agent in ADELFE, see Sect. 2.3) are identified and local AMAS adequacy is studied to conclude the phase.

The heaviest phase of ADELFE is the Design Phase (see Fig. 5) which consists in defining the multi-agent oriented architecture of the solution. It starts with the

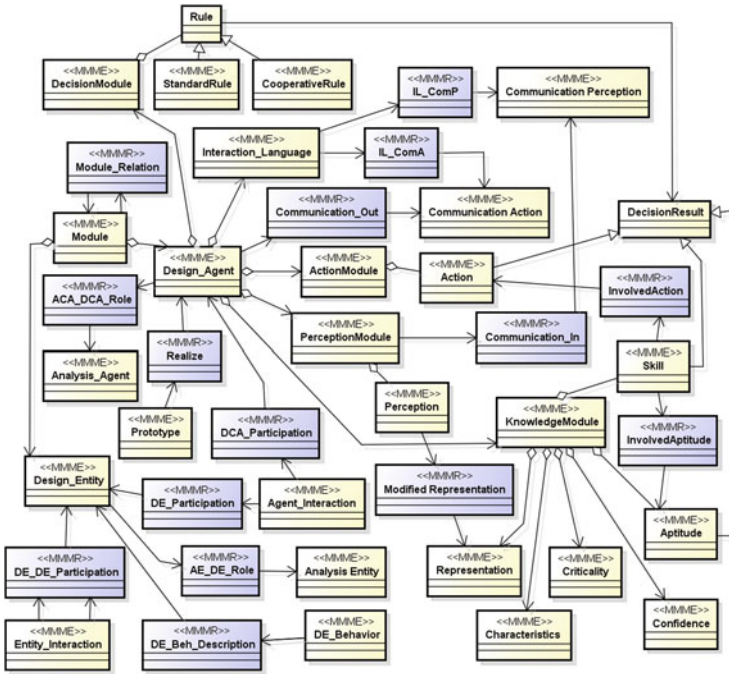


Fig. 5 The design phase MAS metamodel

definition of a module view. Then, all the communication acts are defined in order to define precisely the entity interactions and agent interactions that will be useful to complete the module view and to define a component-connector view of the agents with their neighbourhood (agents, active and passive entities). The definition of the structure and behaviour of the agents is made in two steps which lead to the definition of the knowledge module, the action module, the perception module and the decision module. These two steps concern respectively the nominal behaviour of the agent, which enables the agent to reach its goal, and the cooperative behaviour which enables the agent to self-adapt to abnormal situations. Finally, a prototype is defined to validate the result.

Currently the last phase of ADELFE, the Implementation Phase (see Fig. 6), focuses on the definition of the component-oriented architecture that will support the design. It is mainly composed of automated activities for model or code generation. However, in this document, we deliberately describe the process as manual operations in order to give more details on the activities and in order to simplify the metamodel. The implementation of an AMAS is not dependent on any MAS platform. On the contrary, it is recommended to produce a dedicated framework in order to gain in software quality. Actually, this framework is defined in terms of components (with provided and required ports, composite components,

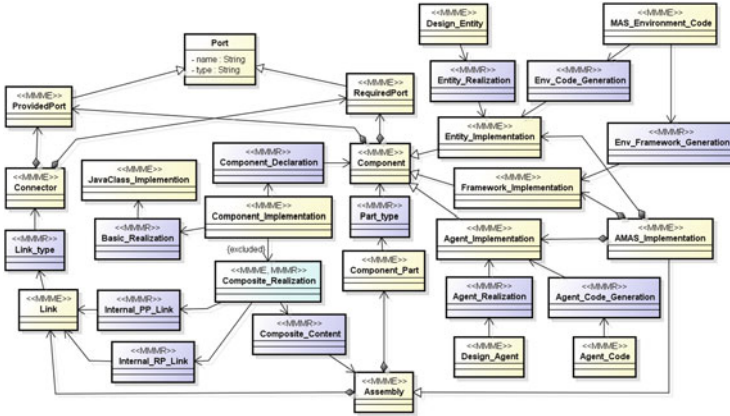


Fig. 6 The implementation phase MAS metamodel

assembling) which are specified and implemented. First, the implementation focuses on the components defining entities, action and perception modules. Finally, decision module is implemented with standard behaviour rules and cooperative behaviour rules.

Like in every process, unit testing, integration testing and functional testing are taken into account but not described here.

1.2.2 Definition of MAS Metamodel Elements

The table below gives a set of concepts definitions related to AMAS theory and ADELFE method. It covers the entire MAS metamodel used during Requirements, Analysis and Design.

Concept	Definition	Referred Concepts	Domain
(Software) System	A (software) system is the term describing the software to be produced, the application to be designed. Anything outside the system is called system environment.	System Environment	Requirements
System Environment	It is the entire environment into which the software system is plunged and which is not under design. In UML, the term Actor (primary or secondary) is often used to characterize the environment. Of course, the frontier between environment and software system will use software entities that will deal with the interactions between environment and system.	Software System	Requirements

(continued)

Concept	Definition	Referred Concepts	Domain
MAS Environment	The MAS environment is composed of all software elements that surround the MAS and which are not agents. The elements of the environment are called MAS entities (active or passive) and all have at minimum an interaction that is defined by means of sensors or effectors.	Passive Entity, Active Entity	Analysis
Agent Environment	The environment of an agent is the union of its neighbourhoods during its life. It represents all the knowledge it has on other agents and on the MAS environment.	Agent Neighbourhood, Agent	Analysis
Agent Neighbourhood	The neighbourhood of an agent is a part of the agent's environment at a particular time.	Agent Environment, Agent	Analysis
Multi-Agent System (MAS)	An MAS is the set of elements, called agents, which are not part of the environment. In an MAS, no agent can be isolated, that is to say without any link with another system component. Therefore, it can be considered as the set of agents that communicate (directly or via the environment) to achieve a common goal.	Goal, Agent	Analysis
Goal	The goal is an objective set by the designer to an agent or to the entire system.	Agent, (Software) System	Analysis
Passive Entity	In the MAS environment, passive entities are related to resources or to data. This implies that they have no autonomy and that a state transition can only be the result of an interaction with another system component. Moreover, a passive entity is unable to send or receive messages.	MAS Environment	Analysis
Active Entity	Unlike a passive entity, an active entity is given behavioural autonomy, allowing it to change state without necessarily interacting with another entity. An active entity can send messages, possibly proactively, and receive messages.	Passive Entity	Analysis
(Cooperative) Agent	In ADELFE, an element of an MAS (i.e. an element that is not part of the MAS environment) is an agent. This agent is characterized by a cooperative attitude.	MAS, Cooperative Attitude	Analysis

(continued)

Concept	Definition	Referred Concepts	Domain
Cooperation	It is a behavioural principle for an entity that avoids being placed in a situation of misunderstanding, ambiguity, incompetence, unproductiveness, conflict, concurrence or uselessness. This principle is applied to agents designed with ADELFE, but can also describe the behaviour of the software system as a whole. The definition of cooperation goes beyond the simple sharing of resources or collaboration. This cooperation includes all behaviours that allow the agent to prevent and to resolve conflicts that occur during system execution.	Agent, Software System	Analysis
Cooperative Attitude	An agent has a cooperative attitude when its activity tends to give priority to anticipate and solve all the Non-cooperative Situations (NCS) it might encounter with its environment. This implies the following properties: (i) <i>Sincerity</i> : If an agent knows a proposition p is true, it cannot say anything different to others. (ii) <i>Compassion</i> : An agent temporarily leaves its individual goal to help another agent in greater difficulty (temporary change of goal). (iii) <i>Reciprocity</i> : An agent knows that it has a cooperative attitude, like all other agents have.	Cooperative Attitude, Non-cooperative Situations, Agent Environment, Agent	Analysis
Communication Acts	A communication act is a mean implemented by an agent to interact with an agent and/or its environment. A speech act (e.g. FIPA ACL) is a communication act.	Agent, Agent Environment	Analysis
Behaviour	The behaviour of an agent is a life cycle consisting of the sequence: (i) perception of the environment (including communication aspects), (ii) decision that allows it to identify the state in which it lies and actions to be performed, (iii) execution of decided actions. The life cycle starts when the agent is created and completes when the agent dies. Agent behaviour can be represented as an automaton whose states are the situations that the agent can identify and transitions are actions it decides to execute.	Agent Environment, Agent	Design

(continued)

Concept	Definition	Referred Concepts	Domain
Nominal Behaviour	The nominal behaviour is the part of the behaviour which enables the agent to reach its goal when it is in cooperative situation.	Goal, Agent, Behaviour	Design
Cooperative Behaviour	The cooperative behaviour of an agent enables it to detect the set of states identifying NCS and to describe the repairing actions to return to a cooperative situation or anticipatory attempt to avoid NCSs. In addition, an agent tries to help the most critical agent in its neighbourhood. In certain conditions, it spontaneously communicates information to agents that it thinks the information will be useful. Such a cooperative behaviour can be divided into three distinct steps: (i) tuning which consists in the modification of parameter values for parameters that influence the behaviour of the agent; (ii) reorganization which consists in the modification of the acquaintances of the agent that will lead to the reorganization of the system to make the resulting global function proper; (iii) evolution which consists in creation or suicide of agents.	Goal, Agent, Behaviour, NCS	Design
Criticality	For an agent, criticality represents the degree of non-satisfaction of its own goal. It enables an agent to determine the relative difficulty of agents in its neighbourhood. Evaluation methods and calculation of the criticality are specific to each type of agent.	Goal, Agent, Neighbourhood	Design
Behaviour Confidence	The behaviour confidence of an agent is an internal measure that provides information on the reliability of the decision on actions intended.	Agent, Behaviour	Design
Skills	The skills of an agent are capabilities in a domain that enables an agent to perform actions to achieve its goal.	Agent, Goal	Design
Characteristics	A characteristic is an intrinsic property of the agent. It can be visible or not and it can be modified by the agent or other agents.	Agent	Design
Aptitudes	The aptitudes of an agent are generic capabilities which are independent of its competence domain.	Agent	Design

(continued)

Concept	Definition	Referred Concepts	Domain
Representations	Representations of an agent are the image that the agent has of its environment and itself, that is to say all of its perceptions and beliefs. They can be updated by means of its perceptions.	Agent, Agent Environment	Design
Interaction Language	The interaction language is a set of tools required by the agent to communicate with other agents. This communication can be done through messages (direct) or via the environment (indirect communication).	Agent	Design

1.3 Guidelines and Techniques

ADELFE is based on object-oriented methodology, inspired from the Rational Unified Process (RUP). Some steps have been added in the classical workflow fitting with the adaptive MAS.

ADELFE is based on UML2 notation with the complementary use of DSML AMASML (AMAS Modelling Language) and SpeADL (Species-based Architecture Description Language), a design methodology, several model-driven tools and a library of components that can be used to facilitate the application development. Guidance of the development process is supported by AdelfeToolkit (Fig. 7a). The general idea is to help the designer to follow the process, with descriptions, examples and presents a summary of works and artifacts already performed and those remaining.

As mentioned in A11 and A13 activities of the ADELFE process, the analyst must verify whether the problem needs the AMAS or not. For this, a tool (Fig. 7b) is provided to answer the questions at the macro-level (eight criteria) and the micro-level (three criteria). Answers to questions are graded from 0 to 20.

2 Phases of the ADELFE Process

ADELFE defines its five phases from the RUP definition. They are respectively dedicated to Preliminary Requirements, Final Requirements, Analysis, Design and Implementation.

2.1 Preliminary Requirements Phase (WD1)

The Preliminary Requirements Phase involves traditional software development stakeholders which are assigned classical activities. The goal of this phase is to

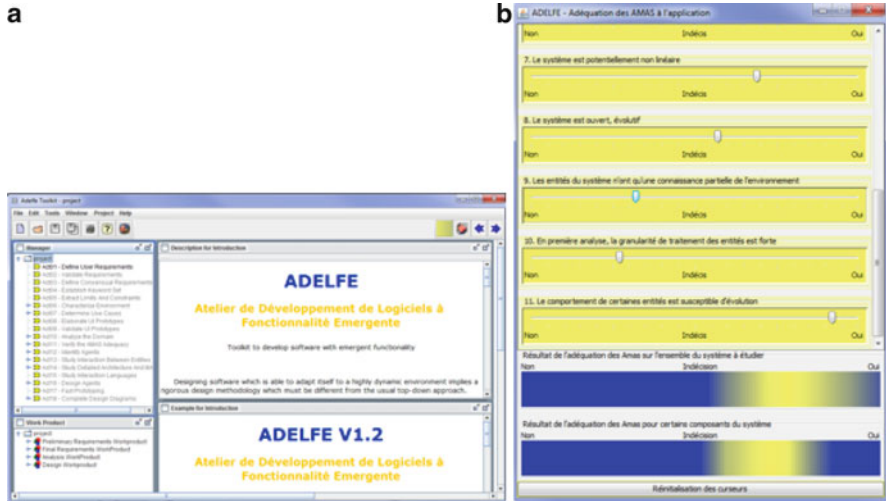


Fig. 7 Tool for monitoring the ADELFE process: AdelfeToolkit (a); tool for AMAS adequacy (b)

obtain a precise and consensual description of the problem as well as the client's business. No specific modelling language is used. The process flow at the level of activities is depicted in Fig. 8, and Fig. 9 depicts this phase according to documents, roles and work products involved.

2.1.1 Process Roles

Four roles are involved in the Preliminary Requirements Phase: the End-user, the Client, the Software analyst and the Business analyst.

- *End-User*: The end-user is responsible for the list of functional and non-functional requirements during the *Define Users Requirements* activity. These requirements are used to define the system and its environment.
- *Client*: The main role of a client is to validate product documents drawn up by other experts. The client is responsible for approving the set of requirements during the *Define Users Requirements* activity and during the *Define Consensual Requirements* activity.
- *Software Analyst*: Actually the software analyst gives a definition for the main concepts used to describe the system and its environment. He/she is responsible for: consensual requirements list during the *Define Consensual Requirements* activity, keywords during the *Establish Keywords* activity and limits and constraints of the system during the *Extract Limits and Constraints* activity.
- *Business Analyst*: A business analyst is responsible for the business model during the *Determine Business Model* activity. He/she defines the business concept and the relationships between them. He/she also describes formally what are the business activities involved, what are the products provided or required and who are the persons responsible for these activities.



Fig. 8 The preliminary requirements phase flow of activities

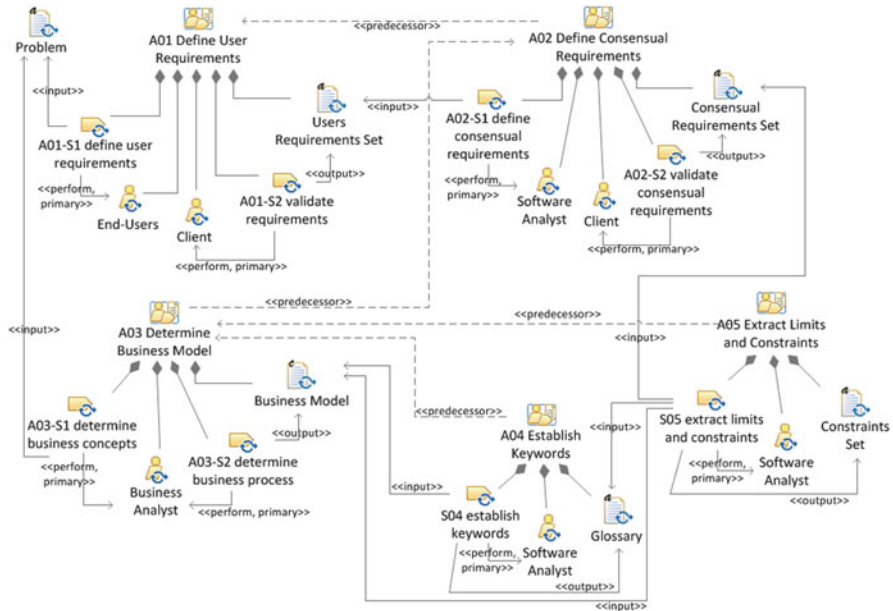


Fig. 9 The preliminary requirements phase described in terms of activities and work products

2.1.2 Activities Details

The flow of activities inside this phase is depicted in Fig. 8 and detailed in the following.

A01: Define Users Requirements

This first activity concerns the description of the system and the environment in which it will be deployed. This activity consists in defining what to build or what is the most appropriate system for end-users. End-users and clients have to list, check and approve the requirements. The context in which the system will be deployed must be understood. The functional and non-functional requirements must be established. The flow of tasks inside this activity is depicted in Fig. 10, and the tasks are detailed in the following table.



Fig. 10 Flow of tasks of the *Define Users Requirements* activity

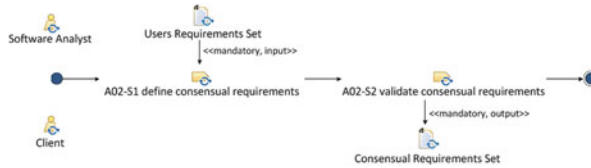


Fig. 11 Flow of tasks of the *Define Consensual Requirements* activity

A01: Define Users Requirements		
Tasks	Task Descriptions	Roles Involved
S1: define users' requirements	End-users list the functional and non-functional requirements.	End-users
S2: validate users' requirements	The client has to check and approve the set of requirements. If this document is not validated, the requirements have to be improved; the previous step is repeated again.	Client

A02: Define Consensual Requirements

This activity consists in defining what conditions or capabilities the system has to conform. The consensual requirements set is defined by the software analyst. The flow of tasks inside this activity is depicted in Fig. 11, and the tasks are detailed in the following table.

A02: Define Consensual Requirements		
Tasks	Tasks Descriptions	Roles Involved
S1: define consensual requirements	The software analyst defines the requirements set with the consensual requirements.	Software Analyst
S2: validate consensual requirements	If there is no agreement on the Requirements Set document, a backtrack must be performed to study again the previous step.	Client

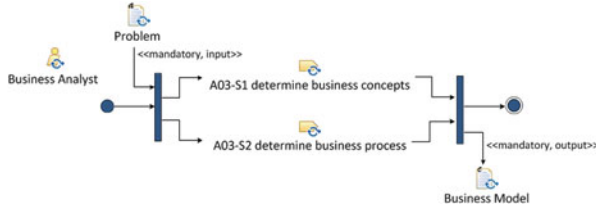


Fig. 12 Flow of tasks of the *Determine Business Model* activity

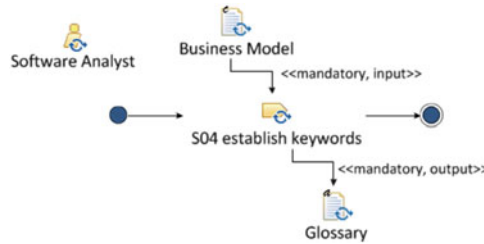


Fig. 13 Flow of tasks of the *Establish Keywords* activity

A03: Determine Business Model

This activity provides an overview of the problem, the related concepts and the activities linked to it. The flow of tasks inside this activity is depicted in Fig. 12, and the tasks are detailed in the following table.

A03: Determine Business Model		
Tasks	Tasks Descriptions	Roles Involved
S1: determine business concepts	This step enables one to understand the static and dynamic structure of the system.	Business Analyst
S2: determine business process	In this step, the sequence of actors’ actions that achieve the goal of the system is determined.	Business Analyst

A04: Establish Keywords

The main concepts used to describe the application and its business model are listed. This activity, carried out by the software analyst, is composed of one task giving the definition of each keyword. These definitions will be stored in the glossary. The flow of this activity is depicted in Fig. 13.

A05: Extract Limits and Constraints

In this activity, the limits and constraints of the system are defined by a software analyst. They can be found in the expression of non-functional requirements and in the definition of the context in which the system will be deployed. This information

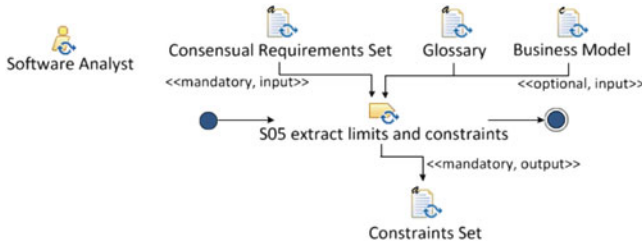


Fig. 14 Flow of tasks of the *Extract Limits and Constraints* activity

will be defined mainly from the consensual requirements set documents. The flow of this activity is depicted in Fig. 14.

2.1.3 Work Products

The Preliminary Requirements Phase generates five work products (text document including textual description and/or diagrams). Their relationships with the MAS metamodel elements are depicted in Fig. 15.

Work Products Kind

Name	Description	Work Product Kind
Users Requirements Set	Textual description of the functional and non-functional requirements	Free Text
Consensual Requirements Set	Textual description composed of consensual requirements	Free Text
Business Model	A document composed of: 1) a diagram modelling the domain-specific data structure; 2) a diagram showing the workflow of activities performed by the business actors	Composite (Structural and Behavioural)
Glossary	A glossary of terms	Free Text
Constraints Set	Textual description composed of the limits and constraints of the system	Free Text

Example: Conference Management Study

The description of the system has already been provided in the introductory chapter. As the requirements are common to all methods and because ADELFE is not really dedicated to preliminary requirements, only the business model is shown in this phase. Figure 16 shows the business concepts and Fig. 17 shows the business process.

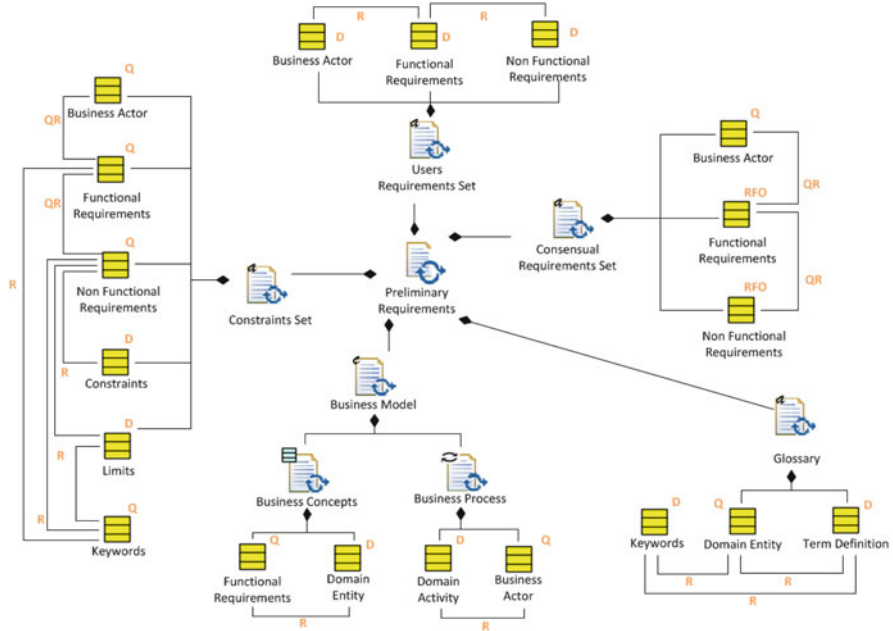


Fig. 15 The preliminary requirements documents structure

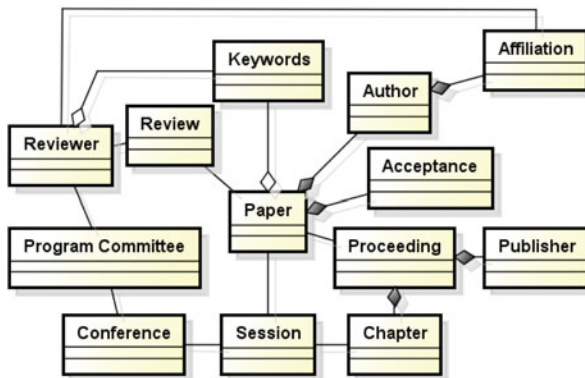


Fig. 16 Business concepts

2.2 Final Requirements Phase (WD2)

The Final Requirements Phase is a classical requirement-oriented phase where the Business Analyst gives a detailed description of the system environment. It also embeds MAS-oriented tasks. The analysis, done by an MAS specialist, must add sufficient details to the description of the system environment in order to conclude

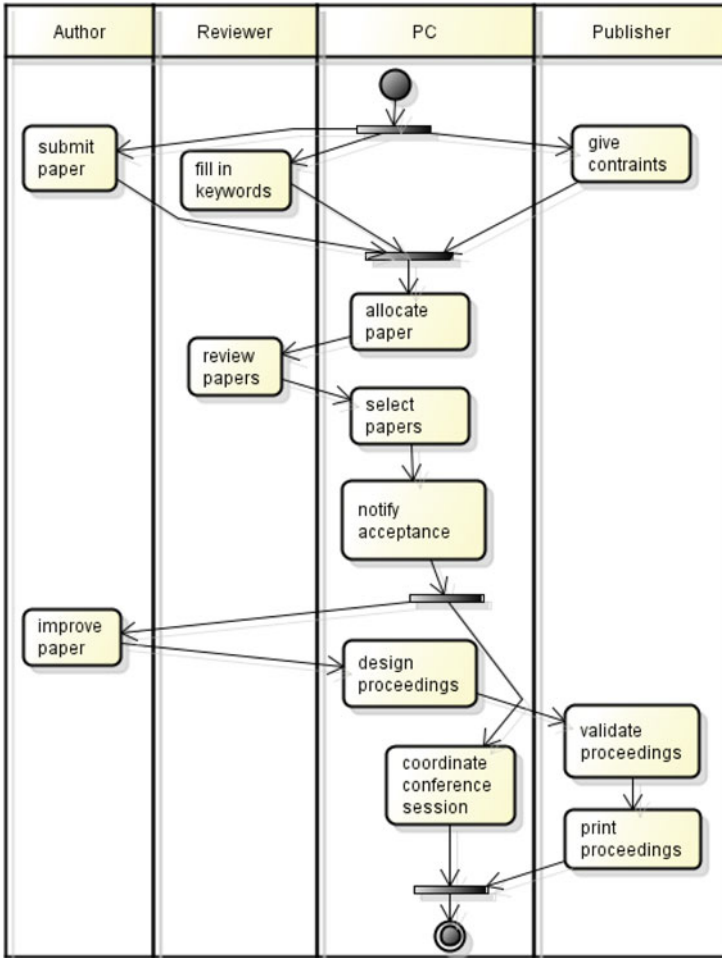


Fig. 17 Business process

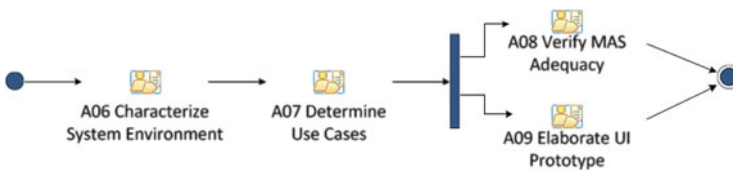


Fig. 18 The final requirements phase flow of activities

if an MAS approach is needed to solve the problem with gains. The process flow at the level of activities is depicted in Fig. 18, and Fig. 19 depicts this phase according to documents, roles and work products involved.

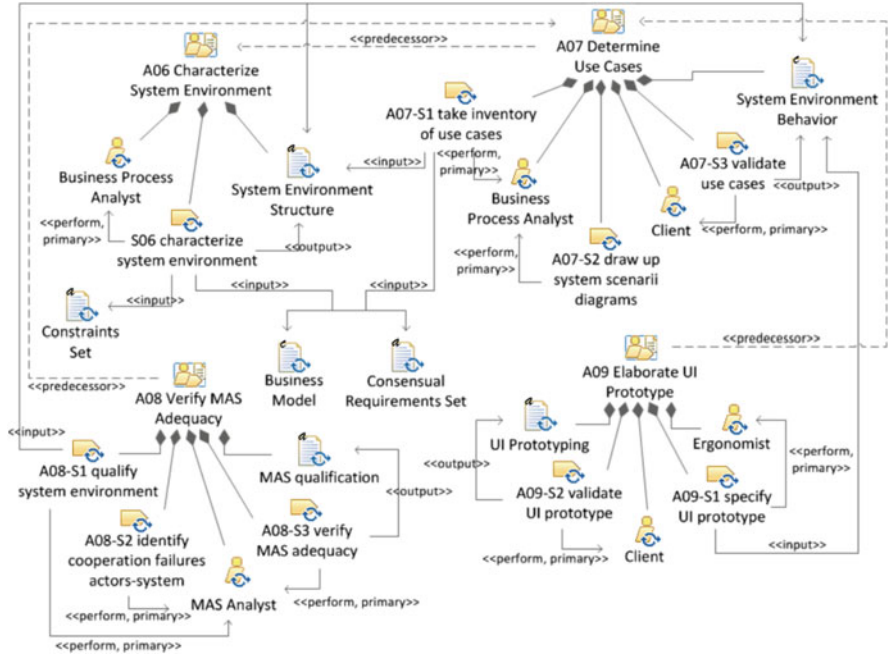


Fig. 19 The preliminary requirements phase described in terms of activities and work products

2.2.1 Process Roles

Four roles are involved in the Final Requirements Phase: the Business Process Analyst, the Client, the MAS Analyst and the Ergonomist.

- *Business Process Analyst:* A business process analyst is responsible for use cases identification during the *Determine Use Case* activity, drawing diagrams that represent the interactions between actors and the system.
- *Client:* The main role of a client is to validate product documents drawn up by other experts. He/she is responsible for approving the use cases defined during the *Determine Use Case* activity. Besides, the client agrees the UI prototype during the *Elaborate UI Prototype* activity.
- *MAS Analyst:* An MAS analyst is responsible for verifying the MAS adequacy during the *Verify MAS Adequacy* activity. It consists in (1) the characterization of the system environment according to Russel and Norvig definition, (2) the identification of the possible “bad” interactions between the actors and the system, and (3) the analysis of the previous results to justify the MAS use.
- *Ergonomist:* An ergonomist is responsible for graphic user interfaces prototype during the *Elaborate UI Prototype* activity. He/she understands the interactions among humans and the system, and designs a prototype which optimizes human well-being and overall system performance.

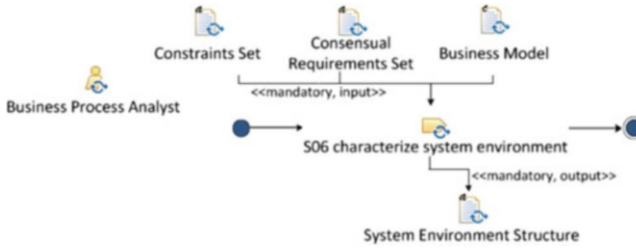


Fig. 20 Flow of tasks of the *Characterize System Environment* activity

2.2.2 Activities Details

The flow of activities inside this phase is depicted in Fig. 18 and is detailed in the following.

A06: Characterize System Environment

The main objective of this activity is to define the system environment in the system environment description document. This activity enables one to identify and to describe briefly actors interacting with the system. The possible encountered constraints are also explained. The flow of tasks inside this activity is depicted in Fig. 20.

A07: Determine Use Cases

The main objective of this activity is to clarify the different functionalities that the studied system must provide. The flow of tasks inside this activity is depicted in Fig. 21, and the tasks are detailed in the following table.

A07: Determine Use Cases		
Tasks	Tasks Descriptions	Roles Involved
S1: take inventory of use cases	A set of steps defining interactions between an actor and a system are listed.	Business Process Analyst
S2: draw system scenario diagrams	This step explicates the system behaviour from users' point of view. The interactions between the actors and the system are drawn.	Business Process Analyst
S3: validate use cases	Approval of the System Environment Description document by the client. If the use cases have to be improved, the two previous steps have to be repeated again.	Client

A08: Verify MAS Adequacy

In this activity, one must verify that a Multi-agent System (MAS) approach is needed to realize the system to be built. The question to answer is “Is a traditional (Object-Oriented) approach sufficient to solve the problem or has the problem some characteristics which implies an MAS approach for the solving?”. The flow of

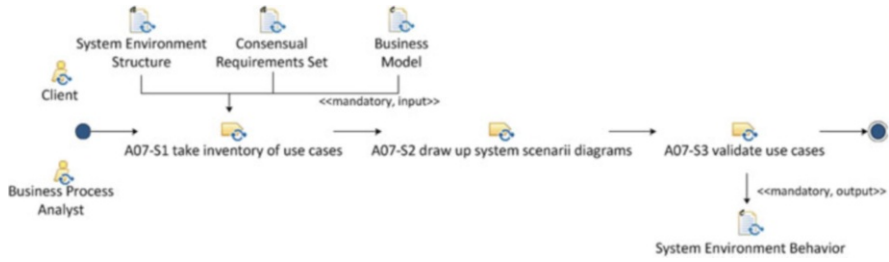


Fig. 21 Flow of tasks of the *Determine Use Cases* activity

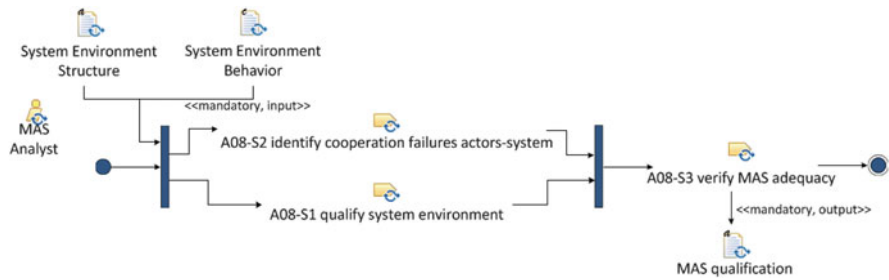


Fig. 22 Flow of tasks of the *Verify MAS Adequacy* activity

tasks inside this activity is depicted in Fig. 22, and the tasks are detailed in the following table.

A08: Verify MAS Adequacy		
Tasks	Tasks Descriptions	Roles Involved
S1: qualify system environment	During this step, the MAS analyst characterizes the system environment according to the Russel and Norvig definition.	MAS Analyst
S2: identify cooperation failures actors-system	The aim of this step is to show the inadequate interactions that may occur between the actors and the system.	MAS Analyst
S3: verify MAS adequacy	This step verifies the MAS adequacy by analysing the results obtained during the two previous steps.	MAS Analyst

A09: Elaborate UI Prototypes

The GUIs described in the UI Prototype document have to be defined, judged and validated from functional or non-functional (ergonomic, design, etc.) points of view. The flow of tasks inside this activity is depicted in Fig. 23, and the tasks are detailed in the following table.

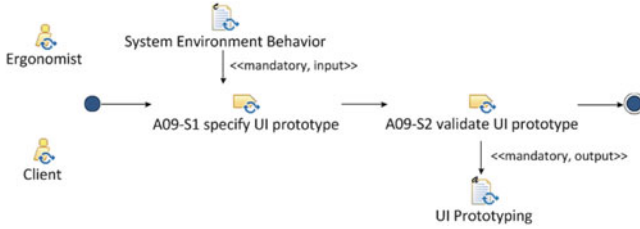


Fig. 23 Flow of tasks of the *Elaborate UI Prototypes* activity

A09: Elaborate UI Prototypes		
Tasks	Tasks Descriptions	Roles Involved
S1: specify UI prototypes	In this step, the interfaces supplying all needed functionalities are specified.	Ergonomist
S2: validate UI prototypes	The UIs are used and assessed from functional and non-functional points of view.	Client

2.2.3 Work Products

The Final Requirements Phase generates four work products (text document including textual description and/or diagrams). Their relationships with the MAS meta-model elements are depicted in Fig. 24.

Work Products Kind

Name	Description	Work Product Kind
System Environment Structure	A textual description describing the actors which interact with the system and the possible constraints. Moreover, this document contains a brief text of actors description.	Free Text
System Environment Behaviour	A document composed of: 1) a use case diagram representing actors and the functionalities assigned to them; 2) a structured text description of the actors; 3) diagrams representing the interactions between the actors and the system.	Composite (Structural and Behavioural and Free text)
MAS Qualification	A text document composed of the description of the environment according to the Russel and Norvig definition, the description of “bad” interaction between actors and system and the justification of an implementation that using an MAS is needed.	Free Text
UI Prototype	This document is composed of the GUIs description through which the user(s) interact with the system and the links between the GUIs.	Free Text

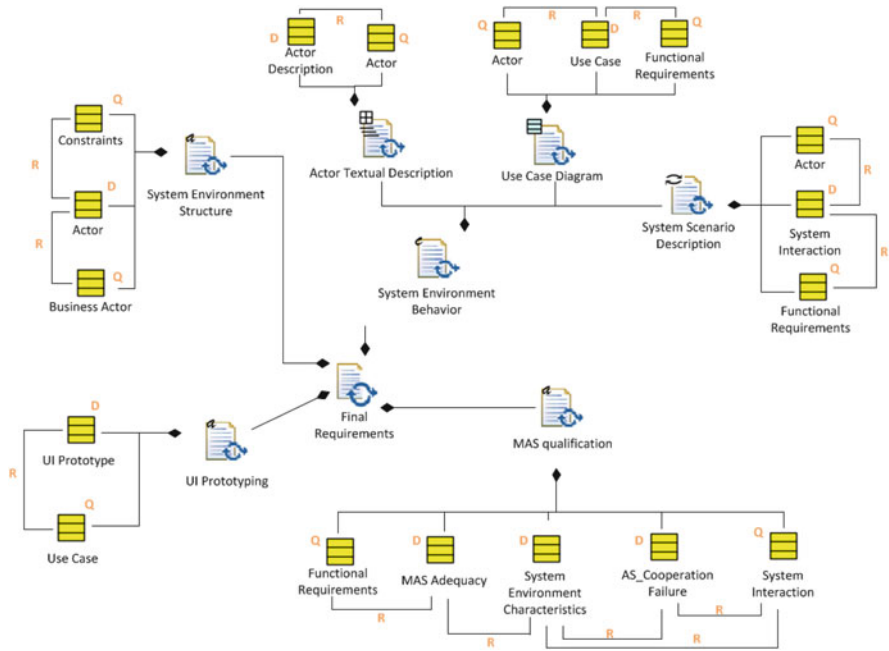


Fig. 24 The final requirements documents structure

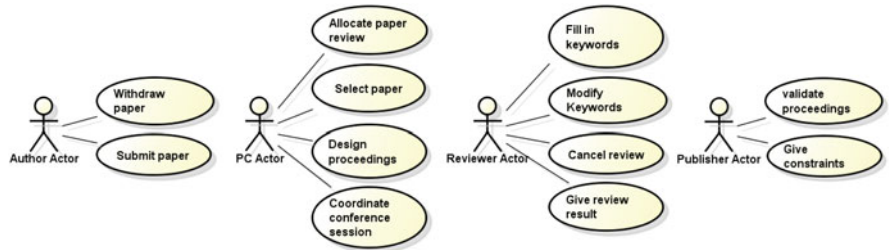


Fig. 25 Use cases diagram

Example: Conference Management Study

From the business model and the requirements previously established, four actors are defined. For each of them, the functionalities are detailed and depicted in a use cases diagram (see Fig. 25).

Moreover the interactions between the system and the actors are studied and shown in Fig. 26. From these diagrams, the following cooperative failures have been identified:

- at least one paper is not allocated
- not enough papers are accepted for the conference
- the chair committee disagrees with the paper allocation

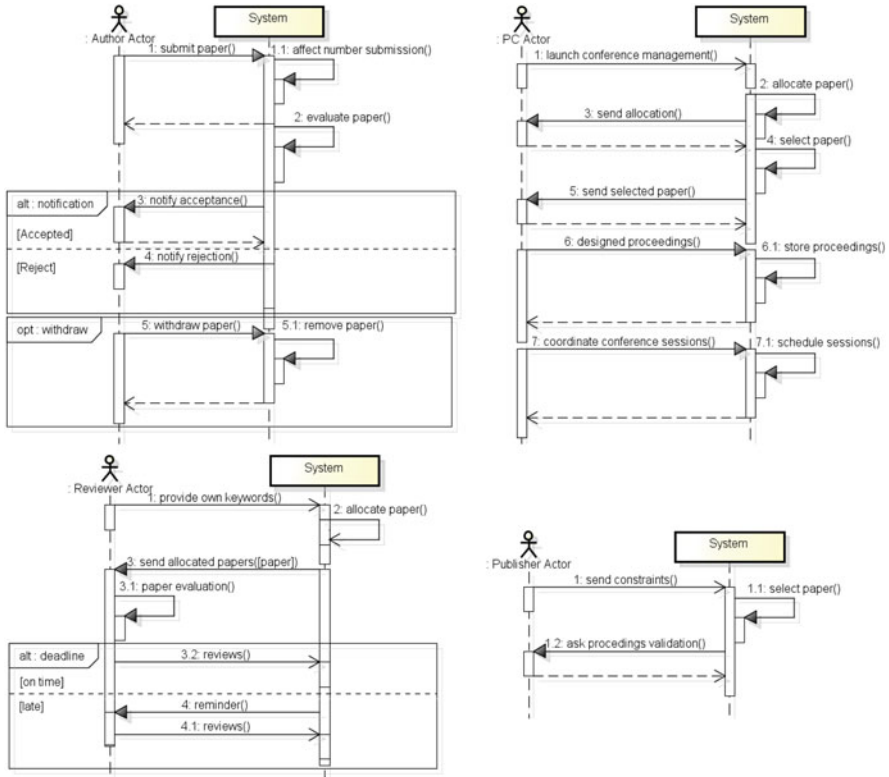


Fig. 26 System sequence diagrams

- a reviewer does not have a paper to review
 - the reviewer disagrees with the paper allocation
- Besides, according to Russel and Norvig’s definition, the environment of the system is described as
- *inaccessible* because knowing all about the environment (papers, keywords, etc.) is difficult
 - *discrete* because the number of distinct percepts and actions is limited
 - *non-deterministic* because the actions have multiple unpredictable outcomes
 - *dynamic* because the state of the environment depends upon actions of the system that is within this environment

2.3 Analysis Phase (WD3)

The Analysis Phase aims at identifying the system structure and justifying the AMAS adequacy. This phase is composed of four activities enabling one to analyse the domain characteristics, determine the agents and validate an AMAS approach



Fig. 27 The analysis phase flow of activities

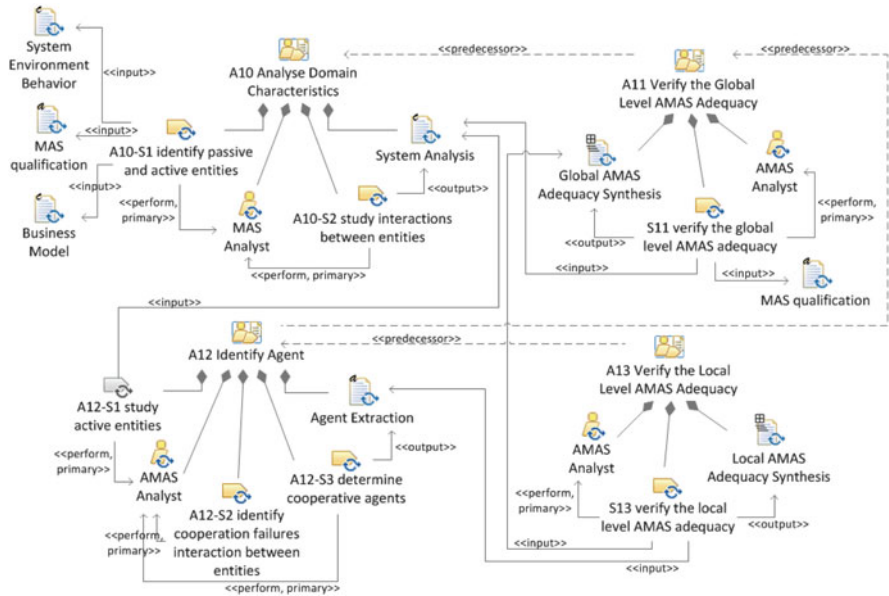


Fig. 28 The analysis phase described in terms of activities and work products

at the global and local level. The process flow at the level of activities is depicted in Fig. 27, and Fig. 28 depicts this phase according to documents, roles and work products involved.

2.3.1 Process Roles

Two roles are involved in the Analysis Phase: the MAS Analyst and AMAS Analyst.

- *MAS Analyst*: An MAS analyst is responsible for detailing the MAS Environment in *Analysis Domain Characteristics* activity. It consists in (1) the identification of what are the entities which are active and the ones which are not (passive), (2) the identification of the interactions between the entities. An MAS analyst is also responsible for *Identify agent* of the step which consists in defining autonomy, goal and negotiation abilities of active entities.
- *AMAS Analyst*: An AMAS analyst is responsible for every activity dealing with the specificities of AMAS principles. They can be found in *Verify the global level AMAS adequacy* activity, in *Identify agent* activity and in *Verify the local level AMAS adequacy* activity. The identification of (cooperative) agents needs

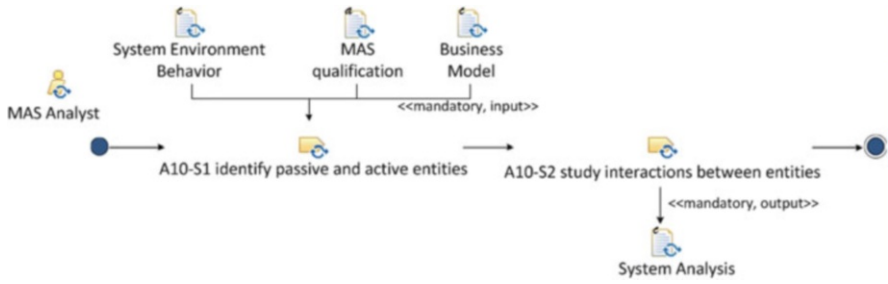


Fig. 29 Flow of tasks of the *Analyse Domain Characteristics* activity

to determine the cooperation failures that can occur between entities and then to define the agents regarding the results of previous steps.

2.3.2 Activity Details

The flow of activities inside this phase is depicted in Fig. 27 and detailed in the following.

A10: Analyse Domain Characteristics

The main goal of this activity is to analyse the Business Domain and the System Environment Description in order to detail the entities of the domain and their interactions. The flow of tasks inside this activity is depicted in Fig. 29, and the tasks are detailed in the following table.

A10: Analyse Domain Characteristics		
Tasks	Tasks Descriptions	Roles Involved
S1: identify passive and active entities	The MAS analyst splits the system into passive and active entities.	MAS Analyst
S2: study interactions between entities	This step shows the interactions between entities.	MAS Analyst

A11: Verify the Global Level AMAS Adequacy

In this activity, the AMAS analyst must verify that an AMAS approach is needed to realize the system to be built. For example, having a system which is able to adapt itself is sometimes completely useless if the algorithm required to solve the task is already known, if the task is not complex or if the system is closed and nothing unexpected can occur. In this activity, the adequacy at the global level is studied to answer the question “is an AMAS required to implement the system?”. This is done through several simple questions related to the global level. The flow of this activity is depicted in Fig. 30.

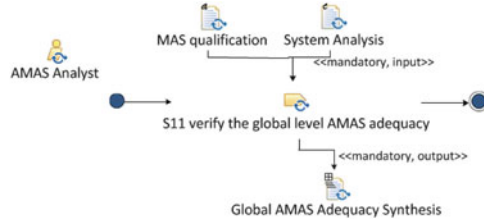


Fig. 30 Flow of tasks of the *Verify the Global Level AMAS Adequacy* activity

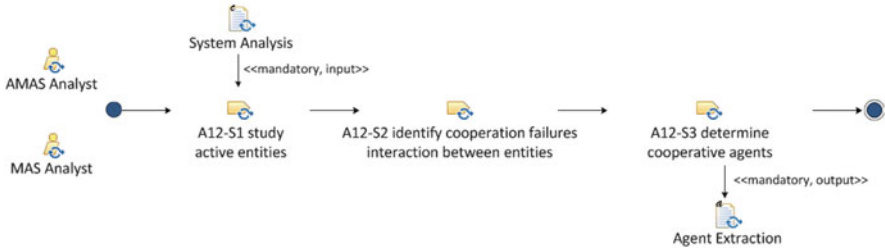


Fig. 31 Flow of tasks of the *Identify Agent* activity

A12: Identify Agent

This activity aims at finding what will be considered as agents in the desired system. These agents are defined among the previously defined entities. The flow of tasks inside this activity is depicted in Fig. 31, and the tasks are detailed in the following table.

A12: Identify Agent		
Tasks	Tasks Descriptions	Roles Involved
S1: study active entities	For each previously defined active entity, its autonomy, its goal and its negotiation abilities are studied.	MAS Analyst
S2: identify cooperation failures during interaction between entities	During its interactions with other entities, an entity can encounter failures to respect the protocol or failures in the content of the interaction (misunderstanding, etc.). This step extracts this kind of interactions.	AMAS Analyst
S3: determine cooperative agents	The entities pertaining to the previous step are considered as agents. In addition, the AMAS diagram is drawn.	AMAS Analyst

A13: Verify the Local Level AMAS Adequacy

In this activity, the AMAS adequacy is studied at the local level in order to determine if some agents are needed to be implemented as an AMAS i.e. if a certain kind of decomposition or recursion is required during the building of the system. The flow of this activity is depicted in Fig. 32.

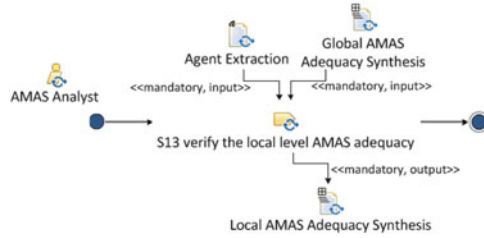


Fig. 32 Flow of tasks of the *Verify the Local Level AMAS Adequacy* activity

2.3.3 Work Products

The Analysis Phase generates four work products (text document including textual description and/or diagrams). Their relationships with the MAS metamodel elements are depicted in Fig. 33.

Work Products Kind

Name	Description	Work Product Kind
System Analysis	A document composed of: 1) a textual description of the entities described as active or passive; 2) diagrams depicting the interactions between entities.	Composite (Free Text and Behavioural)
Global AMAS Adequacy Synthesis	This document stores the answers to the questions regarding the global level about an implementation using an AMAS.	Structured Text
Agent Extraction	This document supplements the System Analysis document with: 1) the definition of the goal, the study of autonomy and the negotiation abilities for each active entity; 2) the list of the cooperation failure interactions between entities or between entity and its environment; 3) the definition of the cooperative agent and the AMAS diagram which represents them.	Composite (Free Text and Behavioural)
Local AMAS Adequacy Synthesis	This document completes the Global AMAS adequacy synthesis with the answers to the questions regarding the local level about an implementation using an AMAS.	Structured Text

Example: Conference Management Study

From the business concepts model, we define the active and passive entities. In our case, we define six active entities and the other concepts as passive entities. The entities system structure is depicted in Fig. 34. Broadly speaking, entities which are

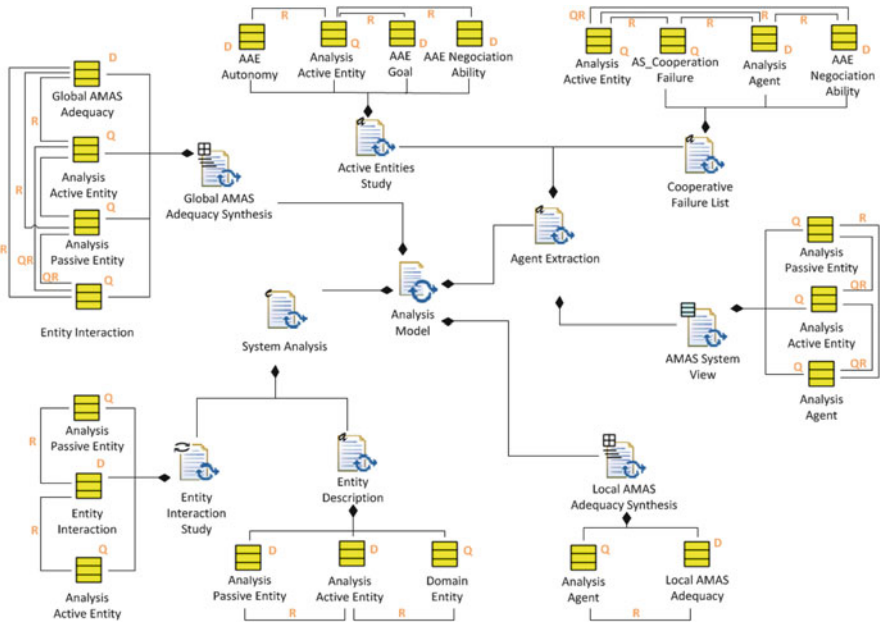


Fig. 33 The analysis documents structure

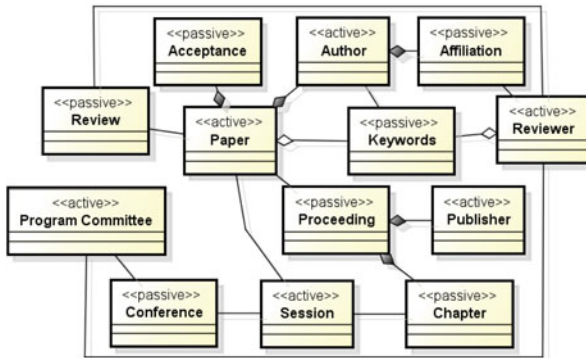


Fig. 34 Entities structure system diagram

linked to users are considered as active because users may change their mind, which implies a change in the state of the related entity. Moreover, papers and sessions are considered as active because they will need negotiation while finding a review or a session organization.

The following step is the interactions study between entities shown in Figs. 35 and 36. The started interactions and the other one show the interaction after the paper notification, they are represented in Fig. 35.

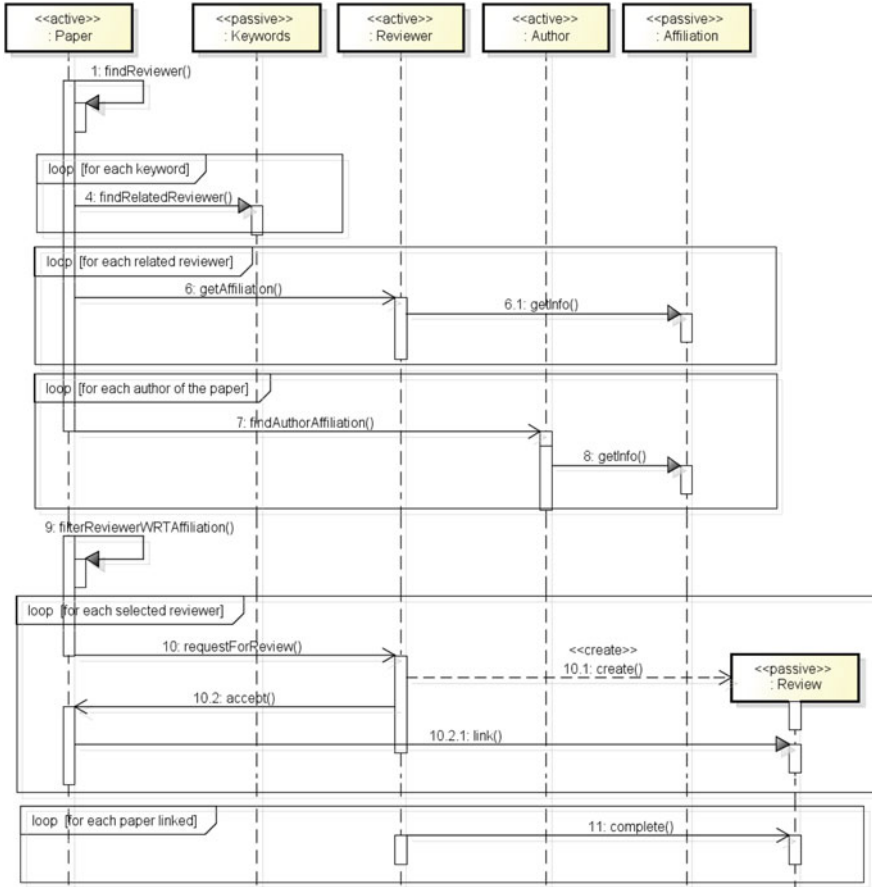


Fig. 35 Entities interactions system structure (beginning)

Verifying the AMAS adequacy consists in studying some specific features of AMAS with respect to the target application. The designer is provided with a tool which helps him to answer some questions. Here are the eight questions that are asked and some answers for our case study:

- *Is the global task incompletely specified? Is an algorithm a priori unknown?* YES: the CMS is precisely defined and some algorithms may be found to solve this kind of problem.
- *If several entities are required to solve the global task, do they need to act in a certain order?* YES: there are dependencies in the interactions needed between entities.
- *Is the solution generally obtained by repetitive tests? Are different attempts required before finding a solution?* NO: no need for that.

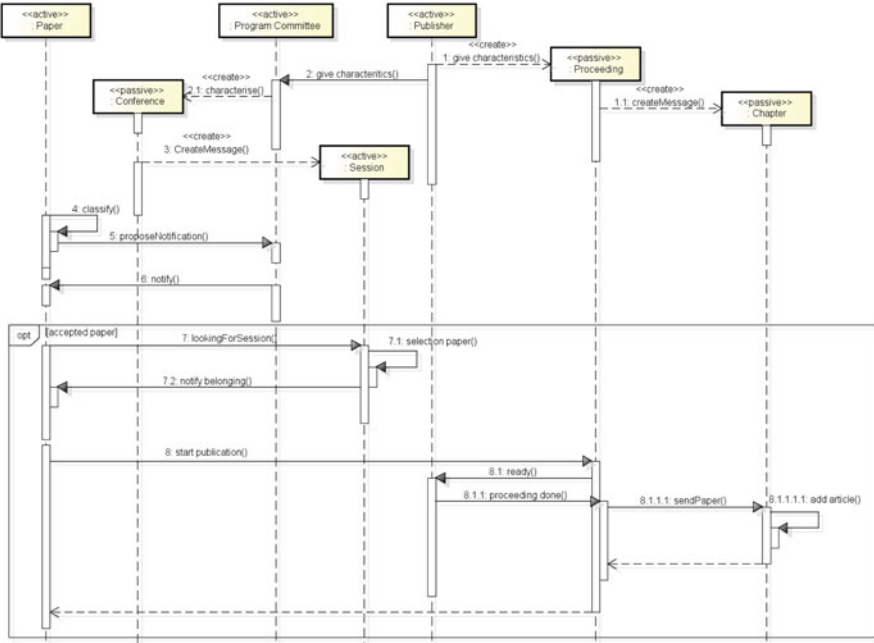


Fig. 36 Entities interactions system structure (ending)

- *Can the system environment evolve? Is it dynamic?* YES: it is highly dynamic. Authors, reviewers and papers may appear or disappear while a solution is calculated.
- *Is the system process functionally or physically distributed? Are several physically distributed entities needed to solve the global task? Or is a conceptual distribution needed?* YES/NO: people is physically distributed but there is no need for a distributed solving of the problem.
- *Are a great number of entities needed?* YES: depending on the conference, but potentially, in conferences like AAMAS, there are a great number of entities.
- *Is the studied system non-linear?* RATHER YES: in the nominal case, it can be rather easy to find a linear decomposition of the problem, but with the openness described earlier, the great number of interactions may lead to a complex system.
- *Is the system evolutionary or open? Can new entities appear or disappear dynamically?* YES: it is highly dynamic. Authors, reviewers and papers may appear or disappear while a solution is calculated.

In the “Identify Agents” activity, the active entities previously defined are studied. The following table depicts the autonomy, the goal and the negotiation abilities of each of them.

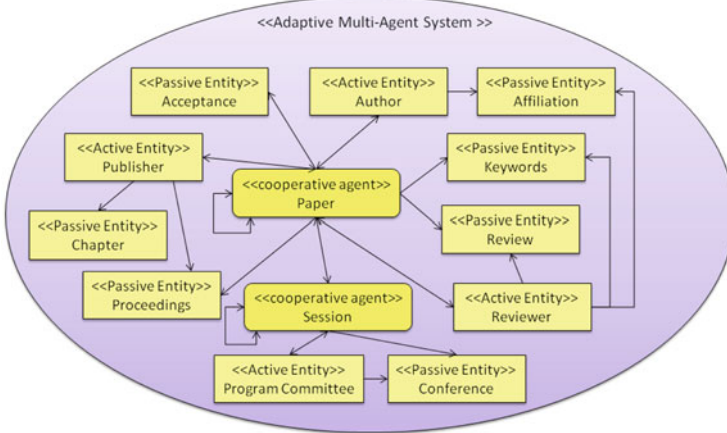


Fig. 37 AMAS system diagram

Active Entities	Autonomy	Goal	Negotiation Abilities
Paper	Take its own decision	Know its result: acceptance or rejection	Talk with other papers to choose reviewer and find its position in the acceptance
Reviewer	Act according to the paper and users	Review papers	
Program Committee	Act according to the users	Validate	
Publisher	Act according to the users	Give constraints and proceedings	
Author	Act according to the users	Deal with a paper	
Session	Select itself the right papers to insert for the conference management	Select paper to be full	Discuss with the other sessions to select papers

From the interactions between entities which have been previously identified, three other cooperative failure interactions are identified: (1) a paper does not find a review; (2) several papers want to rise to the same rank; (3) sessions select the same paper. Two kinds of agent are therefore deduced: the paper agent and the session agent. The resulting AMAS is represented in Fig. 37.

2.4 Design Phase (WD4)

The Design Phase aims at providing a detailed architecture of the system. During this phase, the definition of a module view is proposed, the communication acts are studied and the different behaviours are determined. The process flow at the

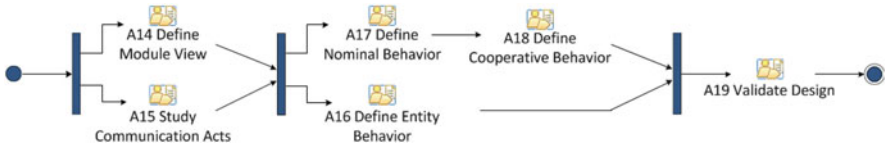


Fig. 38 The design phase flow of activities

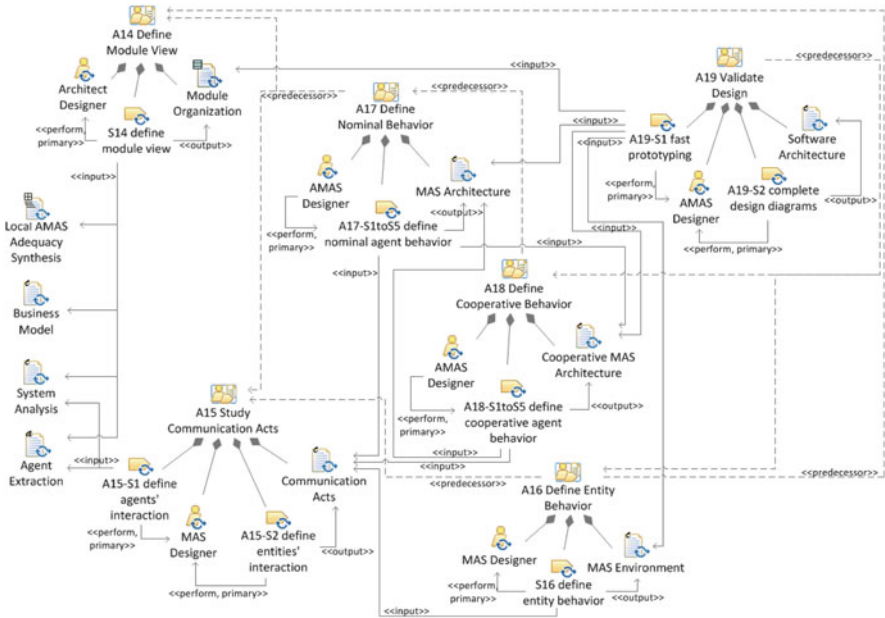


Fig. 39 The design phase described in terms of activities and work products

level of activities is depicted in Fig. 38, and Fig. 39 depicts this phase according to documents, roles and work products involved.

2.4.1 Process Roles

Three roles are involved in the Design Phase: the architectural designer, the MAS designer and the AMAS designer.

- *Architectural Designer*: An architectural designer is responsible for module organization during the *Define Module View* activity. He/she defines the detailed architecture of the system in terms of modules.
- *MAS Designer*: An MAS designer is responsible for communication acts during the *Study Communication Acts* activity and the definition of the entities behaviour during the *Define Entity Behaviour* activity. He/she defines how the entities and the agents interact together or with their own environment.
- *AMAS Designer*: An AMAS designer is responsible for nominal behaviour of agents during the *Define Nominal Behaviour* activity, cooperative behaviour

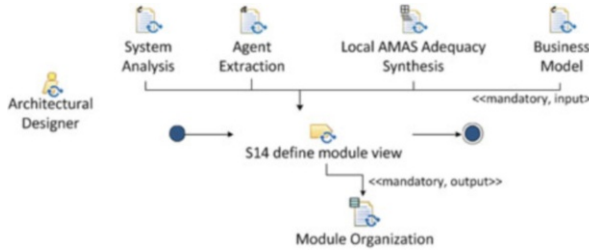


Fig. 40 Flow of tasks of the *Define Module View* activity

of agents in the *Define Cooperative Behaviour* activity and fast prototyping during the *Validate Design Phase* activity. Indeed, from the structure analysis and the communication acts previously detailed, an AMAS designer defines skills, aptitudes, an interaction language, a world representation, a criticality and the characteristics of an agent. He/she fulfils the agent behaviour by adding a cooperative attitude i.e. giving rules which enable anticipating or detecting and repair of the non-cooperative situations. For that, skills, aptitudes, an interaction language, a world representation, a criticality and the characteristics are filled out. Finally, an AMAS designer tests the behaviour of agents i.e. the protocols, the methods and the general behaviour of agents.

2.4.2 Activity Details

The flow of activities inside this phase is depicted in Fig. 38 and detailed in the following.

A14: Define Module View

This activity shows how the architectural designer maps the key elements of the software to the modules. Their organization and dependencies are defined. The following kinds of dependencies can be used: *use*, *allowToUse*, *include/decompose*, *CrossCut*, *EnvModel*. The flow of this activity is depicted in Fig. 40.

A15: Study Communication Acts

This activity aims at making clear interactions between the entities and/or the agents previously identified. The flow of tasks inside this activity is depicted in Fig. 41, and the tasks are detailed in the following table.

A15: Study Communication Acts		
Tasks	Tasks Descriptions	Roles Involved
S1: define agents interaction	This step consists in defining the way in which an agent is going to interact with the others and its environment.	MAS Designer
S2: define entities interaction	This step consists in defining the way in which an entity is going to interact with others entities.	MAS Designer

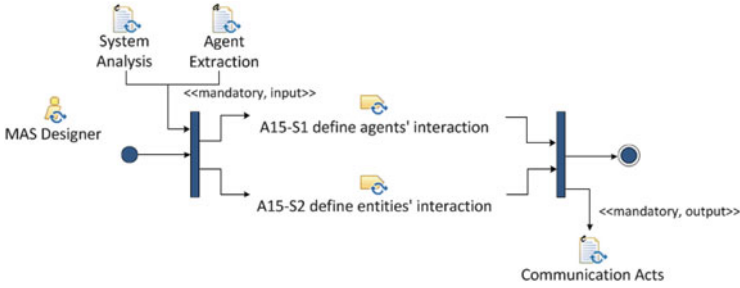


Fig. 41 Flow of tasks of the *Study Communication Acts* activity

A16: Define Entity Behaviour

The aim of this activity is to define the entity behaviour. It can be illustrated by an inner state related to its current role. This activity is performed by an MAS analyst. The flow of this activity is depicted in Fig. 42.

A17: Define Nominal Behaviour

The purpose of this activity is to define the nominal behaviour. The AMAS designer has to define skills, aptitudes, an interaction language, a world representation and a criticality, which compose the nominal behaviour. Agents may also have physical characteristics such as weight, colour, etc. which may be necessarily found during this activity. The structural diagrams of agents are drawn and the structural rules are described. In addition, an agent can be defined by an inner state related to its current role in the MAS organization. The flow of tasks inside this activity is depicted in Fig. 43, and the tasks are detailed in the following table.

A17: Define Nominal Behavior		
Tasks	Tasks Descriptions	Roles Involved
S1: define its skills	The knowledge about a domain allowing the agent to execute actions is defined.	AMAS Designer
S2: define its aptitudes	The aim of this activity is to determine the capabilities of an agent to reason on its knowledge about the domain or on its representation of the world.	AMAS Designer
S3: define its interaction language	This step consists in defining the way in which agents are going to interact. Actually, if agents interact to communicate, information exchanges between agents are described. Technically, these protocols are specified through protocol diagrams.	AMAS Designer
S4: define its world representation	The AMAS designer defines the way to describe the representations of an agent about other agents, itself and its environment.	AMAS Designer
S5: define criticality and confidence of agent behaviour	This step determines the relative difficulty of agents in its neighbourhood and its internal measure that provides information on the reliability of the decision on actions intended.	AMAS Designer

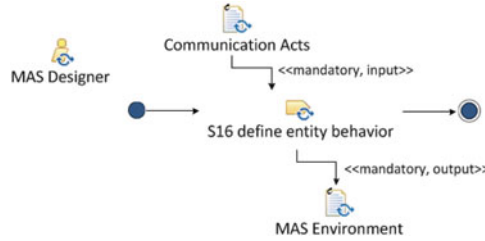


Fig. 42 Flow of tasks of the *Define Entity Behaviour* activity

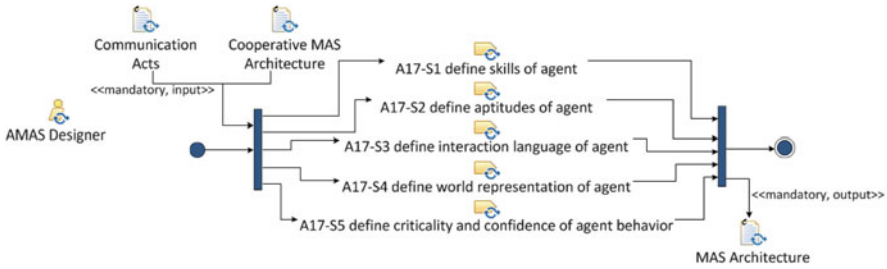


Fig. 43 Flow of tasks of the *Define Nominal Behavior* activity

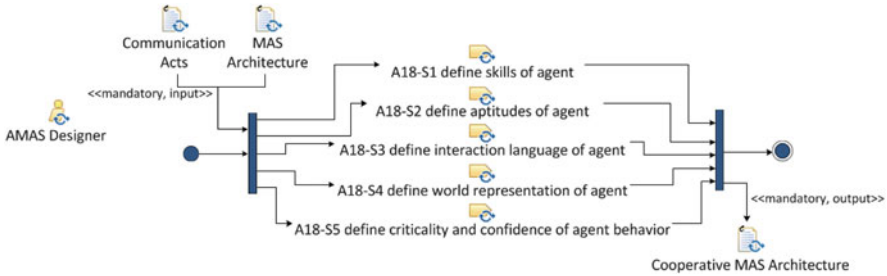


Fig. 44 Flow of tasks of the *Define Cooperative Behaviour* activity

A18: Define Cooperative Behaviour

This activity is a key step. Indeed, the AMAS designer defines the cooperative behaviour by the allocation of cooperation rules. These rules enable an agent to have a cooperative attitude i.e. anticipate or detect and repair the non-cooperative situations. During this activity, the structural diagram is completed by appropriated skills, representations, the attitudes or any other agent characteristic. The flow of tasks inside this activity is depicted in Fig.44, and the tasks are detailed in the following table.

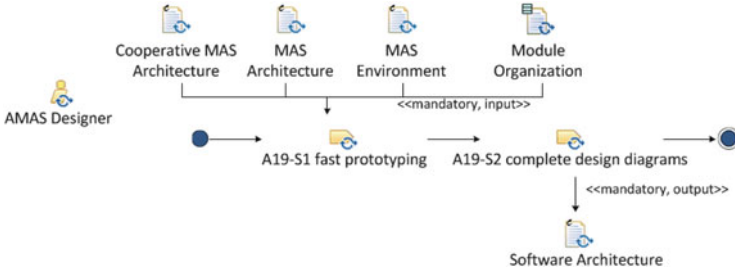


Fig. 45 Flow of tasks of the *Validate Design Phase* activity

A18: Define Cooperative Behaviour		
Tasks	Tasks Descriptions	Roles Involved
S1: define its skills	The knowledge about a domain allowing the agent to execute actions is defined.	AMAS Designer
S2: define its aptitudes	The aim of this activity is to determine the capabilities of an agent to reason on its knowledge about the domain or on its representation of the world.	AMAS Designer
S3: define its interaction language	This step consists in defining the way in which agents are going to interact. Actually, if agents interact to communicate, information exchanges between agents are described. Technically, these protocols are specified through protocol diagrams.	AMAS Designer
S4: define its world representation	The AMAS designer defines the way to describe the representations of an agent about other agents, itself and its environment.	AMAS Designer
S5: define criticality and confidence of agent behaviour	This step determines the relative difficulty of agents in its neighbourhood and its internal measure that provides information on the reliability of the decision on actions intended.	AMAS Designer

A19: Validate Design Phase

During this activity, the AMAS designer may test the behaviour of the agents. This test can lead to improve an agent’s behaviour if it is not adequate. The flow of tasks inside this activity is depicted in Fig.45, and the tasks are detailed in the following table.

A19: Validate Design Phase		
Tasks	Tasks Descriptions	Roles Involved
S1: fast prototyping	During this step, the agents' behaviour is tested. The prototype has to point out the possible lack of an agent behaviour and of cooperative attitude.	AMAS Designer
S2: complete design diagrams	The aim of this step is to finalize the module organization and finish the Design Phase.	AMAS Designer

2.4.3 Work Products

The Design Phase generates six work products. Their relationships with the MAS metamodel elements are depicted in Fig. 46.

Work Products Kind

Name	Description	Work Product Kind
Module Organization	This document depicts the organization and the dependencies of the key elements of the software.	Structural
Communication Acts	This document is composed of the specific textual description of the entity interactions and the agent interactions and the precise diagrams depicting this.	Composite (Free and Behavioural)
MAS Environment	This document contains the description of the entities behaviour. It is illustrated with inner state related to their current role.	Composite (Free and Behavioural)
MAS Architecture	This document is composed of the agent nominal behaviour description, illustrated with inner state related to their current role and depicted by structural diagram of the agents. Skills, aptitudes, an interaction language, a world representation and a criticality define cooperative agent behaviour. Moreover, it contains the physical characteristics of the agent and its structural rules.	Composite (Free and Structural and Behavioural)
Cooperative MAS Architecture	This document contains the elements of a cooperative agent behaviour, enabling anticipation or detection and repair of the non-cooperative situations. A cooperative agent behaviour is composed of skills, aptitudes, an interaction language, a world representation and a criticality.	Composite (Free and Structural and Behavioural)
Software Architecture	This document is composed of the fast prototyping of the agent behaviour and the refinement of the Software architecture entities, Software architecture nominal and Software architecture cooperative document.	Composite (Free and Structural and Behavioural)

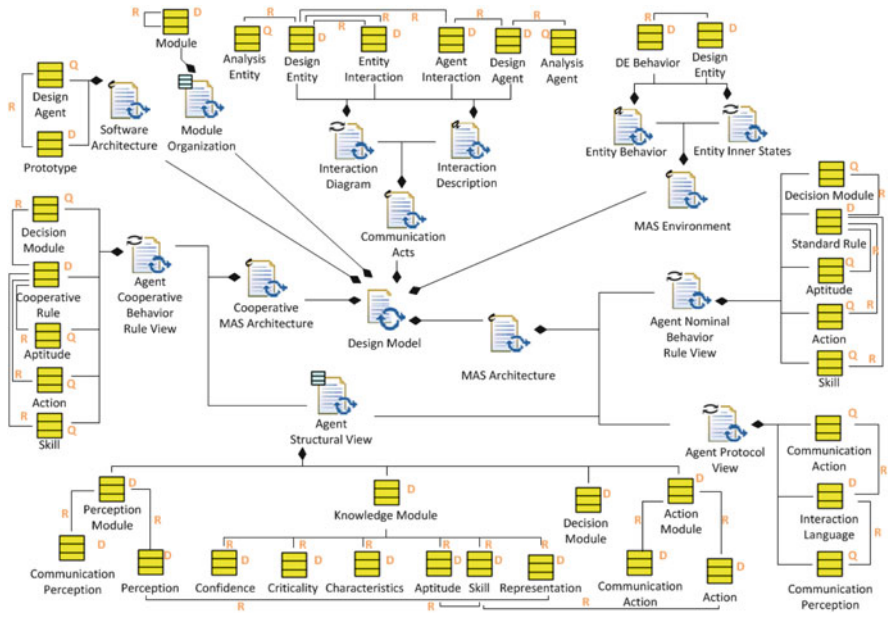


Fig. 46 The design documents structure

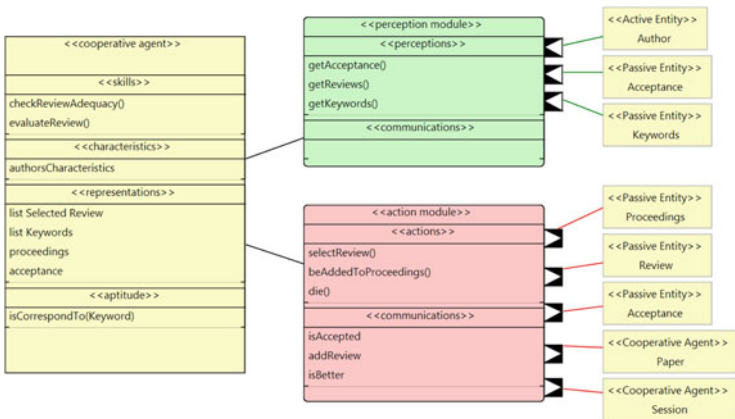
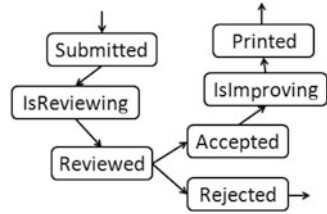


Fig. 47 Paper agent structural diagram

Example: Conference Management Study

In this section where the phase settles the design of the software architecture of each agent and entity, we only describe the paper agent behaviour. Actually, its skills, its aptitudes, its interaction language and its representations are depicted in Fig. 47. Figure 48 represents an inner state related to the current state of the paper agent. The nominal behaviour of a paper agent starts in a *Submitted* state which

Fig. 48 Inner state related to paper agent behaviour



corresponds to the creation of a paper agent. In this state, the paper agent is looking for reviews. It is in *IsReviewing* state when it finds all reviews it needs. It becomes *Reviewed* when all its reviews are complete. In order to reach the following state, *Accepted* or *Rejected*, the paper agent self-evaluates the reviews' results and changes its state. If the paper agent is *Accepted*, it informs the session of its state and becomes *IsImproving*. Finally, the paper agent is *Printed* when the proceedings are published.

Moreover, this cooperative agent can meet some non-cooperative situations. For a paper agent, two situations are detected: (1) a review can be linked to a limited number of papers; it can happen that several paper agents want the same review; (2) when the paper agents have to deal with the acceptance, they can be put in concurrence or competition.

2.5 Implementation Phase (WD5)

The Implementation Phase aims at providing the desired system. Actually, the aspects of the detailed architecture are first described using SpeADL, then implemented using the Java programming language by relying on code generated from the ADL, and finally executed to deliver the desired system. The process flow at the level of activities is depicted in Fig. 49, and Fig. 50 depicts this phase according to documents, roles and work products involved.

2.5.1 Process Roles

Two roles are involved in the Implementation Phase: the AMAS Framework Developer, and the AMAS Developer.

- *AMAS Framework Developer*: An AMAS framework developer is responsible for the description of the system architecture in the SpeAD (Species-based Architecture Description) model language during the *Implement Framework* activity and the implementation of everything that is not an agent. Actually, the AMAS framework developer implements the passive entities, the active entities and all programs required by the system such as a scheduler.
- *AMAS Developer*: An AMAS developer is responsible for the agent behaviour implementation during the *Implement Agent Behaviour* activity. He/she implements the nominal and cooperative behaviour according to the designed software architecture.

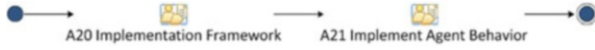


Fig. 49 The implementation phase flow of activities

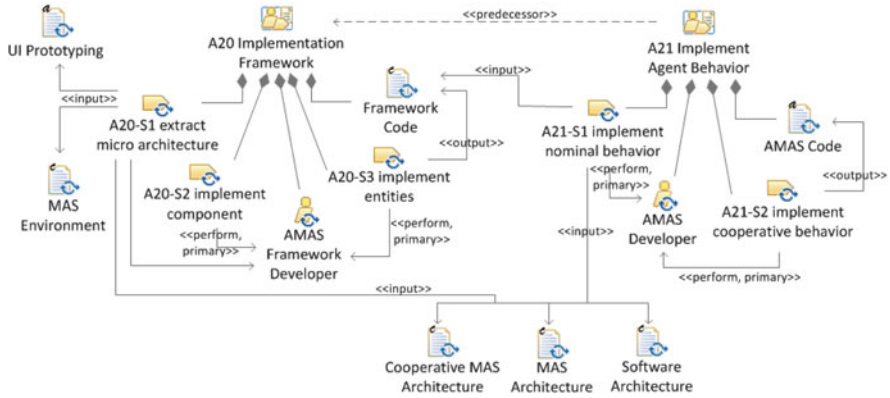


Fig. 50 The implementation phase described in terms of activities and work products

2.5.2 Activities Details

The flow of activities inside this phase is depicted in Fig. 49, and the tasks are detailed in the following table.

A20: Implement Framework

During this activity, the mechanisms are software components with provided and required services that can be composed together to form the architecture of the system. Entities and agents’ architecture is therefore described in terms of components. The architecture is described using the textual architecture description language SpeADL (Species-based Architecture Description Language). Then the architectural elements which are not a cooperative agent are implemented. The flow of tasks inside this activity is depicted in Fig. 51, and the tasks are detailed in the following table.

A20: Implement Framework		
Tasks	Tasks Descriptions	Roles Involved
S1: extract micro architecture	The previously defined software architecture is translated into SpeAD model language.	AMAS Framework Developer
S2: implement component	The AMAS framework developer implements everything that is not related to the agent or entity behaviour in the system.	AMAS Framework Developer
S3: implement entities	The AMAS framework developer implements the active and passive entities behaviour.	AMAS Framework Developer

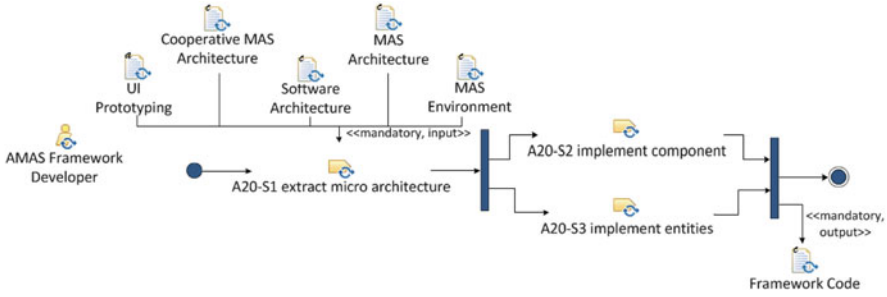


Fig. 51 Flow of tasks of the *Implement Framework* activity

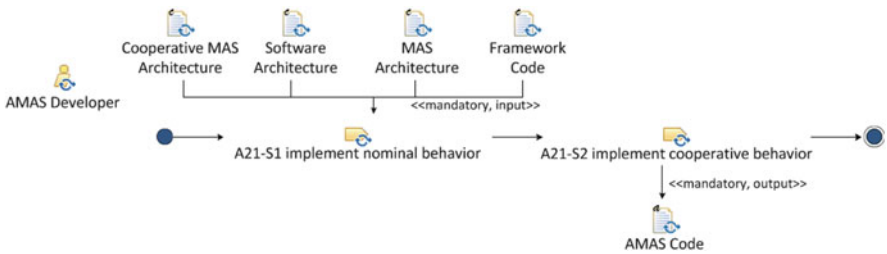


Fig. 52 Flow of tasks of the *Implement Agent Behaviour* activity

A21: Implement Agent Behaviour

During this activity, the behaviour of the cooperative agent is implemented. The flow of tasks inside this activity is depicted in Fig. 52, and the tasks are detailed in the following table.

A21: Implement Agent Behaviour		
Tasks	Tasks Descriptions	Roles Involved
S1: implement nominal behaviour	The nominal behaviour of agents is implemented by working out the agents' process decision.	AMAS Developer
S2: implement cooperative behaviour	The cooperative behaviour of agents is implemented by working out the agents' process decision which enables anticipation or detection and repair of the non-cooperative situations.	AMAS Developer

2.5.3 Work Products

The Implementation Phase generates two work products. Their relationships with the MAS metamodel elements are depicted in Fig. 53.

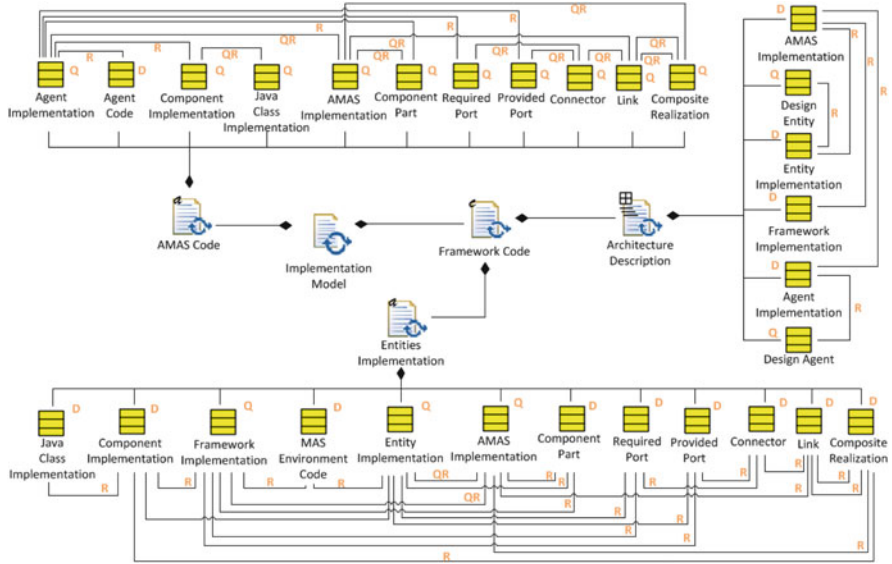


Fig. 53 The implementation documents structure

Work Products Kind

Name	Description	Work Product Kind
Framework Code	This document is composed of: 1) a textual description of the architecture of the system, according to the SpeADL language; 2) the implementation of all what is not agent.	Composite (Structured Text and Free Text)
AMAS code	This document is composed of the implementation of the cooperative agent behaviour (nominal behaviour and cooperative behaviour).	Composite (Structured Text and Free Text)

Example: Conference Management Study

Figure 54 is a graphic description of an SpeADL. This architecture defined with SpeAD is made of components connected together with simple connectors. The components externally provide ports, for which they have an implementation, and require ports that they can use in their implementation. Note that the description of components made with SpeADL will then be translated to Java.

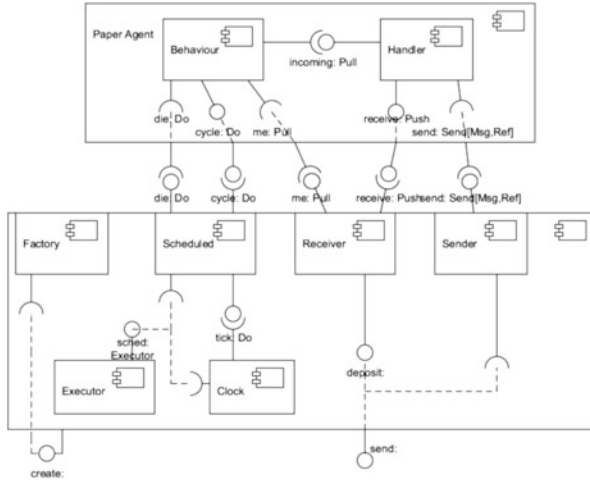


Fig. 54 Architectural description of paper agent

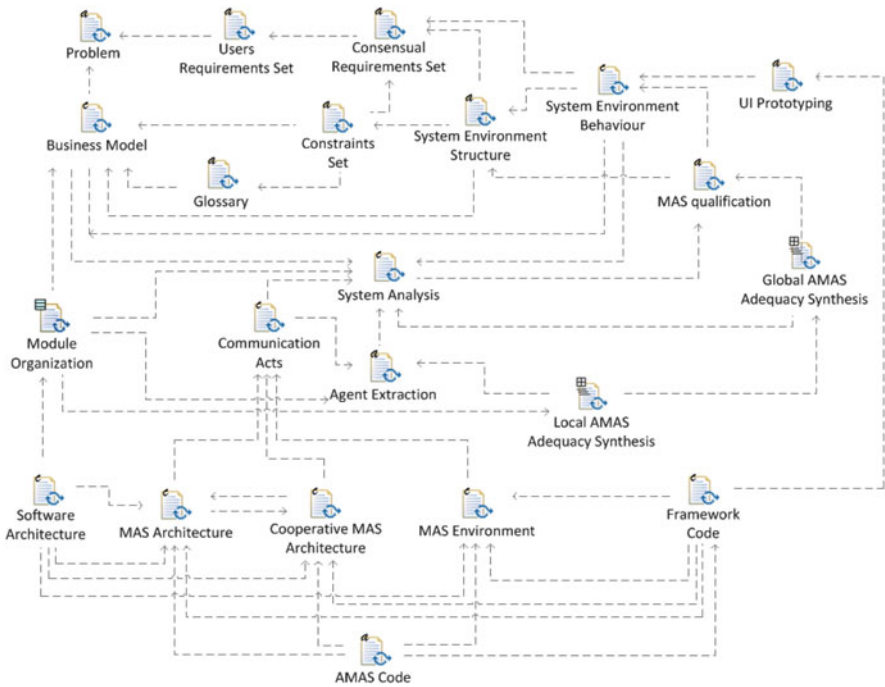


Fig. 55 The work products dependencies

3 Work Product Dependencies

Figure 55 depicts the dependencies among the different work products produced by the process. A dashed arrow is used to relate two of them if one is an input document to the other. Its direction points from the consumer document to the input one.

The ASPECS Process

Massimo Cossentino, Vincent Hilaire, Nicolas Gaud,
Stephane Galland, and Abderrafaa Koukam

Abstract

This chapter introduces an agent-oriented software process for engineering complex systems called ASPECS. ASPECS is based on a holonic organizational metamodel and provides a step-by-step guide from requirements to code, allowing the modeling of a system with different levels of details using a set of refinement methods. This chapter introduces the ASPECS process using the documentation template provided by the IEEE FIPA DPDF Working Group. One of the founding principles of ASPECS is to combine both holonic structures and organizational approaches to ease the modeling and development of complex software applications. The target scope for the proposed approach can be found in complex systems and, especially, hierarchical complex systems. ASPECS is mainly suitable for open large-scale MAS. The main vocation of ASPECS is toward the development of holonic (as well as not-holonic) societies of software agents.

1 Introduction

ASPECS is a step-by-step requirement-to-code software process for engineering Complex Systems using Multiagent Systems and Holonic Multiagent Systems [5]. To deal with all aspects of complex systems, multiagent systems must deal with multiple levels of abstractions and openness. These multiple levels are frequently not taken into account for most solutions [12].

M. Cossentino
ICAR Institute, National Research Council, Palermo, Italy
e-mail: cossentino@pa.icar.cnr.it

V. Hilaire (✉) • N. Gaud • S. Galland • A. Koukam
IRTES-SET, UTBM, UPR EA 7274, 90 010 Belfort cedex, France
e-mail: vincent.hilaire@utbm.fr

The presence of multiple levels is emphasized in many works, such as that of Simon [15], who postulates that complex systems often (if not always) exhibit a hierarchical configuration.¹ The idea is that the architecture of a complex system can be explained and understood using hierarchical organization structures as presented in [16]. Several metamodels and methodologies have been proposed for Multiagent systems [1]. However, most of them consider agents as atomic entities. There is no intuitive or natural way to deal with hierarchical organization structures. Considering agents as composed entities thus enables the modeling of nested hierarchies and proposes a solution to this problem.

Based on this statement, ASPECS exploits the concepts of holons: agents who may be composed of agents. ASPECS combines both holonic and organizational perspectives within a single metamodel, allowing the modeling of a system at different levels of details using a set of refinement methods.

The authors of ASPECS tried to gather the advantages of organizational approaches as well as those of the holonic vision in modeling complex systems. The result is a set of organization-oriented abstractions that have been integrated into a complete methodological process. The target scope for the proposed approach can be found in complex systems and, especially, hierarchical complex systems. The main vocation of ASPECS is toward the development of societies of holonic (as well as not-holonic) multiagent systems. The interested reader could find information about ASPECS on the ASPECS website <http://www.aspecs.org> and in [3–5]. A specific deployment platform has been developed [8] and may be accessed through the website <http://www.janus-project.org>.

1.1 Global Process Overview (Life Cycle)

The ASPECS life cycle consists of three phases organized in an iterative incremental process (see Fig. 1). Each phase is briefly described below.

The *System Requirements* phase aims at identifying a hierarchy of organizations, whose global behavior may fulfill the system requirements under the chosen perspective. It starts with a Domain Requirements Description (DRD) activity where requirements are identified by using classical techniques such as use cases. Domain knowledge and vocabulary associated to the Problem Domain are then collected and explicitly described in the Problem Ontology Description (POD) activity. Then, requirements are associated to newly defined organizations. Each organization will therefore be responsible for exhibiting a behavior that fulfills the requirements it is responsible for. This activity is called Organization Identification, and it produces an initial hierarchy of organizations that will later be extended and updated, with further iterations, in order to obtain the global organization hierarchy representing the system structure and behavior. The behavior of each organization is realized by a set of interacting roles whose goals consist in contributing to the

¹Hierarchical here is meant as a “loose” hierarchy as presented by Simon.

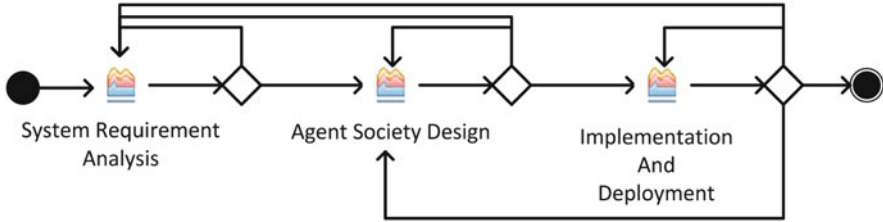


Fig. 1 The ASPECS process phases (and iterations)

fulfillment of (a part of) the requirements of the organization within which they are defined. In order to design modular and reusable organization models, roles are specified without making any assumptions about the structure of the agent who may play them. To meet this objective, the concept of capacity has been introduced. A capacity is an abstract description of a know-how, i.e., a competence of a role. Each role requires certain skills to define its behavior, and these skills are modeled by means of a capacity. Besides, an entity that wants to play a role has to be able to provide a concrete realization for all the capacities required by the role. Finally, the last step of the System Requirements phase, the Capacity Identification activity, aims at determining the capacities required by each role.

The second phase is the *Agent Society Design* phase that aims at designing a society of agents whose global behavior can provide an effective solution to the problem described in the previous phase and to satisfy associated requirements. The objective is to provide a model in terms of social interactions and dependencies between entities (holons and agents). Previously identified elements such as ontology, roles, and interactions are now refined from the social point of view (interactions, dependencies, constraints, etc.). At the end of this design phase, the hierarchical organization structure is mapped into a holarchy (hierarchy of holons) in charge of realizing the expected behaviors. Each of the previously identified organizations is instantiated in form of groups. Corresponding roles are then associated to holons or agents. This last activity also aims at describing the various rules that govern the decision-making process performed inside composed holons as well as the holons' dynamics in the system (creation of a new holon, recruitment of members, etc.). All of these elements are finally merged to obtain the complete set of holons involved in the solution.

The third and last phase, namely *Implementation and Deployment*, firstly, aims at implementing the agent-oriented solution designed in the previous phase by deploying it to the chosen implementation platform, in our case, JANUS. Secondly, it aims at detailing how to deploy the application over various computational nodes (JANUS kernels in our experiments). Based on JANUS, the implementation phase details activities that allow the description of the solution architecture and the production of associated source code and tests. It also deals with the solution reusability by encouraging the adoption of patterns. The Code Reuse activity aims at integrating the code of these patterns and adapting the source code of

previous applications inside the new one. It is worth noting that although we will refer to a JANUS-based implementation, systems developed by using other platforms can be designed as well with the described process. This phase ends with the description of the deployment configuration; it also details how the previously developed application will be concretely deployed; this includes studying distribution aspects, holons physical location(s) and their relationships with external devices and resources. This activity also describes how to perform the integration of parts of the application that have been designed and developed by using other modeling approaches (i.e., object-oriented ones) with parts designed with ASPECS.

1.2 Metamodel

ASPECS has been built by adopting the Model Driven Architecture (MDA) [11], and thus we defined three levels of models, each referring to a different metamodel. We also label the three metamodels “domains,” thus maintaining the link with the PASSI metamodel that was one of our inspiration sources. The three domains we define are:

Problem Domain. It provides the organizational description of the problem independently of a specific solution. The concepts introduced in this domain are mainly used during the analysis phase and at the beginning of the design phase.

Agency Domain. It introduces agent-related concepts and provides a description of the holonic, multiagent solution resulting from a refinement of the Problem Domain elements.

Solution Domain. It is related to the implementation of the solution on a specific platform. This domain is thus dependent on a particular implementation and deployment platform. In our case, this part of the process is based upon the JANUS platform that we specifically designed to ease the implementation of holonic and organizational models. A complete description of the JANUS platform would take too much space to be dealt by this paper, and therefore, we prefer to present only the most significant JANUS issues. The interested reader can find more details in [8] and on the JANUS website².

The following sub-sections detail the three domain metamodels and some fundamental concepts within them. A complete description of all the elements reported in the metamodels is present on the ASPECS website and will not be reported here because of space concerns.

1.2.1 Problem Domain

The Problem Domain metamodel (see Fig. 2) includes elements that are used to catch the problem requirements and perform their initial analysis: Requirements (both functional and non-functional) are related to the organization that fulfills them.

²JANUS: <http://www.janus-project.org/>

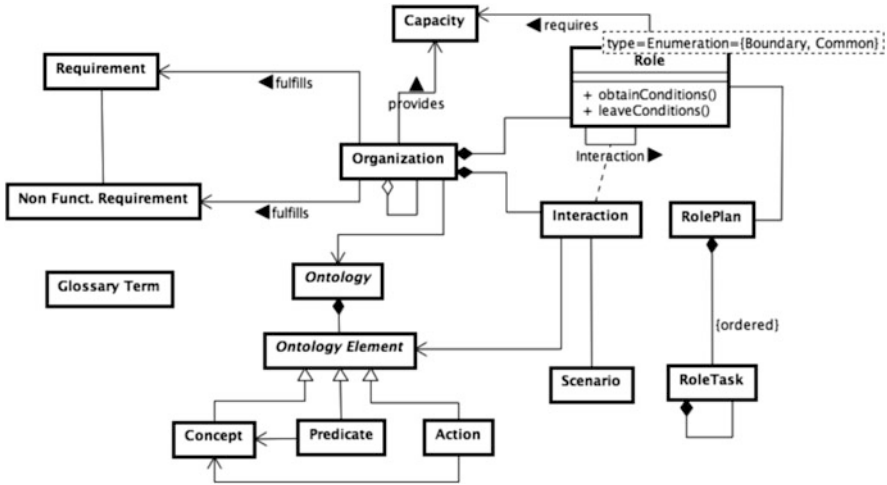


Fig. 2 The ASPECS Problem Domain MAS metamodel

An organization is composed of Roles, which are interacting within scenarios while executing their Role plans. An organization has a context that is described in terms of an ontology. Roles participate in the achievement of their organization goals by means of their Capacities. Definitions of MAS metamodel elements may be found in Table 1, on the ASPECS website, and in [5].

1.2.2 Agency Domain

The Agency Domain metamodel (see Fig. 3) includes the elements that are used to define an agent-oriented solution for the problem analyzed in the previous stage. By adopting an organizational approach, the solution will be mainly composed of the necessary social structures designed in a multi-perspective way. Definitions of the MAS metamodel elements may be found in Table 2, further details on the ASPECS website, and in [5].

1.2.3 Solution Domain

The Solution Domain metamodel contains elements used for the implementation of the designed solution in the chosen platform. These elements are general enough to be applied to several existing platforms with minor or no changes but nonetheless, the most suitable choice is JANUS that directly inspired this portion of the metamodel.

JANUS (see [8]) was designed to facilitate the transition between the design and implementation phases of holonic systems development processes. It is implemented in Java, and it supports the direct implementation of the five key concepts used in the design phase: organization, group, role, agent, and capacity.

The definitions of the ASPECS Solution Domain MAS metamodel elements are given in Table 3, further details on the ASPECS website, and in [5].

Table 1 Definition of the Problem Domain concepts (from [5])

Concept	Definition
Ontology	An explicit specification of a conceptualization of a knowledge domain [10]. An ontology is composed of abstract ontology elements having three possible concrete types: Concept , Predicate or Action .
Concept	A category, an abstraction that shortens and summarizes a variety/multiplicity of objects by generalizing common identifiable properties.
Predicate	Assertions on concepts properties.
Action	A change realized by an entity that modifies one or more properties of one or more concepts.
Organization	An organization is defined by a collection of roles that take part in systematic institutionalized patterns of interactions with other roles in a common context. This context consists in shared knowledge and social rules/norms, social feelings, and is defined according to an ontology. The aim of an organization is to fulfill some requirements.
Role	An expected behavior (a set of role tasks ordered by a plan) and a set of rights and obligations in the organization context. The goal of each Role is to contribute to the fulfillment of (a part of) the requirements of the organization within which it is defined. A role can be instantiated either as a Common Role or Boundary Role. A Common Role is a role located inside the designed system and interacting with either Common or Boundary Roles. A Boundary Role is a role located at the boundary between the system and its outside, and it is responsible for interactions happening at this border (i.e., GUI, Database, etc.).
Interaction	A dynamic, not a priori known sequence of events (a specification of some occurrence that may potentially trigger effects on the system) exchanged among roles, or between roles and entities outside the agent system to be designed. Roles may react to the events according to their behaviors.
Capacity	A capacity is an abstract description of a know-how, i.e., a competence of a role. Each role requires certain skills to define its behavior, and these skills are modeled by means of a capacity. It may be considered as a specification of the pre- and post-conditions of a goal achievement.
Role Task	An activity that defines a part of a role behavior. A <i>Role Task</i> may be atomic or composed by a coordinated sequence of subordinate <i>Role Tasks</i> . The definition of these <i>Role Tasks</i> can be based on capacities, required by roles.
Role plan	The behavior of a <i>Role</i> is specified within a <i>Role plan</i> . It is the description of how to combine and order <i>Role Tasks</i> and interactions to fulfill a (part of) requirement.
Scenario	Describes a sequence of role interactions, which fulfills a (part of) requirement.

2 Phases of the Process

2.1 System Requirement Analysis

The process flow at the level of activities is reported in Fig. 4. The process flow inside each activity will be detailed in the following subsections (after the description of process roles). The System Requirement Analysis phase involves two different process roles, seven work products as described in Figs. 5 and 6. The

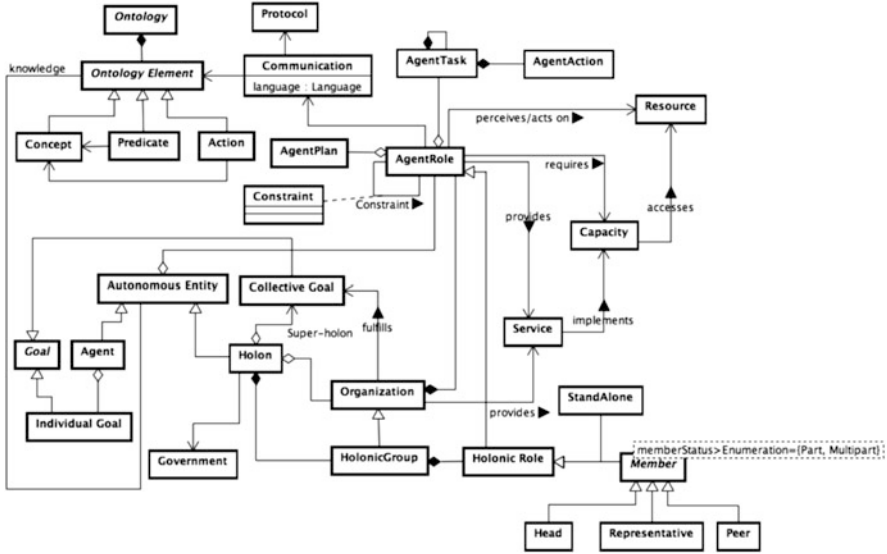


Fig. 3 The ASPECS Agency Domain MAS metamodel

phase has been divided in two figures for clarity reasons. Each figure is complete in terms of activities and corresponding roles and work products. The second figure adds the required predecessor reference(s) to activity(ies) presented in the first figure. The phase is composed of seven activities (i.e., Domain Requirements Description, Problem Ontology Identification, Organization Identification, Interaction and Role Identification, Scenario Description, Role Plan, and Capacity Identification), each of them composed of one or more tasks.

2.1.1 Process Roles

Two roles are involved in the System Requirements phase: the System Analyst and the Domain Expert. They are described in the following subsections.

System Analyst

She/he is responsible for:

1. Use cases identification during the DRD activity. Use cases are used to represent system requirements.
2. Use cases refinement during the DRD activity. Use cases are refined with the help of a Domain Expert.
3. Definition of an ontology for the conceptualization of the problem during the POD activity.
4. Use cases clustering during the Organization Identification (OID) activity. The System Analyst analyzes the use case diagrams resulting from the first activity and the domain concepts resulting from the second activity and attempts to assign use case to organizations in charge of their realization.

Table 2 Definition of the Agency Domain concepts extracted from [5]

Concept	Definition
Communication	An interaction between two or more roles where the content (language, ontology, and encoding) and the sequence of communication acts (protocol) are explicitly detailed. A <i>communication</i> is composed of messages expressing <i>communicative acts</i> [6, 7]. In a communication, participants are <i>Agent Roles</i> , and the knowledge exchanged between them is explicitly represented by a set of ontology elements.
Protocol	The sequence of expected message communicative acts; it represents a common pattern of communication, a high-level strategy that governs the exchange of information between <i>Agent Roles</i> .
Group	An instance in the Agency Domain of an <i>Organization</i> defined in the Problem Domain. It is used to model an aggregation of <i>Agent Roles</i> played by holons.
Agent Role	An instance of the Problem Domain <i>Role</i> . It is a behavior (expressed by a set of Agent Tasks), and it owns a set of rights and obligations in a specific group context. <i>Agent Roles</i> interact with each other by using communications within the context of the group they belong to. Several <i>Agent Roles</i> are usually aggregated in the <i>Autonomous Entity</i> that plays them. An <i>Agent Role</i> may be responsible for providing one of more services to the remaining part of the society.
Holonic Group	A group that is devoted to contain <i>holonic roles</i> and takes care of the holon internal decision-making process (composed-holon's government). Holonic roles are used to represent in an organizational way the notion of moderated group (see [9]). They describe the level of authority of a member inside the holon members' community and the degree of commitment of a member to its super-holon.
Agent Task	An <i>Agent Task</i> is a refinement of a Problem Domain <i>Role Task</i> . It is a portion of a role behavior, and it may be composed by other <i>Agent Tasks</i> or atomic <i>Agent Actions</i> . It may contribute to provide (a portion of) an <i>Agent Role's</i> service.
Agent Action	The atomic composing unit of a behavior. An action takes a set of inputs and converts them into a set of outputs, though either or both sets may be empty. An example of the most basic <i>Agent Action</i> consists in invoking a capacity or a service requiring the same inputs.
Autonomous Entity	An abstract rational entity that adopts a decision in order to obtain the satisfaction of one or more of its own goals. An autonomous entity may play a set of <i>Agent Roles</i> within various groups. These roles interact with each other in the specific context provided by the entity itself. The entity context is given by the knowledge, the capacities owned by the entity itself. Roles share this context by the simple fact of being part of the same entity.
Agent	An autonomous entity that has specific individual goals and the intrinsic ability to realize some capacities.
Goal	A description of an objective to pursue and represents an abstraction of a projected state of affairs to obtain.
Individual Goal	A goal pursued by an individual agent that may be related to its personal desires or intentions. This agent will deliberate to determine a plan or a strategy to achieve its individual goals.

(continued)

Table 2 (continued)

Concept	Definition
Collective Goal	A goal pursued by a community of individuals, which has the commitment of (a part of) the community members. Usually members commit to collective goals because achieving these goals contributes to the achievement of members' individual goals.
Service	It provides the result of the execution of a capacity thus accomplishing a set of functionalities on behalf of its owner: a role, a group, an agent or a holon. These functionalities can be effectively considered as the concrete implementation of various capacities. A role can thus publish some of its capacities and other members of the group can profit of them by means of a service exchange. Similarly a group, able to provide a collective capacity can share it with other groups by providing a service. A capacity is an internal aspect of an organization or an agent, while the service is designed to be shared between various organization or entities. To publish a capacity and thus allow other entities to benefit from it, a service is created.
Resource	The abstraction of an environmental entity. It may be manipulated by roles through specific capacities.

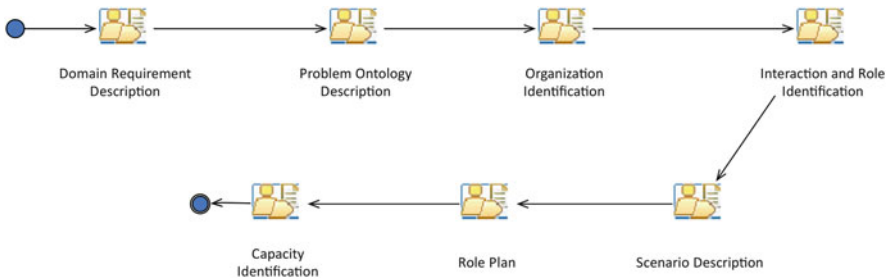
Table 3 Definition of the Solution Domain concepts extracted from [5]

Concept	Definition
JOrganization	A JOrganization is defined by a set of roles and a set of constraints to instantiate these roles (e.g., maximal number of authorized instances).
JGroup	JGroup is the runtime context of interaction. It contains a set of roles and a set of Holons playing at least one of these roles. In addition to its characteristics and its personal knowledge, each agent/holon has mechanisms to manage the scheduling of its own roles. It can change dynamically its roles during the execution of the application (leave a role and request a new one). The life cycle of a Janus agent is composed of three main phases: activation, life, termination. The life of an agent consists in consecutively executing its set of roles and capacities.
JRole	An implementation of the Agency Domain <i>Agent Role</i> .
JHolonRole	Four JHolonRoles are defined to describe the status of a member inside a super-holon: Head, Representative, Part, and Multi-Part.
JHolon	A JHolon is primarily a roles and capacities container. The roles container provides the necessary means for the roles of a holon to interact in the internal interaction context of a holon. The local mechanism of interaction inside a holon is called influence and it is implemented using an event-based communication. Each role can register itself to inform its holon that it wishes to receive all the influences of a given type.
JCapacity Implementation	A capacity can be implemented in various ways, and each of these implementations is modeled by the notion of JCapacityImplementation.

(continued)

Table 3 (continued)

Concept	Definition
JCapacity	The JCapacity concept is an interface between the holon and the roles it plays. The role requires some capacities to define its behavior, which can then be invoked in one of the tasks that make up the behavior of the role. The set of capacities required by a role are specified in the role access conditions.
JCommunication	To communicate, holons must belong to a common group and play a role in this group. If a group is distributed among several kernels, an instance of this group exists in each kernel, and all the instances have the same name. Communication in JANUS is based on the roles. Messages are delivered to agents according to their roles. This mode of interaction allows the implementation of communications between an emitter and several receivers (one-to-many communication). The address of the receiver agents is dynamically discovered according to the roles they play. When several agents play the same role within the same group, a mode of communication based on role and agents identifier may also be used to distinguish which role player will receive the message.

**Fig. 4** The System Requirement Analysis phase flow of activities

5. Identification of interacting roles for the previously identified organizations and use cases constitutes the Interaction and Role Identification (IRI) activity.
6. Refinement of the interactions between roles during the Scenario Description (SD) activity by means of scenarios designed in the form of sequence diagrams thus depicting the details of role interaction.
7. Refinement of role behaviors during Role Plan (RP) activity by means of state-transition diagrams specifying each role behavior.
8. Identification of capacities that are required by roles or provided by the organizations during the Capacity Identification (CI) activity. The capacities are added to the class diagram depicting the organizations composed of interacting roles.

Domain Expert

The Domain Expert has knowledge about the domain of the problem to be solved. S(he) is responsible for:

1. Assisting the System Analyst with the proper knowledge on the domain.
2. Contributing in the decision whether requirements are correctly identified (end of the Domain Requirements phase).

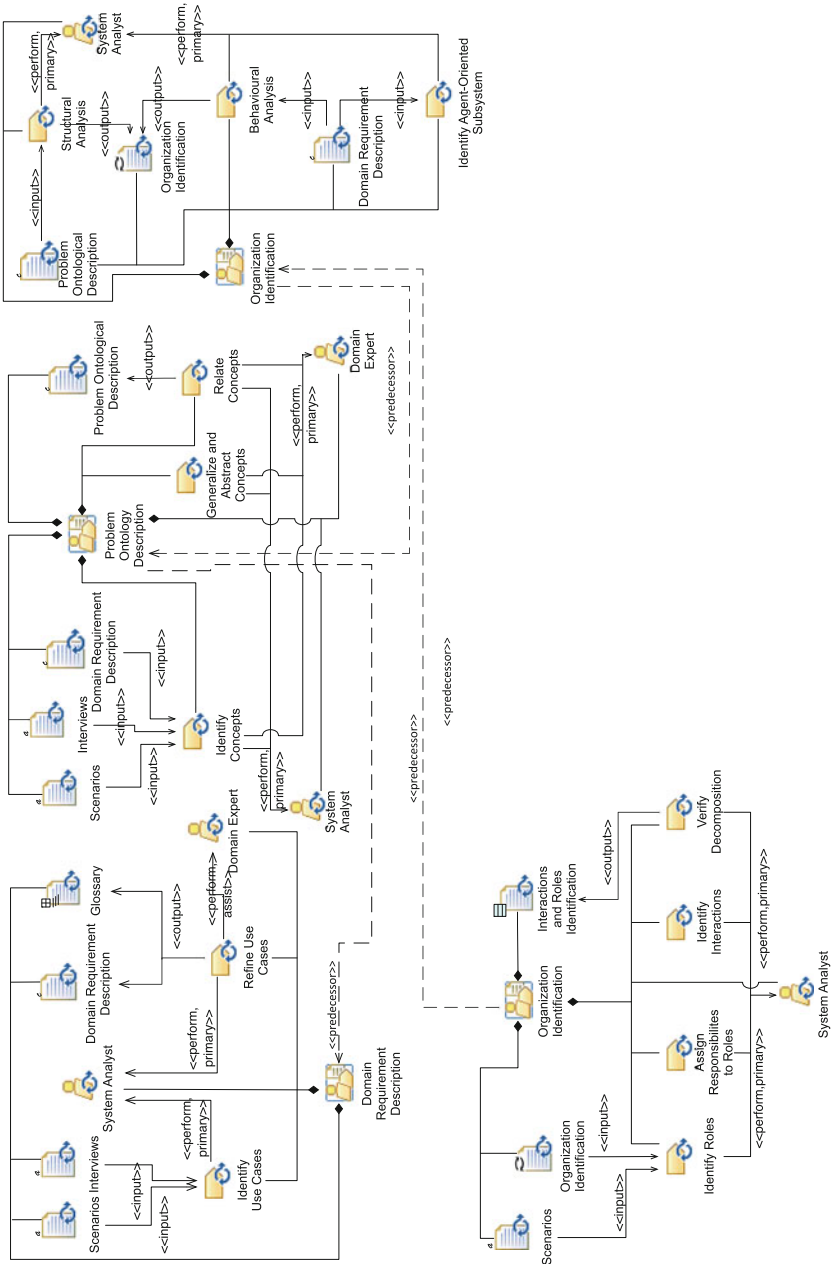


Fig. 5 The System Requirement Analysis phase described in terms of activities and work products I

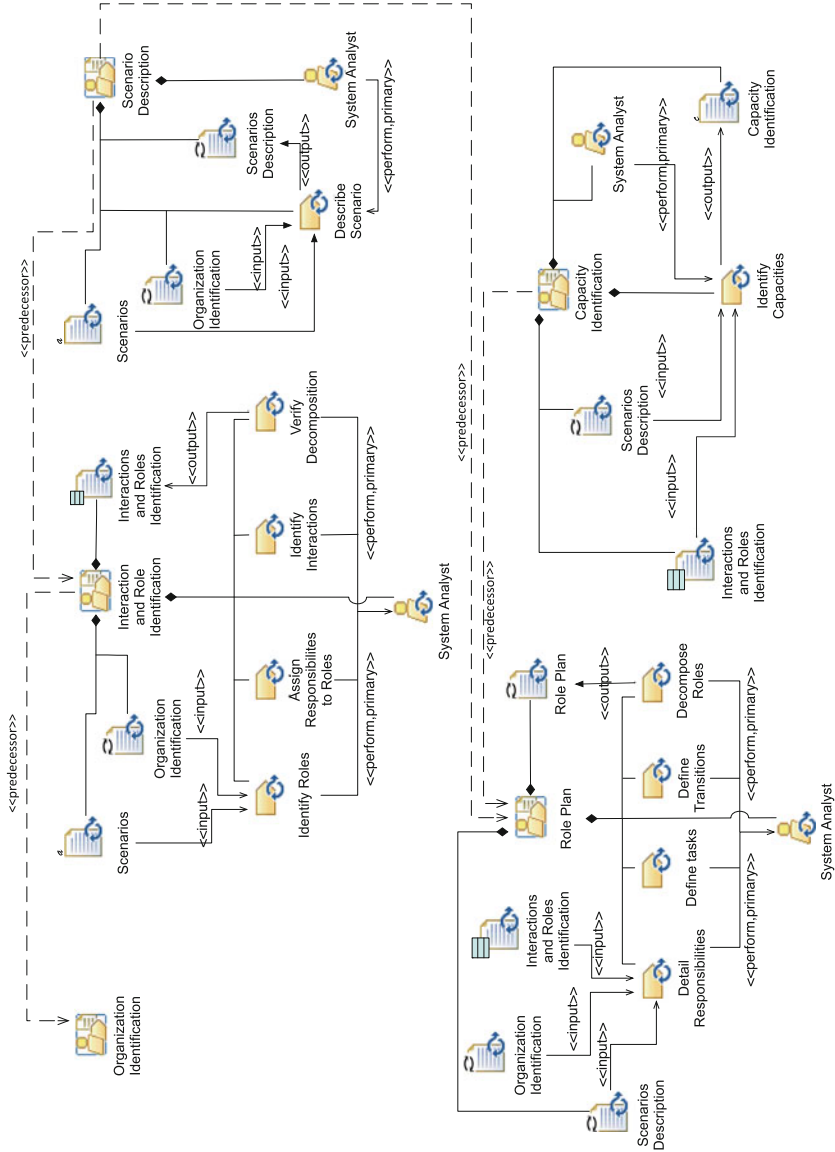


Fig. 6 The System Requirement Analysis phase described in terms of activities and work products II

2.1.2 Activity Details

Domain Requirement Description

The DRD activity aims at an initial requirements elicitation. The expected result is a description of the application behavior. Several different approaches can be used. For example, use cases diagrams and documented version to introduce user annotations can specify functional and non-functional requirements. Tasks of this activity are detailed in Table 4 and their flow is represented in Fig. 7.

Problem Ontology Description

The Domain Ontology Description activity defines a conceptual overview of the domain concerned. This ontology aims at the conceptualization of experts knowledge that will provide the applications context. Moreover, this ontology helps to understand the problem to be solved and allows requirements refinements. Among the subsequent refinements, the identification of organizations, roles, and capacities is one of the most important. Tasks of this activity are detailed in Table 5 and their flow is represented in Fig. 8.

Organization Identification

The Organization Identification activity goal consists in assigning to each requirement a behavior, which is not detailed at this level and which is represented by an organization. The meaning of this assignment is that a requirement should be satisfied by an organization. The behavior represented by the organization is the result of the interacting roles within a common context. The latter is conceptualized in the POD defined in the previous activity. This activity starts from the requirements defined in the DRD activity. From this starting point different approaches are possible to define organizations and assign requirements. A possible way is to jointly use a structural (ontological-based) approach based on the structural features already identified and a functional approach based on requirement clustering. Tasks of this activity are detailed in Table 6 and their flow is represented in Fig. 9.

Interaction and Role Identification

The IRI activity should decompose the behavior represented by an organization into finer grain behaviors represented by roles. Tasks of this activity are detailed in Table 7 and their flow is represented in Fig. 10.

Scenario Description

The Scenario Description activity should describe a set of possible interactions within an organization. The objective is to refine and explore the possible sequences of interactions between roles of a same organization. Obviously the required elements are organizations, with both roles and interactions, and requirements assigned. The challenge of this activity is to specify the fulfillment of requirements by the interactions of organization roles. Tasks of this activity are detailed in Table 8 and their flow is represented in Fig. 11.

Table 4 Task descriptions of the Domain Requirement Description activity

Activity	Task	Task description	Roles involved
Domain Requirement Description	Identify Use Cases	Use cases are used to represent system requirements	System Analyst (perform)
Domain Requirement Description	Refine Use Cases	Use cases are refined with the help of a Domain Expert	System Analyst (perform) Domain Expert (assist)

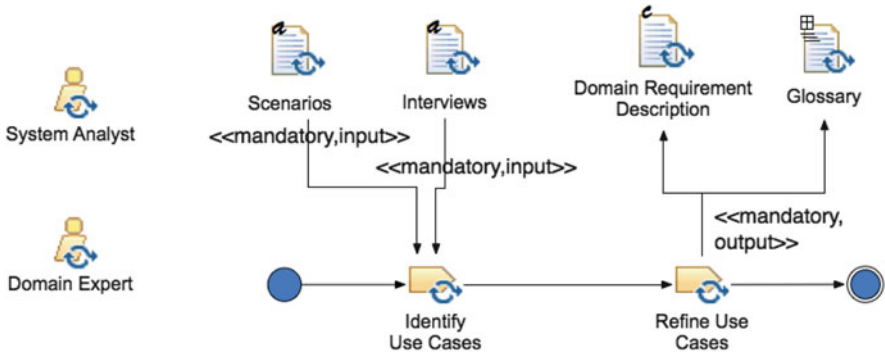


Fig. 7 The flow of tasks of the Domain Requirement Description activity

Table 5 Task descriptions of the Problem Ontology Description activity

Activity	Task	Task description	Roles involved
Problem Ontology Description	Identify Concepts	Recurrent and most significant concepts are identified in use case descriptions	System Analyst (perform), Domain Expert (assist)
Problem Ontology Description	Generalize and Abstract Concepts	Concepts are arranged by means of inheritance and abstraction.	System Analyst (perform), Domain Expert (assist)
Problem Ontology Description	Relate Concepts	Determine concept relationships and especially composition ones.	System Analyst (perform), Domain Expert (assist)

Role Plan

The Role Plan activity details the behavior of each role by a general plan. This plan is a partial fulfillment of the organization objectives. The plan is composed of RoleTasks that are elementary units of action. The different plans should conform to the scenarios described in the previous activity. One different diagram is drawn for detailing the behavior of each organization. Very complex plans may require a specific diagram depicted for each role. Tasks of this activity are detailed in Table 9 and their flow is represented in Fig. 12.

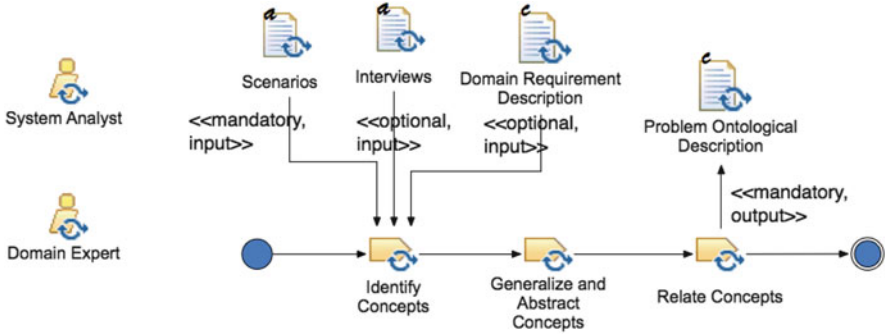


Fig. 8 The flow of tasks of the Problem Ontology Description activity

Table 6 Task descriptions of the Organization Identification activity

Activity	Task	Task description	Roles involved
Organization Identification	Identify Agent-Oriented Subsystem	Identify the parts of the systems that will be realized by using an agent-oriented approach.	System Analyst (perform)
Organization Identification	Structural Analysis	Cluster use cases considering their associations and related ontological concepts.	System Analyst (perform)
Organization Identification	Behavioral Analysis	Cluster use cases dealing with related pieces of system behavior, consider organizational design patterns too.	System Analyst (perform)

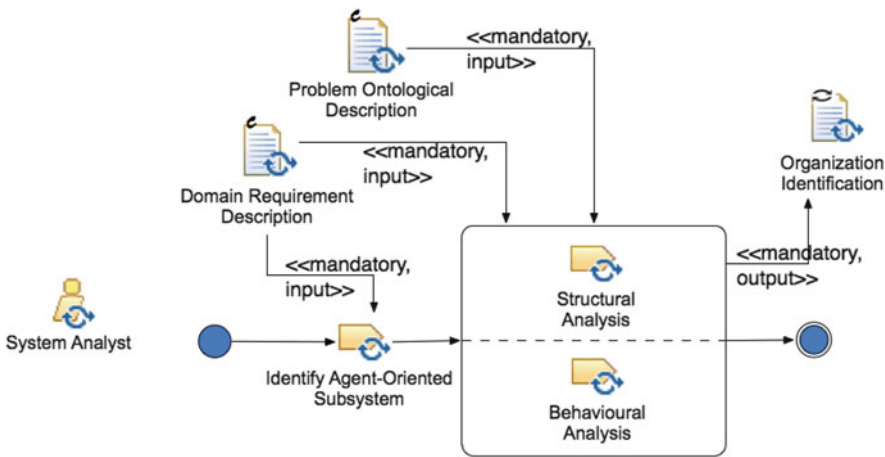


Fig. 9 The flow of tasks of the Organization Identification activity

Capacity Identification

The Capacity Identification activity contributes to the definition of generic behaviors. The underlying principle is to abstract the know-hows that are necessary

Table 7 Task descriptions of the Interaction and Role Identification activity

Activity	Task	Task description	Roles involved
Interaction and Role Identification	Identify Roles	Identify roles from problem ontology and the previously identified organizations.	System Analyst (perform)
Interaction and Role Identification	Assign Responsibilities to Roles	Responsibilities correspond to a part of the requirements associated to the role's organization.	System Analyst (perform)
Interaction and Role Identification	Identify Interactions	Role interactions can be deduced from the study of the relationship between use cases and the associated roles.	System Analyst (perform)
Interactions and Roles Identification	Verify Decomposition	Verify the goodness of the decomposition by studying the contributions from organizations at a given level to roles belonging to the upper level.	System Analyst (perform)

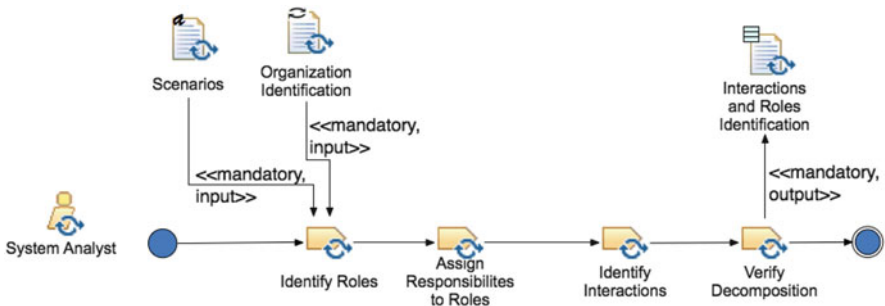


Fig. 10 The flow of tasks of the Interaction and Role Identification activity

Table 8 Task descriptions of the Scenario Description activity

Activity	Task	Task description	Roles involved
Scenario Description	Describe Scenario	Describe scenarios as a set of interacting roles working to achieve a required behavior of the system.	System Analyst (perform)

for playing the roles of an organization. This additional abstraction will allow the modularization and parameterization of the system to be (especially these choices can be made during the next phase). Indeed, the abstraction represented by a capacity can be fulfilled by any means, some unknown at this stage. The identified capacities are added to the already designed IRI diagram by adding relationships between capacities and roles. Tasks of this activity are detailed in Table 10 and their flow is represented in Fig. 13.

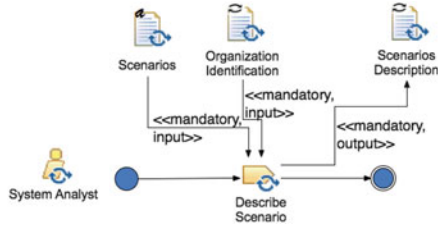


Fig. 11 The flow of tasks of the Scenario Description activity

Table 9 Task descriptions of the Role Plan activity

Activity	Task	Task description	Roles involved
Role Plan	Detail Responsibilities	Detail responsibilities assigned to the role studied in the current diagram.	System Analyst (perform)
Role Plan	Define Tasks	Identify a set of RoleTasks for accomplishing the assigned responsibilities.	System Analyst (perform)
Role Plan	Define Transitions	Define transitions among the various tasks and the set of associated conditions.	System Analyst (perform)
Role Plan	Decompose Roles	Decompose roles if they require external contributions and/or are not correctly encapsulated/coherent.	System Analyst (perform)

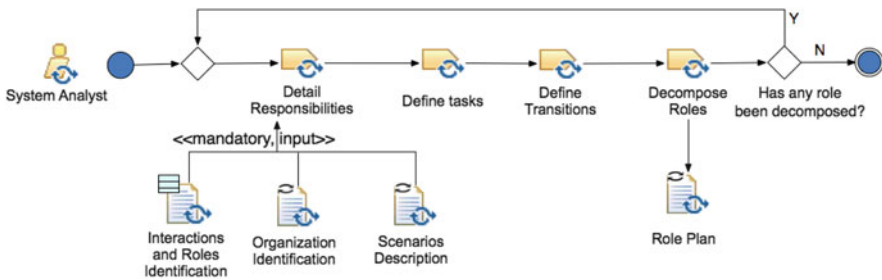


Fig. 12 The flow of tasks of the Role Plan activity

2.1.3 Work Products

The System Requirements phase generates eight work products. Their relationships with the MAS metamodel elements are described in Fig. 14.

This diagram represents the System Requirements model in terms of work products. Each of these reports one or more elements from the ASPECS MAS

Table 10 Task descriptions of the Capacity Identification activity

Activity	Task	Task description	Roles involved
Capacity Identification	Identify Capacities	Identify the generic part of the role behavior and distinguish it from all behaviors which could depend on internal properties and data of the entity which will play the role.	System Analyst (perform)

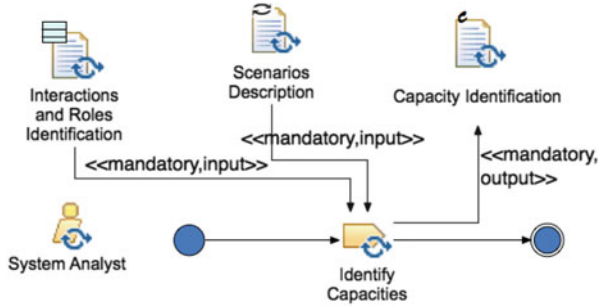


Fig. 13 The flow of tasks of the Capacity Identification activity

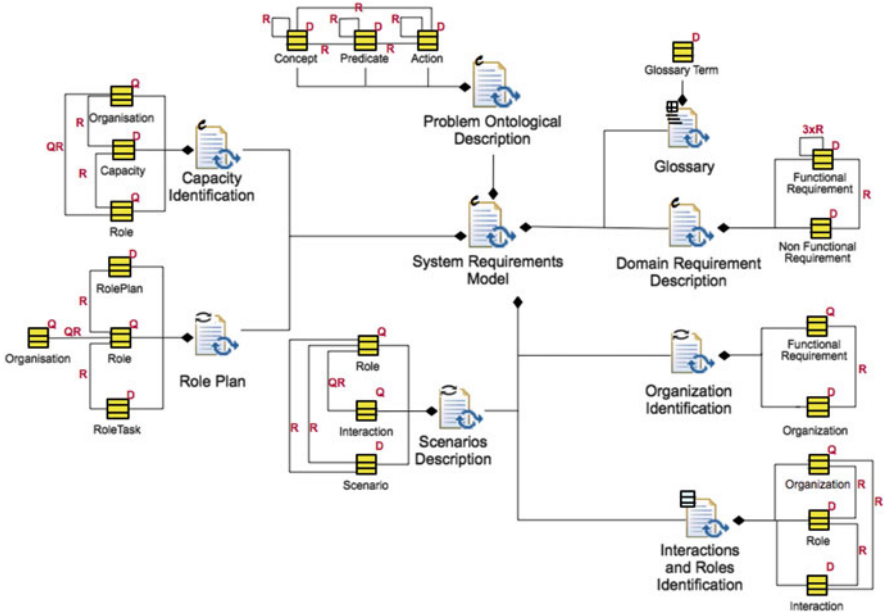


Fig. 14 ASPECS System Requirements phase work products

Table 11 ASPECS System Requirements phase work product kinds

Name	Description	Workproduct kinds
Domain Requirement Description (DRD)	A document composed by standard UML use case diagrams, textual documentation of use cases and textual description of non-functional requirements	Composite (Structured + Behavioral)
Glossary	A structured text document defining terms used in the development phases.	Structured
Problem Ontology Description (POD)	The ontology is described in terms of concepts (categories of the domain), predicates (assertions on concept properties), actions (performed in the domain, affecting the status of concepts), and their relationships. A profile for UML class diagrams is used to describe the ontology.	Composite (Structured + Structural)
Organization Identification (OID)	A class diagram reporting use cases and organizations as packages containing them. First level organizations can be decomposed into smaller ones in order to describe an organizational hierarchy.	Behavioral
Interactions and Roles Identification (IRI)	A class diagram where each role is represented by a stereotyped class and interactions are represented by associations between roles. Roles are positioned inside packages representing organizations they belong to.	Structural
Scenarios Description (SD)	Scenarios are described using stereotyped sequence diagrams	Behavioral
Role Plan (RP)	An activity diagram where swimlanes are used to partition activities of different roles and one swimlane is left for hosting external events.	Behavioral
Capacity Identification (CI)	A stereotyped class diagram where capacities are related to the roles that require them through associations. Capacity description is completed by a table detailing, for each capacity, the following fields: Name, Input, Output, Requires, Ensures, and Textual Description.	Composite (Structured + Structural)

metamodel; each MAS metamodel element is represented using a UML class icon (yellow filled) and, in the documents, such elements can be Defined, reFined, Quoted, Related or their Relationships Quoted.

Work-Product Kinds

Work-product kinds are briefly described in Table 11. In the following paragraphs a description of the notation adopted in each work product will be provided. Generally speaking, ASPECS uses UML as a modeling language but because of the specific needs of agent and holonic organizational design, the UML semantics and notation are used as reference points, and they have been extended using specific profiles (stereotypes); in fact, UML diagrams are often used to represent concepts that are not completely considered in UML and the notation has been

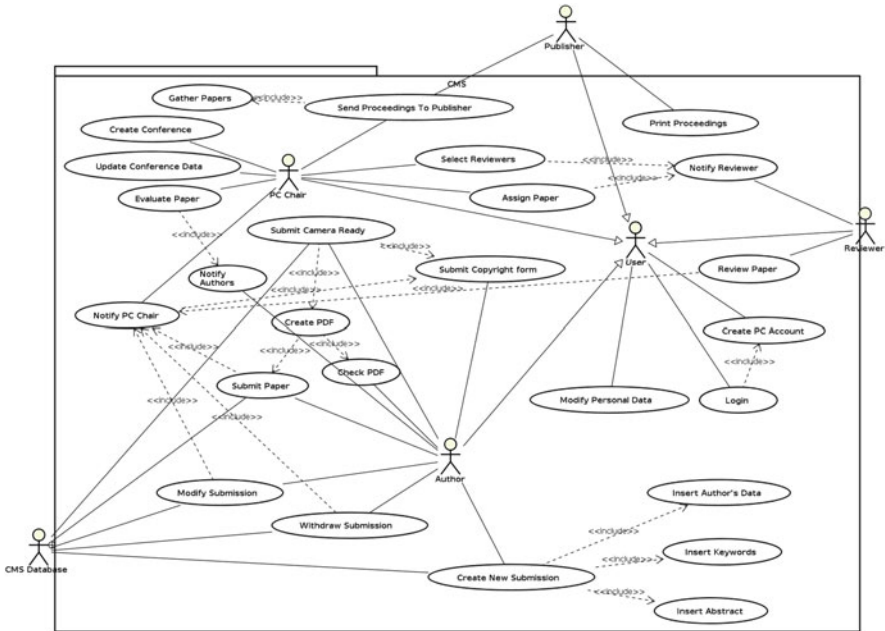


Fig. 15 CMS case study: Domain Requirement Description

modified to better meet the need of modeling agents. Almost all UML diagrams are associated with a textual document documenting and completing the content of the diagram.

Domain Requirement Description

The global objective of the Domain Requirement Description (DRD) activity is gathering needs and expectations of application stake-holders and providing a complete description of the behavior of the application to be developed. In the proposed approach, these requirements should be described by using the specific language of the application domain and a user perspective. This is usually done by adopting use case diagrams for the description of functional requirements (see Fig. 15); besides, conventional text annotations are applied to use cases documentation for describing non-functional requirements. The use cases are elicited from text scenarios and stake-holders interviews.

Problem Ontology Description

The global objective of the POD is to provide an overview of the Problem Domain. Problem ontology is modeled by using a class diagram where concepts, predicates, and actions are identified by specific stereotypes. Problem Ontology of the CMS Case study is depicted in Fig. 16. The POD is the result of the analysis of DRD, scenarios, and stake-holders interviews.

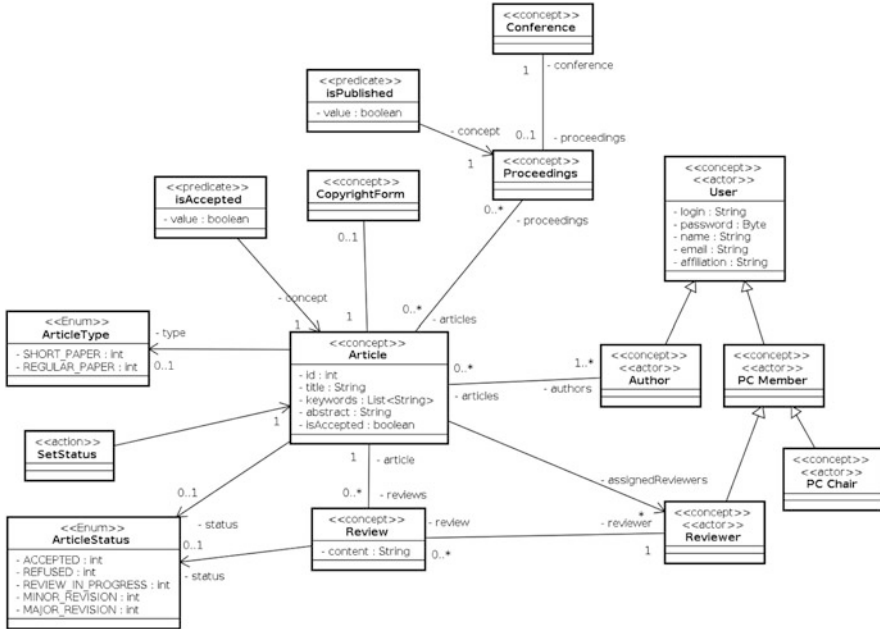


Fig. 16 CMS case study: Problem Ontology Description

Organization Identification

The work product of the Organization Identification activity (OID) refines the use case diagram produced by the DRD activity and adds organizations as packages encapsulating the fulfilled use cases. The use case diagram presented in Fig. 17 presents the organization identification diagram for the CMS Case study. The OID uses DRD and POD.

Interaction and Role Identification

The result of the IRI is a class diagram where classes represent roles (stereotypes are used to differentiate common and boundary roles), packages represent organizations and relationships describe interactions among roles or contributions (to the achievement of a goal) from one organization to another. Figure 18 shows the IRI diagram for the CMS Case study. The organizations come from OID. Scenarios are used to identify roles and interactions.

Scenario Description

Scenarios of the Scenario Description (SD) activity are drawn in the form of UML sequence diagrams and participating roles are depicted as object-roles. The role name is specified together with the organization it belongs to. Figure 19 shows the Scenario Description diagram for the example of the *Paper Submission* use case for the *Paper Management* organization of the CMS Case study. In this use case, only

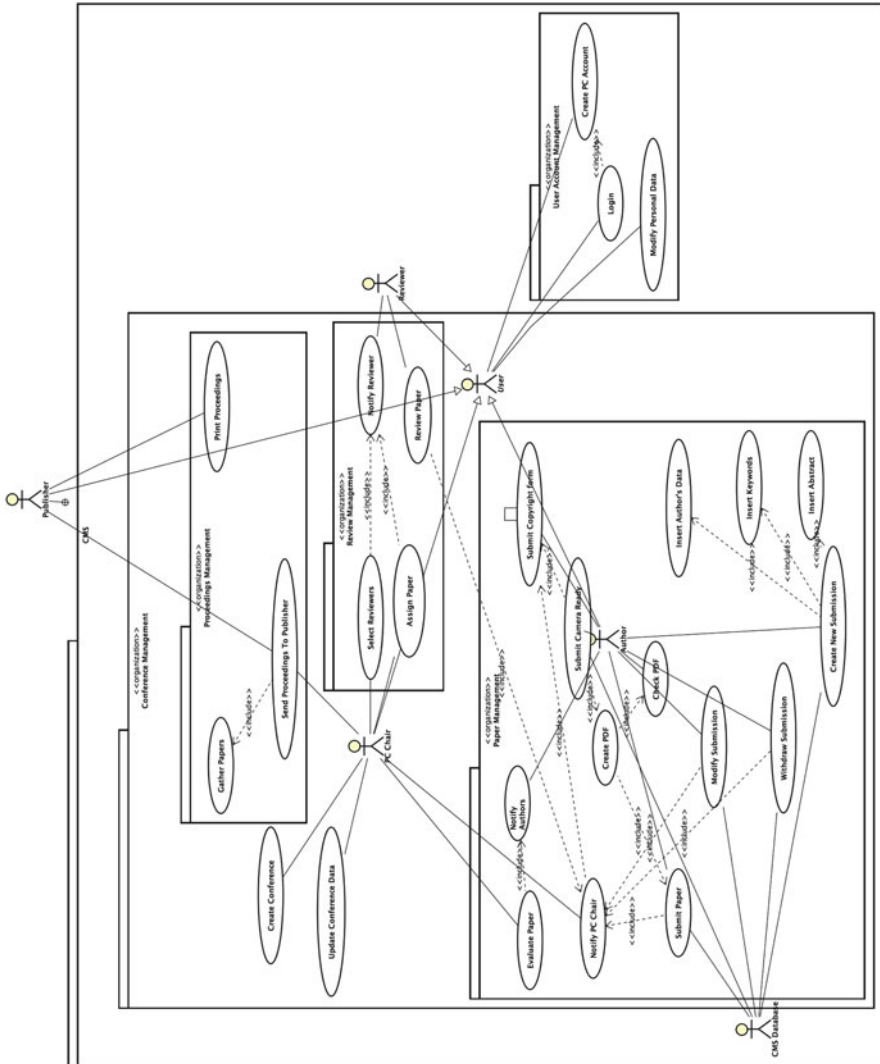


Fig. 17 CMS case study: Organization Identification Description

two of the three roles of the organization are interacting. Organizations, roles, and interactions come from OID and IRI. Obviously, scenarios are used.

Role Plan

The resulting work product of the Role Plan (RP) activity is an UML activity diagram reporting one swimlane for each role. Activities of each role are positioned in the corresponding swimlane and interactions with other roles are depicted in the form of signal events or object flows corresponding to exchanged messages.

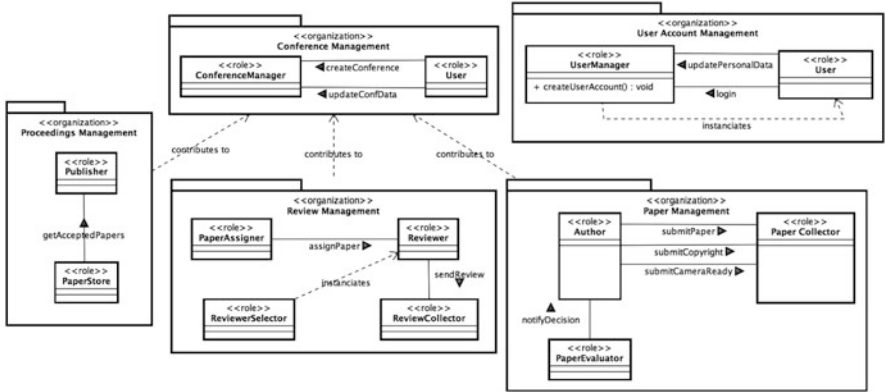


Fig. 18 CMS case study: Interaction and Role Identification

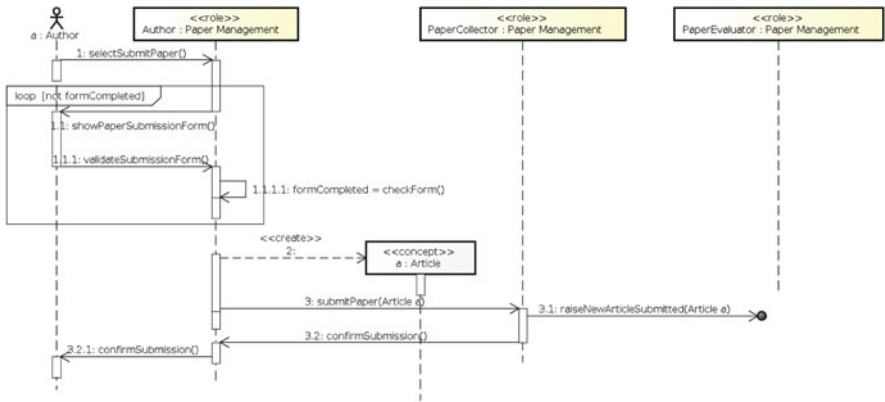


Fig. 19 CMS case study: example of the Scenario Description for the Paper Submission use case for the Paper Management organization

Figure 20 shows part of the Role Plan Description Diagram for the Paper Management organization of the CMS Case study. Roles and interactions come from IRI. SD are also used.

Capacity Identification

The work product produced by the Capacity Identification activity is a refinement of the IRI diagram including capacities (represented by classes) and relating them to the roles that require them. Figure 21 shows the Capacity Identification Diagram for the Paper Management organization of the CMS Case study. Organizations and roles come from OID and IRI.

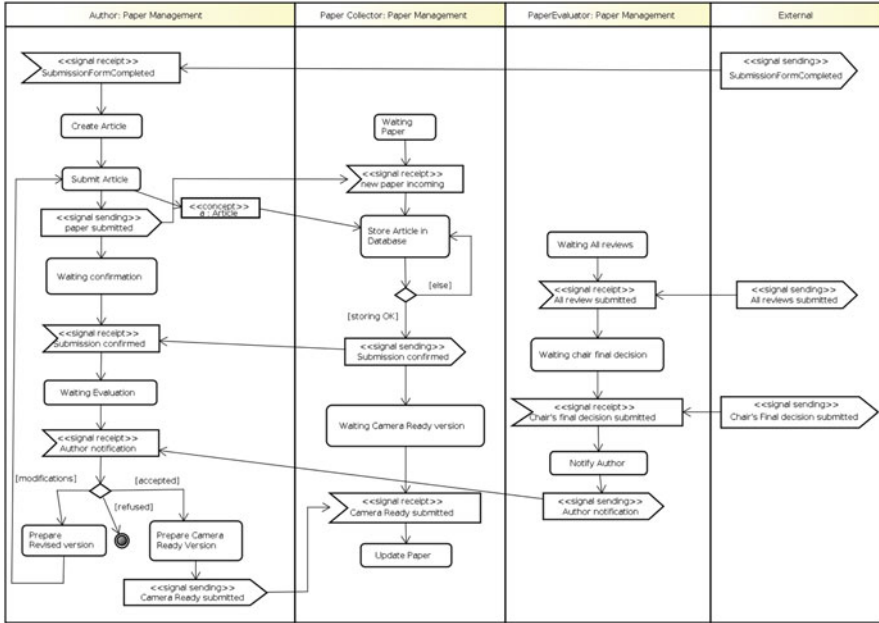


Fig. 20 CMS case study: part of the Role Plan Description for the *Paper Management* organization

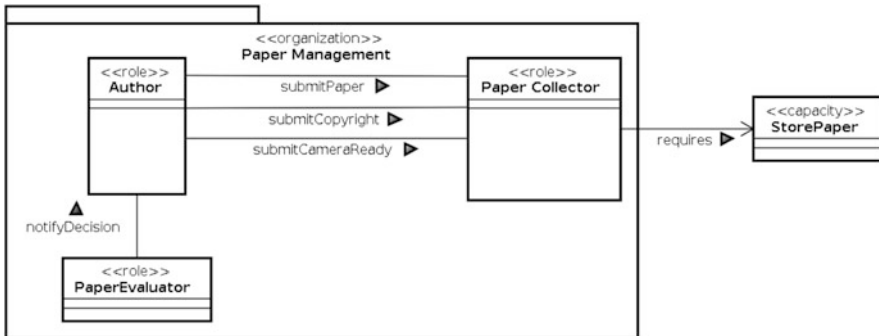


Fig. 21 CMS case study: Capacity Identification for the *Paper Management* organization

2.2 Agent Society Design

The complete sequence of activities is described in Figs. 22, 23 and 24. As it was the case for the first phase the description in terms of activities and work products has been divided in two figures for clarity reasons. Each figure is complete in terms of activities and corresponding roles and work products. The second figure adds the required predecessor reference(s) to activity(ies) presented in the first figure.

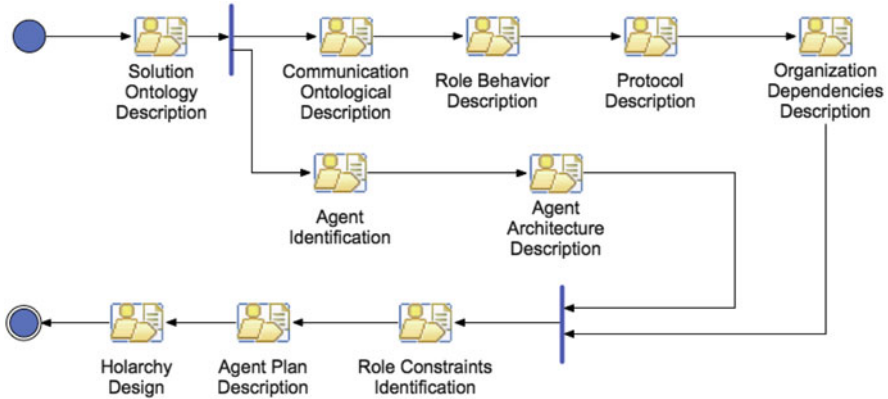


Fig. 22 The Agent Society Design phase flow of activities

2.2.1 Process Roles

In this second phase of the ASPECS process three roles are employed: Agent Designer, Domain Expert, and System Analyst. The first is a generic role which generally corresponds to a senior developer or a software architect with a good expertise in multiagent systems. The second is an expert in the Problem Domain able to help the designer in analyzing the problem and understanding its peculiarities. The third collaborates to some design activities by bringing her/his understanding of requirements to be satisfied.

Domain Expert

She/he assists in:

1. The refinement and identification of concepts.
2. The identification of actions and predicates.

Agent Designer

She/he is responsible for:

1. Refining concepts of Problem Ontology and identification of new solution-dependent concepts during Solution Ontology Description (SOD) activity.
2. Action and predicate identification and their association with previously identified concepts during SOD activity.
3. Identifying individual agents and their personal goals during Agent Identification activity. The Agent designer analyzes the solution ontology to identify individuals who are usually concepts linked to actions. Action's analysis is considered as a useful guideline to identify agent responsibilities since an individual acts according to personal motivations and goals.
4. Defining agent's architecture in Agent Architecture Description (ADD) activity.
5. Clustering interactions in communications or conversations during Communication Ontological Description (COD) activity.
6. Describing role plans using statecharts during the Role Behavior Description activity.

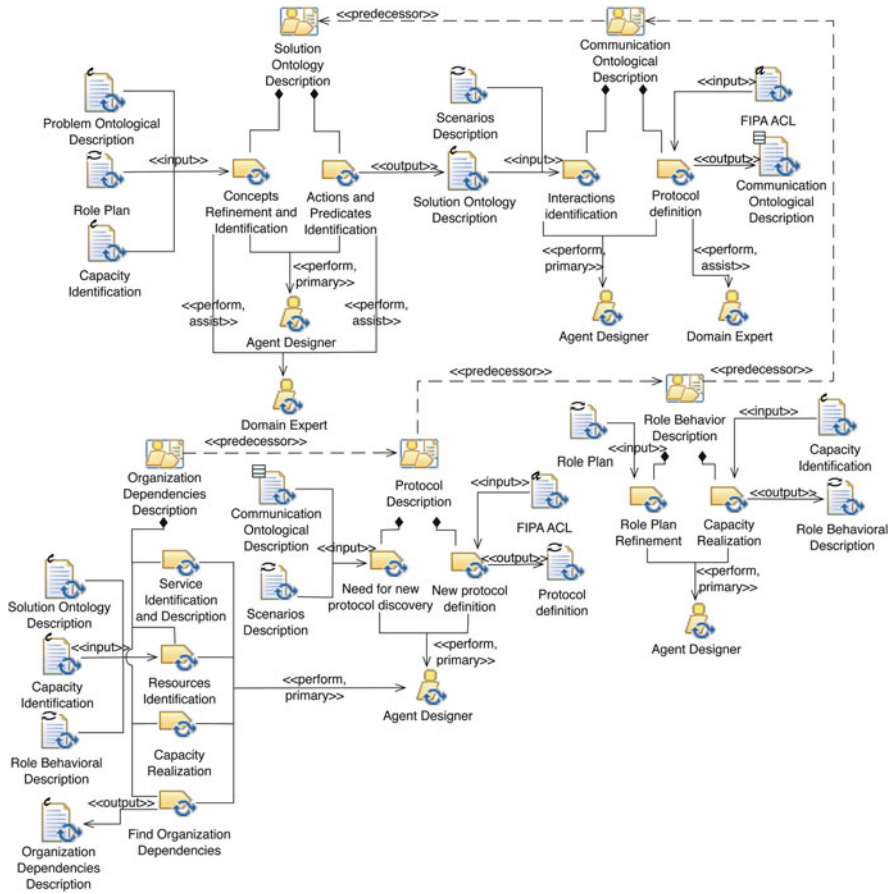


Fig. 23 The Agent Society phase described in terms of activities and work products I

7. Describing tasks and actions needed by roles to realize a capacity.
8. Defining purpose-specific interaction protocols during Protocol Description (PD) activity when the description of communications done during COD and Scenario Description activities does not match any of the existing FIPA protocols.
9. Identifying resources and services related to the different roles of solution's organizations during Organization Dependencies Description (ODD) activity.
10. Identifying constraints between roles during Role Constraints Identification (RCI) activity.
11. Identifying agents' responsibilities, defining lowest level roles and identifying agents during the Agent Identification (AI) activity.
12. Determining agent's personal plan during Agent Plan Description (APD) activity.

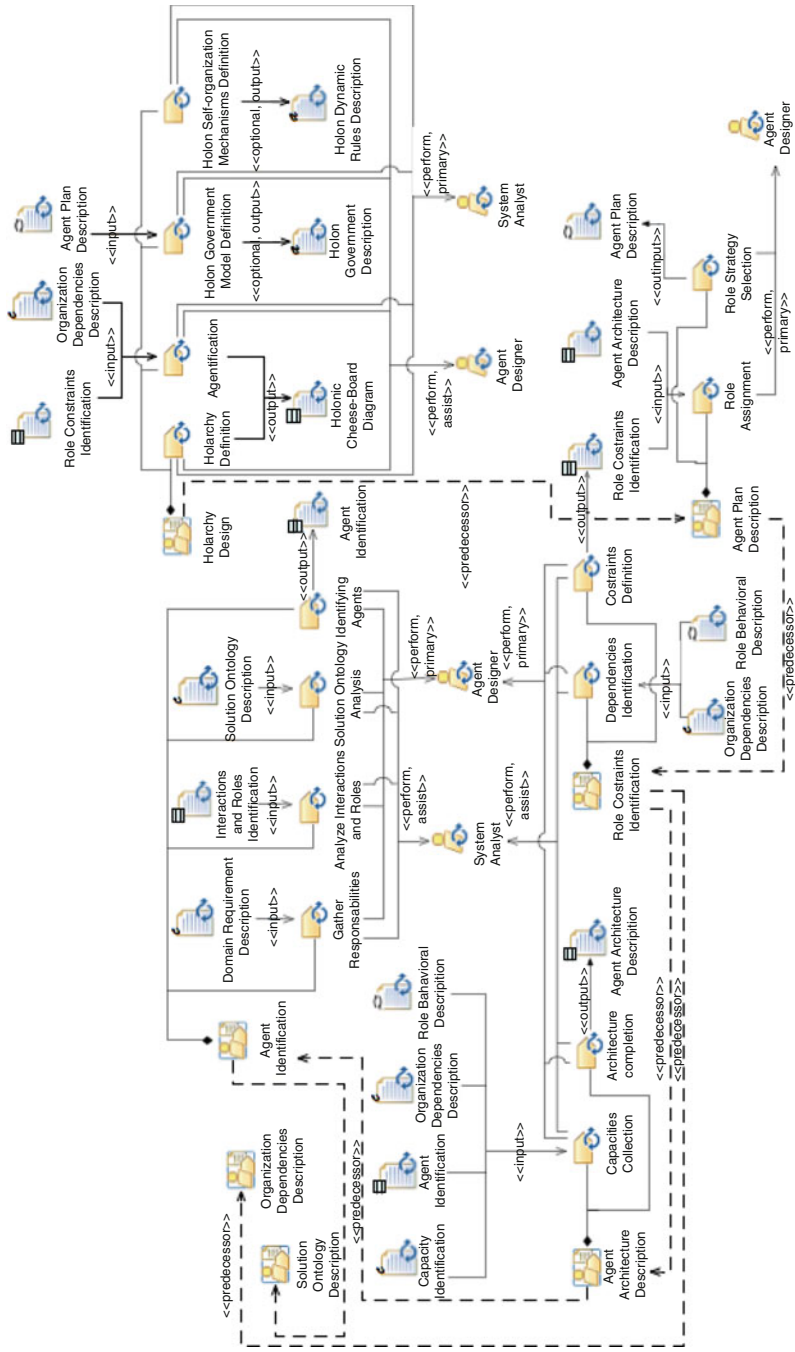


Fig. 24 The Agent Society phase described in terms of activities and work products II

Table 12 Task descriptions of the Solution Ontology Description activity

Activity	Task	Task Description	Roles involved
Solution Ontology Description	Concepts refinement and identification	Existing concepts are refined and new ones may be identified	Agent Designer (perform), Domain Expert (assist)
Solution Ontology Description	Actions and predicates identification	Actions and predicates are identified with the help of the Domain Expert	Agent Designer (perform), Domain Expert (assist)

13. Defining the complete solution structure during Holarchy Design (HD) activity: mapping between agent and role, agent composition.
14. Specifying the rules that govern the dynamics of holarchies during HD activity.

System Analyst

She/he assists in:

1. Identifying agents' responsibilities, defining lowest level roles and identifying agents during the Agent Identification (AI) activity.
2. Defining agent's architecture in ADD activity.
3. Identifying constraints between roles during RCI activity.

2.2.2 Activity Details

Solution Ontology Description

The objective of this activity consists in refining the problem ontology described during POD by adding new concepts related to the agent-based solution and refining the existing ones.

For concept identification, a guideline consists in looking into previously identified organizations. The hierarchical composition of two organizations generally hides a composition between concepts. This can also be done by looking into role plans and scenarios. Identification of actions and predicates can be facilitated by results of the capacity identification activity. As we have already said, if a capacity deals with some ontological knowledge, manipulated concepts should be connected to an action in the corresponding ontology. Moreover, the solution ontology will also be exchanged in agent communications; an indication about which knowledge will be necessary to roles' behaviors and in which activities it will be used can be found in role plans and scenarios. From these, actions and predicates can generally easily be identified. Of course, this description of the ontology is also an iterative process, and it is generally refined during the communication ontology description (the second activity of the Agent Society phase). Concepts, predicates, and actions of this ontology will also be used to describe information exchanged during communications among roles. Tasks of this activity are detailed in Table 12 and their flow is represented in Fig. 25.

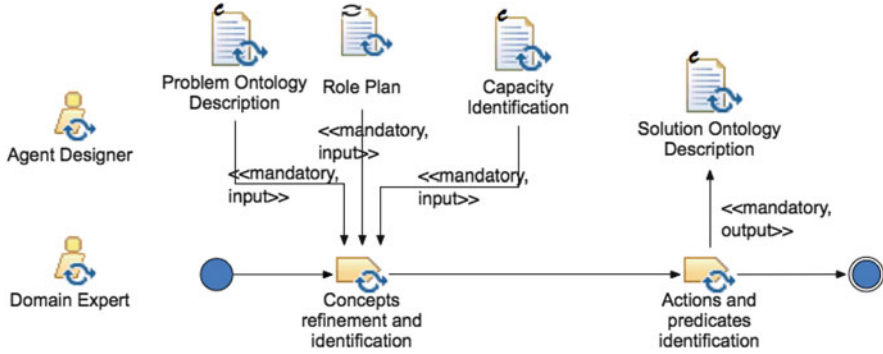


Fig. 25 The flow of tasks of the Solution Ontology Description (SOD)

Communication Ontological Description

This activity aims at describing communications among roles in terms of ontology, interaction protocol, and content language.

A communication is an interaction between two or more roles where the content (language, ontology, and encoding) and the control of the communication (protocol) are explicitly detailed. A communication mainly consists of FIPA speech acts and protocols. The model of role communication we adopt is based on the assumption that two roles during their interaction have to share a common ontology. This common knowledge is represented in the communication by a set of Ontology elements. A communication is an interaction composed of several messages ordered by a Protocol. Each set of interactions between two roles has to be clustered in one or more communications. At this stage, we could regard the previous studied interactions as messages of the communication. Interactions linking a boundary role to another boundary one are generally refined in communications (usually these interactions are mediated by the environment and therefore). They can be based on events or other stimuli sent from one role to the other. Interactions between classical (i.e., non-boundary) roles are refined in communications with a defined protocol, a referred ontology and an associated content language. As a consequence, in communications, the sequence of messages will be ruled by a proper interaction protocol, and the content will be related to the exchanged information or required exhibition of a capacity through an explicit reference to the adopted ontology. This is in accordance with the FIPA specifications [6] (that we largely adopt), where communications consist of speech acts [14]. FIPA also specifies a relevant number of interaction protocols but a new one, if necessary, can be designed.

This activity should also describe data structures required in each role to store exchanged data. These are based upon the elements of the solution ontology. Tasks of this activity are detailed in Table 13 and their flow is represented in Fig. 26.

Table 13 Task descriptions of the Communication Ontology Description activity

Activity	Task	Task Description	Roles involved
Communication Ontology Description	Interactions identification	Among the existing interactions some are refined into communications	Agent Designer (perform)
Communication Ontology Description	Protocol definitions	For each communication, a protocol and speech acts are defined	Agent Designer (perform), Domain Expert (assist)

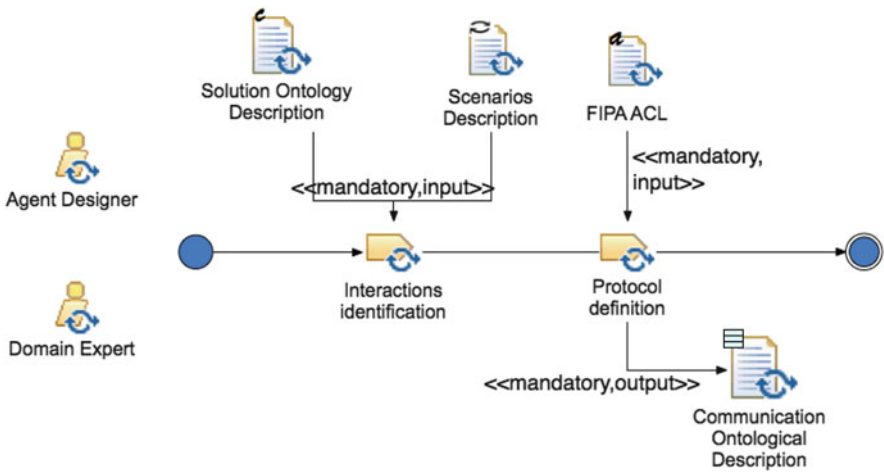


Fig. 26 The flow of tasks of the Communication Ontological Description (COD)

Role Behavior Description

This activity aims at defining the complete life cycle of a role by integrating previously identified RoleTasks, capacities, communications/conversations in which it is involved and the set of (or part of) requirements it has to fulfill. The behavior of the role is described by a set of AgentTasks and AgentActions.

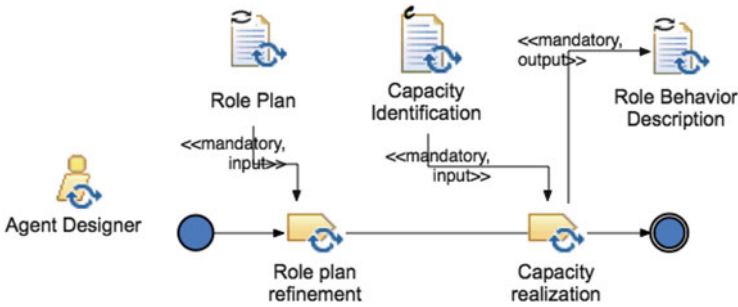
Role Behavior Description is a refinement at the Agent Society level of outputs produced by the Role Plan activity during the System Requirements phase. Each RoleTask is refined and decomposed into a set of atomic behavioral units. The behavior of each role is now described using a statechart or an activity diagram.

If a role requires capacities or provides services, this activity has to describe tasks and actions in which they are really used or provided. The designer describes the dynamical behavior of the role starting from the Role Plan drawn in the previous phase, and the capacities used by the role.

Tasks of this activity are detailed in Table 14 and their flow is represented in Fig. 27.

Table 14 Task descriptions of the Role Behavior Description activity

Activity	Task	Task Description	Roles involved
Role Behavior Description	Role Plan Refinement	Role plans are described in the form of a statechart	Agent Designer (perform)
Role Behavior Description	Capacity Realization	If a role requires capacities or provides services, tasks, and actions in which they are really used or provided have to be described.	Agent Designer (perform)

**Fig. 27** The flow of tasks of the Role Behavior Description (RBD) activity

Protocol Description

The aim of this activity is to define purpose-specific interaction protocols whose need may arise when the description of communications done during the COD (Communication Ontology Description) and SD (Scenario description) activities does not match any of the existing FIPA protocols. The designer starts from scenarios and the ontological description of communications in order to find the need for a new specific interaction protocol. If this is the case, then he can proceed to the definition of a new protocol that is compliant with the interactions described in scenarios and the semantics of the communication. It is advisable to refer to the FIPA Interaction protocols library³ in order to see if a satisfying protocol already exists and if not, probably an existing one can be the basis for changes that can successfully solve the specific problem. Tasks of this activity are detailed in Table 15 and their flow is represented in Fig. 28.

Organization Dependencies Description

The first aim of this activity is to identify resources manipulated by roles; this often implies identifying new capacities devoted to manipulate these resources. Moreover, since organizations depend on each other through service exchange, services provided by roles (while exploiting their capacities and accessing resources) can be identified in this activity.

³FIPA Interaction Protocols specifications: <http://www.fipa.org/repository/ips.php3>

Table 15 Task descriptions of the Protocol Description activity

Activity	Task	Task Description	Roles involved
Protocol Description	Need for new protocol discovery	The designer starts from scenarios and the ontological description of communications in order to find the need for a new specific interaction protocol.	Agent Designer (perform)
Protocol Description	Role plan refinement	Refine the defined role plan by decomposing activities into atomic behavior units	Agent Designer (perform)

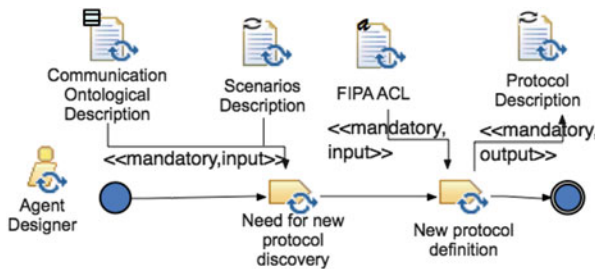


Fig. 28 The flow of tasks of the Protocol Description (PD)

Identification of resources can be done by identifying resources that can be accessed and manipulated by different roles: databases, files, hardware devices, etc. Resources are external to the role; a capacity is generally used to interface the role with the new resource. Each capacity needs a realization in the form of a service that is purposefully defined for that.

When capacities and services are completely described, a verification is necessary to ensure that each capacity is associated with its set of possible service-level realizations. This matching between service and capacity allows the initialization of a repository that may be used to inform agents on how to dynamically obtain a given capacity. Moreover, it is also a verification of the goodness of the system hierarchical decomposition. This matching should validate the contribution that organizations acting at a given level give to upper-level organizations. It will be assumed that an organization can provide a service if the service is provided by one of its roles. A role may provide a service if it owns a capacity with a compatible description, and it manages/supervises a workflow where:

- The service is implemented in one of its own behaviors or
- The service is obtained from a subset of other roles of the same organization by interacting with them using a known protocol; this is a service composition scenario where the work plan is a priori known, and performance may be somehow ensured or
- The service is obtained from the interaction of roles of the same organization, but the work plan is not a priori known and the service results in an emergent way

Table 16 Task descriptions of the Organization Dependencies Description activity

Activity	Task	Task Description	Roles involved
Organizations Dependencies Description	Resource Identification	Identify resources manipulated by roles.	Agent Designer (perform)
Organizations Dependencies Description	Service Identification and Description	Identify and describe services provided by roles and organizations	Agent Designer (perform)
Organizations Dependencies Description	Capacity Realization	Match capacities with services that can realize them and with resources accessed by capacities	Agent Designer (perform)
Organizations Dependencies Description	Find Organization Dependencies	Check dependencies between organizations for possible capacities to be published/realized	Agent Designer (perform)

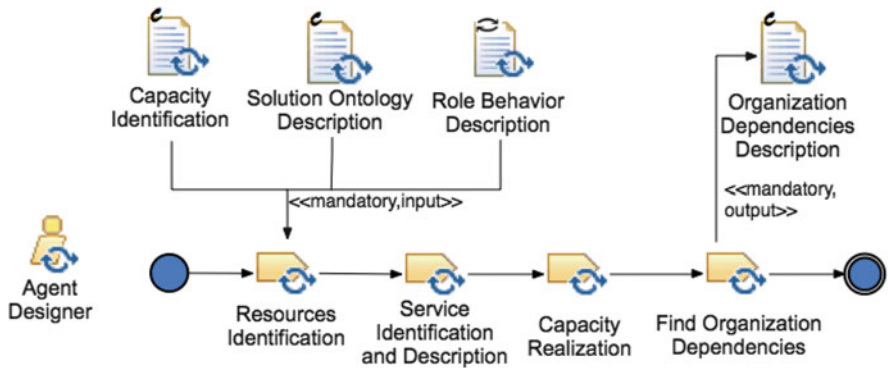


Fig. 29 The flow of tasks of the Organizations Dependencies Description (ODD)

In the previous cases, the management of a service provided by the global organization behavior is generally associated to one of the organization’s roles. It can also be managed by at least one of the representatives of the various holons playing roles defined in the corresponding organization. Service exchange is often a need arising from the beginning of the Requirements Analysis phase. In the proposed approach, some services can be identified by exploiting relationships among use cases, especially those crossing different organizational packages. If two use cases belonging to two different organizations are related by an *include* or *extend* relationship, it is very likely that the corresponding organizations will cooperate in a service exchange. Tasks of this activity are detailed in Table 16 and their flow is represented in Fig. 29.

Agent Identification

The Agent Identification (AI) activity consists in identifying the agents that will compose the lowest level of the system hierarchy and their responsibilities. Respons-

Table 17 Task descriptions of the Agent Identification activity

Activity	Task	Task Description	Roles involved
Agent Identification	Gather responsibilities	Identify the set of lowest level requirements	Agent Designer (perform), System Analyst (assist)
Agent Identification	Analyze Interactions and Roles	Find lowest level roles	Agent Designer (perform), System Analyst (assist)
Agent Identification	Solution Ontology analysis	Analyze the Solution Ontology to identify concepts linked to actions	Agent Designer (perform), System Analyst (assist)
Agent Identification	Identifying agents	Combine the results of previous tasks to identify agents	Agent Designer (perform), System Analyst (assist)

ibilities are modeled using the notion of individual goals, and they will be the basis to determine the agent architecture in the next activity. The Domain Requirements Description, Interactions, and Role Identification and Solution Ontology Description Document are the main inputs for this activity. Results of this activity may be reported by adopting TROPOS[2] goal and actor diagram. Other options may be preferred like, for instance, the system and agent overview diagrams proposed in PROMETHEUS[13].

The work flow of this activity begins with the identification of Agents goals; this mainly consists in gathering organization responsibilities located at the lowest level of the system organizational hierarchy. These responsibilities are expressed in terms of requirements described by using a combination between a use-case driven and a goal-oriented approach. A second task exploits the Interactions and Role Identification activity results. Indeed, Agents are conceived to play lowest-level roles; their personal goals should at least correspond to the set of the goals pursued by these roles. In order to be able to play these roles, agents have also to provide an implementation for the capacities required by these roles. This aspect will be studied in the next activity. A third task, consists in the analysis of the Solution Ontology in order to identify concepts that represent system individuals (concepts linked to ontology actions). The results of these three tasks are effectively considered as useful guidelines to identify agent responsibilities since an individual acts according to personal motivations and goals. Tasks of this activity are detailed in Table 17 and their flow is represented in Fig. 30.

Agent Architecture Description

The AAD activity defines the architecture to be adopted by agents. Such agent architecture should at least define the set of roles that the agent should play and the minimal set of services implementing the capacities required by these roles. The association between Agents and Agent Roles allows the identification of the set of capacities that are required by Agent Roles in order to be played by Agents. In this activity, a UML class diagram is used to describe agents and their capacities realizations in terms of attributes and methods.

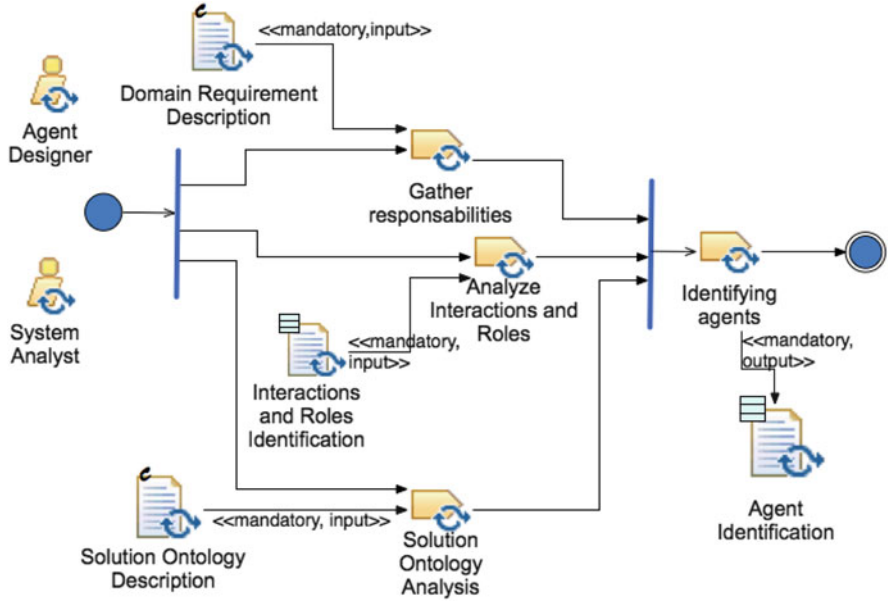


Fig. 30 The flow of tasks of the Agent Identification (AI) activity

Table 18 Task descriptions of the Agent Architecture Description activity

Activity	Task	Task Description	Roles involved
Agent Architecture Description	Capacities Collection	Capacities needed by the agent to play roles are related to the agent.	Agent Designer (perform), System Analyst (assist)
Agent Architecture Description	Agent Architecture Description	Other agents' architectural details (for instance knowledge) are added to the architecture definition.	Agent Designer (perform), System Analyst (assist)

Based on the previous activity, Agent Identification, this activity is composed of two tasks: Capacities collection and Architecture completion. The first task consists in collecting the capacities needed by the roles played by the agent, the second in adding all the necessary agent architectural elements (for instance knowledge slots necessary to manage the concepts related to role playing and capacities enacting). Tasks of this activity are detailed in Table 18 and their flow is represented in Fig. 31.

Role Constraints Identification

This activity aims at identifying constraints among roles. The objective consists in characterizing roles that have to be played simultaneously, priorities or mutual

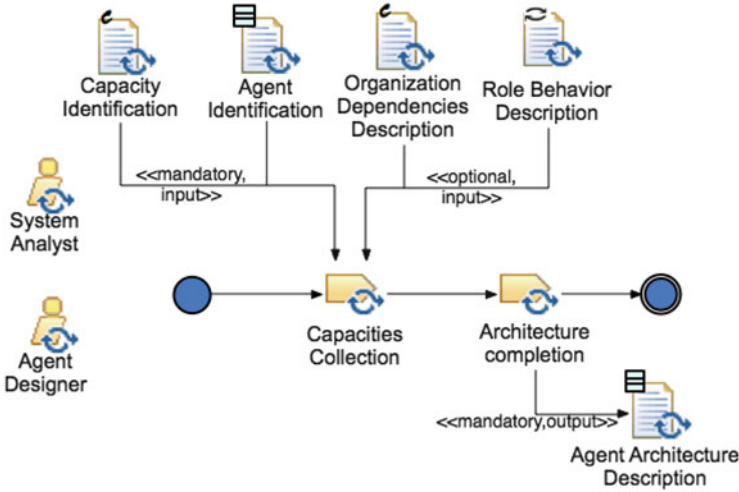


Fig. 31 The flow of tasks of the Agent Architecture Description (AAD) activity

Table 19 Task descriptions of the Role Constraints Identification activity

Activity	Task	Task Description	Roles involved
Role Constraints Identification	Dependencies Identification	Identification of possible conflicts, dependencies, and the constraints that relate roles	Agent Designer (perform), System Analyst (assist)
Role Constraints Identification	Constraints Definition	Formalization of constraints in the output diagram	Agent Designer (perform), System Analyst (assist)

exclusion between roles, preconditions for one role generated by another one, etc. Concurrency constraints are also important because they will drive the definition of role scheduling policies. Constraints between roles that prevent their concurrent execution and force a precise execution sequence have to be defined as well. Roles can be played simultaneously if and only if they allow an exchange of information between two different organizations. A means for realizing this exchange may consist in using the agent internal context when both roles belong to the same agent. This constitutes an alternative to the use of services and a simplification of information transfer.

The first task of the work consists in identifying constraints between roles thanks to roles’ dependencies and associated knowledge as described in the Organization Dependency Description and Role Behavior Description activities. This allows the definition of precise constraints between roles in the second activity’s task. Results of this activity are added to the ODD diagram, thus refining that work product. Tasks of this activity are detailed in Table 19 and their flow is represented in Fig. 32.

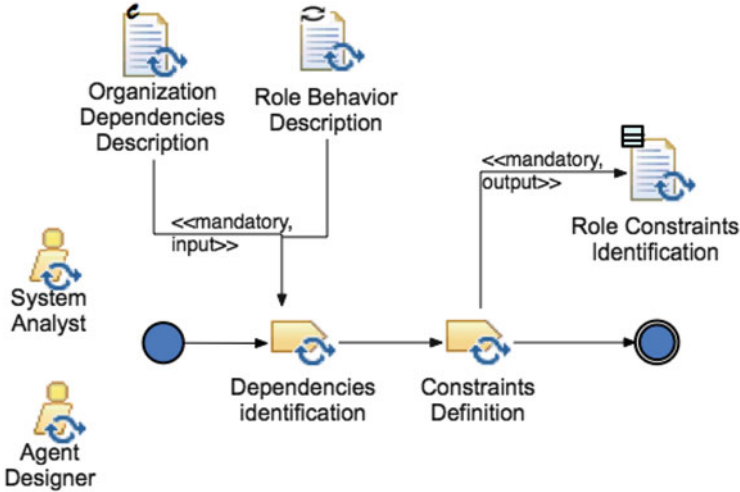


Fig. 32 The flow of tasks of the Role Constraints Identification (RCI) activity

Table 20 Task descriptions of the Agent Plan Description activity

Activity	Task	Task Description	Roles involved
Agent Plan Description	Role Assignment	Select, for each agent the roles it plays	Agent Designer (perform)
Agent Plan Description	Role Strategy Selection	Define a role strategy selection for each agent	Agent Designer (perform)

Agent Plan Description

This activity refines the design of the agent internal architecture. Plans of each agent are defined according to its goals, starting from the Agent Architecture Description and RCI work products. In the plan, the designer makes explicit the strategy the agent will use for choosing the roles it will play. In this activity, a statechart or activity diagram is used to describe the plan of each agent.

In this activity, each agent within the system is associated with the set of roles it has to play according to the set of capacities it owns. In a second step, the strategy for selecting the current role-playing relationship is defined. It may consist in a scheduling algorithm or a rule-based or norm-based mechanism. Tasks of this activity are detailed in Table 20 and their flow is represented in Fig. 33.

Holarchy Design

A super-holon is a community of holons that cooperate to achieve a commonly agreed objective. In order to allow a fruitful evolution of the super-holon structure at runtime, certain rules have to be defined.

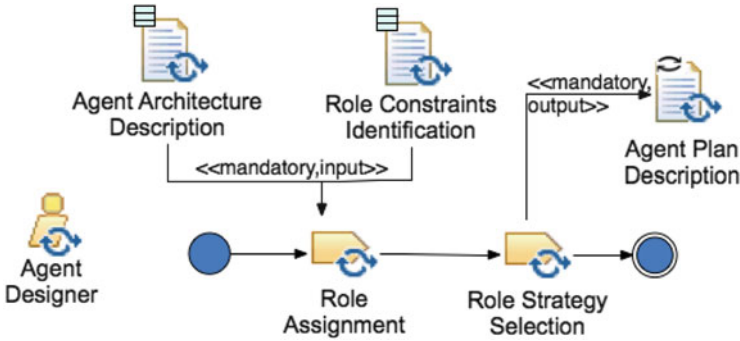


Fig. 33 The flow of tasks of the Agent Plan Description (APD) activity

Trying to enumerate all possible issues would probably result in an incomplete or domain-dependent list. Conversely, we prefer to only discuss the most common and important rules that must be defined in the presented approach. The first rule is about managing *Inclusion/Exclusion of holons members*. Once a super-holon has been created, new members may request to join it or the super-holon itself may require new members to achieve its goal/task (the new member admission process is called *Merging*). Two aspects of that should be analyzed: (1) who is in charge for taking the decisions and how this happens (head, vote, etc.); (2) how the requesting process could be started (who is to be contacted by the external holon who wants to enter the super-holon to play a role within it). The decision process for the admission of a new member can be done accordingly to several different internal policies representing different levels of involvement of the holon members' community: federation is positioned at one side of the spectrum; in this configuration, all members are equal when a decision has to be taken. Opposite to that there is the dictatorship, where heads are omnipotent; a decision taken by one of them does not have to be validated by any other member. In this government form, members loose most of their autonomy having to request permission of the head to provide a service or request a collective action.

Even if these configurations may be useful in specific domains, generally an intermediate configuration will be desirable. In order to identify the right choice, it is, firstly, necessary to analyze the functionalities that will be provided by the holon, and then define the level of authority that the Head will have.

A versatile solution to this problem consists in establishing a voting mechanism for each functionality. In order to parameterize a voting mechanism, three elements have to be defined: *Requester*, *Participants*, and *Adoption Mechanism*.

The vote Requester defines which members are allowed to request for a vote, Participants are members who are authorized to participate in the voting process, and, finally, the adoption mechanism defines how a proposal is accepted or rejected. This scheme can be easily modeled as an organization as shown in Fig. 34.

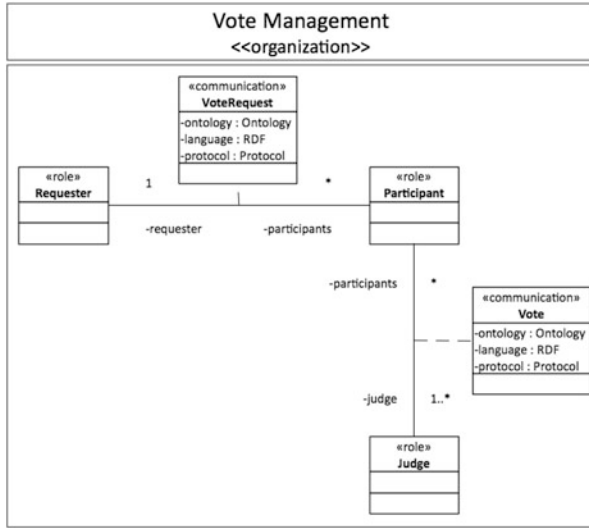


Fig. 34 Voting organization

Table 21 A part of the Holon government description template

Related aspects	Questions to answer
The addition of new functioning rules	<ul style="list-style-type: none"> • Is it possible to add new rules? • By what process can we add a new functioning rule?
Integration/Exclusion of a member	<ul style="list-style-type: none"> • How are the new members integrated? • How to exclude a member?
New tasks acceptance	<ul style="list-style-type: none"> • How are the new tasks accepted or refused?
Choice and Acceptance of new objectives for the super-holon	<ul style="list-style-type: none"> • Is it possible to add new goals to the super-holon? • If yes, how?
Super-holon actions	<ul style="list-style-type: none"> • Who can take decisions about action selection?
Multi-Part authorized	<ul style="list-style-type: none"> • Are the members authorized to join other holons? • If yes, under which conditions?
Holon Destruction: Leaving one of the members	<ul style="list-style-type: none"> • Is it possible that a part leaves the super-holon without implying its destruction? • If yes, under which conditions?

Solution Ontology gives important information for the definition of the holonic structure: rules and constraints are generally described in the ontology. Normally, at this step, these constraints have already been associated to each organization.

Table 21 also provides a set of questions that may guide the design of holon government and the determination of rules that govern holons dynamic. Answer to

Table 22 Task descriptions of the Hierarchy Design activity

Activity	Task	Task Description	Roles involved
Hierarchy Design	Agentification	Identification of the holons for all levels and their composition relationships	Agent Designer (perform), System Analyst (assist)
Hierarchy Design	Holon Government Model Definition	For each holon, definition of a decision making mechanism	Agent Designer (perform), System Analyst (assist)
Hierarchy Design	Hierarchy Definition	Definition of the initial hierarchies properties	Agent Designer (perform), System Analyst (assist)
Hierarchy Design	Holon Self-Organization Mechanisms Definition	Definition of the rules that will describe the dynamics of hierarchies	Agent Designer (perform), System Analyst (assist)

these questions facilitates the definition of the future super-holons functioning: not authorized process, decisions are made by a vote, the heads take the decision, and so on.

This activity aims at refining the solution architecture, and it is composed of four tasks: firstly, the agentification task consists in identifying all the required holons and associating them with the set of roles they can play. This allows the identification of the set of capacities required by each holon, thus giving precise indications on the architecture that should be adopted for it. Indeed, the holon architecture is at least defined by the set of roles that the holon should play, the minimal set of services that implement capacities required by its role. The second task focusses on composed holons and aims at identifying a government type for each of them. The objective consists in describing the various rules used to take decisions inside each super-holon. At this point, all the previously described elements are merged in order to obtain the complete set of holons (composed or not) involved in the solution. In this way, the complete hierarchy of the system at the instance level is described.

The description obtained with the previous tasks is just the initial structure of the system, the last objective is now to specify rules that govern holons' dynamics in the system (creation, new member integration, specific self-organization mechanisms, scheduling policies for roles) in order to support a dynamic evolution of the system hierarchy. Tasks of this activity are detailed in Table 22 and their flow is represented in Fig. 35.

2.2.3 Work Products

The System Requirements phase generates eight work products. Their relationships with the MAS metamodel elements are described in Fig. 36. This diagram represents the Agent Society Model in terms of work products. Each of these latter reports one or more elements from the ASPECS metamodel Agency Domain.

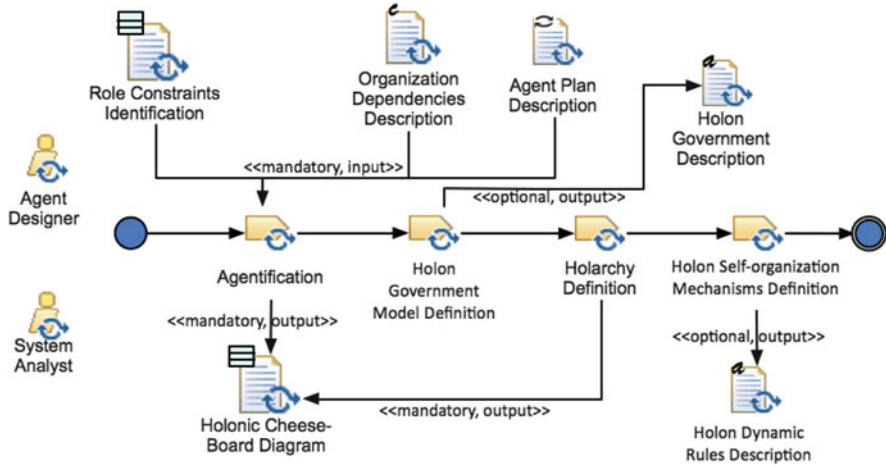


Fig. 35 The flow of tasks of the Holarchy Design (HD)

Work-Product Kinds

Table 23 briefly describes the nature and content of each work product kind. Almost every work-product could be considered as a composite one as each UML diagram is associated to a structured text document detailing the content of the diagram. We consider that this documentation is part of the diagram, the nature of the work product is thus defined with respect to the description given in the diagram.

Due to space concerns we present the three most representative examples of workproducts.

Role Behavior Description

This activity produces a behavioral work product where role behaviors are refined and detailed under the form of states and transitions. Transitions classically specify the dynamic conditions under which roles change their states. The notation is that of the UML state machine diagram for describing a single role behavior. Figure 37 shows an example of notation of the Role Behavior Diagram for the CMS case study. Roles come from the IRI diagram and their general behavior from the PD diagram.

Role Constraints Identification

This activity produces a structural work product that describes roles and the constraints that are to be verified at runtime. The notation used is an UML class diagram where roles are represented as classes and constraints are expressed as stereotyped relationships between roles. Figure 38 shows an example of notation of the RCI diagram for the CMS case study. It is a refinement of the ODD diagram.

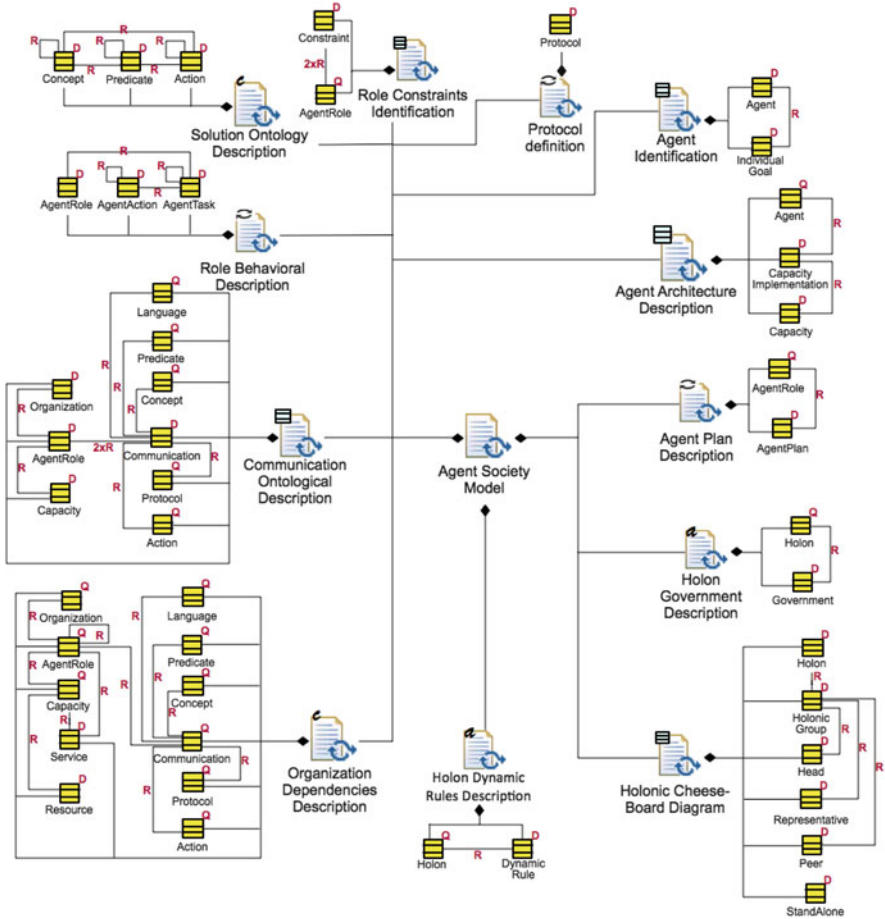


Fig. 36 The Agent Society phase model documents structure

Holarchy Design

The Holarchy Design activity produces a diagram called cheeseboard diagram that represents the different levels of the Holarchy, and, for each level, the different holons and their corresponding groups and roles. A free text description can complement this diagram to explain the adopted design choices. Figure 39 shows an example of notation of the Holarchy Design diagram for the CMS case study. Holons come from AI diagrams, groups and levels come from ODD diagrams.

Table 23 Work-product kinds of the Agent Society phase

Name	Description	Work-Product Kind
Solution Ontology Description (SOD)	A UML Class Diagram with a specific profile and its associated documentation describing concepts, actions, predicates, and their relationships of the agent-based solution ontology	Structural & Structured Text
Agent Identification (AI)	Tropos goal and actor diagram or PROMETEHEUS agent overview diagram and their associated textual documentation to identify agents and their individual goals	Behavioral & Structured Text
Agent Architecture (AA)	A UML Class Diagram with a specific profile and its associated documentation describing attributes and behavioral methods of each agent and their personal capacities	Structural & Structured Text
Communication Ontological Description (COD)	A UML Class Diagram with a specific profile and its associated documentation describing agents' roles and their communications. Interactions previously identified are now clustered in conversation or communications and represented by an association. Attributes of each communication (Ontology) and each conversation (Ontology, Content Language, and Interaction Protocol) are specified in an association class. Each conversation is oriented from the initiator of the conversation to the other participant roles.	Structural & Structured Text
Role Behavior Description (RBD)	UML State machine Diagram describing role's internal behavior in terms of AgentActions, AgentTasks, and AgentRoleStates.	Behavioral & Structured Text
Protocol Description (PD)	A UML Class Diagram with a specific profile (or AUML) and its associated documentation defining purpose-specific interaction protocols when communications' description does not match any of the existing FIPA protocols	Behavioral & Structured Text
Organization Dependencies Description (ODD)	A UML Class Diagram with a specific profile and its associated documentation describing service, resources, capacities, and roles of each organization. Refinement of Capacity Identification diagram	Structural & Structured Text
Role Constraints Identification (RCI)	Refinement of ODD diagram with the additional constraints between roles to be played simultaneously.	Structural & Structured Text
Agent Plan Description (APD)	UML State Machine diagram with specific profile and its associated documentation defining an agent's personal plan according to its individual motivations and pursued goals. The plan represents the strategy used by the	Behavioral & Structured Text

(continued)

Table 23 (continued)

Name	Description	Work-Product Kind
	agent to choose the roles to play. Each agent is associated to the set of roles it has to play according to the set of capacities that it owns.	Behavioral & Structured Text
Holarchy Design (HD), Holonic Cheese-board	Cheese-board diagram and its associated documentation describing the complete structure of the system mapping the hierarchy of organizations to a holarchy. This mapping is based on the association between holons composing the holarchy with the set of roles they play in the various groups (instance of organizations) composing the system.	Structural & Structured Text
Holarchy Design (HD), Holon Government Description	Text document describing the various rules used to take decisions inside each super-holon: the holon decision-making process.	Free text
Holarchy Design (HD), Holon Dynamic Rules Description	Text document specifying holons self-organization mechanisms (creation, new member integration, and scheduling policies for roles) in order to support a dynamic evolution of the system holarchy.	Free text

2.3 Implementation and Deployment

The objectives of the Implementation and Deployment phase are to implement and deploy an agent-oriented solution resulting from the Agent Society Design phase. Due to space restrictions this phase will not be fully described according to the FIPA Standard but only synthesized in this section. The interested reader could find information concerning this phase in [5] and on the ASPECS website.⁴

The complete sequence of activities is described in Fig. 40.

2.3.1 Process Roles

The roles involved in the activities and tasks of this phase are two: Developer and Tester.

2.3.2 Activity Details

Some details about the activities of this phase are presented in Table 24. In the following the different objectives for each activity are introduced.

The Holon Architecture Definition activity should define the architecture of each agent/holon identified within the Agent Society Design phase. For each organization, a description detailed enough to allow implementation should be provided. The

⁴<http://www.aspecs.org>

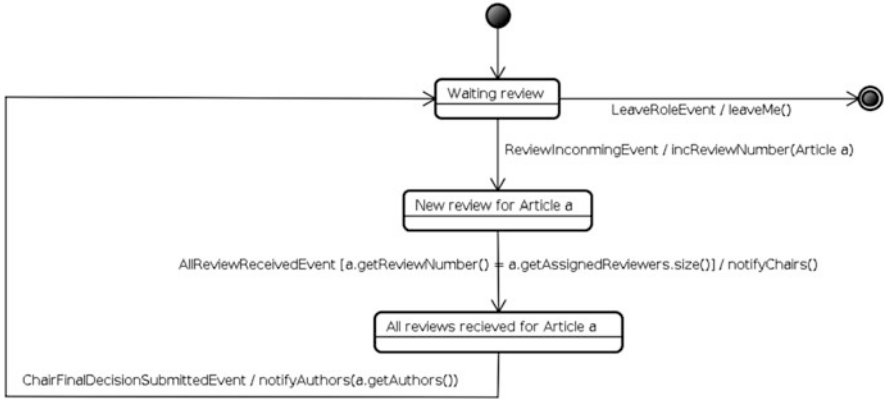


Fig. 37 Role Behavior Description diagram—a portion of the CMS case study

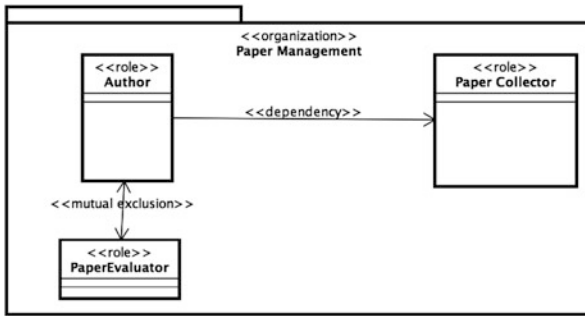


Fig. 38 Role Constraints Identification diagram—a portion of the CMS case study

role playing relationship of each agent/holon and the sets of capacities/services are completely defined.

The Code Reuse activity aims at integrating the set of organizational patterns identified during the OID (Organization Identification) and IRI (Interactions and Roles Identification) activities and for the Holon Government identification task. The second aim is to allow the reuse of code from other existing applications.

The Code Production for Organizations and Roles activity should produce code for each organization and role. The ideal case is when the chosen implementation platform natively supports these concepts as it happens for Janus [8].

The Organizations and Roles Units Tests activity is the first level of test in ASPECS. The principle is to test at organization and role levels for each specific context.

The Code Production for Holons activity focusses on code production for each Holon. The principle is to implement, depending on the platform primitives, the results of the Holarchy Design activity from the previous phase.

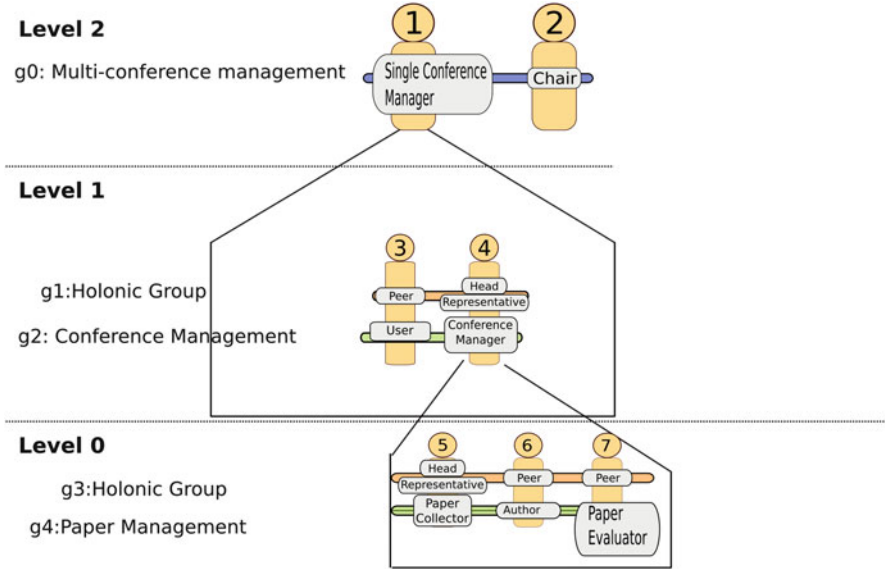


Fig. 39 Hierarchy Design diagram—a portion of the CMS case study

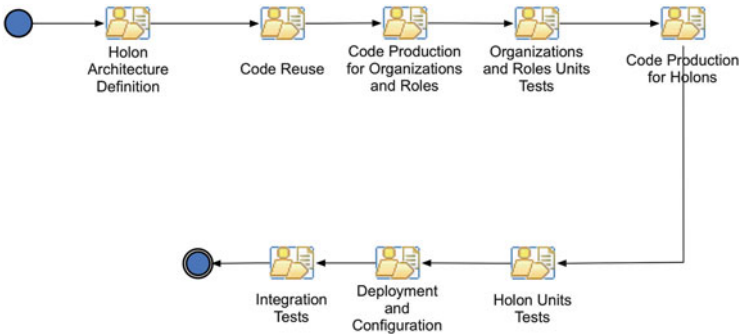


Fig. 40 The Implementation and Deployment phase flow of activities

The Holon Unit Test activity is the second level of test in the ASPECS process. The focus is on holon’s behavior validation. Each holon is thus individually tested.

The Deployment and Configuration activity details the concrete deployment of the application. The elements to be detailed are, for example, a network configuration, holon(s) physical location, external devices, external applications, etc.

Integration Tests is the third and final test activity of the ASPECS process. The testing scope is on system functionalities and on the interfaces between system parts.

Table 24 Activity details for the Implementation and Deployment phase

Activity name	Task	Task description	Roles involved
Holon Architecture Definition	Static architecture definition	Each architecture designed in the Hierarchy Design is detailed	Developer (perform)
	Dynamic architecture definition	A dynamic architecture is defined to take into account the Hierarchy Design	Developer (perform)
Code Reuse	Organizations Pattern Integration	OID model is used to identify organizational patterns	Developer (perform)
	Role-Interactions Pattern Integration	IRI model is used to identify role and interactions patterns	Developer (perform)
	Holonic Pattern Integration	Hierarchy Design is used to identify holonic patterns	Developer (perform)
	Pattern Integration	Identified patterns are merged and adapted	Developer (perform)
Code Production for Organizations and Roles	Organization, Role code production	IRI model and Role plan models are detailed with platform specific primitives	Developer (perform)
Organizations and Roles Units Tests	Organization test	SD model are used to define organizations tests	Tester (perform), Developer (assist)
	Role test	RP models are used to define roles tests	Tester (perform), Developer (assist)
Code Production for Holons	Holons CodeProduction	The HD model is used to define tests dealing with rules about holon government, task management, and new members entrance	Developer (perform)
Holon Unit Test	Unit Test Definition	The HD model is used to define the most adapted implementation and platform specific primitives	Tester (perform), Developer (assist)
Deployment and Configuration	Holon Partitioning	The HD model is used to establish a partition between the various holons used to develop the application	Developer (perform)
	Dynamic Configuration Rules	The configuration specification is used to define rules for kernel distribution/integration	Developer (perform)
Integration Tests	Interfaces Tests	The HD model is used to define interface tests between sub-systems	Tester (perform), Developer (assist)
	Functionality Tests	The DRD model is used to define functionality tests	Tester (perform), Developer (assist)

Table 25 Work-product kinds of the Implementation and Deployment phase

Name	Description	Work-Product Kind
Holarchy Architecture Definition (HAD)	A UML Class Diagram with a specific profile	Structural & Structured Text
Code Reuse (AI)	A document describing the reused code	Structured
Code Production for Organizations and Roles (CPOR)	A document describing code for organizations and roles	Structured
Organizations and Roles Unit Tests (ORUT)	A document describing tests for organizations and roles.	Structured
Code Production for Holons (CPH)	A document describing code for holons.	Structured
Holons Unit Tests (HUT)	A document describing tests for holons.	Structured
Deployment and Configuration	A UML Deployment Diagram with a specific profile representing where the agents are located, the resources and communication channels.	Structural
Integration Test (IT)	A document describing integration tests.	Structured

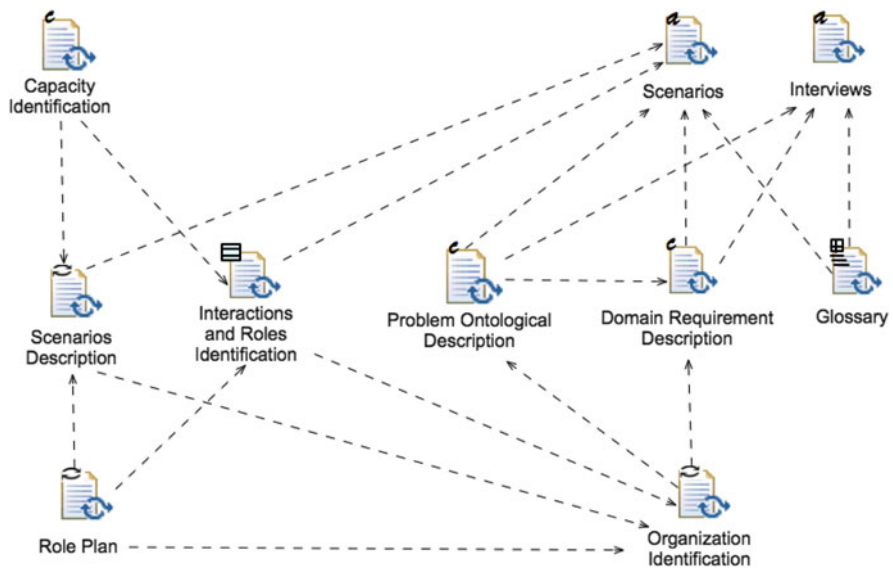


Fig. 41 The work-product dependency diagram of the System Requirements phase

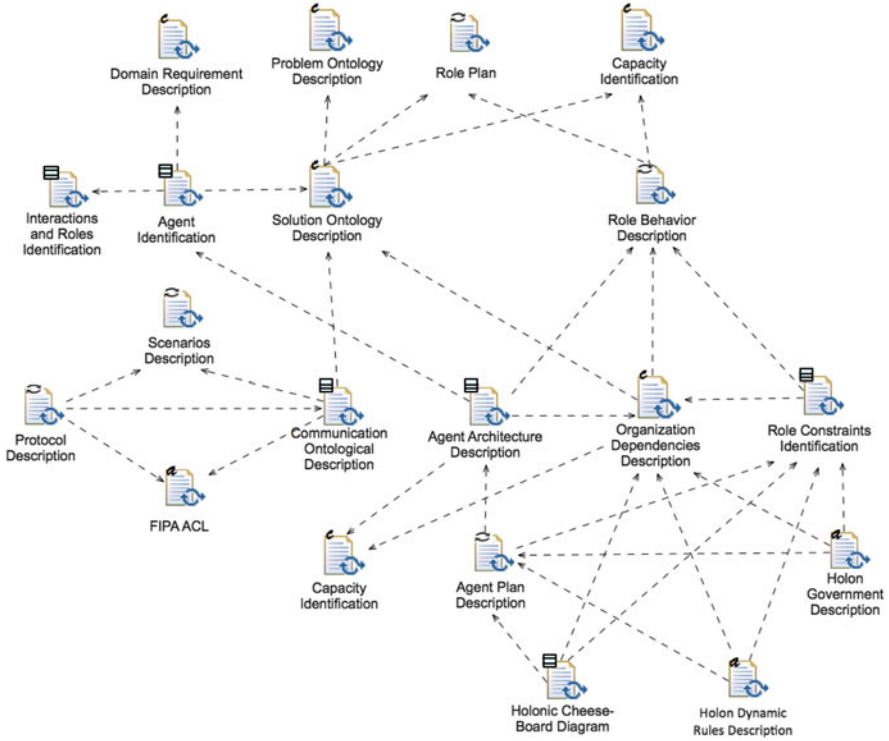


Fig. 42 The work-product dependency diagram of the Agent Society phase

2.3.3 Work Products

The Implementation and Deployment phase generates eight work products which are listed in Table 25.

Work-Product Kinds

Work-product kinds are briefly described in Table 25.

3 Work-Product Dependencies

Figures 41 and 42 describe the dependencies among the work products of the first two ASPECS phases.

References

1. Bernon, C., Cossentino, M., Pavón, J.: An overview of current trends in european aose research. *Informatica* **29**(4), 379–390 (2005)
2. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: an agent-oriented software development methodology. *J. Auton. Agent Multi Agent Syst.* **8**(3), 203–236 (2004)
3. Cossentino, M., Gaud, N., Galland, S., Hilaire, V., Koukam, A.: Holomas'07, Regensburg, Germany, pp. 237–246 (2007)
4. Cossentino, M., Galland, S., Gaud, N., Hilaire, V., Koukam, A.: How to control emergence of behaviours in a holarchy. In: *Self-adaptation for Robustness and Cooperation in Holonic Multi-agent Systems (SARC)*, Workshop of the Second IEEE International Conference on Self-adaptive and Self-organizing Systems (SASO). Isola di San Servolo, Venice, Italy (2008)
5. Cossentino, M., Gaud, N., Hilaire, V., Galland, S., Koukam, A.: Aspecs: an agent-oriented software process for engineering complex systems. *Auton. Agent Multi Agent Syst.* **20**(2), 260–304 (2010). doi:10.1007/s10458-009-9099-4
6. Foundation for Intelligent Physical Agents: FIPA ACL Message Structure Specification (2002). Standard, SC00061G
7. Foundation For Intelligent Physical Agents: FIPA Communicative Act Library Specification (2002). Standard, SC00037J
8. Gaud, N., Galland, S., Hilaire, V., Koukam, A.: An organisational platform for holonic and multiagent systems. In: *PROMAS-6@AAMAS'08*, Estoril, Portugal (2008)
9. Gerber, C., Siekmann, J., Vierke, G.: Holonic multi-agent systems. Tech. Rep. DFKI-RR-99-03, DFKI - GmbH (1999)
10. Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum. Comput. Stud.* **43**(5–6), 907–928 (1995)
11. Object Management Group: MDA Guide, v1.0.1, OMG/2003-06-01 (2003)
12. Odell, J., Nodine, M., Levy, R.: A metamodel for agents, roles, and groups. In: Odell, J., Giorgini, P., Müller, J. (eds.) *Agent-Oriented Software Engineering*. Lecture Notes in Computer Science. Springer, Berlin (2005)
13. Padgham, L., Winikoff, M.: Prometheus: a methodology for developing intelligent agents. In: *Agent-Oriented Software Engineering III*. Lecture Notes in Computer Science, vol. 2585, pp 174–185. Springer, Berlin (2003)
14. Searle, J.: *Speech Acts*. Cambridge University Press, Cambridge (1969)
15. Simon, H.A.: *The Science of Artificial*, 3rd edn. MIT, Cambridge (1996)
16. Wilber, K.: *Sex, Ecology, Spirituality*. Shambhala, Boston (1995). <http://207.44.196.94/~wilber/20tenets.html>

ELDAMeth Design Process

Giancarlo Fortino, Francesco Rango, and Wilma Russo

Abstract

In this paper the design process documentation template defined in the context of the IEEE FIPA DPDF Working Group (FIPA Design Process Documentation Template, <http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/docs.htm>, accessed June 2012) is exploited to describe the ELDAMeth agent-oriented methodology. ELDAMeth, which is based on the ELDA agent model and related frameworks and tools, fully supports the development of distributed agent systems and has been applied both stand-alone and in conjunction with other agent-oriented methodologies to different application domains: e-Commerce, information retrieval, conference management systems, content delivery networks, and wireless sensor networks.

1 Introduction

ELDAMeth [7] is an agent-oriented methodology specifically designed for the simulation-based prototyping of distributed agent systems (DAS). It is centered on the ELDA (Event-driven Lightweight Distilled StateCharts Agent) model [9, 12] and on an iterative development process covering DAS Modeling, simulation, and implementation for a target agent platform (currently JADE [2]) and exploits specifically defined frameworks and CASE tools. In particular, the ELDA model is based on three main concepts which are important for enabling dynamic and distributed computation [15, 16]: (1) lightweight agent architecture and agent behaviors driven by events that trigger reactive and proactive computation; (2) agent interaction and cooperation based on multiple coordination spaces that are exploited by

G. Fortino (✉) • F. Rango • W. Russo

Department of Informatics, Modeling, Electronics and Systems (DIMES), University of Calabria,
Via P. Bucci, Cubo 41C, 87036 Rende (CS), Italy

e-mail: g.fortino@unical.it; frango@deis.unical.it; w.russo@unical.it

the agents at run-time; (3) coarse-grained strong mobility through which agents can migrate across agent locations by transparently retaining their execution state [6].

Moreover, ELDAMeth can be used either stand-alone, according to the ELDAMeth process reported in Fig. 1, or in conjunction/integration with other agent-oriented methodologies which support the analysis and (high-level) design phases.

In particular, ELDAMeth has been integrated with Gaia [11], PASSI [4], and MCP [9] by using a process-driven method engineering approach [3]. Moreover, ELDAMeth (or previously defined models and frameworks that are now in ELDAMeth) was applied in different application domains: e-Commerce [4, 11], distributed information retrieval [7–9, 12, 14], content distribution networks [10], distributed data mining [5], and wireless sensor networks [1].

Useful references for ELDAMeth:

Fortino, G., Russo, W.: ELDAMeth: a methodology for simulation-based prototyping of DAS. *Inform. Softw. Technol.* **54**, 608–624 (2012)

Fortino, G., Garro, A., Mascillaro, S., Russo, W.: Using event-driven light-weight DSC-based agents for MAS modeling. *Int. J. Agent Orient. Softw. Eng.* **4**(2) (2010)

Fortino, G., Rango, F., Russo, W.: Engineering multi-agent systems through statecharts-based JADE agents and tools. *Trans. Comput. Collect. Intell. LNCS 7270 VII*, 61–81 (2012)

Fortino, G., Russo, W., Zimeo, E.: A statecharts-based software development process for mobile agents. *Inform. Softw. Technol.* **46**(13), 907–921 (2004)

Useful references for ELDAMeth integrations and extensions:

Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: PASSIM: a simulation-based process for the development of multi-agent systems. *Int. J. Agent Orient. Softw. Eng.* **2**(2), 132–170 (2008)

Fortino, G., Garro, A., Mascillaro, S., Russo, W.: A multi-coordination based process for the design of mobile agent interactions. In: *Proceedings of IEEE Symposium on Intelligent Agents* (2009)

Fortino, G., Garro, A., Russo, W.: An integrated approach for the development and validation of multi agent systems. *Comput. Syst. Sci. Eng.* **20**(4), 94–107 (2005)

1.1 The ELDAMeth Process Lifecycle

ELDAMeth is based on the three phases of the iterative process model shown in Fig. 2:

- The *Modeling* phase produces a specification of a Multi-Agent System (ELDA MAS) fully compliant with the ELDA MAS Meta-model [9] (see Sect. 1.2). Moreover, the platform-independent code of the ELDA MAS is generated in this phase.
- The *Simulation* phase produces MAS execution traces and computes performance indices that are evaluated with respect to the functional and non-functional requirements of the MAS under-development. On the basis of such evaluation,

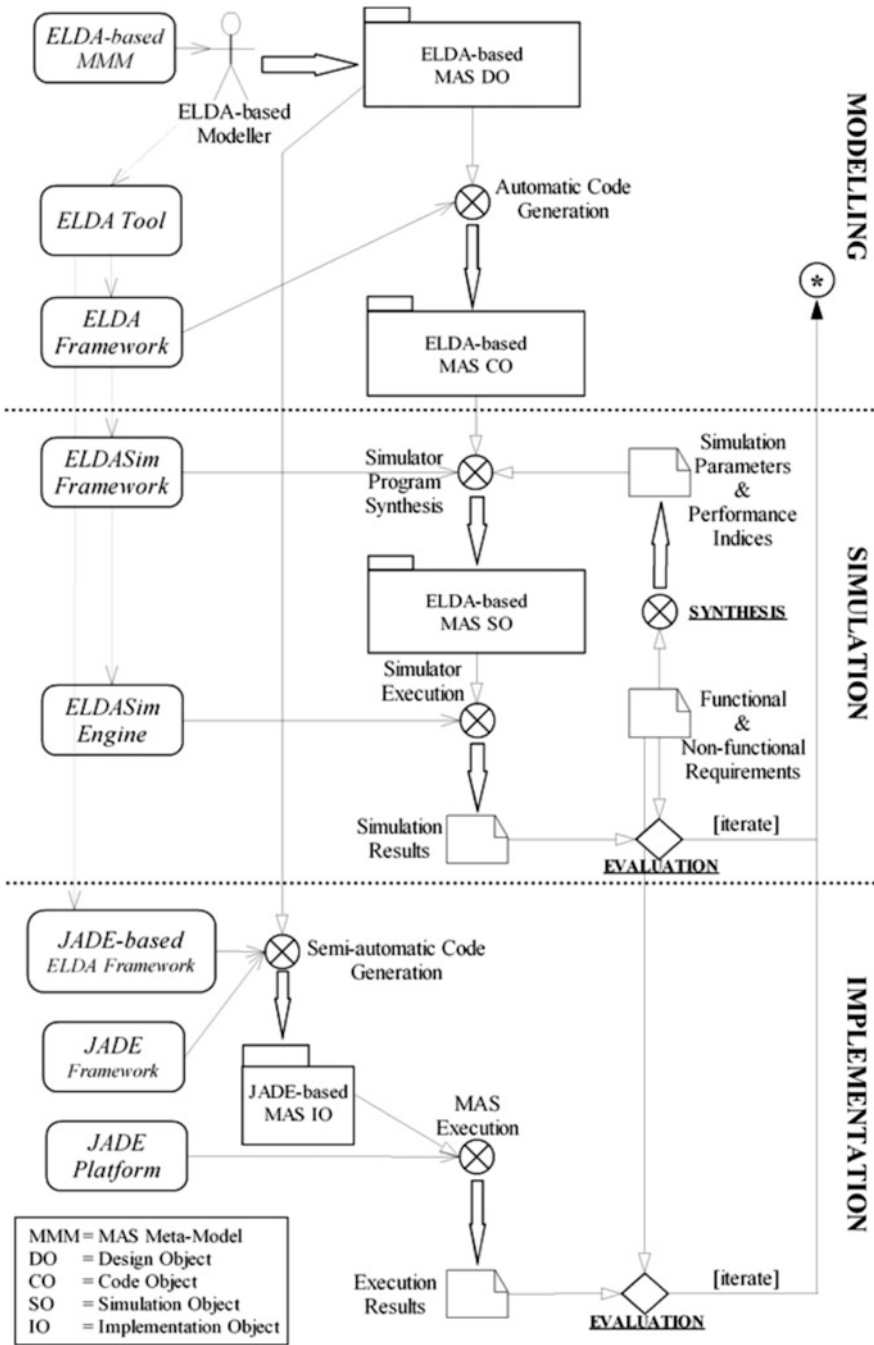


Fig. 1 The traditional ELDAMeth process

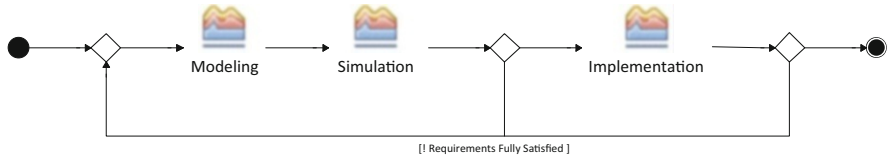


Fig. 2 The ELDAMeth process phases (and iterations)

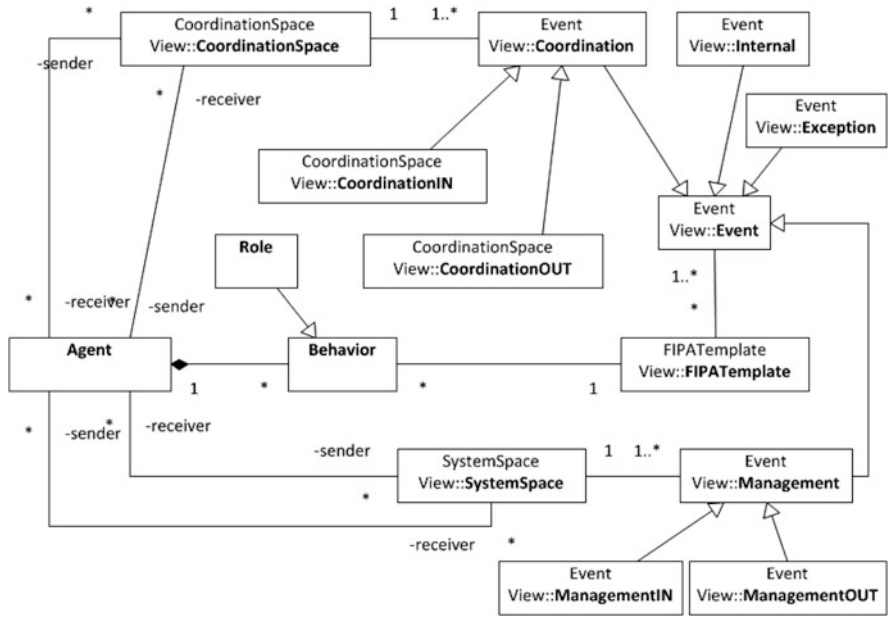


Fig. 3 The ELDA MAS Meta-model

if requirements are satisfied, the *Implementation* phase is carried out; otherwise, the *Modeling* phase is iterated.

- The *Implementation* phase produces the ELDA-based MAS code targeting a specific platform. Currently the JADE platform is exploited [13].

1.2 The ELDA MAS Meta-Model

The MAS Meta-model [9] adopted by ELDAMeth is represented in Fig. 3. The definitions of the MAS Meta-model Elements (MMMElements) are reported in Table 1.

Table 1 Definitions of ELDA MAS Meta-model elements

Concept	Definition
Agent	An ELDA agent with multiple behaviors
Role	A role represented by an agent
Behavior	An ELDA agent's behavior is specified through a Distilled StateChart (DSC) [14], which is a hierarchical state machine obtained from Statecharts and based on ECA rules, OR-decomposition, history entrance mechanisms, and UML-like execution semantics based on the run-to-completion step
FIPATemplate	An ELDA behavior is compliant to an extended version of the FIPA agent lifecycle template that allows restoring the agent execution state after agent migration or agent suspension
Event	The interactions of ELDA agents are based on events: <ul style="list-style-type: none"> – Internal (i.e., self-triggering events) – Management, coordination, and exception (i.e., requests to or notifications from the local agent server). Events can be either OUT-events (generated by the agent and always targeting the local agent server) or IN-events (generated by the local agent server and delivered to target agents)
SystemSpace	SystemSpace provides extensible system services through management (ManagementOUT and ManagementIN) events which allow for agent lifecycle management, timer setting, and resource access
CoordinationSpace	CoordinationSpace provides extensible coordination services through Coordination (CoordinationOUT and CoordinationIN) events which enable coordination acts between agents and between agents and non-agent components (e.g. remote objects, web services) according to specific coordination models. The currently defined inter-agent coordination models are: Direct (synchronous and asynchronous), Tuple-based, and publish/subscribe event-based. The interactions between agent/non-agent components can be based on a general RMI object model or on the Web Services model

2 Phases of the ELDAMeth Process

2.1 The Modeling Phase

The goal of the *Modeling* phase is to provide a detailed design of the MAS under-development in terms of a set of interconnected DSCs [14] representing agent behaviors and/or roles. Figure 4 presents the flow of activities of the Modeling phase. In particular, the two main activities are ELDA Modeling and ELDA Coding. Figure 5 shows the *Modeling* described in terms of activities, roles, and work products. The *Modeling* involves a process role and five work products. The *Modeling* is fully supported by ELDATool, a CASE tool specifically developed to automate modeling, validation and implementation of ELDA-based MAS.



Fig. 4 The *Modeling* flow of activities

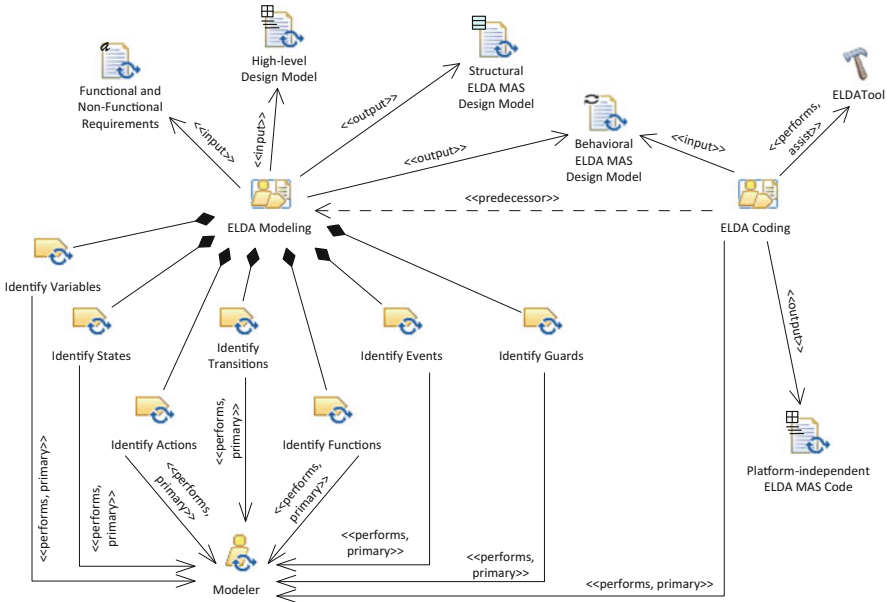


Fig. 5 The *Modeling* phase described in terms of activities, roles, and work products

2.1.1 Process Roles

One role is involved in the Modeling: the Modeler.

Modeler

The Modeler produces a detailed design of the MAS under-development and generates a platform-independent code through the following activities:

- ELDA Modeling: this activity allows the design of the MAS under-development, specifying agent behaviors and/or roles.
- ELDA Coding: the objective of this activity is to generate a platform-independent code for the MAS under-development through ELDATool.

2.1.2 Activity Details

The ELDA Coding activity is an atomic activity that has no tasks and is usually carried out by the Modeler with support of the ELDATool that is able to automatically translate the models produced in the ELDA Modeling activity into

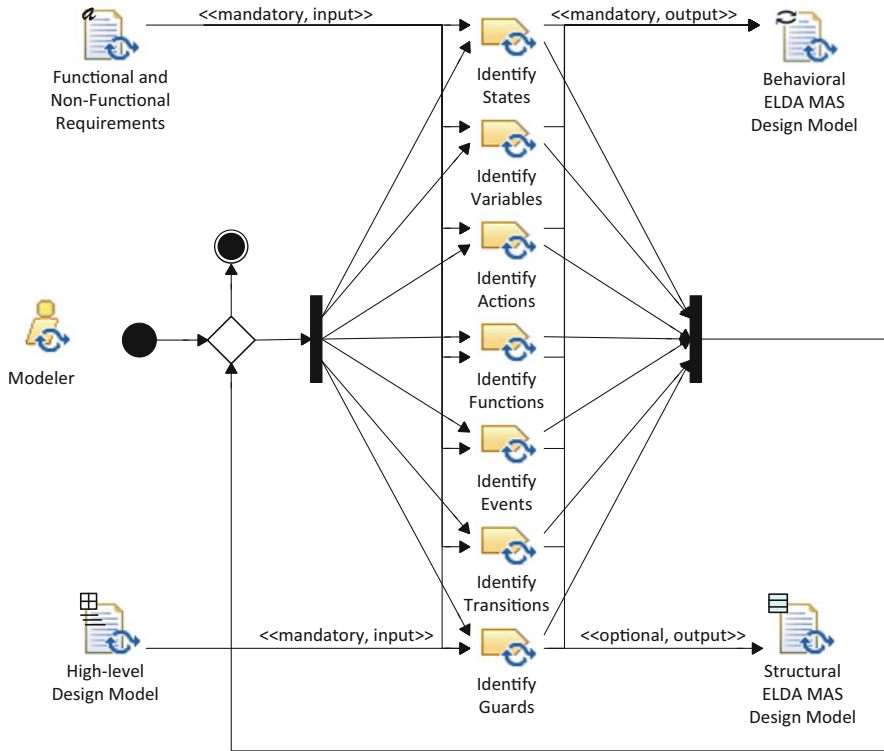


Fig. 6 The flow of tasks of the ELDA Modeling activity

platform-independent code according to the ELDA Framework [9]. Conversely, the ELDA Modeling activity has seven tasks described in the following.

ELDA Modeling Activity

The ELDA Modeling activity is a fundamental activity carried out by the Modeler that produces the behavioral and structural ELDA MAS design based on defined functional and non-functional requirements and on a high-level design model, both deriving from an external system analysis phase not included in ELDAMeth. This activity is composed of seven tasks as shown in Fig. 6; their description is reported in Table 2.

Tasks can be carried out in parallel and iteratively. The mandatory inputs to the *ELDA Modeling* are the *Functional and Non-Functional Requirements* document and the *High-Level Design Model*. The outputs are the (mandatory) *Behavioral ELDA MAS Design Model* and the (optional) *Structural ELDA MAS Design Model*. The former is a set of DSCs, representing agent behaviors and/or roles, whereas the latter is a class diagram representing the interaction relationships among agents and/or roles.

Table 2 Tasks of ELDA Modeling activity

Activity	Task	Task description	Role involved
ELDA Modeling	Identify states	Identification of DSC states	Modeler (perform)
ELDA Modeling	Identify variables	Identification of DSC variables	Modeler (perform)
ELDA Modeling	Identify actions	Identification of DSC actions	Modeler (perform)
ELDA Modeling	Identify functions	Identification of DSC functions	Modeler (perform)
ELDA Modeling	Identify events	Identification of DSC events	Modeler (perform)
ELDA Modeling	Identify transitions	Identification of DSC transitions	Modeler (perform)
ELDA Modeling	Identify guards	Identification of DSC guards	Modeler (perform)

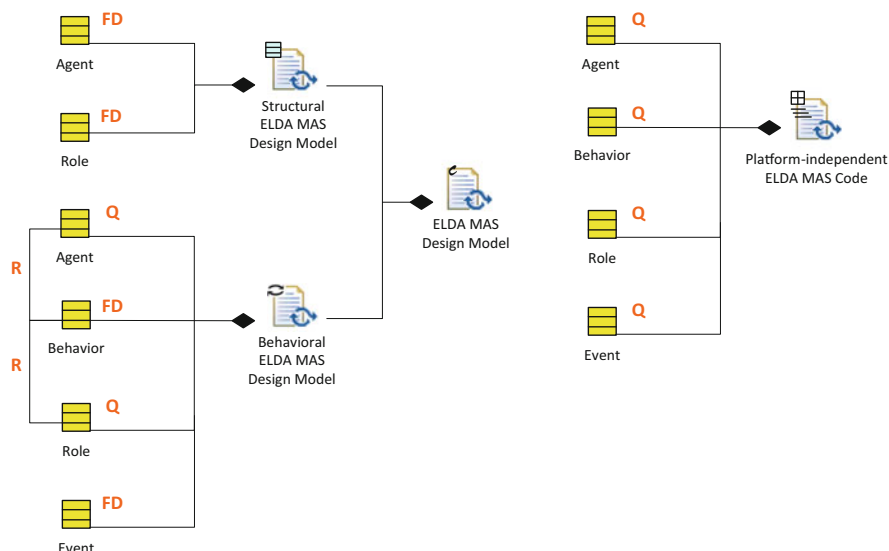


Fig. 7 The *Modeling* work products

2.1.3 Work Products

Figure 7 reports the relationships among the work products of this step (*Modeling* phase) and the ELDA MMMElements (see Sect. 1.2).

Work Product Kinds

Table 3 describes the work products of the *Modeling*.

In the following the *Structural ELDA MAS Design Model*, the *Behavioral ELDA MAS Design Model*, and the *Platform-independent ELDA MAS Code* (specifically the *Reviewer* role code) produced for the CMS case study will be described.

Table 3 Modeling work products kinds

Name	Description	Work product kind
Functional and non-functional requirements	A document defining functional and non-functional requirements of the MAS under-development	Free text
High-level design model	A high-level design model produced by an external method/methodology	Structured
ELDA MAS design model	The detailed design of the MAS under-development	Composite
Structural ELDA MAS design model	The class diagram of the MAS under-development	Structural
Behavioral ELDA MAS design model	The DSC design of the MAS under-development	Behavioral
Platform-independent ELDA MAS code	The platform-independent code generated for the MAS under-development	Structured

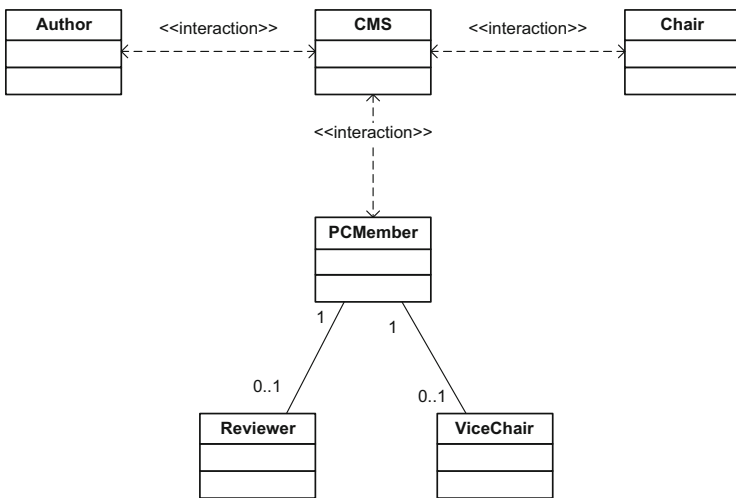


Fig. 8 Class diagram of agents and roles interactions in the CMS case study

Structural ELDA MAS Design Model

In Fig. 8 the *Structural ELDA MAS Design Model* of the CMS case study is portrayed. In particular, five roles are identified: Author, Chair, and PCMember, where a PCMember could be either a Reviewer, or a Vice-Chair, or both. Moreover, CMS is an agent representing the CMS system.

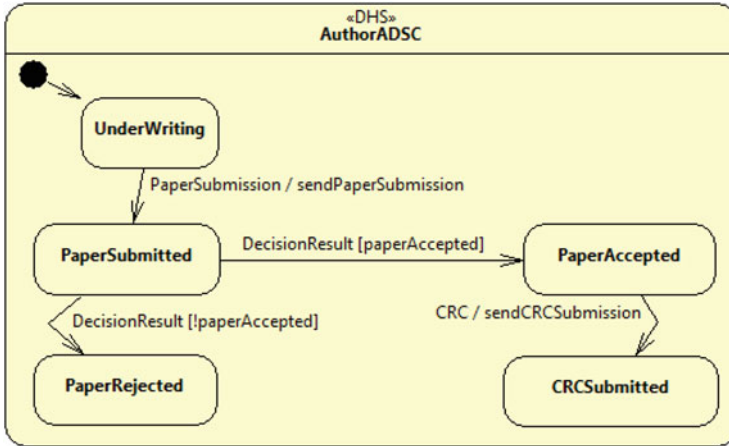


Fig. 9 Author DSC diagram

Table 4 Author actions

Action	Description
sendPaperSubmission	Author submits the paper to the CMS system
sendCRCSubmission	Author submits the CRC to the CMS system

Table 5 Author guards

Guard	Description
paperAccepted	Author checks if the submitted paper has been accepted

Behavioral ELDA MAS Design Model

The *Behavioral ELDA MAS Design Model* of the CMS case study is composed of the DSCs of the five defined roles (Author, Chair, PCMember, Reviewer, and Vice-Chair) and the CMS agent. In the following they are detailed in terms of DSC diagram and event, action and guard tables. The PCMember specification is based on the specifications of Reviewer and Vice-Chair, so a further specification for the PCMember was not defined.

Author

See Fig. 9 and Tables 4, 5 and 6.

Reviewer

See Fig. 10 and Tables 7, 8 and 9.

Table 6 Author events

Event	Sender	Description
PaperSubmission	Author	Internal event sent when Author decides to submit the paper
DecisionResult	CMS	Coordination event containing decision about submitted paper (if it has been accepted or rejected)
CRC	Author	Internal event sent when Author decides to submit the CRC

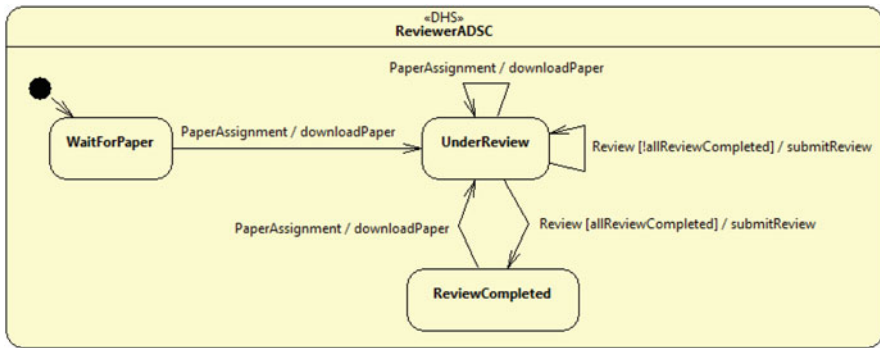


Fig. 10 Reviewer DSC diagram

Table 7 Reviewer actions

Action	Description
downloadPaper	Reviewer downloads papers that have been assigned to it
submitReview	Reviewer sends to the CMS system a review

Table 8 Reviewer guards

Guard	Description
allReviewCompleted	Reviewer checks if all the assigned papers were reviewed

Table 9 Reviewer events

Event	Sender	Description
PaperAssignment	CMS	Coordination event indicating the papers assigned to the Reviewer
Review	Reviewer	Internal event sent when Reviewer completes a review

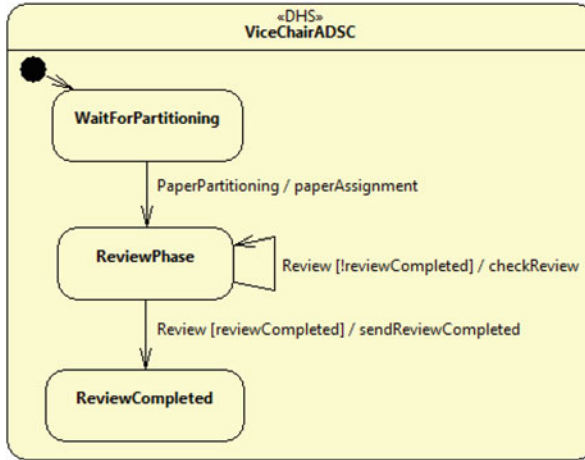


Fig. 11 Vice-Chair DSC diagram

Table 10 Vice-Chair actions

Action	Description
paperAssignment	Vice-Chair sends the event to CMS, indicating the assignment of papers to the reviewers that it manages
checkReview	Vice-Chair checks a review
sendReviewCompleted	Vice-Chair sends the event to CMS, indicating that all the expected reviews (sent from the reviewers managed by this Vice-Chair) were received

Table 11 Vice-Chair guards

Guard	Description
reviewCompleted	Vice-Chair checks if all the expected reviews (sent from the reviewers managed by this Vice-Chair) were received

Table 12 Vice-Chair events

Event	Sender	Description
PaperPartitioning	CMS	Coordination event indicating which reviewers must be managed by Vice-Chair and how to distribute the papers to be reviewed
Review	CMS	Coordination event containing a review

Vice-Chair

See Fig. 11 and Tables 10, 11 and 12.

Chair

See Fig. 12 and Tables 13, 14 and 15.

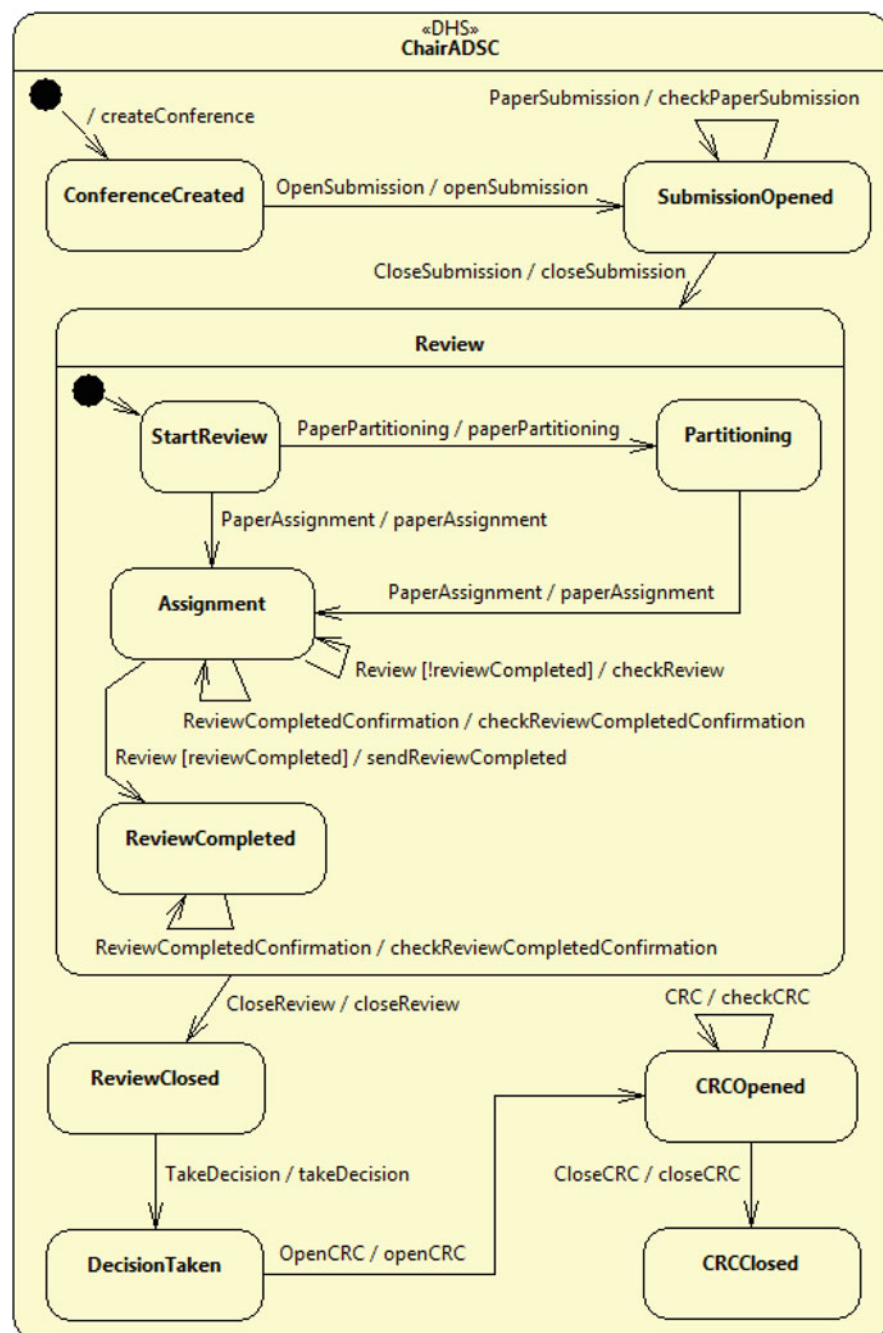


Fig. 12 Chair DSC diagram

Table 13 Chair actions

Action	Description
createConference	Chair creates and initializes the CMS conference system
openSubmission	Chair sends the event to CMS, indicating the opening of the paper submission phase
checkPaperSubmission	Chair checks a submitted paper
closeSubmission	Chair sends the event to CMS, indicating the closure of the paper submission phase
paperPartitioning	Chair sends the event to CMS, indicating how to distribute the papers to be reviewed among the reviewers and how to involve one or more Vice-Chairs in the management of a part of them
paperAssignment	Chair sends the event to CMS, indicating the assignment of some papers to be reviewed to the reviewers that it manages
checkReview	Chair checks a review
checkReviewCompletedConfirmation	Chair checks if all the reviews were successfully received
sendReviewCompleted	Chair sends the event to CMS, indicating that all the expected reviews (sent from the reviewers managed by this Chair) were received
closeReview	Chair sends the event to CMS, indicating the closure of the review phase
takeDecision	Chair sends the event to CMS, indicating the decisions taken on the submitted papers (if they have been accepted or rejected)
openCRC	Chair sends the event to CMS, indicating the opening of the CRC submission phase
checkCRC	Chair checks a submitted CRC
closeCRC	Chair sends the event to CMS, indicating the closure of the CRC submission phase

Table 14 Chair guards

Guard	Description
reviewCompleted	Chair checks if all the expected reviews (sent from the reviewers managed by Chair) were received

CMS

See Fig. 13 and Tables 16 and 17.

Platform-Independent ELDA MAS Code

In Fig. 14 part of the code (variables, actions, guards, and events) of the active behavior of the Reviewer role (see earlier section “[Reviewer](#)”) produced in the ELDA Coding activity is reported.

Table 15 Chair events

Event	Sender	Description
OpenSubmission	Chair	Internal event sent when Chair decides to open the paper submission phase
PaperSubmission	CMS	Coordination event containing a submitted paper
CloseSubmission	Chair	Internal event sent when Chair decides to close the paper submission phase
PaperPartitioning	Chair	Internal event sent when Chair decides to involve Vice-Chair in the management of the papers to be reviewed
PaperAssignment	Chair	Internal event sent when Chair decides to assign some papers to be reviewed to the reviewers that it manages
Review	CMS	Coordination event containing a review
ReviewCompletedConfirmation	CMS	Coordination event indicating that all the reviews were received
CloseReview	Chair	Internal event sent when Chair decides to close the review phase
TakeDecision	Chair	Internal event sent when Chair wants to start the decision process about the submitted papers
OpenCRC	Chair	Internal event sent when Chair decides to open the CRC submission phase
CRC	CMS	Coordination event containing a submitted CRC
CloseCRC	Chair	Internal event sent when Chair decides to close the CRC submission phase

2.2 The Simulation Phase

The goal of the *Simulation* phase is to support the functional validation and performance evaluation of the MAS model produced in the *Modeling* phase (see Sect. 2.1). Specifically, the ELDASim simulation framework is exploited to fully support such phase. The *Simulation* process is composed of three main activities: *Performance Indices Definition*, *Simulation Implementation*, and *Simulation Execution*, as shown in Fig. 15. *Simulation* specifically involves a process role and five work products, as described in Fig. 16.

2.2.1 Process Roles

One role is involved in the Simulation: the Simulation Designer.

Simulation Designer

The Simulation Designer is responsible for the functional validation and performance evaluation of the MAS under-development through the following activities:

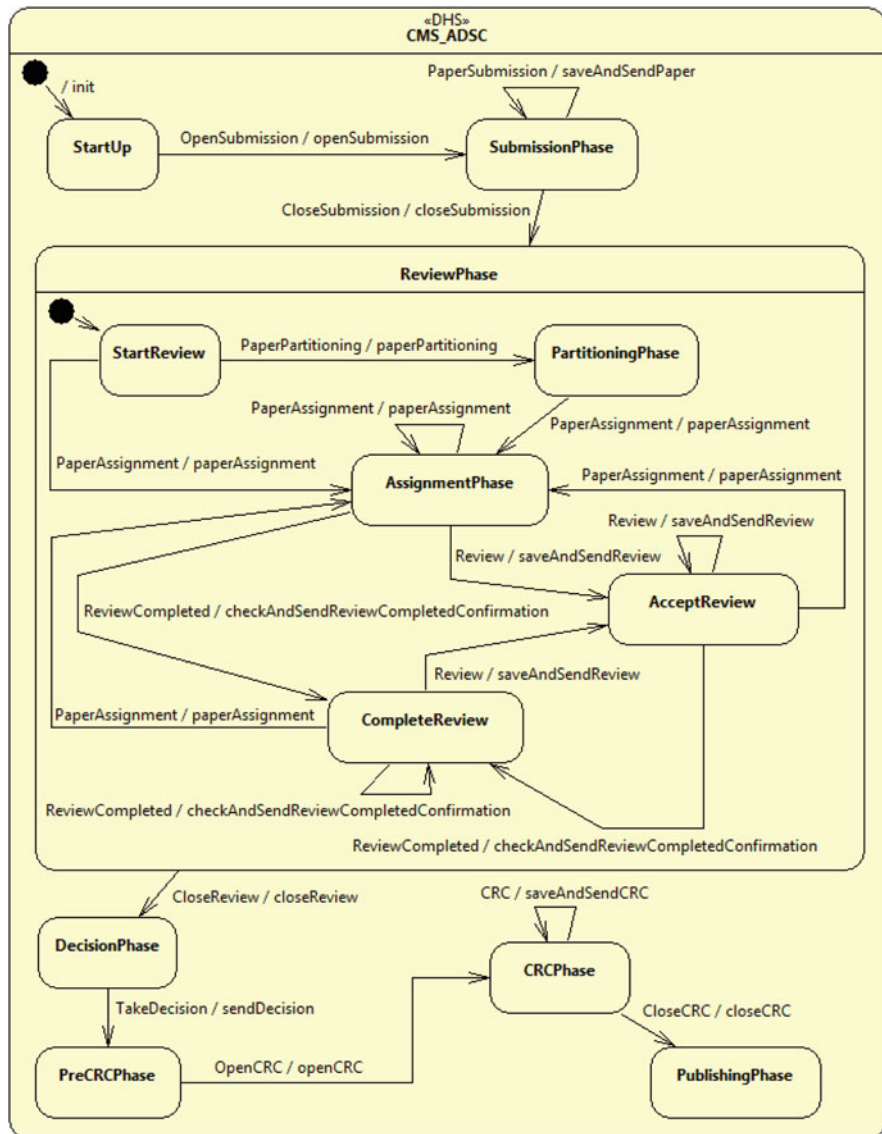


Fig. 13 CMS DSC diagram

- *Performance Indices Definition*: this activity allows the definition of the performance indices which will be evaluated during the simulation.
- *Simulation Implementation*: it produces a simulator program that allows executing the simulation.
- *Simulation Execution*: in this activity the simulation is executed and the simulation results are obtained.

Table 16 CMS actions

Action	Description
init	CMS initializes the conference with parameters decided by Chair at the conference creation and sends the call for paper (CFP) to authors
openSubmission	CMS opens the paper submission phase
saveAndSendPaper	CMS saves a submitted paper and sends a paper submission notification to Chair
closeSubmission	CMS closes the paper submission phase
paperPartitioning	CMS sends the event to Vice-Chair, indicating which reviewers this Vice-Chair must manage and how to distribute the papers to be reviewed among the different reviewers
saveAndSendReview	CMS stores a review and sends it to the corresponding reviewer manager (Chair or Vice-Chair)
saveAndSendCRC	CMS saves and sends a CRC to Chair
closeReview	CMS closes the review phase
sendDecision	CMS sends the decisions, taken by Chair about submitted papers, to authors
closeCRC	CMS closes the CRC submission phase
paperAssignment	CMS assigns some papers to be reviewed to reviewers
openCRC	CMS opens the CRC submission phase
checkAndSendReview CompletedConfirmation	If all the reviewer managers (Chair or Vice-Chair) received all reviews from the reviewers that they manage, CMS sends the event to Chair, indicating that all the reviews have been received

2.2.2 Activity Details

Performance Indices Definition and *Simulation Implementation* are atomic activities that have no tasks, whereas *Simulation Execution* activity has two tasks as described here.

Simulation Execution Activity

The *Simulation Execution* activity comprises the two tasks described in Table 18. The flow of tasks in the *Simulation Execution* activity is reported in Fig. 17.

2.2.3 Work Products

Figure 18 reports the relationships among the work products of this step and the ELDA MMMElements (see Sect. 1.2).

Work Product Kinds

Table 19 describes the work products of the *Simulation*.

Simulator Program

In Fig. 19 the *Simulator Program* template produced for the CMS case study is described.

Table 17 CMS events

Event	Sender	Description
OpenSubmission	Chair	Coordination event indicating the opening of the paper submission phase
PaperSubmission	Author	Coordination event containing a submitted paper
CloseSubmission	Chair	Coordination event indicating the closure of the paper submission phase
PaperPartitioning	Chair	Coordination event sent when Chair decides to involve the Vice-Chair in the management of the papers to be reviewed
PaperAssignment	Chair/Vice-Chair	Coordination event sent by a reviewer manager (Chair or Vice-Chair) to assign some papers to be reviewed to the reviewer that it manages
Review	Reviewer	Coordination event containing a review
ReviewCompleted	Chair/Vice-Chair	Coordination event sent by a reviewer manager (Chair or Vice-Chair) indicating that all the reviews have been received from the reviewers it manages
CloseReview	Chair	Coordination event indicating the closure of the review phase
TakeDecision	Chair	Coordination event sent when Chair wants to decide about submitted papers
OpenCRC	Chair	Coordination event indicating the opening of the CRC submission phase
CRC	Author	Coordination event containing a submitted CRC
CloseCRC	Chair	Coordination event indicating the closure of the CRC submission phase

The methods of the CMS class are:

- `void resetSimulationParams()`: resets the simulation parameters
- `void loadParams(XMLTree configuration)`: loads and initializes the simulation parameters
- `void setupAS()`: performs the setup of the agent servers of the distributed simulated agent platform
- `void createSimPerformanceParamsTabs()`: creates database tables for storing the results obtained from the simulations
- `void setupAndStartCustomSimulation()`: starts the simulation up
- `void setupAgent()`: allows the setup of the agents involved in the simulation
- `void setAgentCodeDimension()`: sets the code dimension of the agents
- `void startAgent()`: starts the agents up
- `void traceSimPerformanceParams()`: traces the simulation performance parameter values obtained from the simulation
- `void clearAS()`: clears the agent servers up
- `void resetSimPerformanceParams()`: resets the tracing of the simulation results.

STATE	VARIABLES	
ReviewerADSC	int reviewCount ELDAId cms	

ACTION	CODE
downloadPaper	PaperAssignment evt = (PaperAssignment) e; String paperCode = (String) evt.getData(); download(paperCode); reviewCount++;
submitReview	Object review = ((Review) e).getData(); generate(new ELDAEventMSGRequest(self(), new Review(self(), cms, review)); reviewCount--;

GUARD	CODE
allReviewCompleted	return reviewCount == 0;

EVENT	SENDER	TYPE
PaperAssignment	CMS	ELDAEventMSG
Review	Reviewer	ELDAEventInternal

Fig. 14 ELDAFramework-based code of the Reviewer role

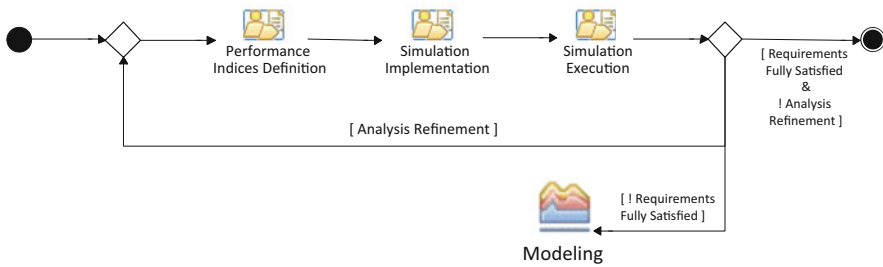


Fig. 15 The Simulation flow of activities

2.3 The Implementation Phase

The goal of the *Implementation* phase is to translate the MAS design model into code for a target platform. In particular, the translation is semi-automatic, supported by the ELDATool, and targeting the JADE platform. The *Implementation* process is composed of two main activities (*Platform-specific ELDA Implementation* and *Testing*), as shown in Fig. 20. In particular, *Implementation* involves two process roles and five work products (see Fig. 21).

2.3.1 Process Roles

Two roles are involved in the *Implementation*: Developer and Tester.

Developer

The Developer is responsible for:

- Platform-specific ELDA Implementation—this activity translates the MAS design model into code generated according to a real target platform (e.g., JADE) through ELDATool.

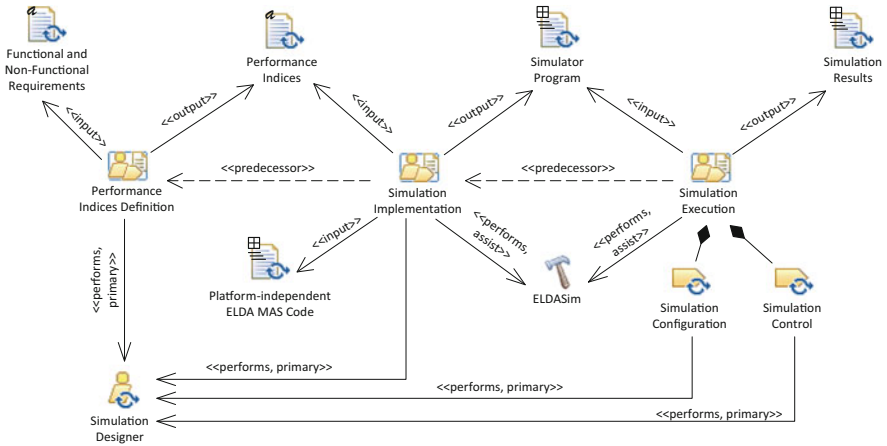


Fig. 16 The Simulation phase described in terms of activities, roles, and work products

Table 18 Tasks of Simulation Execution activity

Activity	Task	Task description	Role involved
Simulation Execution	Simulation Configuration	Configuration of simulation parameters	Simulation Designer (perform)
Simulation Execution	Simulation Control	Control of Simulation Execution	Simulation Designer (perform)

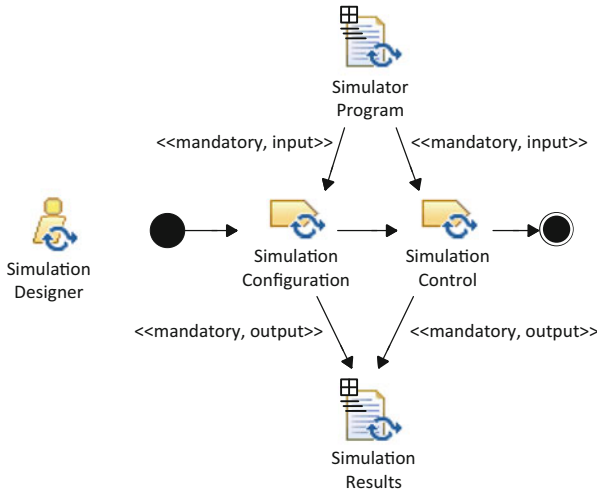


Fig. 17 The flow of tasks of the Simulation Execution activity

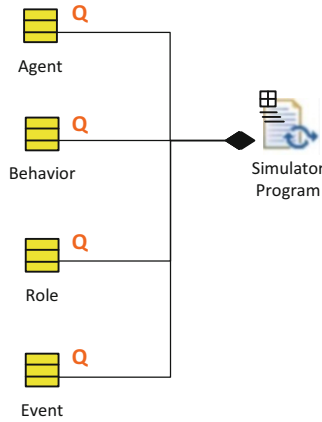


Fig. 18 The Simulation work products

Table 19 Simulation work products kinds

Name	Description	Work product kind
Functional and non-functional requirements	A document defining functional and non-functional requirements of the MAS under-development	Free text
Performance indices	The definition of the performance indices which will be evaluated during the simulation	Free text
Simulator Program	The resulting simulator program that allows executing the simulation	Structured
Platform-independent ELDA MAS code	Platform-independent code generated for the MAS under-development	Structured
Simulation results	Results of executed simulation	Structured

Tester

The Tester is responsible for:

- Testing—this activity executes some tests on the MAS under-development considering the performance indices evaluated during the simulation and produces a document containing the test results.

2.3.2 Activity Details

Platform-specific ELDA implementation and testing activities are atomic and do not have any tasks.

2.3.3 Work Products

The work products produced in this phase are the platform-specific ELDA MAS Code, which is the code of the MAS under-development for the JADE platform, and Testing Results, which is a set of real execution traces and table/plots of computed performance indices.

```

public class CMS extends MASSimulation {

    private static int nReviewerAgent;
    private static int nAuthorAgent;
    private static int nChairAgent;
    private static int nViceChairAgent;

    private static Hashtable<String, SimConfig> simsConfiguration;

    protected void resetSimulationParams () {}
    protected void loadParams (XMLTree configuration) throws Exception {}
    private static void initializeSimsConfiguration (Vector<XMLNode>
        simsCfg) throws ClassNotFoundException
        InvalidNodeException, InvalidAttributeException {}
    protected void setupAS () {}
    protected void createSimPerformanceParamsTabs () throws Exception {}
    protected void setupAndStartCustomSimulation () throws Exception {}
    protected void setupAgent () throws Exception {}
    protected void setAgentCodeDimension () {}
    protected void startAgent () {}
    protected void traceSimPerformanceParams () throws Exception {}
    protected void clearAS () {}
    protected void resetSimPerformanceParams () {}
}

```

Fig. 19 Code template of the Simulator Program

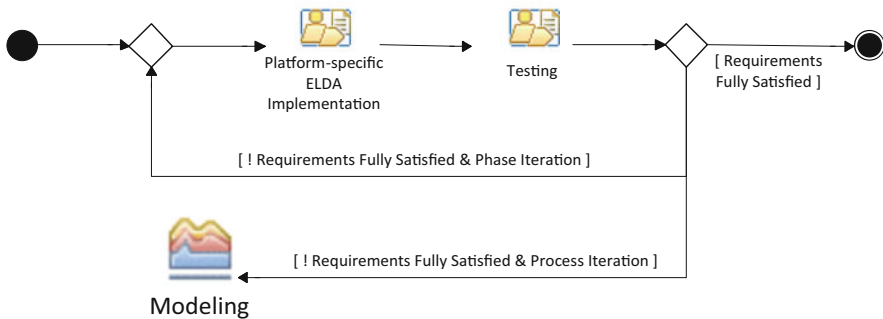


Fig. 20 The Implementation flow of activities

Work Product Kinds

Table 20 describes the work products of the Implementation:

Platform-Specific ELDA MAS Code

In Fig. 22 part of the JADE-based code (variables, actions, guards, and events) of the *Reviewer* role of the *Platform-specific ELDA MAS Code* produced for the CMS case study will be described.

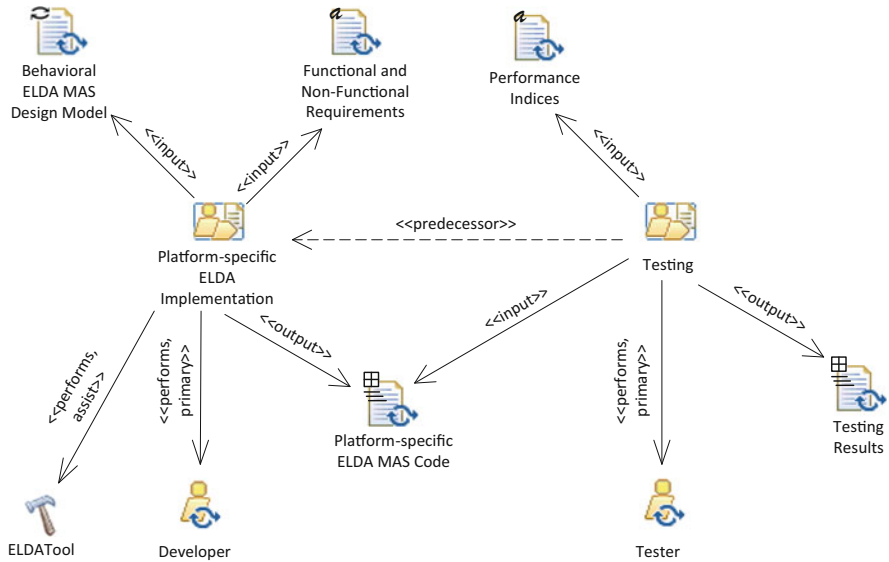


Fig. 21 The Implementation phase described in terms of activities, roles, and work products

Table 20 Implementation work products kinds

Name	Description	Work product kind
Functional and non-functional requirements	A document defining functional and non-functional requirements of the MAS under-development	Free text
Behavioral ELDA MAS design model	The DSC design of the MAS under-development	Behavioral
Platform-specific ELDA MAS code	The MAS code generated according to a real target platform (e.g. JADE)	Structured
Performance indices	The definition of the performance indices evaluated during the simulation	Free text
Testing results	A document containing the results of executed tests	Structured

3 Work Products Dependencies

The diagram in Fig. 23 depicts the dependencies among the different work products.

STATE	VARIABLES
ReviewerADSC	int reviewCount AID cms

ACTION	CODE
downloadPaper	PaperAssignment evt = (PaperAssignment) e; String paperCode = (String) evt.getData(); download(paperCode); reviewCount++;
submitReview	Serializable review = ((Review) e).getData(); ArrayList<AID> target = new ArrayList<AID>(); target.add(cms); generate(new ELDAEventMSGRequest(self(), new Review(self(), target, review))); reviewCount--;

GUARD	CODE
allReviewCompleted	return reviewCount == 0;

EVENT	SENDER	TYPE
PaperAssignment	CMS	ELDAEventMSG
Review	Reviewer	ELDAEventInternal

Fig. 22 The JADE-based code of the Reviewer role

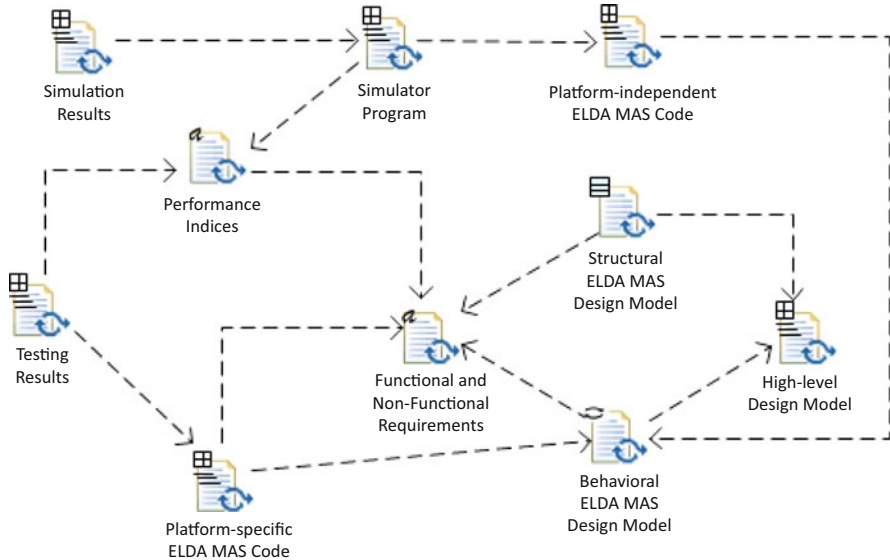


Fig. 23 Work products dependencies diagram

References

1. Aiello, F., Fortino, G., Gravina, R., Guerrieri, A.: A Java-based agent platform for programming wireless sensor networks. *Comput. J.* **54**(3), 439–454 (2011)
2. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. *Softw. Pract. Exper.* **31**(2), 103–128 (2001)
3. Brinkkemper, S., Lyytinen, K., Welke, R.: Method engineering: principles of method construction and tool support. In: *Proceedings of the IFIP TC8 WG8.1/8.2 Working Conference on Method Engineering* (1996)
4. Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: PASSIM: a simulation-based process for the development of multi-agent systems. *Int. J. Agent Orient. Softw. Eng.* **2**(2), 132–170 (2008)
5. Di Fatta, G., Fortino, G.: A customizable multi-agent system for distributed data mining. In: *Proceedings of the 22nd Annual ACM Symposium on Applied Computing* (2007)
6. Fortino, G., Rango, F.: An application-level technique based on recursive hierarchical state machines for agent execution state capture. *Sci. Comput. Program.* **78**(6), 725–746 (2013)
7. Fortino, G., Russo, W.: ELDAMeth: a methodology for simulation-based prototyping of distributed agent systems. *Inform. Softw. Technol.* **54**(6), 608–624 (2012)
8. Fortino, G., Garro, A., Mascillaro, S., Russo, W.: A multi-coordination based process for the design of mobile agent interactions. In: *Proceedings of IEEE Symposium on Intelligent Agents* (2009)
9. Fortino, G., Garro, A., Mascillaro, S., Russo, W.: Using event-driven lightweight DSC-based agents for MAS modeling. *Int. J. Agent Orient. Softw. Eng.* **4**(2) (2010)
10. Fortino, G., Garro, A., Mascillaro, S., Russo, W., Vaccaro, M.: Distributed architectures for surrogate clustering in CDNs: a simulation-based analysis. In: *Proceedings of the 4th International Workshop on the Use of P2P, GRID and Agents for the Development of Content Networks* (2009)
11. Fortino, G., Garro, A., Russo, W.: An integrated approach for the development and validation of multi agent systems. *Comput. Syst. Sci. Eng.* **20**(4), 94–107 (2005)
12. Fortino, G., Frattolillo, F., Russo, W., Zimeo, E.: Mobile active objects for highly dynamic distributed computing. In: *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2002)
13. Fortino, G., Rango, F., Russo, W.: Engineering multi-agent systems through statecharts-based JADE agents and tools. In: Nguyen, N.T. (ed.) *Transactions on Computational Collective Intelligence VII. Lecture Notes in Computer Science*, vol. 7270, pp. 61–81. Springer, Heidelberg (2012)
14. Fortino, G., Russo, W., Zimeo, E.: A statecharts-based software development process for mobile agents. *Inform. Softw. Technol.* **46**(13), 907–921 (2004)
15. Luck, M., McBurney, P., Preist, C.: A manifesto for agent technology: towards next generation computing. *Auton. Agents Multi-Agent Syst.* **9**(3), 203–252 (2004)
16. Omicini, A., Zambonelli, F.: Challenges and research directions in agent-oriented software engineering. *Auton. Agents Multi-Agent Syst.* **9**(3), 253–283 (2004)

The Gaia Methodology Process

Luca Cernuzzi, Ambra Molesini, and Andrea Omicini

Abstract

Gaia was the first complete methodology proposed for the development of multi-agent systems (MASs), and was subsequently improved to designing and building systems in complex, open environments. Gaia focuses on the use of the organizational abstractions to drive the analysis and design of MAS. Gaia models both the macro (social) aspects and the micro (agent internals) aspects of MAS, and devotes a specific effort to model the organizational structure and the organizational rules that govern the global behavior of the agents in the organization. In this chapter we present the complete documentation of the Gaia process following the IEEE-FIPA Documentation Template.

1 Introduction

The Gaia methodology was initially proposed by Wooldridge et al. in [5,6], extended toward social issues and open systems in [7], and largely revised and improved by Zambonelli et al. in [8]. The original version of Gaia focuses on the analysis and design of closed Multi-agent Systems (MASs), assuming the benevolence

L. Cernuzzi

Departamento de Ingeniería Electrónica e Informática, Universidad Católica “Nuestra Señora de la Asunción” Campus Universitario, C.C. 1683, Asunción, Paraguay
e-mail: lcernuzz@uca.edu.py

A. Molesini (✉)

DISI, Alma Mater Studiorum – Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
e-mail: ambra.molesini@unibo.it

A. Omicini

DISI, Alma Mater Studiorum – Università di Bologna, Via Sacchi 3, 47521 Cesena, Italy
e-mail: andrea.omicini@unibo.it

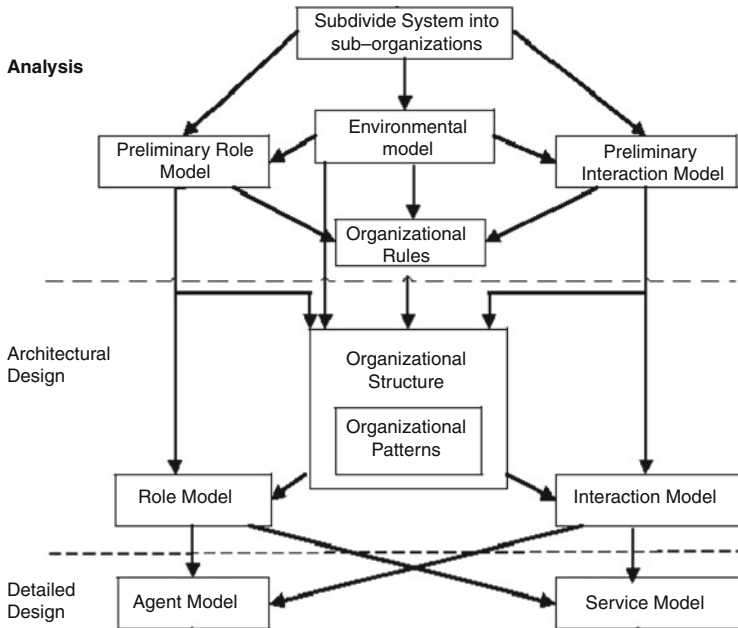


Fig. 1 An overview of the Gaia process

and predisposition to collaborate of the agents in the organizations. The extended version of Gaia is more oriented to designing and building systems in complex, open environments. The Gaia methodology was one of the first appeared in the AOSE field, and offered interesting insights into the development of different other Agent Oriented Software Engineering methodologies. It explicitly focuses on using organizational abstractions to drive the analysis and design of MAS usually characterizing complex and open environments. It models both the macro (social) aspect and the micro (agent internals) aspects of a MAS. Moreover, Gaia devotes a special effort to model the organizational structure and to specify the organizational rules that govern the global behavior of the agents in the organization, avoiding conflict based on self-interest actions. An overview of the Gaia methodologies is presented in Fig. 1. The following are useful references for the Gaia process:

- M. Wooldridge, N.R. Jennings, D. Kinny. *A Methodology for Agent-oriented Analysis and Design* [5]
- M. Wooldridge, N.R. Jennings, D. Kinny. *The Gaia Methodology for Agent-oriented Analysis and Design* [6]
- F. Zambonelli, N.R. Jennings, A. Omicini, M. Wooldridge. *Agent-Oriented Software Engineering for Internet Applications* [7].
- F. Zambonelli, M. Wooldridge, N.R. Jennings. *Developing Multiagent Systems: The Gaia Methodology* [8].

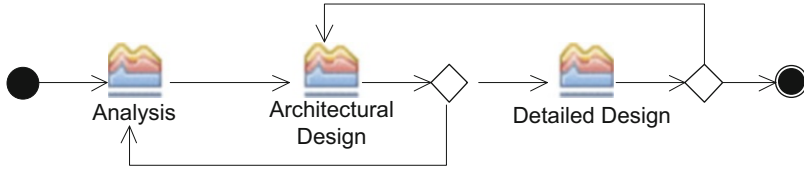


Fig. 2 The Gaia process phases

- F. Zambonelli, N.R. Jennings, M. Wooldridge. *Multiagent Systems as Computational Organizations: The Gaia Methodology* [9].
- L. Cernuzzi, F. Zambonelli. *Experiencing AUML in the Gaia Methodology* [1].
- L. Cernuzzi, F. Zambonelli. *Adaptive Organizational Changes in Agent-Oriented Methodologies* [2].
- L. Cernuzzi, A. Molesini, A. Omicini, F. Zambonelli. *Adaptable Multi-Agent Systems: The Case of the Gaia Methodology* [4].

1.1 Process Life Cycle

In the original version [6] the development process underpinning Gaia was composed by only two phases: Analysis and Design. However, due to different limitation of the original version of Gaia [3], the work in [8] introduced relevant improvements to better exploit the organizational perspective in the analysis and design activities.

These improvements led to a reformulation of the Gaia process, including the introduction of a new phase—Architectural Design—and a set of new models. Therefore, Gaia currently proposes three main phases—namely, the Analysis, the Architectural Design, and the Detailed Design, as represented in Fig. 2.

It is worth noting that Gaia does not directly deal with the requirement phase, although the difference with the analysis phase is not always obvious, and implementation issues. The authors believe that requirement elicitation activities should be neutral with respect to the adopted paradigm for the design and the eventual solution technology. Finally, the result of the design phase is assumed to be something that could be implemented in a technology neutral way, that is, by using a traditional method (such as object orientation or component-ware) or an appropriate agent-programming framework. The details of each step will be discussed in the following sections.

1.2 Meta-model

The MAS meta-model of Gaia is mostly devoted to represent a MAS as a social organization, and adopts *organizations*, *agents*, *roles*, *protocols*, *organizational rules*, and *environment* as its basic building blocks. The meta-model adopted by

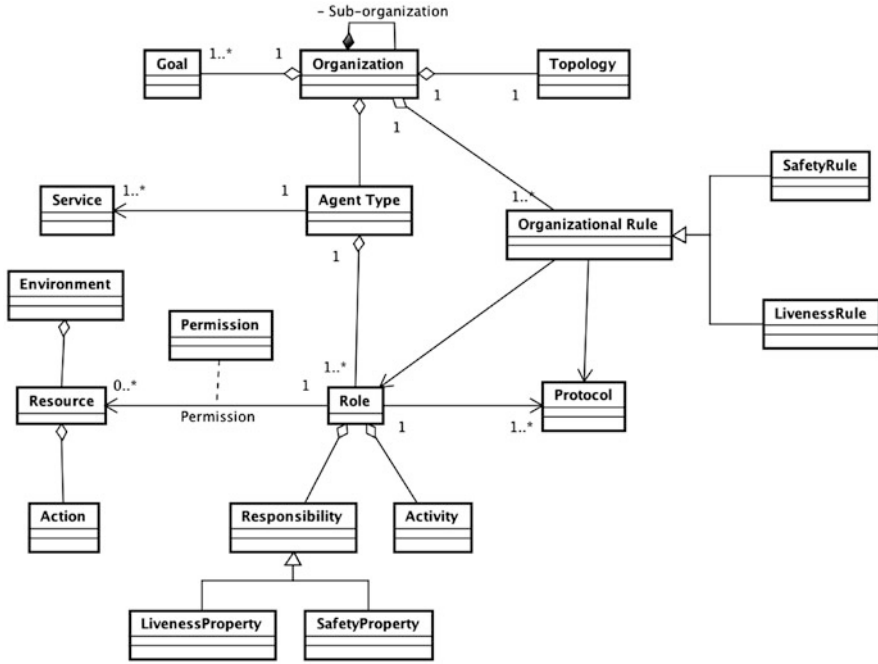


Fig. 3 The Gaia meta-model

Gaia is represented in Fig. 3, where Gaia abstract entities—MAS Meta-Model Elements (MMMEs)—are depicted along with their mutual relations. In particular: the Gaia agent is an entity that plays one or more roles; a role represents specific behaviors to be played by an agent, defined in terms of responsibilities and activities, and of its interactions with other roles (i.e., protocols); an agent plays a role by actualizing the behavior in terms of services.

The primary role of organizational abstractions is represented by means of the organization and the associated organizational rules elements. The MAS organization could be composed by different sub-organizations, each one with a specific structure (i.e., topology) and goals. In turn, each sub-organization is defined as a composition of agents and organizational rules.

The organizational structure describes the overall architecture of the system characterizing the position of each agent (better, of the agents playing specific roles) in it, and the structural relationships with the other agents. Organizational rules have the goal of specifying some constraints that agents in the organization (or sub-organization) have to observe. To some extent, they could be considered as the social laws that have to drive all interactions in the organization and the evolution of the organization itself. They may be either global, affecting the behavior of the society as a whole, or concerning only specific roles or protocols.

Finally, the environment abstraction explicitly specifies all the entities and resources a MAS may interact with, according to defined permissions.

Analysis

In the Analysis, Gaia covers different concepts of the meta-model. Gaia starts with the identification of the *Sub-organizations* for the MAS with their respective *Goals*.

Then, for each sub-organization it prescribes to specify the Preliminary Role Model identifying a set of *Roles* (i.e., the functional requirement to achieve the sub-organizational goals). Each *Role* includes a set of Activities and Responsibilities that are in turn composed by *LivenessProperty* and *SafetyProperty* (to state the active and safe behaviors of an agent playing such a role).

Roles are usually interacting among them according to different *Protocols* that are specified in the Preliminary Interaction Model. Moreover, in order to achieve their objectives, *Roles* have to interact with the MAS operation *Environment* (i.e., the Environmental Model) that is composed of a set of computational *Resources*. Each resource enables the role to perform different *Actions* according to specific *Permission*. Finally, system analysts have to specify the *OrganizationalRules* that govern the general behavior of the MAS. Such rules include *LivenessRules* and *SafetyRules* (to specify norms and constraints on the desirable and undesirable behaviors in the sub-organization or in the overall MAS).

Architectural Design

The main goal of the Architectural Design phase is defining the most proper Organizational Structure for the MAS, i.e., the *Topology* of the overall MAS and/or of each sub-organization. The *Topology* complements the *Goals* and the *Organizational Rules* (i.e., the control regime) of the sub-organizations. It is possible that new *Roles* and *Protocols* arise according to the adopted organizational structure.

Detailed Design

Finally, the Detailed Design covers the specification of the set of *Agent Types*, i.e., the agent classes in the MAS. Each instance of an *Agent Type* is an Agent playing one or more *Roles*. In addition, the set of activities an agent is able to perform are described in terms of *Services* (Table 1).

2 Phases of the Gaia Process

2.1 Analysis

The Analysis phase in Gaia covers the requirements in terms of functions and activities adopting the agent paradigm according to the organizational perspective. This involves firstly identifying which loosely coupled sub-organizations possibly

Table 1 The Gaia entities definitions

Concepts	Definition	Phase
Role	<i>Entity representing specific behaviors to accomplish some tasks</i>	Analysis, Architectural Design
Activity	<i>action that a role can perform to reach some specific goal</i>	Analysis, Architectural Design
Responsibility	<i>is the key attribute associated with a role and determines functionality of the role</i>	Analysis, Architectural Design
LivenessProperty	<i>describes those states of affairs that an agent must bring about (intuitively states that "something good happens", and hence that the agent carrying out the role is alive)</i>	Analysis, Architectural Design
SafetyProperty	<i>an acceptable state of affairs maintained across all states of execution (intuitively states that "nothing bad happens")</i>	Analysis, Architectural Design
Environment	<i>involves determining all the entities and resources that the MAS, as a whole, can exploit, control, or consume when it is working toward the achievement of the organizational goal</i>	Analysis
Resource	<i>elements in the environment for agents to interact with</i>	Analysis
Action	<i>an activity impacting resources of the environment according to roles objectives</i>	Analysis
Permission	<i>identifies the resources that are available to a role in order to realize its responsibilities (i.e., the "rights" associated with a role)</i>	Analysis, Architectural Design
Protocol	<i>defines the way that a role can interact with other roles</i>	Analysis, Architectural Design
OrganizationalRule	<i>expresses general requirements for the proper instantiation and execution of a MAS in terms of relationships and constraints between roles, between protocols, and between roles and protocols in the organization</i>	Analysis, Architectural Design
SafetyRule	<i>properties that the system must guarantee to prevent undesirable behaviors in the organization</i>	Analysis, Architectural Design
LivenessRule	<i>properties that the system must guarantee to enable desirable behaviors in the organization</i>	Analysis, Architectural Design
Organization	<i>a group of agents interacting according to a specific organizational structure (i.e., topology) and respecting some defined organizational rules</i>	Architectural Design
Topology	<i>describes the relationships between roles (and therefore their positions) in an organization</i>	Analysis, Architectural Design
Goal	<i>a single objective of an organization that has to be accomplished by one or more agents in the organization</i>	Analysis, Architectural Design
AgentType	<i>defines a class of pro-active, autonomous entity that plays one or more roles</i>	Detailed Design
Service	<i>a function of the agent; that is, simply a single, coherent block of activity that an agent will engage in</i>	Detailed Design

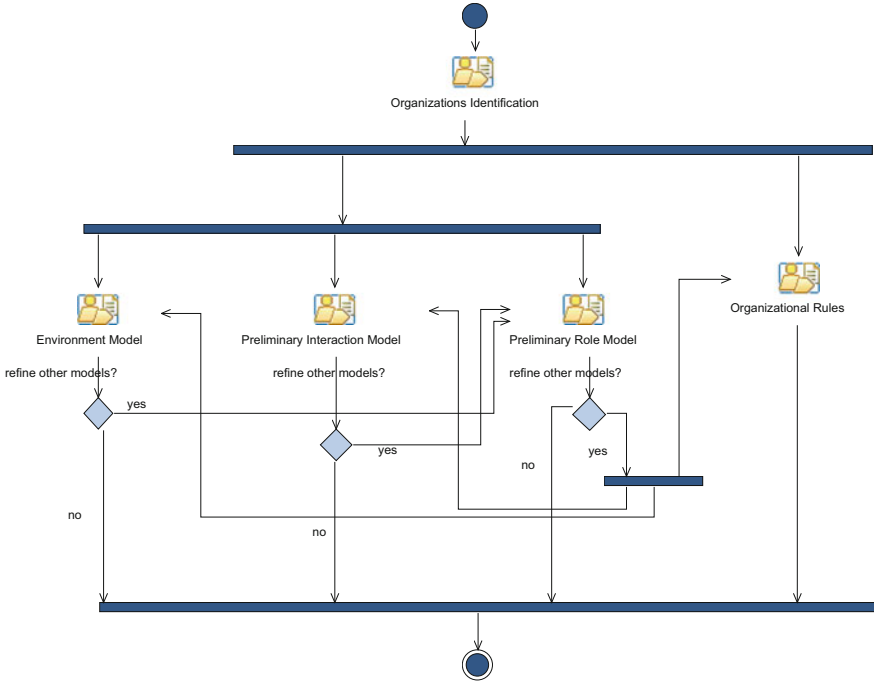


Fig. 4 The Analysis process

compose the whole systems; then, for each of these, producing four basic abstract models: (1) the environmental model, to capture the characteristics and their computational representation of the MAS operational environment; (2) a preliminary roles model, to capture the key task-oriented activities to be played in the MAS; (3) a preliminary interactions model, to capture basic inter-dependencies between roles and their corresponding protocols; and (4) a set of organizational rules, expressing global constraints/directives that must underlie the MAS functioning. The different models can in principle be performed concurrently. However, there exists a strict interdependence between them, since the modification of one specific diagram (e.g., the introduction of a new preliminary role in the sub-organization) typically implies the modification of other diagrams, too.

Figure 4 shows the Analysis process, while Fig. 5 presents the flow of activities, the involved roles, and the work products.

2.1.1 Process Roles

One role is involved in the Analysis phase: the System Analyst.

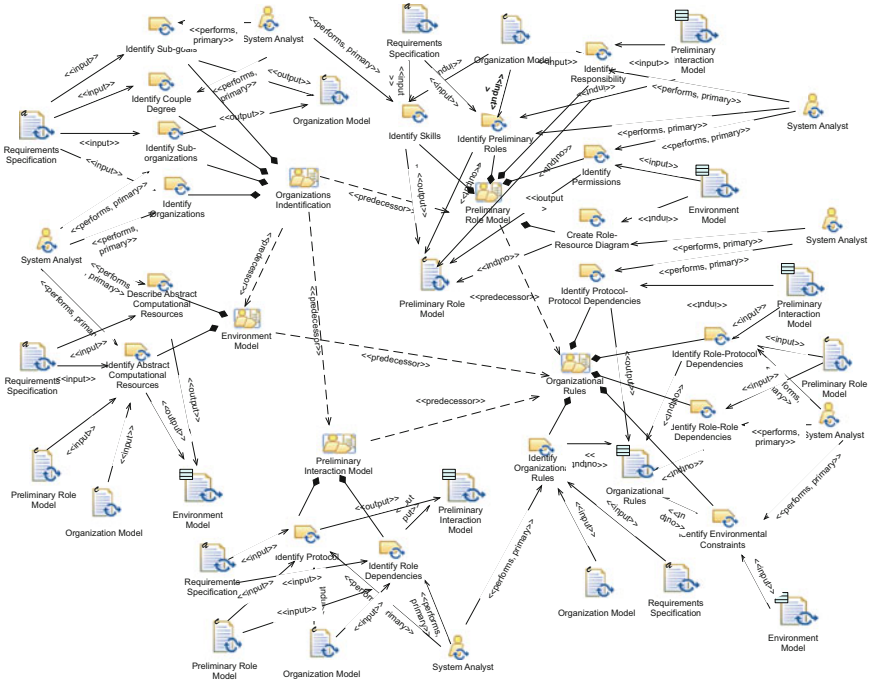


Fig. 5 The Analysis flow of activities, roles, and work products

System Analyst

A System Analyst is responsible for:

- Identification of the organization structure and its main characteristics such as the couple degree, sub-organizations, and organizational rules
- Identification of the preliminary roles and their main characteristics such as skills, permissions, and responsibilities
- Identification of the preliminary protocols
- Identification of the resources presented in the system environment.

2.1.2 Activity Details

Organizations Identification Activity

The flow of tasks in the *Organizations Identification* activity is reported in Fig. 6 and the tasks are detailed in the following table.

Environment Model Activity (Fig. 7)

Preliminary Role Model Activity

The flow of tasks in this activity is reported in Fig. 8 and the tasks are detailed in the following table.

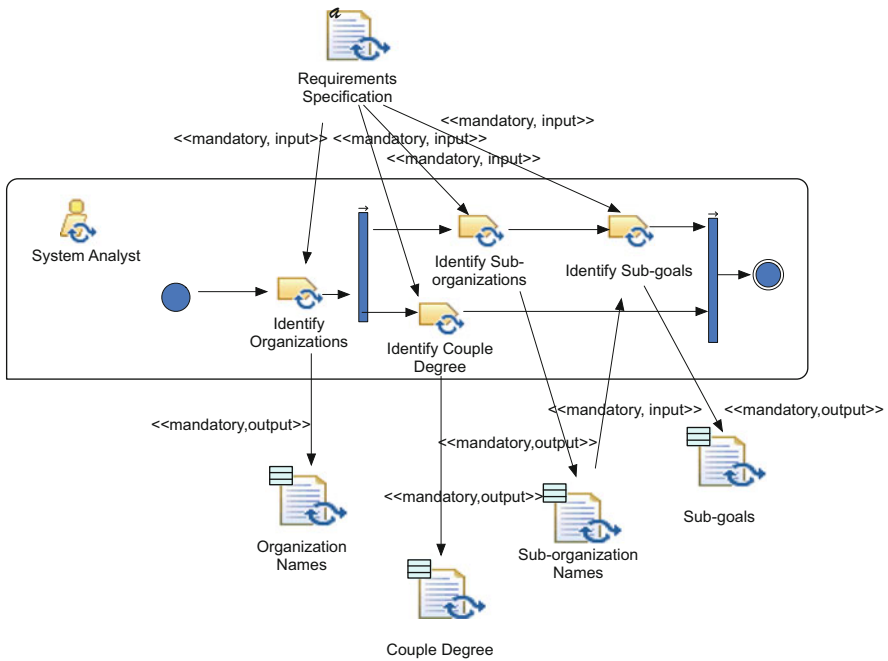


Fig. 6 The flow of tasks in the Organizations Identification activity

Activity	Task	Task Description	Role Involved
Organizations identification	Identify organization	<i>the MAS could be seen as a global organization with its general goals</i>	System Analyst (perform)
Organizations identification	Identify couple degree	<i>identifying the couple degree of the sub-organizations</i>	System Analyst (perform)
Organizations identification	Identify sub-organizations	<i>identifying how to fruitfully decompose the global organization into loosely coupled sub-organizations</i>	System Analyst (perform)
Organizations identification	Identify sub-goals	<i>each sub-organization has its own goals that contribute to the goals of the overall MAS</i>	System Analyst (perform)

Preliminary Interaction Model Activity

The flow of tasks in this activity is reported in Fig. 9 and the tasks are detailed in the following table.

Organizational Rules Activity

The flow of tasks in this activity is reported in Fig. 10 and the tasks are detailed in the following table.

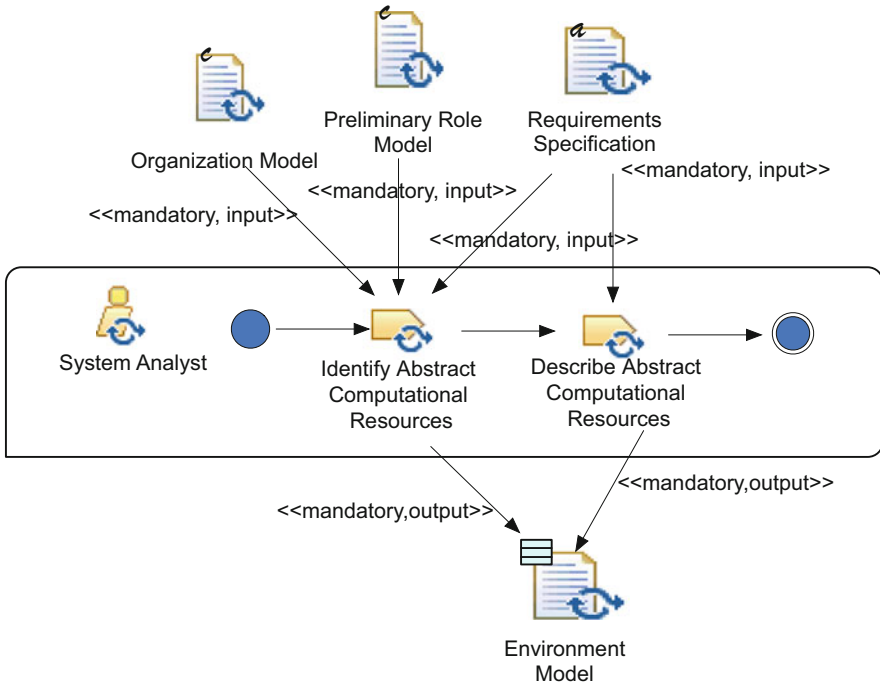


Fig. 7 The flow of task in the Environment Model activity

Activity	Task	Task Description	Role Involved
Environment Model	Identify abstract computational resources	<i>A set of variables or tuples, made available to the agents for sensing (e.g., reading their values), for effecting (e.g., changing their values) or for consuming (e.g., extracting them from the environment)</i>	System Analyst (perform)
Environment Model	Describe abstract computational resources	<i>Each resource is described in terms of a symbolic name, is characterized by the type of actions that the agents can perform on it, and could present additional textual comments and descriptions</i>	System Analyst (perform)

2.1.3 Work Products

Figure 11 reports the relationships among the work products of this phase and the MMMEs of the Analysis.

Kinds of Work Products

Table 2 describes all the work products of the Analysis.

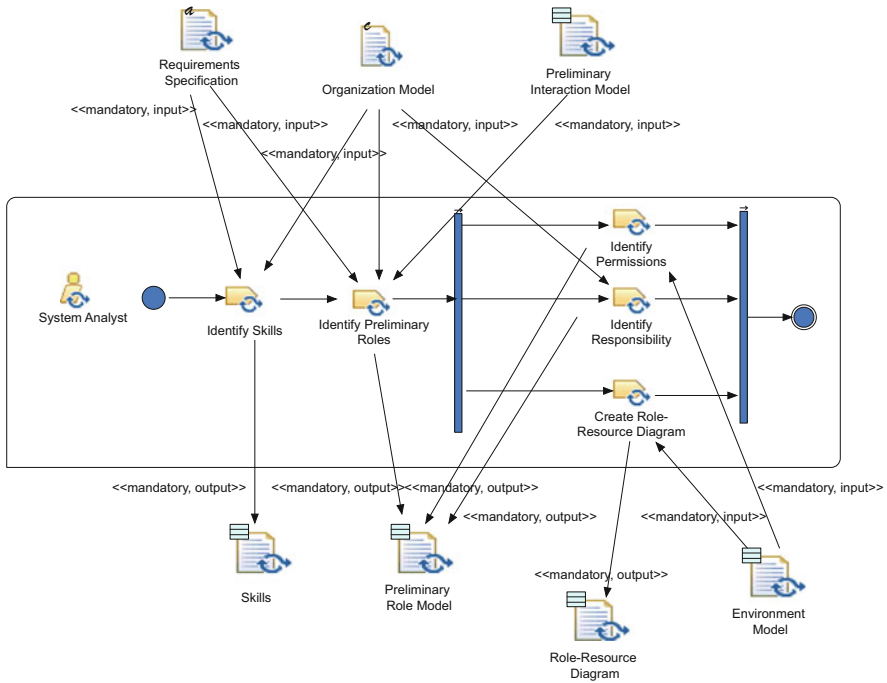


Fig. 8 The flow of tasks in the Preliminary Role Model activity

Activity	Task	Task Description	Role Involved
Preliminary Role Model	Identify skills	<i>some characteristics that are required by the organization to achieve its goals</i>	System Analyst (perform)
Preliminary Role Model	Identify preliminary roles	<i>basic skills of the organization. Each of them, with their activities, can be carried out (from) by an agent</i>	System Analyst (perform)
Preliminary Role Model	Identify permissions	<i>permissions relate agent roles to the environment identifying the resources that can be used to carry out the role and stating the resource limits within which the role must operate (e.g., possibly change or consume them)</i>	System Analyst (perform)
Preliminary Role Model	Identify responsibilities	<i>determine the expected behavior of a role in terms of liveness properties and safety properties</i>	System Analyst (perform)
Preliminary Role Model	Create a role-resource diagram	<i>a graphical diagram representing what the agents playing the role must be allowed to do or must not be allowed to do with the available environmental resources</i>	System Analyst (perform)

Organization Model

The following table presents an example of the Organization Model for the conference management case study.

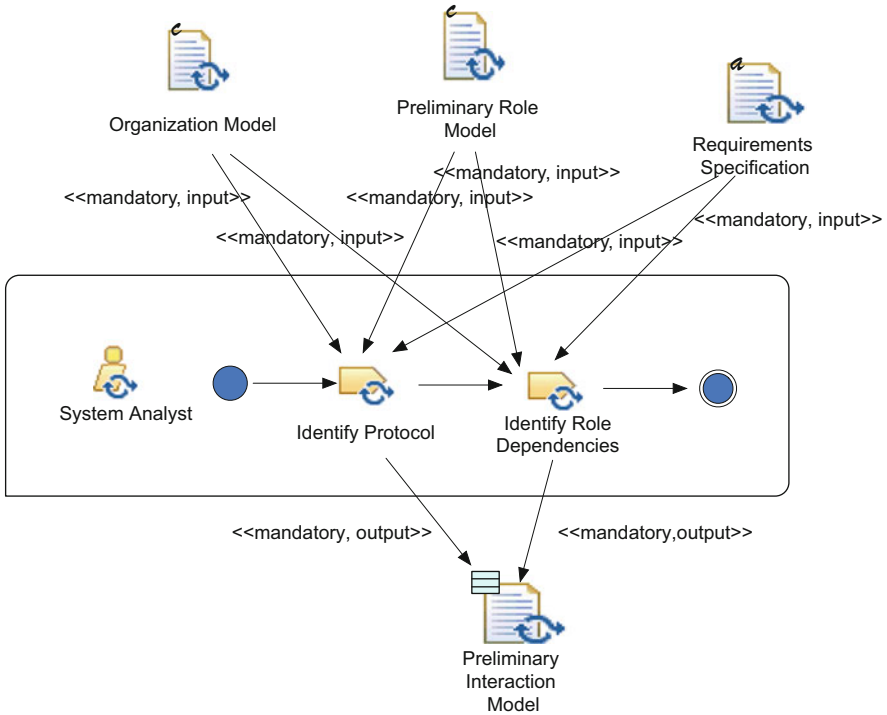


Fig. 9 The flow of tasks in the Preliminary Interaction Model activity

Activity	Task	Task Description	Role Involved
Preliminary Interaction Model	Identify protocol	<i>A pattern of interaction that has been formally defined and abstracted away from any particular sequence of execution steps</i>	System Analyst (perform)
Preliminary Interaction Model	Identify role dependencies	<i>Relationships and interactions between the various roles in the MAS organization that allow roles to reach their own goals</i>	System Analyst (perform)

Environment Model

Table 3 presents an example of the Environment Model for the conference management case study.

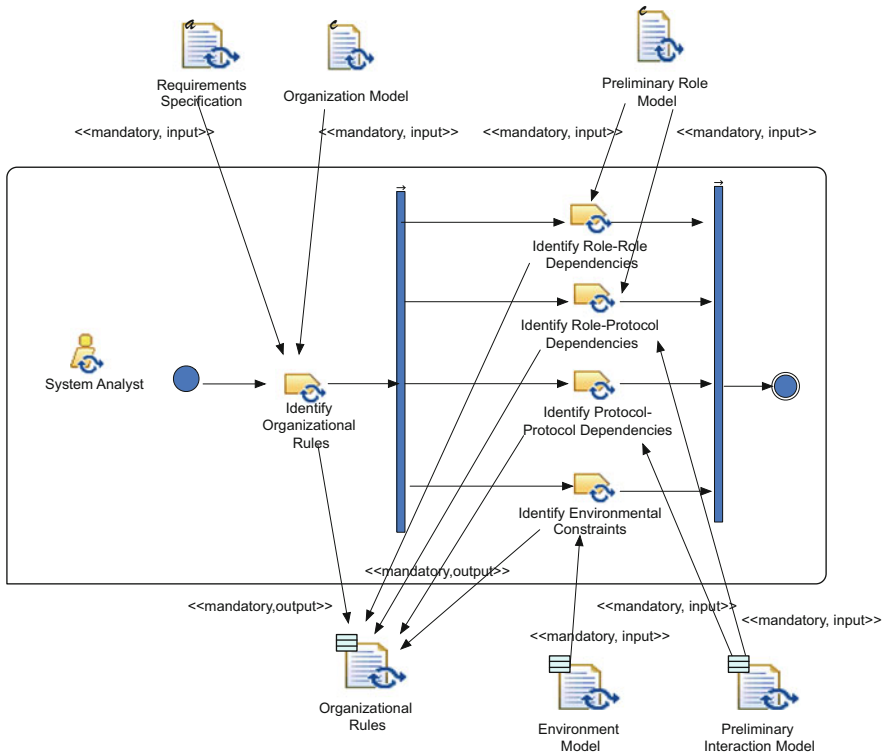


Fig. 10 The flow of tasks in the Organizational Rules activity

Activity	Task	Task Description	Role Involved
Organizational Rules	Identify Organizational Rules	Organizational rules are considered as responsibilities of the organization as a whole and are represented in terms of liveness and safety organizational rules	System Analyst (perform)
Organizational Rules	Identify role-role dependencies	Norms and constraints on relationships between roles (e.g., a role can be played only if another role has been played before; or that two roles can never be played concurrently by the same entity)	System Analyst (perform)
Organizational Rules	Identify role-protocol dependencies	Norms and constraints that relate roles and their interactions (e.g., the protocol must be executed only once by the role)	System Analyst (perform)
Organizational Rules	Identify protocol-protocol dependencies	Norms and constraints on the interactions among roles (e.g., the protocol P1 must necessarily be executed before the execution of protocol P2)	System Analyst (perform)
Organizational Rules	Identify environmental constraints	Norms and constraints on the environmental resources (e.g., each paper will receive three reviews from different referees)	System Analyst (perform)

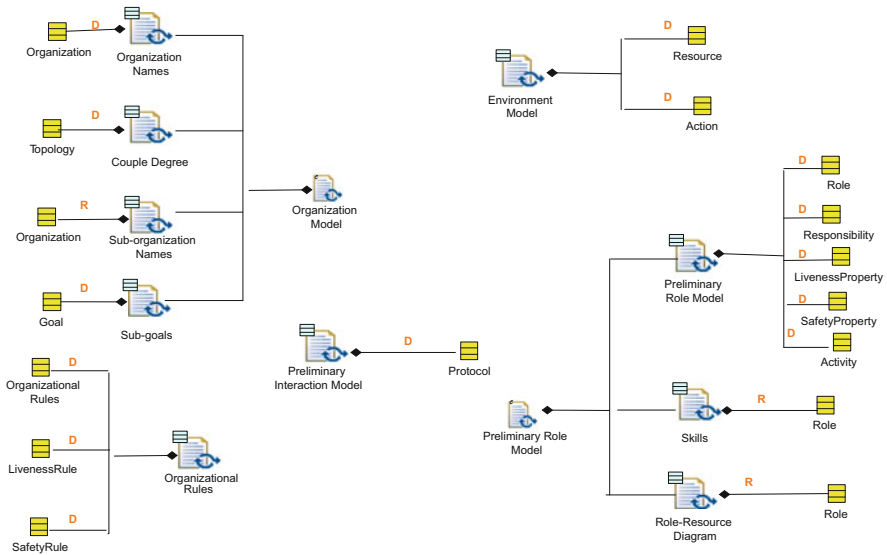


Fig. 11 The Analysis work products

Table 2 Analysis work products kinds

Name	Description	Work Product Kind
Requirement Specification	<i>A description of the problem to be solved (out of the scope of Gaia)</i>	Free text
Organization Model	<i>A composition of other work products that defines the MAS and its components</i>	Composite (structural)
Organization Names	<i>The name of the overall organization</i>	Structural
Couple Degree	<i>The degree of relationships and dependencies among sub-organizations</i>	Structural
Sub-organization Names	<i>The name of each sub-organization</i>	Structural
Sub-goals	<i>The goals of a sub-organization</i>	Structural
Environment Model	<i>A set of computational resources with their description and the actions an agent can perform on them</i>	Structural
Preliminary Role Model	<i>A composition of other work products that defines the roles</i>	Composite (structural)
Preliminary Role Model	<i>The basic skills of the organization that can be carried out from an agent</i>	Structural
Skills	<i>Set of basic characteristics that are required by the organization to achieve its goals</i>	Structural
Role-Resource Diagram	<i>A diagram representing the actions allowed or not acceptable on the available environmental resources</i>	Structural
Organizational Rule	<i>A set of norms and constraints that governs the organization (or a sub-organization) as a whole</i>	Structural
Preliminary Interaction Model	<i>Relationships and interactions between the various roles in the MAS</i>	Structural

Table 3 The environment model for the review sub-organization

Action	Environment Abstraction	Description
Reads	Paper submitted	the web site receives a paper
	Review submitted	the web site receives a review
Changes	DB submission	insert in the database the paper or the review received; one per each track
	DB reviewer	insert in the database the personal data of the reviewer, the topic of expertise, and the maximum number of papers the referee accepted to review

Table 4 The ReviewCatcher functional role schema

Role name: ReviewCatcher		
Description: This role is in charge of selecting reviewers and distributing papers among them.		
Protocol and activities: <u>GetPaper</u> , <u>CheckPaperTopic</u> , <u>CheckRefereeExpertise</u> , <u>CheckRefereeConstraints</u> , <u>AssignPaperReferee</u> , <u>ReceiveRefereeRefuse</u> , <u>UpdateDBSubmission</u> , <u>UpdateDBReferee</u>		
Permissions:		
Reads	<i>paper_submitted</i> <i>referee-data</i>	in order to check the topic and authors in order to check the expertise and constraint (i.e., the referee is one of the authors, or belongs to the same organization)
Changes	<i>DB Submission</i> <i>DB Referee</i>	assigning a referee to the paper assigning the paper to the referee incrementing the number of assigned papers
Responsibilities:		
<i>Liveness:</i>	<u>ReviewCatcher = (GetPaper.CheckPaperTopic.CheckRefereeExpertise. CheckRefereeConstraints.AssignPaperReferee.[ReceiveRefereeRefuse] UpdateDBSubmission.UpdateDBReferee)"</u>	
<i>Safety:</i>	$\forall \text{ paper: number_of_referees} \geq n$ Referee \neq Author Referee_organization \neq Author_organization	

Preliminary Role Model

Table 4 presents an example of the Preliminary Role Model for the conference management case study.

Preliminary Interaction Model

Figure 12 presents an example of the Preliminary Interaction Model for the conference management case study.

Sub-organization	Couple degree	Sub-goals
Submission Org	Loosely coupled	Distribution of CFP; paper submission; papers classification; assignment of a submission number; authors notification
Review Org	Loosely coupled	Papers assignment; providing reviews; collections of reviews; acceptance/rejection; authors notification
Publication Org	Loosely coupled	Camera-ready production; collecting camera ready; program preparation; composing the proceedings

2.2 The Architectural Design

The Architectural Design phase defines the most proper organizational structure for the MAS, i.e., the topology of interactions in the MAS and the control regime of these interactions, which most effectively enables the MAS goals to be fulfilled. The definition of the organizational structure has to account for a variety of factors, including the characteristics of the environment, the need of reflecting the structure of the real-world organization, the need of respecting and enforcing the organizational rules, as well as the obvious need to keep the design as simple as possible. In choosing the most appropriate organizational structure it is strongly recommended to exploit well-known organizational patterns. Once the most appropriate organizational structure is defined, the roles and interactions models identified in the analysis phase (which were preliminary, in that they were organizational-independent aspects) can be finalized, to account for all possibly newly identified roles and interactions. These activities result in two models: the Complete Role Model and Complete Interaction Model. The Complete Role and Complete Interaction models are strongly related to each other since any modification of one of the models directly affects the other model.

Figure 13 presents the Architectural Design process, while Fig. 14 presents the flow of activities, the involved roles, and the work products.

2.2.1 Process Roles

One role is involved in the Architectural Design: the Architectural Designer.

Architectural Designer

An Architectural Designer is responsible for:

- Identifying the best organizational structure according to a specific evaluation of organizational forces
- Completing role model according to the chosen organizational structure
- Completing interaction model according to the chosen organizational structure and the role model
- Categorizing roles and protocols

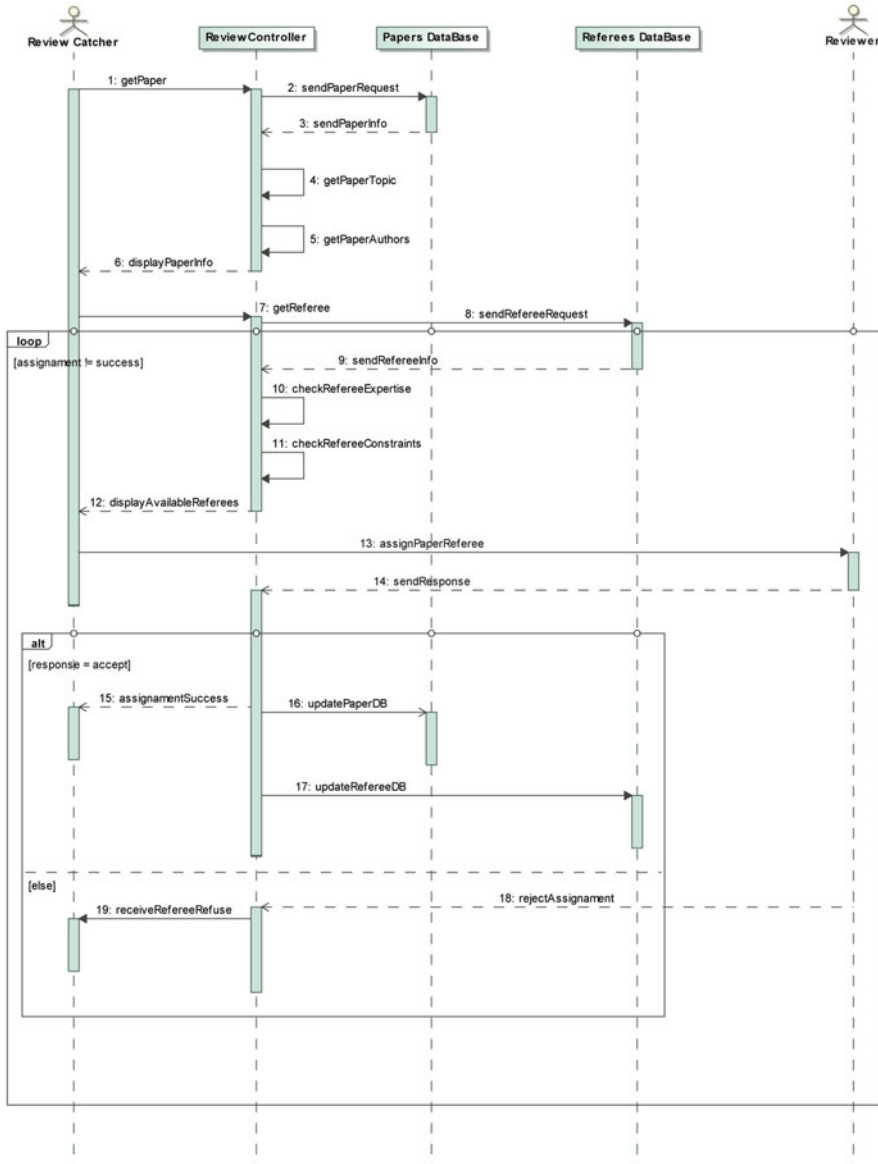


Fig. 12 The *AssignPaperReferee* interaction protocol

2.2.2 Activity Details

Organizational Structure Activity

The flow of tasks in the *Organizational Structure* activity is reported in Fig. 15; the tasks are detailed in the following table.

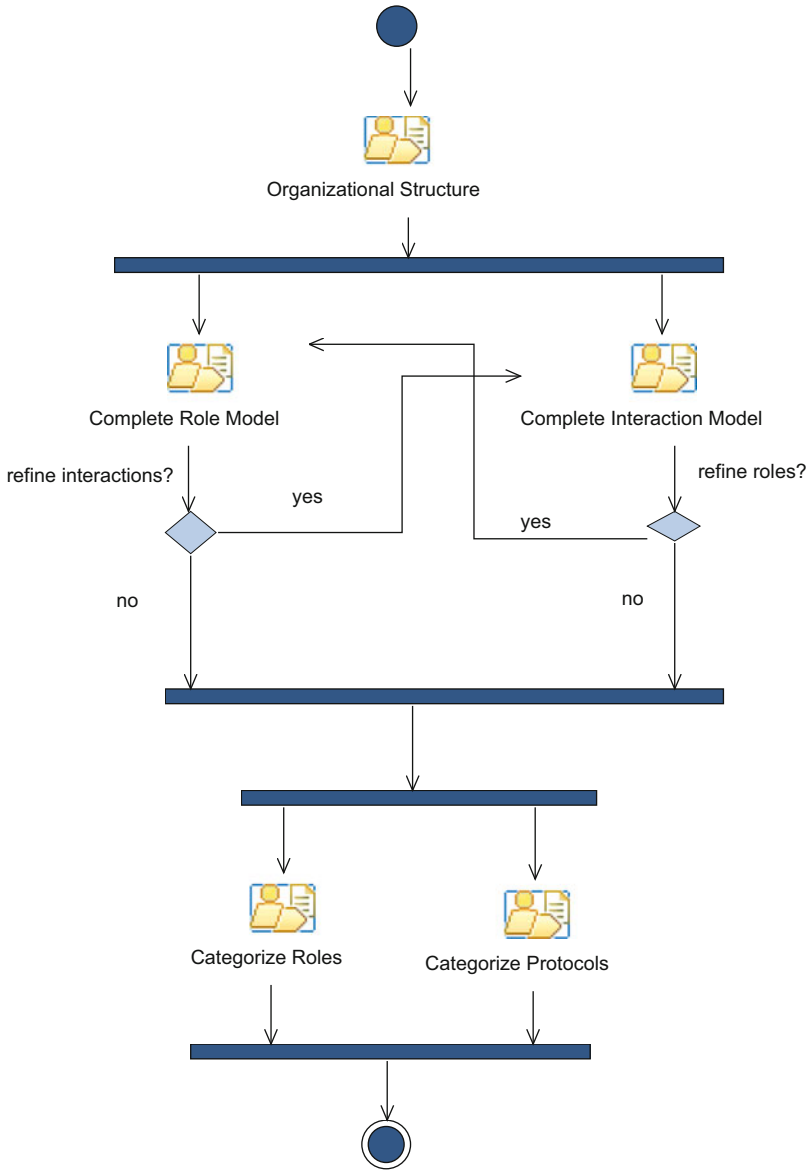


Fig. 13 The Architectural Design process

Complete Role Model Activity

The flow of tasks in the *Complete Role Model* activity is reported in Fig. 16; the tasks are detailed in the following table.

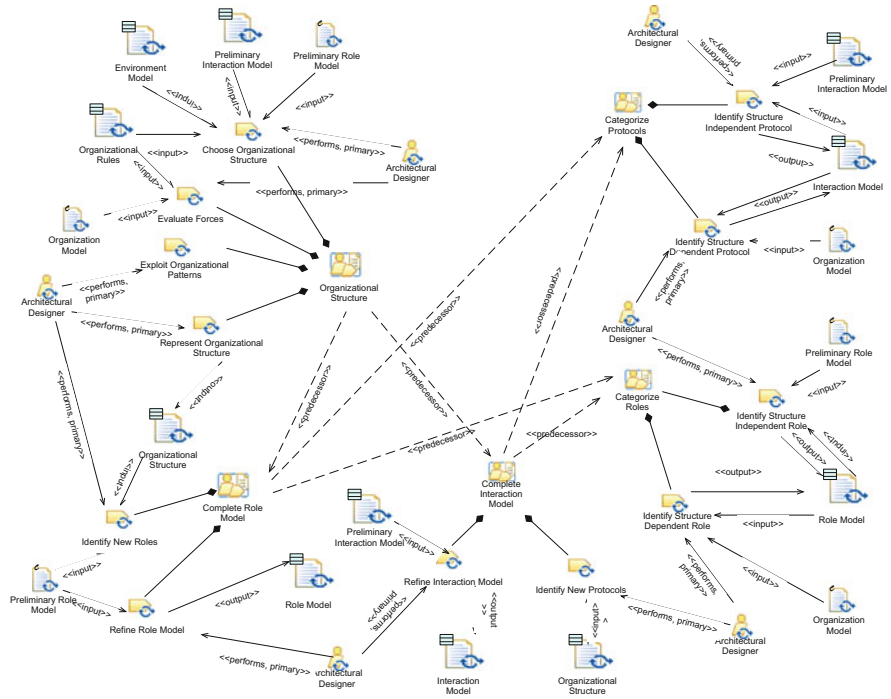


Fig. 14 The Architectural Design flow of activities, roles, and work products

Activity	Task	Task Description	Role Involved
Organizational structure	Evaluate forces	<i>considering forces like the need to achieve organizational efficiency, the need to respect organizational rules (i.e., control regime), and the need to minimize the distance from the real-world organization</i>	Architectural designer (<i>perform</i>)
Organizational structure	Choose organizational structure	<i>identifying an efficient and reliable way to structure the MAS organization (i.e., topology)</i>	Architectural designer (<i>perform</i>)
Organizational structure	Exploit organizational patterns	<i>analyzing and possibly adopting well-documented patterns that already demonstrated how and when to exploit a specific organizational structure</i>	Architectural designer (<i>perform</i>)
Organizational structure	Represent organizational structure	<i>using a diagram to represent the topology of the MAS organization</i>	Architectural designer (<i>perform</i>)

Complete Interaction Model Activity

The flow of tasks in the *Complete Interaction Model* activity is reported in Fig. 17; the tasks are detailed in the following table.

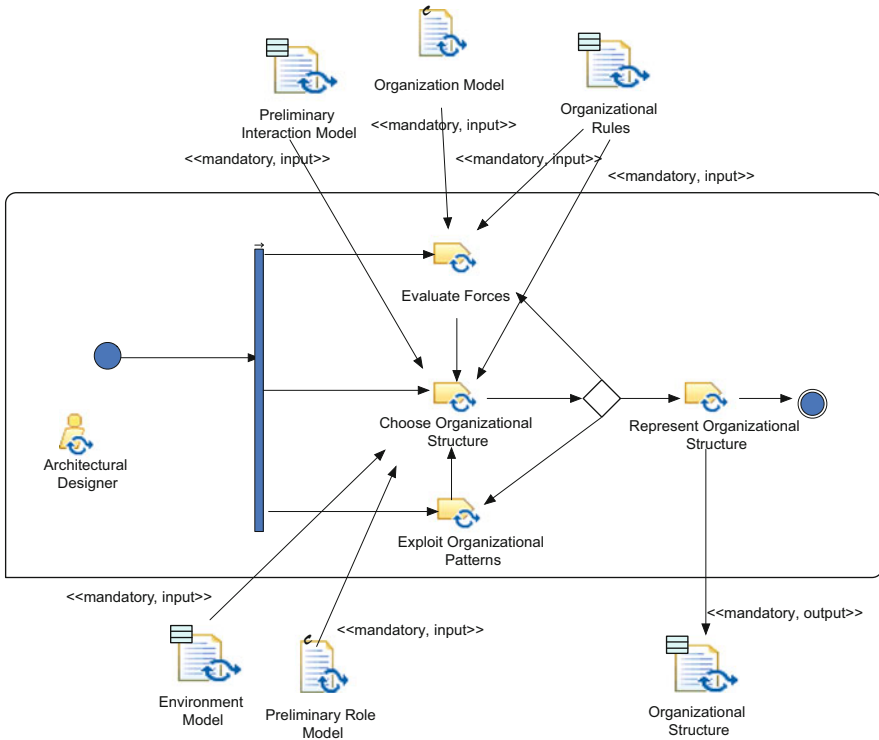


Fig. 15 The flow of tasks in the Organizational Structure activity

Activity	Task	Task Description	Role Involved
Complete Role Model	Identify new roles	<i>identification of new roles deriving directly from the adoption of a given organizational structure (i.e., organizational roles)</i>	Architectural designer (<i>perform</i>)
Complete Role Model	Refine role model	<i>the definitive identification of activities and services of the roles, including the new organizational roles</i>	Architectural designer (<i>perform</i>)

Activity	Task	Task Description	Role Involved
Complete Interaction Model	Identify new protocols	<i>identification of new protocols deriving from the adopted organizational structure (i.e., organizational protocols)</i>	Architectural designer (<i>perform</i>)
Complete Interaction Model	Refine interaction model	<i>completing the definition of the protocols required by the MAS, by specifying which roles the protocol will involve</i>	Architectural designer (<i>perform</i>)

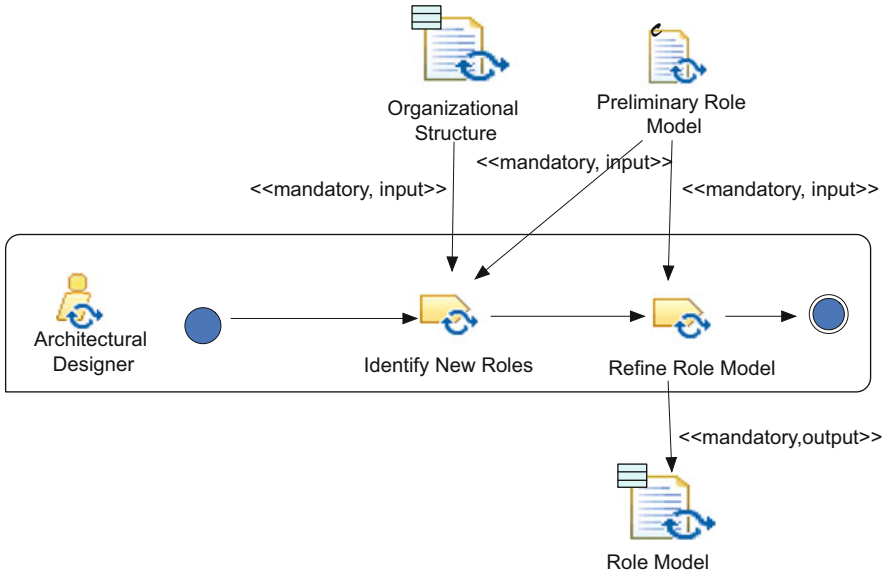


Fig. 16 The flow of tasks in the Complete Role Model activity

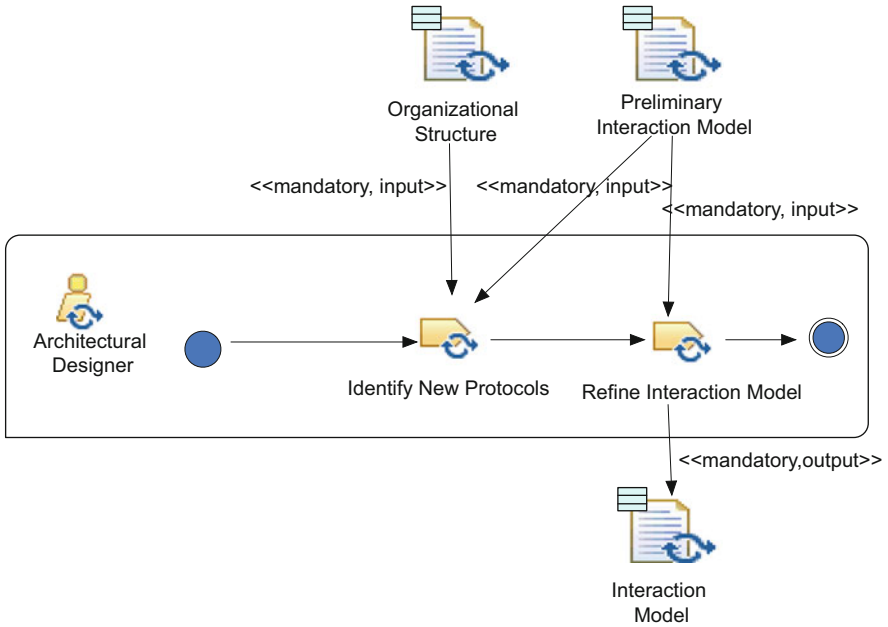


Fig. 17 The flow of tasks in the Complete Interaction Model activity

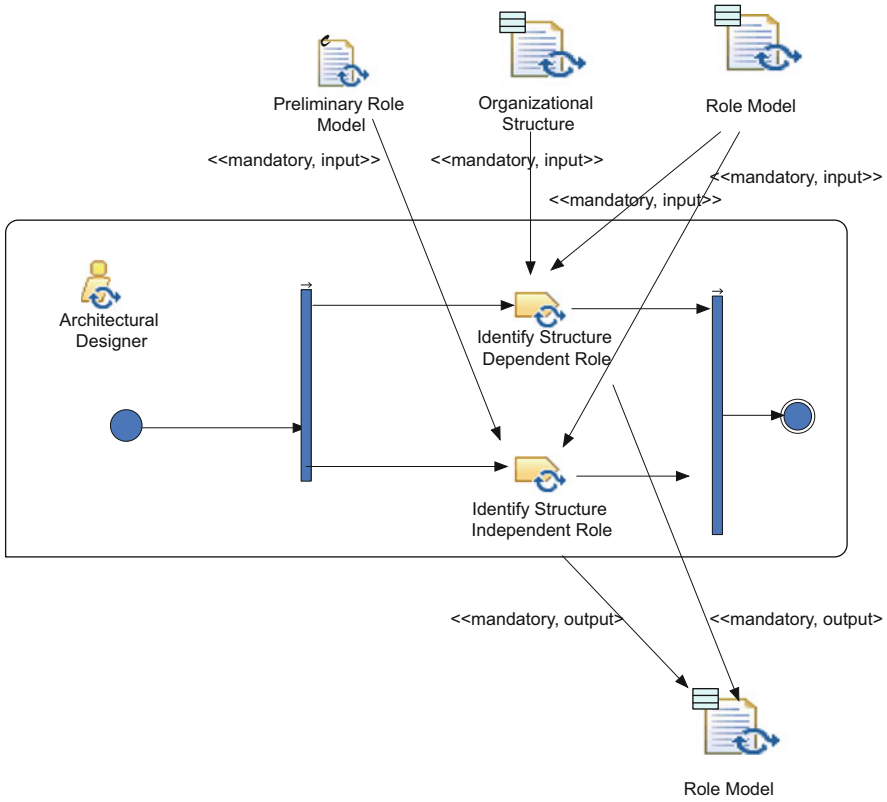


Fig. 18 The flow of tasks in the Categorize Role activity

Categorize Role Activity

The flow of tasks in the *Categorize Role* activity is reported in Fig. 18; the tasks are detailed in the following table.

Activity	Task	Task Description	Role Involved
Categorize Role	Identify structure dependent role	<i>specify which roles are deriving from the adopted organizational structure (i.e., organizational roles)</i>	Architectural designer (<i>perform</i>)
Categorize Role	Identify structure independent role	<i>specify which roles represent basic skills of the MAS that are independent from the adopted organizational structure</i>	Architectural designer (<i>perform</i>)

Categorize Protocols Activity

The flow of tasks in the *Categorize Protocols* activity is reported in Fig. 19; the tasks are detailed in the following table.

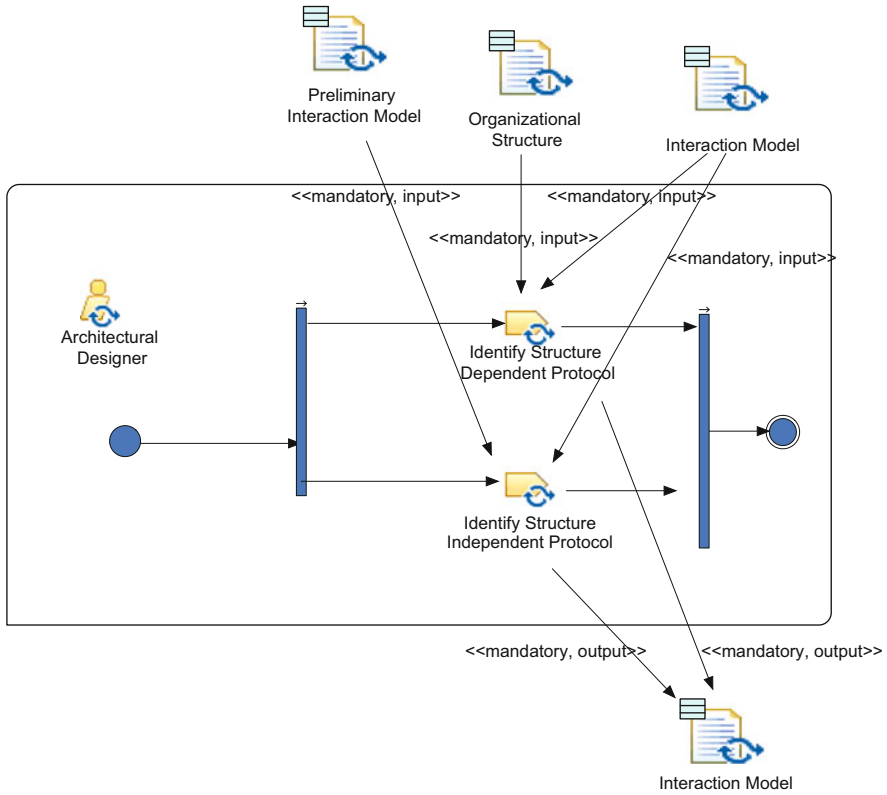


Fig. 19 The flow of tasks in the Categorize Protocol activity

Activity	Task	Task Description	Role Involved
Categorize Protocol	Identify structure dependent protocol	<i>specify which protocols are deriving from the adopted organizational structure (i.e., organizational protocols)</i>	Architectural designer (<i>perform</i>)
Categorize Protocol	Identify structure independent protocol	<i>specify which protocols are independent from the adopted organizational structure</i>	Architectural designer (<i>perform</i>)

2.2.3 Work Products

Figure 20 reports the relationships among the work products of this phase and the MMEs of the Architectural Design phase.

Kinds of Work Products

Table 5 describes all the work products of the Architectural Design.

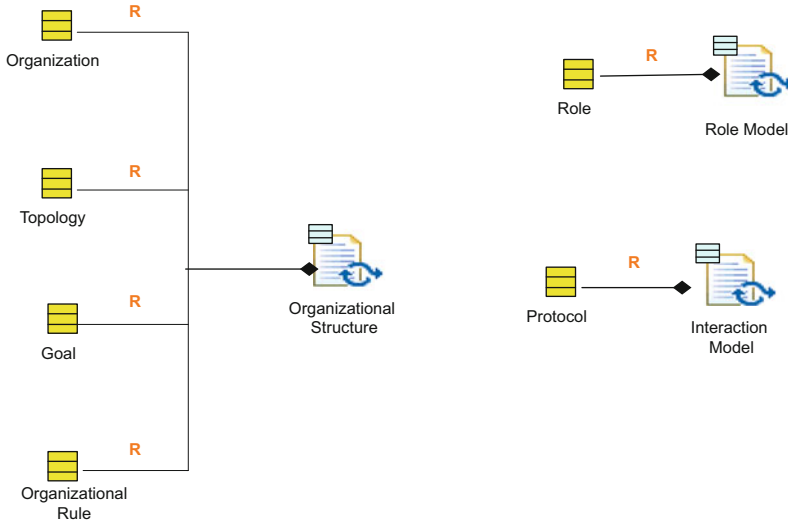


Fig. 20 The Architectural Design work products

Table 5 Architectural Design work products kinds

Name	Description	Work Product Kind
Organizational structure	<i>Diagram representing the topology of the MAS organization</i>	Structural
Role model	<i>The definitive specification of activities and services of the basic roles and the organizational roles (depending on the adopted organizational structure)</i>	Structural
Interaction model	<i>The definitive specification of the protocols (and the corresponding roles involved) required by the MAS</i>	Structural

Organizational Structure

Figure 21 presents an example of the Organizational Structure for the conference management.

Complete Role Model

Table 6 presents an example of the Complete Role Model for the conference management case study.

Complete Interaction Model

Figure 22 presents an example of the Complete Interaction Model for the conference management case study.

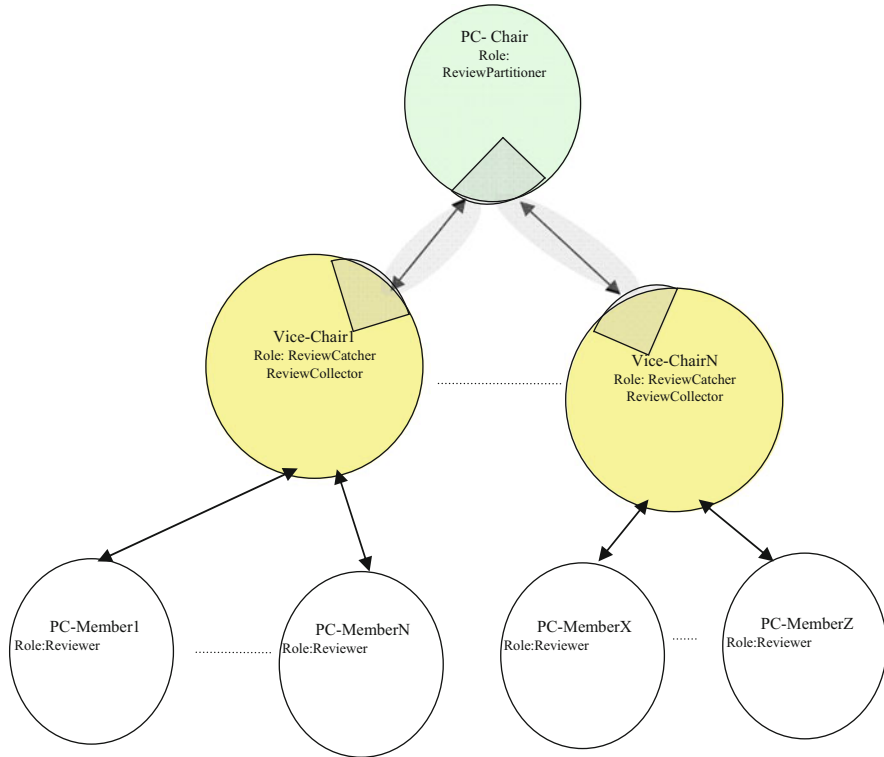


Fig. 21 Organizational Structure in the conference management case study

2.3 The Detailed Design

Past the Architectural Design phase, the Detailed Design involves identifying: (1) an agent model, i.e., the set of agent classes in the MAS, implementing the identified roles, and the specific instances of these classes; and (2) a services model, expressing services (a service is a single, coherent block of activity in which an agent will engage) and interaction protocols to be provided within agent classes. An agent class could package a number of closely related roles for the purposes of convenience or efficiency. It is worth remembering that the Detailed Design models avoid any specification oriented to the implementation in a specific technology, leaving this issue to the programmer. Figure 23 presents the Detailed Design process, while Fig. 24 presents the flow of activities, the involved roles, and the work products.

2.3.1 Process Roles

Detailed Designer

Detailed Designer is responsible for:

- identifying agent-types and their cardinalities
- identifying services
- specifying actions

Table 6 The ReviewPartitioner organizational role schema

Role name: ReviewPartitioner		
Description: This role is in charge of distributing papers among Vice-Chairs according the area of competence.		
Category: organizational role		
Protocol and activities: <u>GetPaper</u> , <u>CheckPaperTopic</u> , <u>CheckViceChairArea</u> , <u>AssignPaperViceChair</u> , <u>UpdateDBSubmission</u> .		
Permissions:		
Reads	<i>paper_submitted</i> <i>Vice-Chair-data</i>	in order to check the topic and authors in order to check the area
Changes	<i>DB Submission</i>	assigning the paper to a Vice-Chair area
Responsibilities:		
Liveness:	ReviewPartitioner = (GetPaper.CheckPaperTopic.CheckViceChairArea. AssignPaperViceChair.UpdateDBSubmission) ^W	
Safety:	\forall paper assigned to a ViceChairArea	

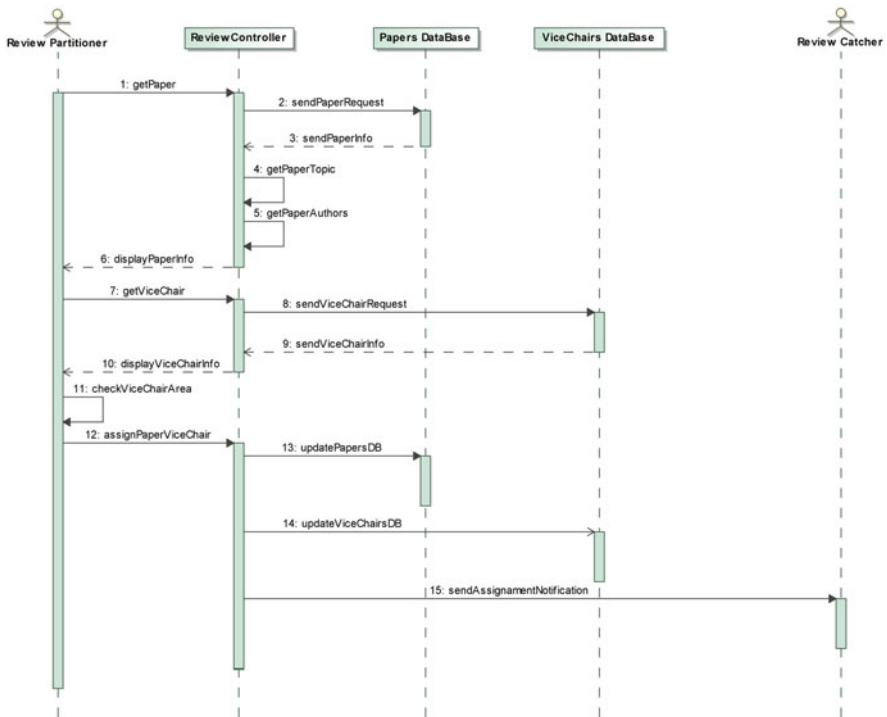


Fig. 22 The *ReceivePaperAssignment* interaction protocol

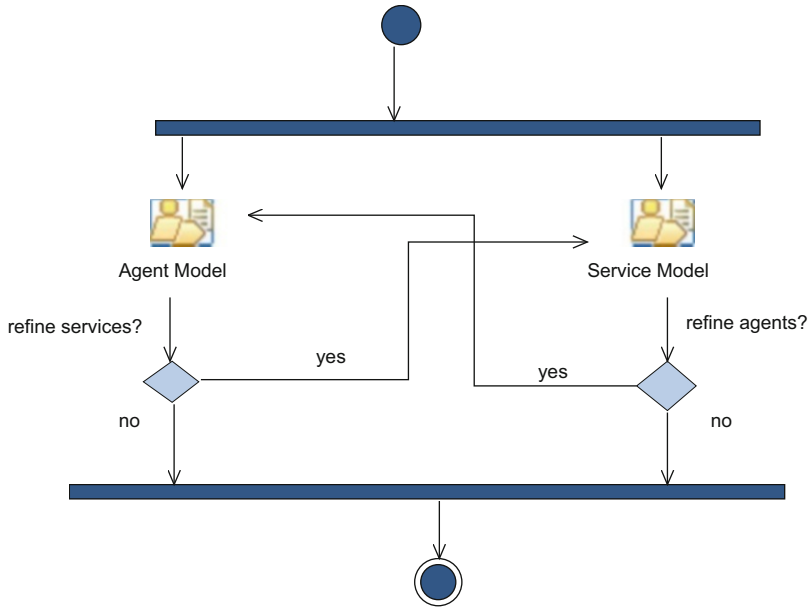


Fig. 23 The Detailed Design process

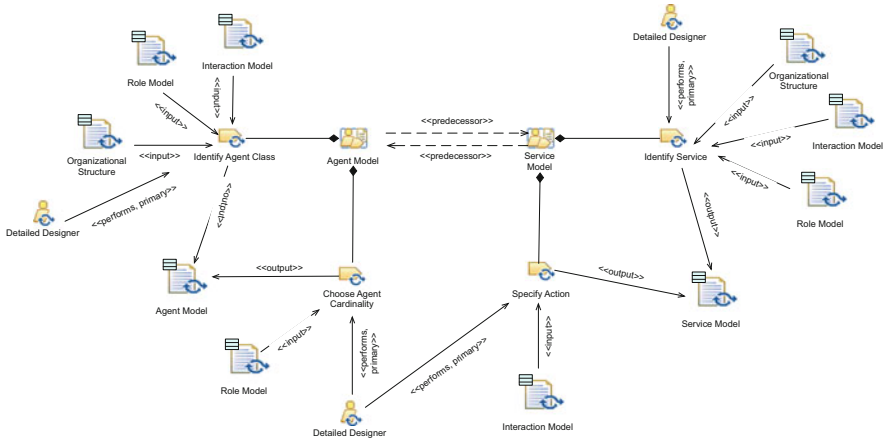


Fig. 24 The Detailed Design flow of activities, roles, and work products

2.3.2 Activity Details

Agent Model Activity

The flow of tasks in the *Agent Model* activity is reported in Fig. 25; the tasks are detailed in the following table.

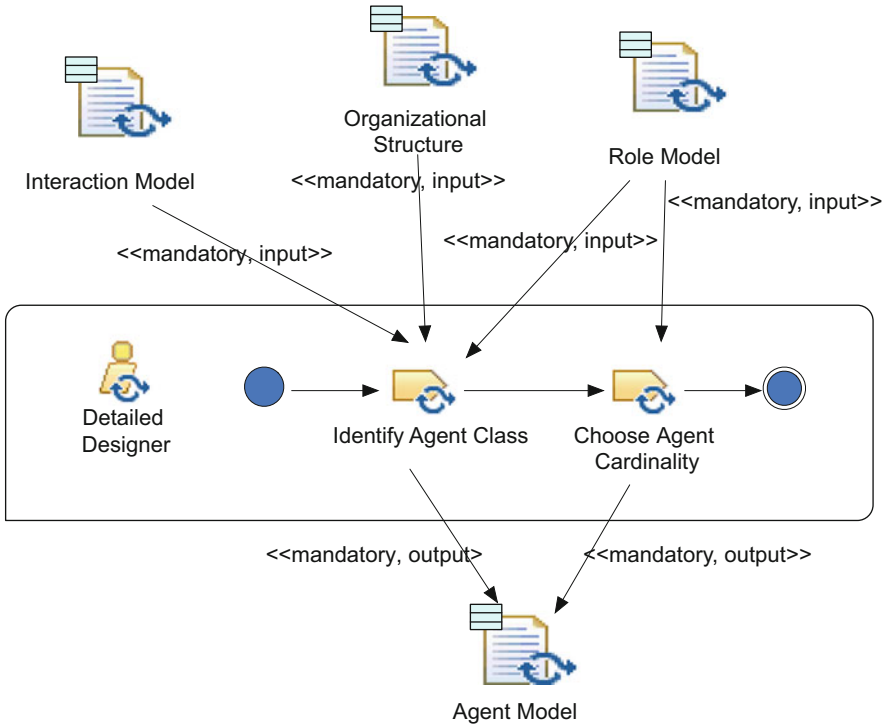


Fig. 25 The flow of tasks in the Agent Model activity

Activity	Task	Task Description	Role Involved
Agent Model	Identify agent class	<i>identification of a set of pro-active, autonomous entities (i.e., agents), each of them playing one or more roles, and the instances of each class that will appear in the MAS</i>	Detailed designer (perform)
Agent Model	Choose agent Cardinality	<i>specifying how many instances of each class will appear in the MAS using the cardinality standard</i>	Detailed designer (perform)

Service Model Activity

The flow of tasks in the *Service Model* activity is reported in Fig. 26; the tasks are detailed in the following table.

2.3.3 Work Products

Figure 27 reports the relationships among the work products of this step and the MMEs of the Detailed Design step.

Kinds of Work Products

Table 7 describes all the work products of the Detailed Design.

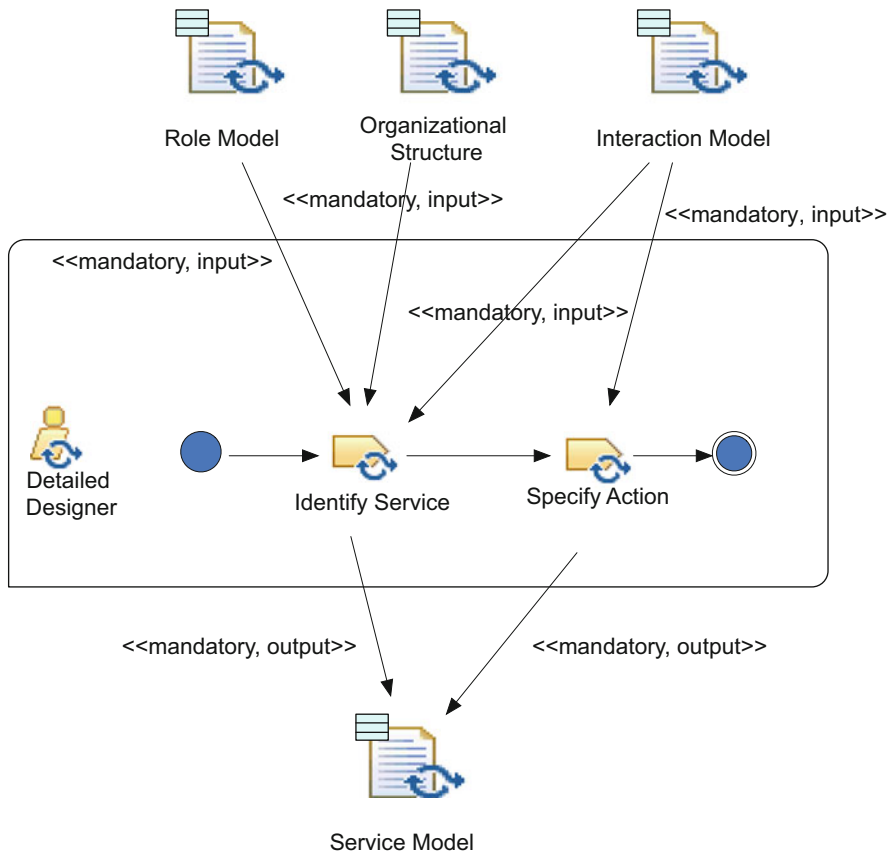


Fig. 26 The flow of tasks in the Service Model activity

Activity	Task	Task Description	Role Involved
Service Model	Identify service	<i>identify the services associated with each agent class with their inputs, outputs, pre-conditions, and post-conditions</i>	Detailed designer (perform)
Service Model	Specify action	<i>each service specifies a single, coherent block of activity in which an agent will be engaged</i>	Detailed designer (perform)

Agent Model

Figure 28 presents an example of the Agent Model for the conference management case study.

Service Model

Table 8 presents an example of the Service Model for the conference management case study.

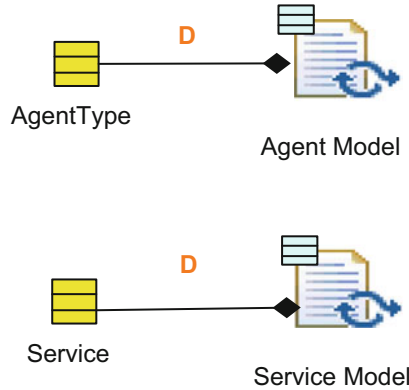


Fig. 27 The Detailed Design work products

Table 7 Detailed Design work products kinds

Name	Description	Work Product Kind
Agent Model	Diagram of the agents and the roles they are playing	Structural
Service Model	Specification of the actions associated with each agent class	Structural

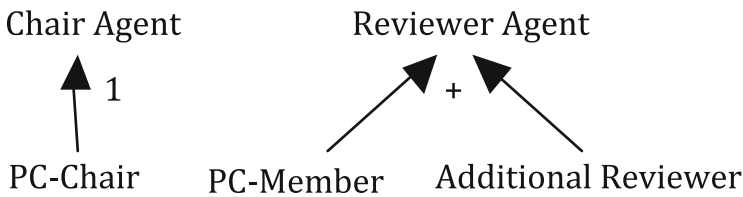


Fig. 28 An example of the Agent Model

Table 8 An example of the Service Model

Service	Input	Output	Pre-conditions	Post-conditions
Accept/Decline papers assignment	Request for reviewing papers	Papers to be reviewed	Papers are available for reading (see Environmental Model)	List of papers accepted for reviewing
Accepted review	Assigned paper	Review form fulfilled	Paper is available for reading. Agent has not already reviewed the maximum allowed number of papers	The Review Form has been correctly completed

3 Work Products Dependencies

The diagram in Fig.29 describes the dependencies among the different work products.

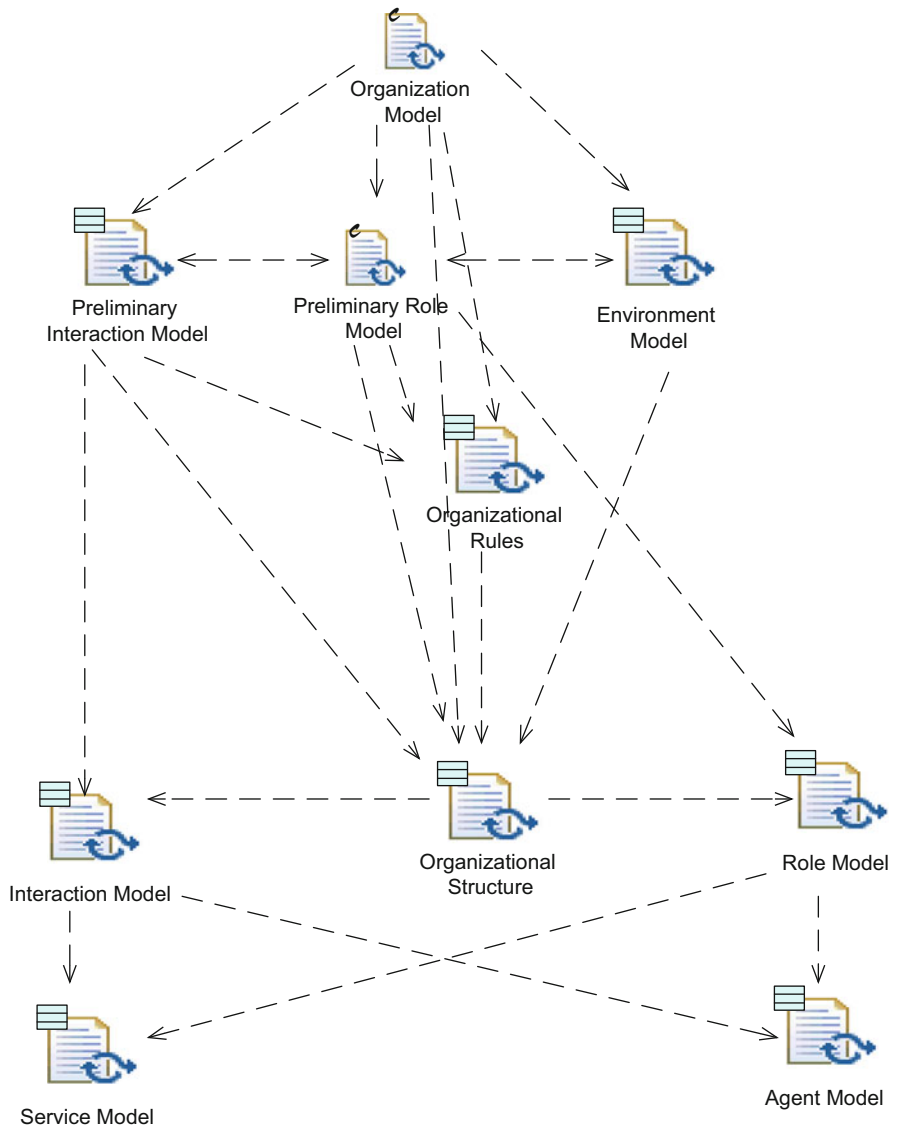


Fig. 29 The work products dependencies

Acknowledgment This work has been partially supported by the EU-FP7-FET Proactive project SAPERE—Self-aware Pervasive Service Ecosystems, under contract no. 256873.

References

1. Cernuzzi, L., Zambonelli, F.: Experiencing AUML in the Gaia methodology. In: 6th International Conference on Enterprise Information Systems (ICEIS 2004), Porto, Portugal, vol. 3, pp. 283–288 (2004)
2. Cernuzzi, L., Zambonelli, F.: Adaptive organizational changes in agent-oriented methodologies. *Knowl. Eng. Rev.* **26**(2), 175–190 (2011). doi:10.1017/S0269888911000014
3. Cernuzzi, L., Juan, T., Sterling, L., Zambonelli, F.: The Gaia methodology. In: Bergenti, F., Gleizes, M.P., Zambonelli, F. (eds.) *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Multiagent Systems, Artificial Societies, and Simulated Organization, Chap. 4, vol. 11, pp. 69–88. Kluwer, Boston (2004). doi:10.1007/1-4020-8058-1_6
4. Cernuzzi, L., Molesini, A., Omicini, A., Zambonelli, F.: Adaptable multi-agent systems: the case of the Gaia methodology. *Int. J. Softw. Eng. Knowl. Eng.* **21**(4), 491–521 (2011). doi:10.1142/S0218194011005384
5. Wooldridge, M.J., Jennings, N.R., Kinny, D.: A methodology for agent-oriented analysis and design. In: Etzioni, O., Müller, J.P., Bradshaw, J.M. (eds.) *3rd Annual Conference on Autonomous Agents (AGENTS'99)*, pp. 69–76. ACM, New York (1999). doi:10.1145/301136.301165
6. Wooldridge, M.J., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. *Auton. Agent Multi Agent Syst.* **3**(3), 285–312 (2000). doi:10.1023/A:1010071910869
7. Zambonelli, F., Jennings, N.R., Omicini, A., Wooldridge, M.J.: Agent-oriented software engineering for Internet applications. In: Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R. (eds.) *Coordination of Internet Agents: Models, Technologies, and Applications*, Chap. 13, pp. 326–346. Springer, Berlin (2001)
8. Zambonelli, F., Jennings, N.R., Wooldridge, M.J.: Developing multiagent systems: the Gaia methodology. *ACM Trans. Softw. Eng. Methodol.* **12**(3), 317–370 (2003). doi:10.1145/958961.958963
9. Zambonelli, F., Jennings, N., Wooldridge, M.J.: Multiagent systems as computational organizations: the Gaia methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) *Agent Oriented Methodologies*, Chap. 6, pp. 136–171. Idea Group Publishing, Hershey (2005). doi:10.4018/978-1-59140-581-8.ch006

GORMAS: A Methodological Guideline for Organizational-Oriented Open MAS

Sergio Esparcia, Estefanía Argente, Vicente Julián, and Vicente Botti

Abstract

The design of Organization-Oriented Multi-Agent Systems requires methodologies that use the agent organization as a key concept, defining concepts such as organizational goals, roles, norms, organizational services, etc., in an explicit way. GORMAS (Guidelines for Organizational Multi-agent Systems) is an Agent-Oriented Software Engineering methodology to develop Virtual Organizations that allows describing and designing Organization-Oriented Multi-Agent Systems. This methodology is composed of four phases: Mission Analysis, Service Analysis, Organizational Design and Organization Dynamics Design. Moreover, it is supported by a specific metamodel, named Virtual Organization Model, which defines five organizational dimensions, by which an agent organization can be specified: Structural, Functional, Dynamical, Environment and Normative. This chapter describes the four phases of the GORMAS methodology, following the template developed within the FIPA Design Process Documentation and Fragmentation working group. Thus, the activities developed on each phase, and the products generated during the whole process are detailed.

1 Introduction

Organization-Centered Multi-agent Systems (OCMAS) [1, 2] are a specific type of MAS where the organization is explicitly defined, and the agents populating it are focused on achieving the organizational goals. Organizations describe system functionality, structure, environment and dynamics, and an OCMAS is defined following a top-down approach (i.e., first, the organization is defined, and then the

S. Esparcia • E. Argente (✉) • V. Julián • V. Botti
Dept. Sistemas Informáticos y Computación, Universitat Politècnica de València, Grupo de Tecnología Informática - Inteligencia Artificial, Camino de Vera, s/n 46022, Valencia, Spain
e-mail: sesparcia@dsic.upv.es; eargente@dsic.upv.es; vinglada@dsic.upv.es; vbotti@dsic.upv.es

agents that will populate it). OCMAS are open, so external agents coming from other organizations or the environment will enter into the organization. Thus, it is necessary to introduce regulative elements such as norms to avoid interested or malicious behavior by these agents.

Therefore, it is required to be provided with an Agent-Oriented Software Engineering (AOSE) methodology to help designers and developers to define OCMAS. The Guidelines for Organizational Multi-agent Systems (GORMAS) methodology is focused on designing large scale, open, service-oriented OCMAS, where organizations are able to accept external agents into them. GORMAS is based on a specific method for designing human organizations [3], which consists on diverse phases for analysis and design: Mission Analysis, Service Analysis, Organizational Design and Organizational Dynamics Design. These phases have been appropriately adapted to the MAS field, this way to catch all the requirements of the design of an organization from agents' perspective. Thus, the methodological guidelines proposed in GORMAS cover the common requirement analysis, architectural and detailed designs of many relevant AOSE methodologies (such as PASSI [4], SODA [5], and INGENIAS [6]), but it also includes a deeper analysis of the system as an open organization that provides services to its environment.

GORMAS is supported by a CASE tool named EMFGormas [7], which uses the MDA Eclipse Technology. This technology requires defining a platform independent unified metamodel that describes the modeling language in a formal way, establishing the primitives and syntactic-semantic properties of organizations and MAS. The tool offers several graphical editors, one for each view of the model, but diagrams are stored in a unique model. Thus, GORMAS is supported by the Virtual Organization Model (VOM) [8], a metamodel that describes a Virtual Organization by means of five dimensions: *Structural*, *Functional*, *Dynamical*, *Environment* and *Normative Dimensions*.

GORMAS methodological guideline can be integrated into a complete software development process, which may include the analysis, design, implementation, deployment and maintenance phases of a MAS. Implementation of GORMAS designs of several cases of study and actual problems have been carried out using the THOMAS framework [9].

In this chapter, GORMAS methodology will be described using the template [10] proposed by the FIPA Design Process Documentation and Fragmentation working group (using SPEM 2.0 notation [11]), by initially considering its whole process and its metamodel, and then its four phases, describing the roles, activity details and work products involved on each phase. All phases are exemplified by means of a common case of study based on a Conference Management System (CMS), which is a real-life domain that can be easily translated into MAS domain. Then, this definition will be used to identify and extract the fragments that compose GORMAS. The rest of this chapter is structured as follows: Sect. 1.1 gives an overview on the GORMAS process life cycle and details VOM. Section 2 details all phases followed by GORMAS.

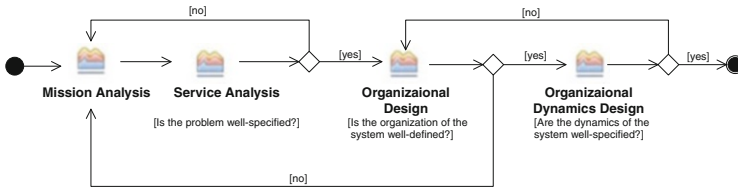


Fig. 1 GORMAS life cycle design process

Some interesting GORMAS references are as follows:

- E. Argente. *GORMAS: Guías para el desarrollo de Sistemas multi-agente abiertos basados en organizaciones*. PhD thesis, Universitat Politècnica de València, 2008.
- E. Argente, V. Julian, and V. Botti. *MAS Modelling based on Organizations*. In Proc. Agent Oriented Software Engineering 2008, pp. 16–30, 2008.
- E. Argente, V. Botti and V. Julian. *GORMAS: An Organizational-Oriented Methodological Guideline for Open MAS*. In Proc. Agent Oriented Software Engineering 2009, pp. 85–96, 2009.
- E. Argente, V. Botti, V. Julian. *GORMAS: An Organizational-Oriented Methodological Guideline for Open MAS*. In: Agent-Oriented Software Engineering X, pp. 32–47, Springer, Berlin (2011).

1.1 The GORMAS Process Life Cycle

GORMAS follows a four-phase, iterative life cycle (see Fig. 1):

- *Mission Analysis*: Describes the global goals of the system, the services and products that the system provides to other entities, the stakeholders and the conditions of the environment.
- *Service Analysis*: Describes the type of products and services, and the tasks and goals related to its production; the resources and applications needed for offering the system functionality and the roles related with the stakeholders.
- *Organizational Design*: Organizational dimensions are defined and they are employed to define a suitable structure for the organization.
- *Organization Dynamics Design*: Communication between agents, processes concerning roles and agents, mechanisms for the control of the system (like norms) and a reward system, are defined.

On each phase, one or some of the diagrams that represent the five dimensions of the VOM (that will be described in the next subsection) are updated. Additionally, some documents or guidelines for describing the environment conditions, system mission, stakeholders, services and products, are fulfilled in the first two phases of the methodology and are lately employed throughout the whole process. All four phases will be detailed in Sect. 2.

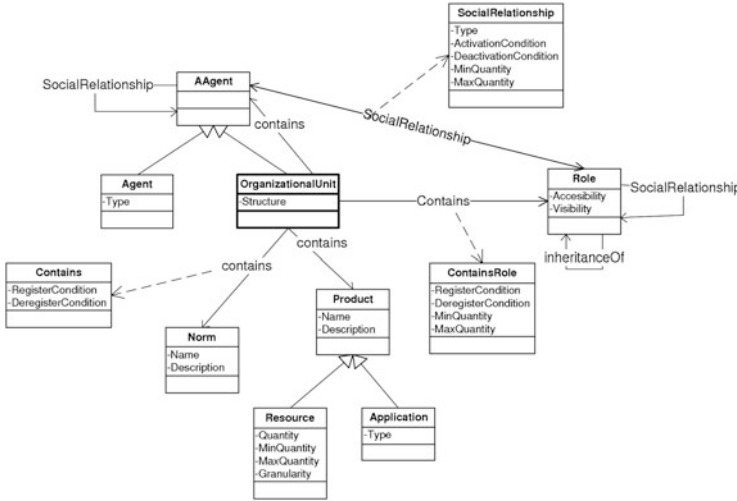


Fig. 2 Structural Dimension metamodel of VOM

1.1.1 The GORMAS MAS Metamodel

This section describes the metamodel that supports GORMAS, named VOM [8]. Organizations are structured by means of Organizational Units (OU), which represent a set of agents that carry out some specific and differentiated activities or tasks, following a predefined pattern of cooperation and communication. An OU is made up of different entities along its life cycle that can be both single agents or other Organizational Units, viewed as a single entity. OUs act as a link between the different GORMAS diagrams.

System entities are capable of offering and/or requesting services and their behavior is motivated by their pursued goals. Services represent the functionality that agents offer to other entities, independently from the specific agent that makes use of it. Moreover, an OU can also publish its service requirements, so then external agents can decide whether to participate inside, providing these services.

The VOM is made up of five dimensions: Structural, Functional, Dynamical, Environment and Normative Dimensions.

The *Structural Dimension* (Fig. 2) describes the components of the system and their relationships. It allows defining all elements that are independent from the final executed entities. More specifically, it defines:

- The entities of the system (*AAgent*), so that an entity represents an atomic entity (*Agent*) or a group of members of the organization (*Organizational Unit*), seen as a unique entity from outside.
- The *Organizational Units* (OUs) of the system, which can also include other units in a recursive way, as well as single agents.
- The *Roles* defined inside the OUs. In the *contains* relationship, a minimum and maximum quantity of entities that can acquire this role can be specified. For each role, the *Accessibility* attribute indicates whether a role can be adopted by an

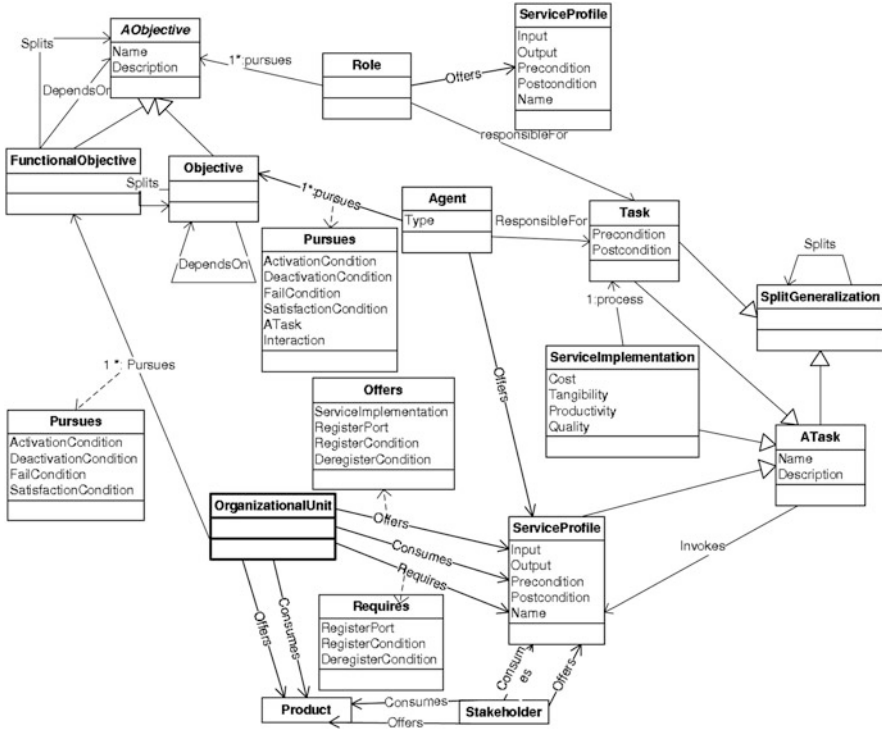


Fig. 3 Functional Dimension metamodel of VOM

entity on demand (external) or it is always predefined by design (internal). The *Visibility* attribute indicates whether entities can obtain information from this role on demand, from outside the OU (public role) or from inside, once they are already members of this OU (i.e., private role). A hierarchy of roles can also be defined using the *InheritanceOf* relationship.

- The organizational social relationships (*SocialRelationship*). The type of a social relationship between two entities is related to their position in the structure of the organization (i.e., information, monitoring, supervision), but other types are also possible. Moreover, a condition on when this social relationship is active or not, and the number of minimum and maximum allowed relationships is established.
- The *products* (resources/applications) available to the OU. They are the tangible results of the organization, which will be consumed by its members or its clients.
- The *norms* that control the global behavior of the members of the OU.
- All contains relationships include conditions for enabling a dynamical registration/deregistration of the elements of an OU through its lifetime.

The **Functional Dimension** (Fig. 3) details the specific functionality of the system, based on services, tasks and goals, as well as system interactions, activated by means of goals or service usage. It allows defining the functionality of the organizational units, roles and agents of the MAS. More specifically, it defines:

- The functionality of the Organizational Units:
 - The *functional objectives* that are pursued by the organization, i.e., non-functional requirements (softgoals) that can be defined for describing the global behavior of this organization.
 - The *stakeholders* that interact with the OU.
 - The results that the organization offers (*products* and *services*, which are described using service profiles) to those stakeholders. In case of services, a specific implementation of the service can be defined in the *offers* relationship (*ServiceImplementation* attribute), which may be registered in a service directory (*RegisterPort* attribute), so then other entities can find it. Conditions for controlling this registration process can also be specified (*RegisterCondition* and *DeregisterCondition* attributes).
 - The services that are required by the organization (*Require* relationship). This “requires” relationship is similar to job offer advertising in human organizations, in the sense that it represents a necessity of finding agents capable of providing these required services as members of the organization.
 - The organization needs from its providers (*Consumes* relationship).
- The composition of goals:
 - The *AObjective* components, which can be *functional objectives* (i.e., softgoals or non-functional requirements) or operational goals (i.e., hardgoals or *objective*).
 - The *Functional objectives* represent the expected results of organizational units, which are split into the specific and measurable results that their members are expected to achieve.
 - The *Objectives* represent operational goals, that is, specific goals that agents have to fulfill. They can be refined into more specific objectives. They might be related with a Task or Interaction needed for satisfying this objective.
- The functionality of the Roles:
 - The *goals* (*AObjective*) pursued by a role (i.e., attached to a role), which can be *functional objectives* (i.e., softgoals or non-functional requirements) or operational goals (i.e., hardgoals or simply *objective*). In the *pursues* relationship, activation and deadline conditions can be defined to establish a temporal timeline in which the objective must be followed by the entity playing this role. In case of an operational goal (*objective*), a satisfaction or fail condition can be defined in order to establish when this objective has to be fulfilled, as well as a Task or Interaction that enables reaching this goal.
 - The services (*ServiceProfile*) related to the role, that is, the services that the role is assumed to offer or provide to other entities.
 - The *tasks* that the entities playing this role are assumed to be responsible for, that is, the specific functionality that an entity playing this role is expected to be able to carry out.
- The functionality of the Agents:
 - The *objectives* pursued by agents. Activation and deadline conditions can be defined to establish a temporal timeline in which the objective is followed. Moreover, a satisfaction or fail condition can be defined in order to establish when this objective has been fulfilled.

– The services (*ServiceProfile*) related to the agent, that is, the services that the agent might offer to other entities. When adopting a role as a member of an organization, the concrete set of services that the agent will be allowed to provide is determined by its own set of offered services and those ones related to the adopted role.

– The *tasks* that the agent is responsible for, that is, the set of tasks that the agent is capable of carrying out.

- The composition of tasks and services:

– The *Service* component describes the service functionality and represents both specific tasks, taskflows or service composition (*Invokes* relationship). This *Service* component can be split into other *Service* components, thus allowing service refinement or task composition.

– A *Task* represents a basic functionality that consumes and produces changes in the agent's Mental States.

– The *order* relationship between tasks, in which ordering conditions can be defined, as well as interactions. The entity *Condition* allows defining the sequence of tasks depending on a condition.

– The service interface (*ServiceProfile*), which indicates activation conditions of the service (preconditions), its input and output parameters and its effects over the environment (postconditions).

– The service specific functionality (*ServiceImplementation*), which describes a particular implementation of a service profile.

The ***Dynamical Dimension*** (Figs. 4 and 5) defines the role enactment process, the interactions between agents, as well as the mental states of the entities of the system. More specifically, it defines:

- The *roles* that the Organizational Unit may play inside other Organizational Units (*Plays* relationship) when considered as a unique entity. *ActivationCondition* and *LeaveCondition* attributes of this relationship indicate in which situation an OU acquires or leaves a role.
- The *roles* played by each agent. *ActivationCondition* and *LeaveCondition* attributes of this *play* relationship indicate in which situation an agent can acquire or leave a role.
- The *Mental States* of the agent, using believes (something that an agent, or a role taken by an agent, thinks that it is true or will happen), events (something that changes the state of the system when it occurs) and facts (something that it is true at the domain of the system).
- The sequence of interactions (Fig. 5):
 - The participants of the interaction (*Executer*). The *Initiates* and *Collaborates* relationships indicate the sequence of activities (tasks and services) that have been executed in an interaction. The *Collaborates* relationship represents a response activity.
 - The performatives (*InteractionUnit*) employed during the interaction.
 - The entity *Condition* allows changing the sequence of interactions depending on a condition.

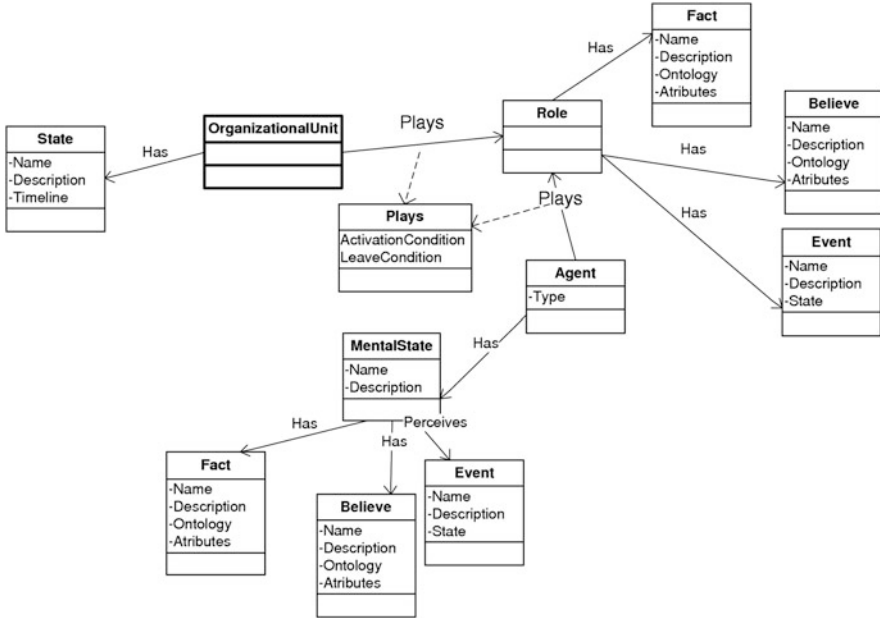


Fig. 4 Dynamical Dimension metamodel of VOM (I)

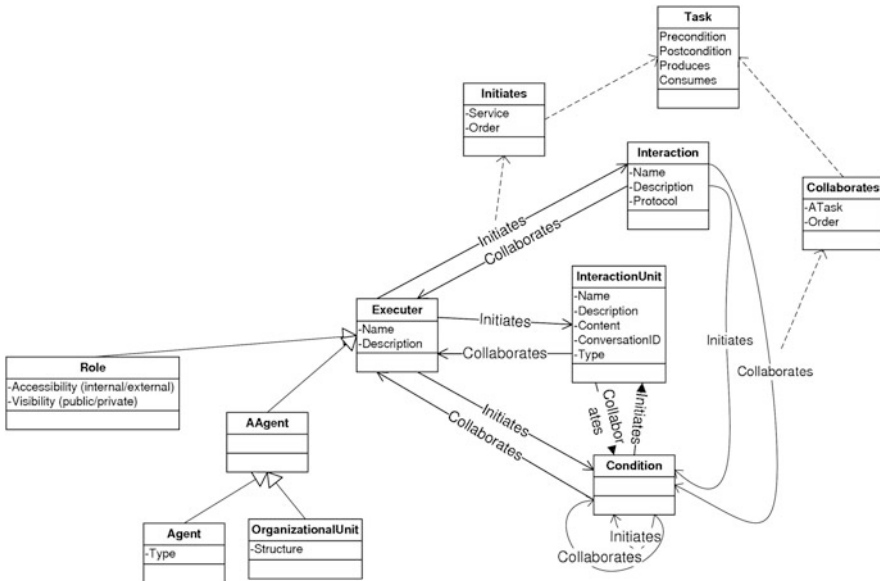


Fig. 5 Dynamical Dimension metamodel of VOM (II)

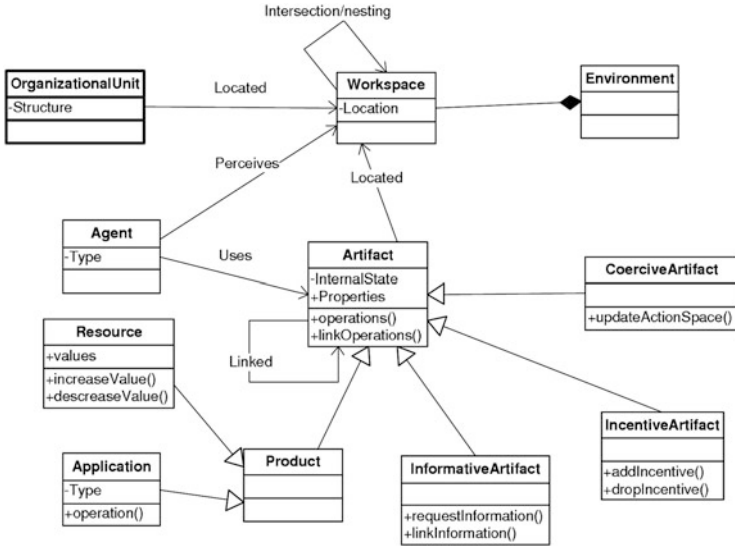


Fig. 6 Environment Dimension metamodel of VOM

The *Environment Dimension* [12] (Fig. 6) describes the environment of a system by means of the Agents & Artifacts (A&A) conceptual framework [13]. In this approach, the environment is structured by means of workspaces, where organizations are located, and each workspace has a set of artifacts, which are passive entities that agents use in order to achieve their objectives. More specifically, it defines:

- The *workspaces* that structure the environment, where organizational units are located. Agents are able to perceive a set of workspaces, defining the visibility of the organization that they have. The combination of all the workspaces defines the *environment* of the organization.
- The *artifacts*, which are the passive entities that agents will use in order to achieve their objectives. An artifact has a internal state, a set of properties and a set of operations and link operations that define its functionality. They encapsulate the functionalities of products, resources and applications defined in other models, and they are divided into:
 - *Informative artifacts*, which provide information to an agent, based on the internal state of this agent and the partial view of the environment that the artifact has.
 - *Incentive artifacts*, which modify the global behavior of the system by changing the reward system of the MAS.
 - *Coercive artifacts*, which update the actions an agent is able to execute.

Finally, the *Normative Dimension* (Fig. 7) describes normative restrictions over the behavior of the system entities, including organizational norms and normative goals that agents must follow. It defines:

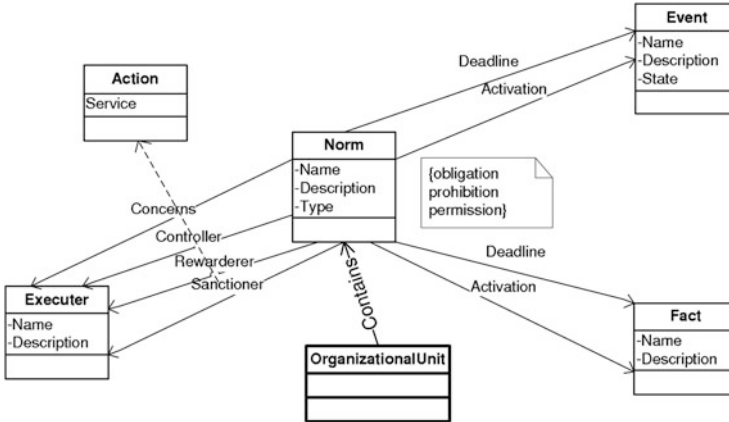


Fig. 7 Normative Dimension diagram

- The *Norm* concept, which represents a specific regulation. The properties of the norm detail all facts and events of the environment that provoke the activation or deactivation of the norm.
- The entity (*Executer*) to whom the norm is applied (using the *Concerns* relation).
- The *Executer* that is responsible of: monitoring the norm satisfaction (*Controller* relation); applying punishments (*Sanctioner* relationship); and/or applying rewards (*Rewarderer* relation).
- The *Service* attribute of all these relationships indicates which task or service will be invoked when monitoring this norm and when punishing or rewarding agents. Table 1 defines the elements of the GORMAS metamodel.

2 Phases of the GORMAS Methodology

In this section, the phases that compose the GORMAS methodology will be described. GORMAS offers a set of guidelines to analyze the requirements of the system, to design the structure of the organization and to design the dynamics of an organization for an Organization-Centered MAS. This set of guidelines is aimed to define the services provided by an organization, its structure and the norms governing the organization, making it easy to analyze and design OCMAS.

This section will follow the template made within the FIPA Design Process Documentation and Fragmentation working group. For each phase, a brief description of its scope will be given. Then, the process roles that are involved in this phase will be defined and activities and tasks from that phase will be described in detail. Finally, work products will be identified. Relationships between VOM elements and work products will be identified. In the following subsections, each step of the GORMAS process will be detailed.

Table 1 *Definition of MAS metamodel elements*

Concept	Definition	Domain
Organizational Unit (OU)	A set of agents that carry out some specific and differentiated activities or tasks by following a predefined pattern of cooperation and communication. An OU is formed by different entities along its life cycle that can be both single agents or other organizational units, viewed as a single entity.	Structural, Functional, Dynamical and Environment dimensions
AAgent	An entity of the system, which represents an atomic entity or a group of members of the organization, seen as unique entity from outside	Structural and Dynamical dimensions
Agent	An entity capable of perceiving and acting into an environment, communicating with other agents, providing and requesting services/resources and playing several roles.	Structural, Functional, Dynamical and Environment dimensions
Product	An entity that is contained into an OU or that belongs to an specific agent. It can be an application or a resource.	Structural and Functional dimensions
Resource	It is an environment object that does not provide a specific functionality, but is essential for task execution.	Structural and Environment dimension
Application	It is a functional interface that does not satisfy a rational criteria	Structural and Environment dimensions
Role	An entity representing a set of goals and obligations, defining the services that an Agent or an OU could provide and consume.	Structural, Functional and Environment dimensions
AObjective	It is a goal pursued by a role. An AObjective could be a functional objective or an operational goal.	Functional dimension
Functional Objective	A functional objective is a non-functional requirement (softgoals) that could be defined to describe the global behavior of the organization.	Functional dimension
Objective	An objective is a specific goal that agents or roles have to fulfill. It can be refined into specific objectives.	Functional dimension
Service Profile	It is the description of a service that the agent might offer to other entities	Functional dimension
Service Implementation	It is a service specific functionality which describes a concrete implementation of a service profile	Functional dimension
Service	An entity describing the service functionality and represents concrete tasks, taskflows or service composition	Functional dimension
Task	An entity that represents a basic functionality, which consumes resources and produces changes in the agent's Mental State.	Functional and Dynamical dimensions
Norm	It is a coordination mechanism and represents a specific regulation for the system.	Structural and Normative dimensions
Stakeholder	It is a group that the organization is oriented to and interacts with the OUs.	Functional dimension
Mental State	It is a set of believes, events and facts that define the current state of an agent.	Dynamical dimension
Believe	It is something that an agent (or a role taken by an agent) thinks that it is true or will happen.	Dynamical dimension
Fact	It is something that is true at the system's domain.	Dynamical and Normative dimensions
Event	It is something that changes the state of the system when it occurs.	Dynamical and Normative dimensions
Interaction	An entity defining an interaction between agents.	Dynamical dimension
Interaction Unit	A performative employed during the interaction.	Dynamical dimension
Condition	An entity that allows defining the sequence of tasks depending on a condition.	Dynamical dimension
Executer	A participant in an interaction. It can be an AAgent or a Role.	Dynamical, Environment and Normative dimensions
Environment	It is the physical location of the organization.	Environment dimension
Workspace	An entity that can be nested and aggregated. The sum of all workspaces built the environment.	Environment dimension

(continued)

Table 1 (continued)

Concept	Definition	Domain
Artifact	A non-proactive, but reactive entity that agents use to help them to fulfil their objectives.	Environment dimension
Informative Artifact	A type of artifact which provides information to the agents.	Environment dimension
Incentive Artifact	A type of artifact that changes the incentive system of the organization.	Environment dimension
Coercive Artifact	A type of artifact that is aimed to modify the action space of an agent.	Environment dimension

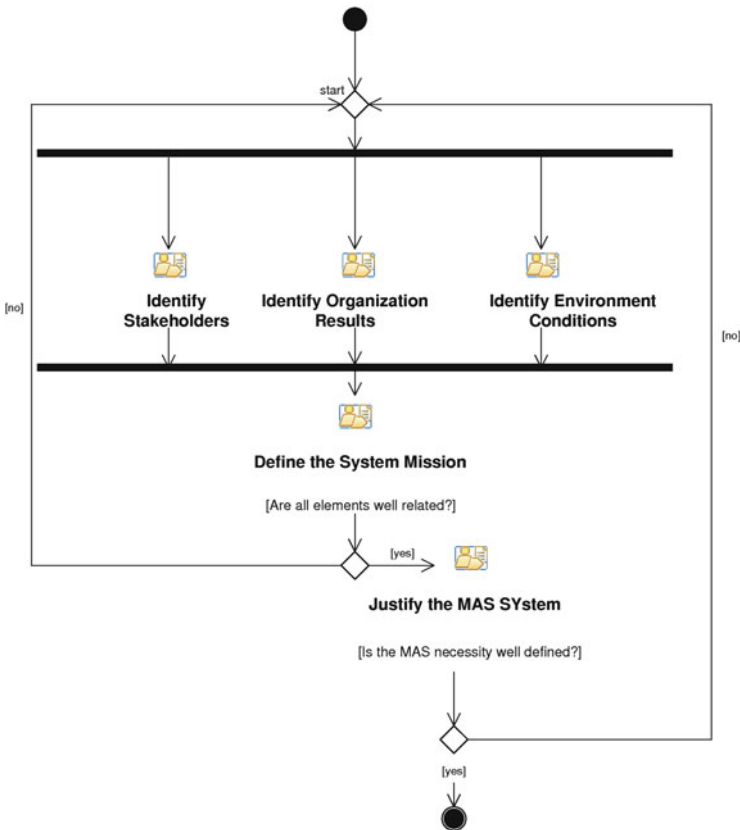


Fig. 8 Activity diagram of the *Mission Analysis* phase

2.1 Mission Analysis Phase

The first phase of GORMAS (Fig. 8) implies the analysis of the system requirements (i.e., the mission of the organization), identifying the stakeholders, the results (products/applications) that the system provides to these stakeholders and which

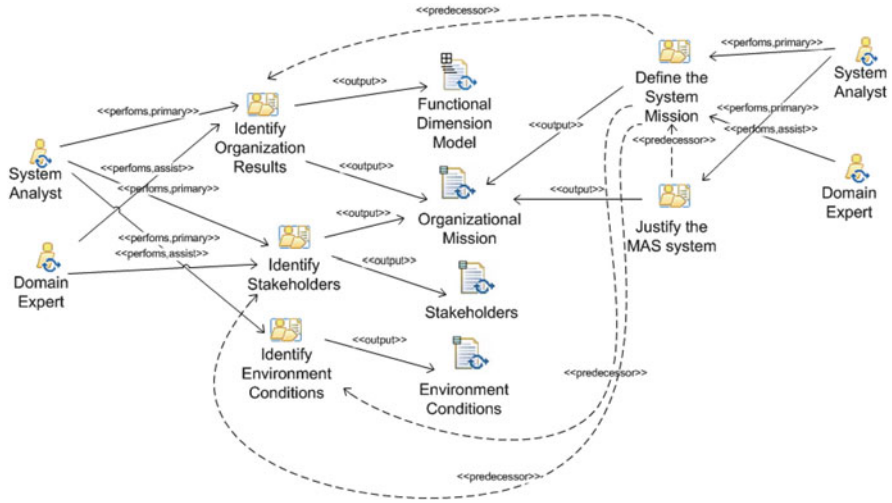


Fig. 9 Resources and products used in *Mission Analysis*



Fig. 10 Key for the elements contained in SPEM 2.0 diagrams

are the environment conditions that affect to the organization. All these issues are lately employed as a basis to define the global goals of the system, reflected by its mission. Moreover, this phase involves two different process roles (System Analyst and Domain Expert), and four work products (one model diagram and three text documents), as described in Fig. 9. Thus, this phase is composed of five activities (Fig. 8): Identify Stakeholders, Identify Organization Results, Identify Environment Conditions, Define the System Mission and Justify the MAS System. As a result, a diagram of the *Functional Dimension Model* is drawn, detailing the products and services offered by the system, the global goals (mission) pursued, the stakeholders and the existing links between them, the system results as well as the resources or services needed. The key to understand Figs. 8 and 9, and the forthcoming, similar figures represented using SPEM 2.0 is found in Fig. 10.

2.1.1 Process Roles

There are two roles involved in the *Mission Analysis* phase: the System Analyst and the Domain Expert:

- *System Analyst*. He/she is responsible of (1) defining the mission and the context of the organization, by means of identifying the system results, the stakeholders and the environment of the organization; (2) creating the documents that define

Table 2 *Mission Analysis* phase tasks

Task	Task description	Roles involved
Define the System Mission	Identify the global goals pursued by the system. These goals compose the mission of the organization.	System analyst and domain expert
Identify organization results	Describe the results (products or services) that the system provides, to understand what the result is, what it does and who is interested in.	System analyst and domain expert
Identify stakeholders	Identify and describe the main stakeholders that the organization is related to (external actors, clients, users, etc.)	System analyst and domain expert
Identify environment conditions	Identify and define the kind of environment in which the organization will be developed, knowing if it is physical or virtual; if it is distributed, etc.	System analyst and domain expert
Justify the MAS system	Justify the existence of this system, comparing it to other existing similar systems (that can use agents or not), and analyzing the advantages and disadvantages, and the singularities of the proposed system.	System analyst

Table 3 Products for *Mission Analysis* phase

Name	Description	Work product kind
Functional Dimension Model	A diagram using the GORMAS graphical notation (based on GOPPR notation) that details the specific functionality of the system, based on services, tasks and goals.	Behavioral
Organizational Mission	A document describing the basic aspects of the organization that will be defined.	Structured text
Stakeholders	A document describing the stakeholders that will take part on the organization.	Structured text
Environment conditions	A document describing the conditions that the environment of the organization will have.	Structured text

the mission of the system; and (3) instantiating the *Functional Dimension Model* diagram from the VOM metamodel.

- *Domain Expert*. He/she is responsible of supporting the System Analyst during the *Mission Analysis* phase, by giving him/her all the information that could be needed.

2.1.2 Activity Details

In the *Mission Analysis* phase, five tasks are defined: (1) to define the global goals of the system (mission); (2) to identify the services and products that the system provides to other entities; (3) to identify the stakeholders with whom the system contacts (clients or suppliers of resources/services), describing their needs and requirements; (4) to identify the conditions of the environment or context in which the organization exists (i.e., complexity, diversity, restrictions, etc.); and (5) to justify the existence of the MAS system that is being designed, in order to prove whether the GORMAS definition could contribute to define an organization or not. Table 2 summarizes these five tasks.

2.1.3 Work Products

This section defines the products that are generated in this phase (see Table 3): one behavioral diagram (Functional Dimension Model) and three structured text products (Organizational Mission, Stakeholders and Environment conditions). Figure 11 describes the relationship of the generated products on the Mission

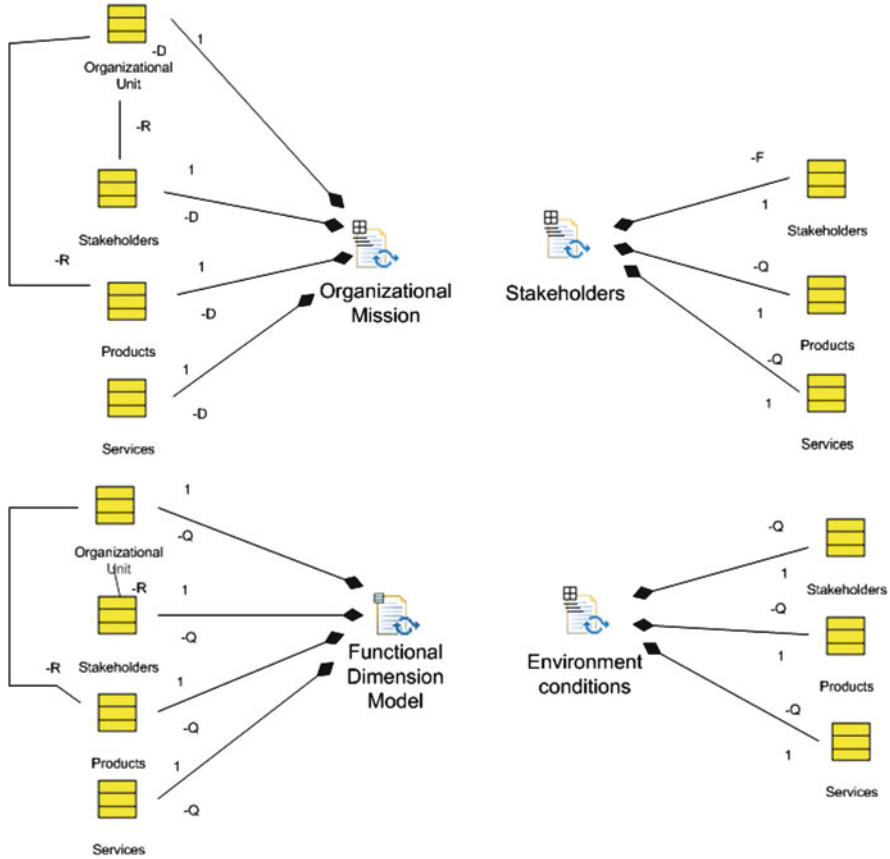


Fig. 11 Mission Analysis phase. Relations between work products and metamodel elements

Analysis phase and the elements of VOM. In this figure, each of the work products reports one or more elements from VOM; each element is represented using an UML class icon and, in the documents, such elements can be [10]: (1) *Defined*: D label near the element symbol, this means that the element is introduced for the first time in the design of this work product (i.e., the metamodel element is instantiated in this diagram); (2) *reFined*: F label means that the metamodel element is refined in the work product (for instance by means of attribute definition); (3) *Related*: R label, this means that an already defined element is related to another, or from a different point of view, that one of the metamodel relationships is instantiated in the document; (4) *Quoted*: Q label, means that the element has been already defined and it is reported in this work product only to complete its structure but no work has to be done on it; or (5) *Relationship Quoted*: RQ label means that the relationship is reported in the work product but it has been defined in another part of the process. As explained before, four work products are generated on this phase: Organizational Mission, Stakeholders, Environment Conditions, and Functional Dimension Model.

Table 4 Template for *Organizational Mission* document

Organizational Mission
Name: general name of the system or organization to be generated
Domain: kind of market or interest area of the organization
Results: set of products or services offered by the organization to its clients - <i>Purpose:</i> Description of the motivation by which this result is offered. - Is it <i>intangible?</i> : If the result is storable, printable and/or reusable, it is a “product”. If it is a used up functionality, it is a “service”.
Stakeholders: actors that set up the market of the organization. - Is it a <i>consumer?</i> : the actor consumes the products or services that the organization provides. - Is it a <i>producer?</i> : the actor provides some resources or services that are required by the organization to work.
Kind of environment: location of the system (unique or distributed). Ability to access to the real and physical world.
Context restrictions: a set of restrictions that are imposed by the context or environment of the organization, and could affect to its structure, services, etc.
Justification: reason of the existence of the organization - <i>Similar systems:</i> to detail the existing systems that provide a similar orientation than the one we are considering. - <i>Advantages:</i> set of advantages that we want to afford with the new proposal. i.e., optimal use of the resources or services. - <i>Disadvantages:</i> limitations that the new proposal has. - <i>Singularities:</i> competitive elements of the organization.

Table 5 Example of *Organizational Mission* document

<i>Organizational Mission</i>
Name: Conference Management System (CMS)
Domain: Scientific conference: education and research
Results: <ul style="list-style-type: none"> • <i>Product: Reviews Purpose:</i> The reviews for conference papers done by reviewers, including marks. • <i>Product: Proceedings Purpose:</i> A set of camera-ready versions of all accepted papers of the conference. • <i>Service: Submit paper. Purpose:</i> A service to upload a paper to the CMS. • <i>Service: Review paper. Purpose:</i> A service to review the submitted papers, distributing the work between reviewers and taking a final decision. • <i>Service: Prepare proceedings. Purpose:</i> A service to collect and format the camera-ready versions of the accepted papers of the organization
Stakeholders: <ul style="list-style-type: none"> • Authors: They write papers for the conference. • Reviewers: They review the papers of a conference. • Program committee members: They handle the paper review, contact the potential referees and ask them to review one or more papers. <ul style="list-style-type: none"> • Chair: distribute papers depending on expertise; accepts or changes system suggestions. • Vice-Chair: if the conference is big, the chair delegates his/her work on them.
Kind of environment: The environment is distributed between the different members of the organization. The system will be in touch with the real world.
Context restrictions: The organization must have a chair, and at least one paper (with at least one author), one PC member and one reviewer.
Justification: This is a system developed to manage the conference management system. <ul style="list-style-type: none"> • <i>Similar systems:</i> Other conferences or journals may have similar systems. • <i>Advantages:</i> To facilitate the tasks and procedures that must be performed by the system users. • <i>Singularities:</i> The performance of the organization is better using this system that makes the review process faster.

Organizational Mission. This document is employed to define the mission of the organization that will be described. It is a structured text document. Its template is shown in Table 4, whereas Table 5 gives an example of this work product. As shown, it is necessary to give a name, a domain and an environment description for

Table 6 Stakeholders document

Stakeholders	
Name	An identifier for the stakeholder.
Beneficiary	Indicate whether the stakeholder is a primary (essential) or a secondary beneficiary.
Type	Indicate whether the stakeholder is a client, a provider or a regulator.
Objectives	Describe the objectives pursued by the stakeholder.
Requires	A set of products and/or services that the stakeholder consumes.
Provides	A set of products and/or services that the stakeholder offers to the organization.
Frequency	To point out whether this stakeholder contacts with the organization frequently, occasionally or in an established period of time.
Benefits	Describe the benefits that the stakeholder wants to achieve.
Decision power	Indicate whether their needs are affecting to the requirements of products or services.
Under the influence of the system?	Indicate whether the organization can affect the interests of the stakeholder.
Contribution	To point out what the organization obtains from its relationship with the stakeholder.

Table 7 Example of a Stakeholders document

Stakeholders			
	Authors	Reviewers	PC members
<i>Beneficiary</i>	Primary	Primary	Primary
<i>Type</i>	Client	Client	Provider
<i>Objectives</i>	Upload papers	Review papers	Manage conference
<i>Requires</i>			
<i>Services</i>	Submit paper		
<i>Products</i>	Reviews, proceedings		Reviews
<i>Provides</i>			
<i>Services</i>			Review paper, prepare proceedings
<i>Products</i>		Reviews	Proceedings
<i>Frequency</i>	Frequent	Frequent	Frequent
<i>Benefits</i>	Publish a paper	CV merits	CV merits
<i>Decision power</i>	No	No	Yes
<i>Under the influence of the system?</i>	Yes	Yes	No
<i>Contribution</i>	No authors means no conference	Reviews are needed to run a conference	A chair is required

the organization. Additionally, it is necessary to set the results that the system will provide with and the stakeholders that are interested on keeping a relationship with the organization. Finally, a justification for designing the system must be provided.

Stakeholders. This document is employed to describe the stakeholders of the organization that have been previously identified in the *Organizational Mission* document. It is a structured text document. Its template is shown in Table 6, and Table 7 gives an example of this work product, describing three of the stakeholders (authors, reviewers, chair) from the CMS case study. The description of the stakeholders is done by providing the type of stakeholder, the objectives that they follow, their products and services provided and required, the benefits obtained by them and their position into the organization.

Table 8 *Environment Conditions* document

Environment conditions	
Change rate:	Are the stakeholders constant through time? Are their requirements constant? Are they modified in a cyclical and a predictable way? Is it possible to estimate the consumption of a product? Is the demand of a product or a service constant through time? If the answer is affirmative, the environment is stable. If not, it is an unstable or dynamic environment.
Complexity:	Are there a lot of different elements? Are there a lot of clients? Are there a lot of types of products and services to offer? Are there a lot of types of providers? Are providers not related between them? If any of the answers is affirmative, the environment is complex. If not, it is a simple environment.
Uncertainty:	If the environment is dynamic and complex, uncertainty is high. If the environment is stable and simple, uncertainty is low.
Receptivity:	Are the inputs and resources available? Are they obtained in an easy and secure way? If the answer is affirmative, the environment is munificent. If not, it is a hostile environment.
Diversity:	Are different groups of clients served? Is it provided a set of different products or services, with no relationship between them? If any of the answers is affirmative, the environment is diverse. If not, it is a uniform environment.

Table 9 Example of an *Environment Conditions* document

Environment conditions		
Condition	Value	Justification
Change rate	Stable	Stakeholders and activities remain unchanged through time.
Complexity	Simple	Low number of clients, products, and services to be offered.
Uncertainty	Low	The environment of the organization is stable and simple.
Receptivity	Munificent	The inputs and resources are obtained in an easy and controlled way from previously known providers.
Diversity	Uniform	There are a few different types of clients, products and services.

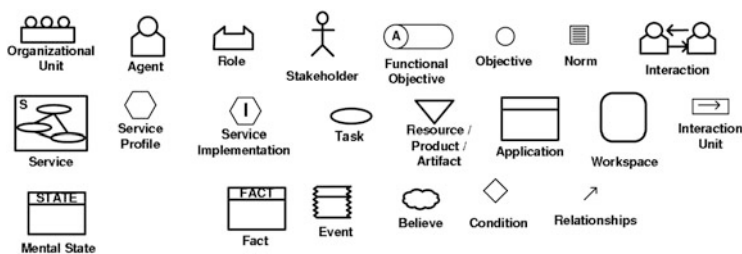


Fig. 12 Entities from the GORMAS graphical notation

Environment Conditions. This document is employed to describe the environment conditions in which the organization will be placed. It is a structured text document. Its template is shown in Table 8, and Table 9 gives an example of this work product. This document analyzes five conditions: the change rate, the complexity, the uncertainty, the receptivity and the diversity of an environment.

Functional Dimension Model. This work product is a GORMAS diagram. GORMAS uses an UML-like graphical notation called GOPPR [14] (used to define diagrams on INGENIAS and ANEMONA methodologies), but adding some entities proposed by GORMAS such as services and norms. A caption to understand the elements of the diagram is shown in Fig. 12.

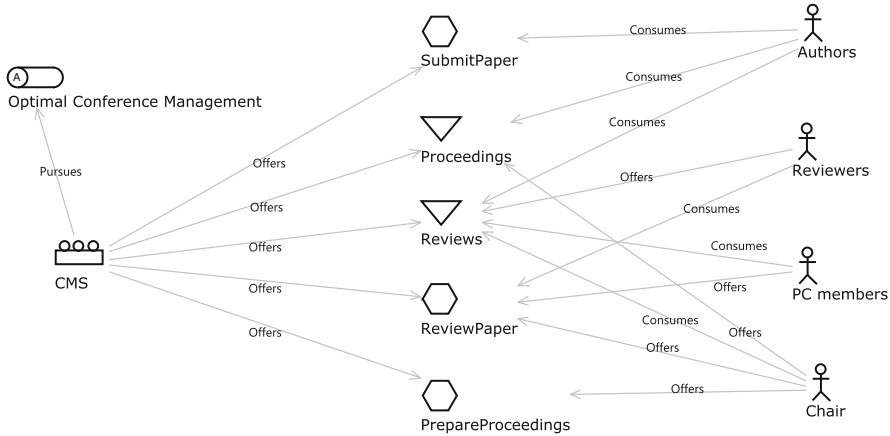


Fig. 13 Example of a *Functional Dimension Model* diagram

As stated before, the *Functional Dimension Model* details the specific functionality of the system, based on services, tasks and goals, as well as system interactions. In this phase of the methodology, the OU representing the system, along with the stakeholders, the global goals, the products and the services of the system are instantiated on this diagram. Figure 13 shows an example of a *Functional Dimension Model* diagram. An OU is defined (CMS), containing three services (Submit Paper, Review Paper, Prepare Proceedings) and two resources (Reviews and Proceedings). The Organization pursues one objective (Optimal Conference Management) and there are four groups of stakeholders (Authors, Reviewers, Program Committee members, and Chair), which are consuming and offering services (represented by a service profile entity) and resources.

2.2 Service Analysis Phase

In this phase (Fig. 14), the services offered by the organization to its clients are specified, as well as how these services behave (inputs/output description and internal tasks) and which are the relationships (interdependencies) between these services. Furthermore, goals associated with services are detailed.

Therefore, the *Service Analysis* phase involves (Fig. 15) two different process roles (System Analyst and Domain Expert), and five work products (three GORMAS model diagrams and two text documents). This step is composed of five activities (Organization Technology Analysis, Organizational Unit Technology Analysis, Work Flow and Technological Interdependence Analysis, Define Organization Functionality and Analysis of the goals of the organization). The flow of tasks of this phase is shown in Fig. 14.

Taking the Organization Theory as a basis, three existing types of technology are considered: (1) the *Organization Technology*, which refers to the whole organization

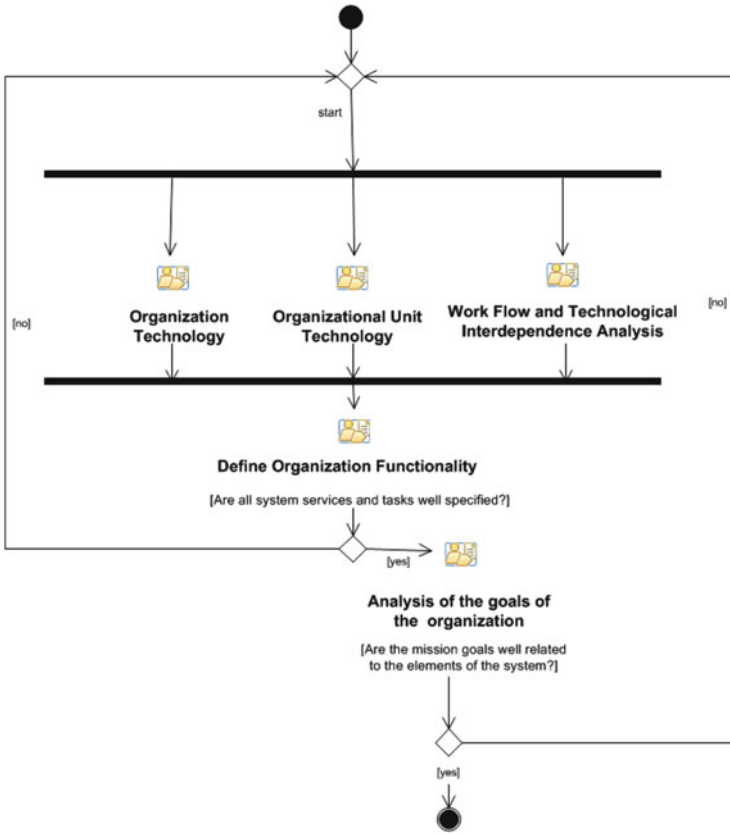


Fig. 14 Activity diagram of *Service Analysis* phase

and determines if the client has influence in the process of production and the final aspect of the product or how services are related between them, and with regard to the clients; (2) the *Organizational Unit Technology*, which contemplates the diversity and complexity of the different organizational tasks, identifying the existing flows of work; and (3) *Work Flow and Technological Interdependence*, which defines the interdependent relations originated as a result of the workflow between OUs.

As a result of this phase, the diagrams of the *Structural* and *Environment Dimension Models* are instantiated and the *Functional Dimension Model* diagram is updated. Specifically, in the *Functional Dimension Model*, both resources and applications of the system are identified. In the *Structural Dimension Model*, the entities representing the clients and/or providers of the system are established. Moreover, the services required and offered by the system are identified, as well as the roles that provide or make use of these services. For each service, the *Functional Dimension Model* diagram must be updated, detailing the service profile (inputs,

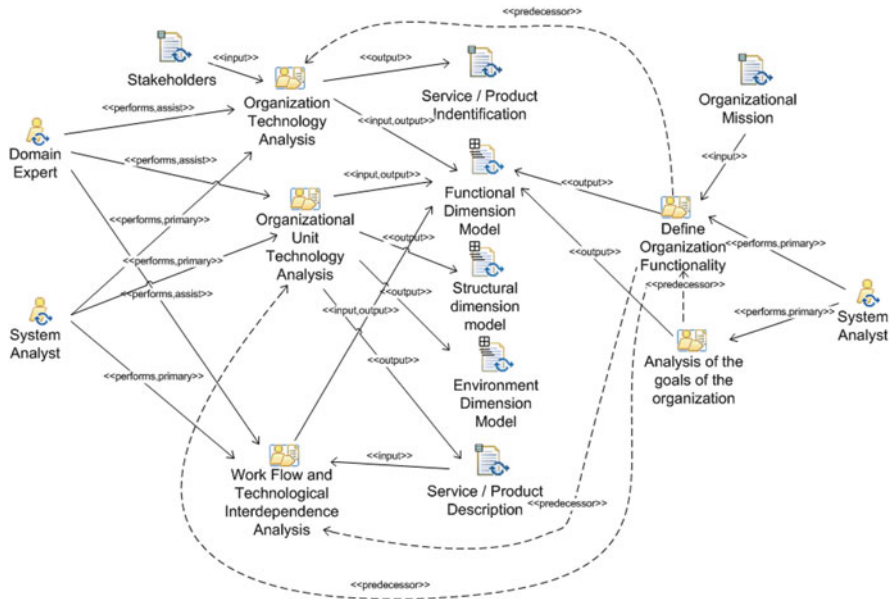


Fig. 15 Resources and products used in *Service Analysis* phase

outputs, preconditions and postconditions) and its tasks. Finally, in the *Functional Dimension Model*, the mission is split into functional goals, which are related to the system entities and its services. The functional goals represent the specific actions of the organizational units and their expected results.

2.2.1 Process Roles

There are two roles involved in this phase:

- *System Analyst.* He/she is responsible of (1) analyzing the technology of the system, along with the goals of the organization; (2) identifying and describing the products and services of the organization; (3) splitting the mission goals of the organization into functional goals; (4) modeling the Structural and the Environment Dimension Model diagrams; and (5) updating the Functional Dimension Model.
- *Domain Expert.* He/she is in charge of supporting the System Analyst on the analysis of the technology, the goals, the services and the products of the system.

2.2.2 Activity Details

For the MAS field, we have mapped the technology concept to the set of resources, applications and knowledge required by agents, as well as the set of processes and tasks that are necessary to carry out with the services offered by the organization. The Service Analysis phase is aimed to identify the kind of technology that the organization will use. Moreover, the designer will describe on detail the generated products and the provided services of the organization.

Table 10 Tasks of the *Service Analysis* phase

Task	Task description	Roles involved
Organization Technology Analysis	Determine the way in which products and services are produced. It can be “ <i>organization directed</i> ”, “ <i>client directed</i> ” or “ <i>standard production</i> ”.	System Analyst and Domain Expert
Organizational Unit Technology Analysis	Existing workflows are determined and services and products are analyzed using the templates provided by the methodology.	System Analyst and Domain Expert
Work Flow and Technological Interdependence Analysis	Identify what is the relationship between organizational units in order to reach the organizational goals. There are three kinds of interdependence: <i>independent</i> (coordination between units must be minimum), <i>sequential</i> (tasks are linked or follow a required workflow) and <i>reciprocal</i> (units depend on each other).	System Analyst and Domain Expert
Define organization functionality	It must be checked whether services and their related tasks are well specified. Additionally, for every Agent or Organizational Unit identified in the functional dimension model, it is assigned a split of the mission goals, by means of the products it generates, its clients or the services it attends.	System Analyst
Analysis of the goals of the organization	For every identified Service, a link is designed between one or some functional objectives and its Organizational Unit.	System Analyst

Besides the description of services and products, there is another main activity in this phase of the methodology. The mission of the system must be derived into other goals or objectives: functional objectives and operative objectives. *Functional objectives* represent the results that organizational units are expected to achieve. The *operative objectives* are the specific and measurable results that are expected to be achieved by the members of a unit. Therefore, this activity aims to split the goals of the system into functional objectives and to set their relationships with organizational units, agents and services of the organization. Tasks of this activity are detailed in Table 10. More specifically, at the *Service Analysis* phase it is specified: (1) the type of products and services that the system offers to or consumes from its stakeholders; (2) the tasks related to the production of the products and services, defining the steps necessary for obtaining them, their relationships and interdependencies between the different services and tasks; (3) the goals related with the achievement of these products or services; (4) the resources and applications needed for offering the system functionality; and (5) the roles related with the stakeholders, on the basis of the type of services or tasks that they provide or consume.

2.2.3 Work Products

This section describes the products generated by the *Service Analysis* phase (see Table 11). Two models from VOM are instantiated (the *Structural* and *Environment Dimension Model* diagrams), whereas the *Functional Dimension Model* diagram is updated. Moreover, two kinds of documents are generated: the *Service/Product Identification*, which is a structured text document that allows defining the type of production of the organization and the kind of technology that the system will use; and the *Product/Service Description*, a document describing the features of the products and services of the organization. Figure 16 describes the relationship of the generated products in this phase and the elements of VOM.

Table 11 Products for *Service Analysis* phase

Name	Description	Work product kind
Functional Dimension Model	See Table 3.	Behavioral
Structural Dimension Model	A diagram that uses the GORMAS notation that describes the components of the system and their relationships. It allows defining the static components of the organization.	Structural
Environment Dimension Model	A diagram that uses the GORMAS notation that describes environmental elements (resources and applications), along with the agents' behavior. It also allows defining the service ports.	Behavioral
Product / Service Identification	A document that identifies the kind of production the system will have and the technology that products will use.	Structured text
Product / Service Description	A document that describes the features of products and services.	Structured text

Product/Service Identification. This document is a couple of structured text documents that describe the kind of production the system will have and the technology that products will use. On the one hand, the technology used to obtain the products is identified. It can be *organization directed*, *client directed* or *standard production*, depending on who is requesting the products. On the other hand, the technology followed by the service organization is described. Services can have *interdependence*, *dependence* or *variability* among them. The template of this document can be found in Tables 12 and 13; an example is shown in Table 14.

Product/Service Description. This document is a couple of structured text documents that describe the products and services that the organization will produce. Services are described by means of their functionality, the roles involved on them and their profile. For product description, it is necessary to specify the granularity of each product, its margins, and/or parameters and the resources that requires. The template of this document can be found in Tables 15 and 16; an example is shown in Table 17.

Functional Dimension Model. This model was initially instantiated on the *Mission Analysis* phase (see Sect. 2.1.3), and in this phase it is updated, so roles are related with the service they offer or require, services are split into tasks (defining the resources and applications that they offer and consume) and the mission of the system is split into functional objectives. As an example, Fig. 17 depicts the mission of the system split on functional objectives. This split forms the Goal Tree of the organization.

Structural Dimension Model. It describes the components of the system and their relationships, allowing the definition of the static components of the organization. In this phase of the methodology, the Organizational Unit representing the whole system is depicted, as well as those entities that have been identified in the previous steps. Figure 18a shows an example of a *Structural Dimension Model*

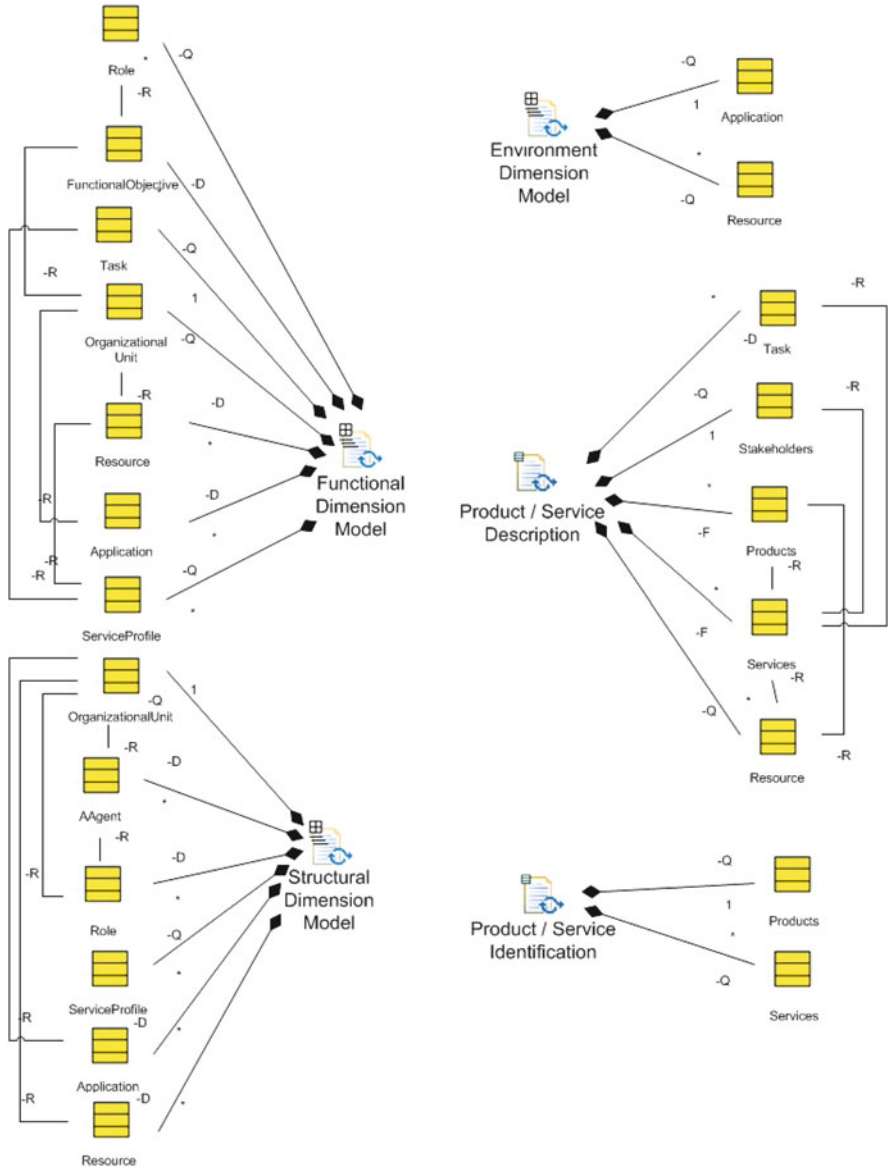


Fig. 16 Service Analysis phase. Relations between work products and VOM elements. *D*: element introduced for first time; *F*: element refined; *Q*: element already defined; *R*: element related with another element

diagram in which a “client directed” technology has been considered. In this figure, four agents (Author, Reviewer, PC member, and Chair) and four roles with the same names as agents are defined along with the OU, named CMS, that represents the Conference Management System, the organization used as case study.

Table 12 *Product Identification* document

Product identification
<p>Are the obtained products used as resources to make new products? Is the production continuous without a clear start and a clear end? Are modules from other processes used? Are the modules assembled in order to obtain products under request? In case of affirmative answer, then it is an organization directed production.</p> <ul style="list-style-type: none"> • <i>Tip:</i> Basic products that are components to make more complex products must be identified. They must be represented as resources.
<p>Do products fit the client’s needs? Is every product made for a concrete client, under his request? Is a wide range of products managed? Is a big variety of the clients’ requirements taken into account? Are changes on product features anticipated? In case of affirmative answer, then it is a client directed production.</p> <ul style="list-style-type: none"> • <i>Tip:</i> Every client must be represented by an A-Agent. It must be included as a member of the Organizational Unit that represents the system.
<p>Could a product be consumed by different kinds of clients? Could a product be consumed by different clients that belong to the same type? Is their production independent from the requirements of their final consumers? In case of affirmative answer, then it is an standard production.</p> <ul style="list-style-type: none"> • <i>Tip:</i> Applications for interacting with the clients must be created. The Organizational Unit that represents the system must contain at least one application per client. All these applications will allow extracting information related to specific clients.

Table 13 *Service Identification* document

Service identification
<p>Is the functionality of every service independent? Is the order that the services are executed making no difference? Are some types of clients connected by means of services? In case of affirmative answer, then it is service independence.</p>
<p>Is a determined order to execute the services required? Are the inputs of a service depending from the outputs of another service? Does a client need to use a previously established service to request another service? In case of affirmative answer, then it is service dependence.</p>
<p>Is the order of the services to offer variable? Does it depend on the decisions and requirements of the clients? Are the needs of the clients unpredictable? In case of affirmative answer, then it is service variability.</p>

Table 14 Example of a *Product/Service Identification* document

Product/service identification
<p>Product technology: <i>standard production</i>. Products are made independently from the requirements of final consumers.</p>
<p>Service technology: <i>client dependence</i>. Products are generated after the petition of the clients of the organization.</p>

Table 15 *Product Description* document

Organizational Unit Technology	
Name	Name of the product
Description	A short description about the product.
Parameters	A set of the product features that will be considered.
Lower margin	Minimum value used for the service.
Upper margin	Maximum value used for the service.
Granularity	Grade of variability that the values for every parameter could have.
Resources	A set of environment entities that the product will use.
<i>Tip</i>	If a “ <i>client directed</i> ” production is used, then variables defined by the clients must be set.

Environment Dimension Model. It describes the environment of the system by means of artifacts and workspaces. Figure 18b shows an example of an *Environment Dimension Model* diagram, depicting the workspace where the organizational unit CMS is located, as well as two artifacts, Proceedings and Reviews.

Table 16 *Service Description* document

<i>Organizational Unit Technology</i>	
Name	<i>Name of the service</i>
Description	A short description about the service functionality.
Conditions	
<i>Context</i>	A specification of the environment where the service is executed.
<i>Exceptions</i>	A set of conditions that prevent the correct execution of the service.
Consumer	An actor that requests the service.
<i>Objective</i>	What the consumer is looking for by using the service.
<i>Price</i>	The value that the consumer should pay while using the service.
<i>Benefit</i>	A description of the benefit obtained by the consumer.
Producer	An actor entrusted to provide and execute a service.
<i>Objective</i>	What the producer is looking for by providing the service.
<i>Cost</i>	A description of the cost to pay while using the service.
<i>Benefit</i>	A description about the benefit obtained while providing the service.
Service Profile	
<i>Inputs</i>	Information that must be supplied to the service.
<i>Preconditions</i>	A set of the input conditions and some environment values in order to obtain a correct execution of the service.
<i>Outputs</i>	Information returned by the service.
<i>Postconditions</i>	Final states of the parameters of the environment, by means of the different kinds of outputs.
Functionality	
<i>Tasks</i>	A set of tasks covered by the service. It must be pointed out whether any of these tasks is another service provided by the organization.
<i>Resources</i>	A set of environment entities that the service will use.
<i>Provider</i>	An actor that provides the given resources.
<i>Products</i>	Tangible results obtained by executing the service.

Table 17 *Service Description* document for the CMS case-study

<i>Organizational Unit Technology</i>	
Service	<i>Submit paper</i>
Description	A service to upload a paper to a conference.
Conditions	
<i>Context</i>	A conference is being organized and papers are required.
<i>Exceptions</i>	There is no conference to upload papers to.
Consumer	Authors
<i>Objective</i>	To upload a paper to be reviewed for a conference.
<i>Price</i>	The previous effort of writing a paper.
<i>Benefit</i>	A possible publication.
Producer	Chair
<i>Objective</i>	To control that the process is correctly carried out.
<i>Cost</i>	
<i>Benefit</i>	Papers to correctly organize a conference
Service Profile	
<i>Inputs</i>	– Paper = (PaperID, authors, content)
<i>Preconditions</i>	$\neg \exists a \in \text{Papers} \mid a.\text{PaperID} = \text{PaperID}$
<i>Outputs</i>	
<i>Postconditions</i>	$\exists a \in \text{Papers} \mid a.\text{PaperID} = \text{PaperID} \vee \text{Papers}$
Functionality	
<i>Tasks</i>	SendPaper.AssID – SendPaper: Send a paper to the CMS – AssID: Assign an ID to the received paper
<i>Resources</i>	Paper database Author database
<i>Provider</i>	
<i>Products</i>	

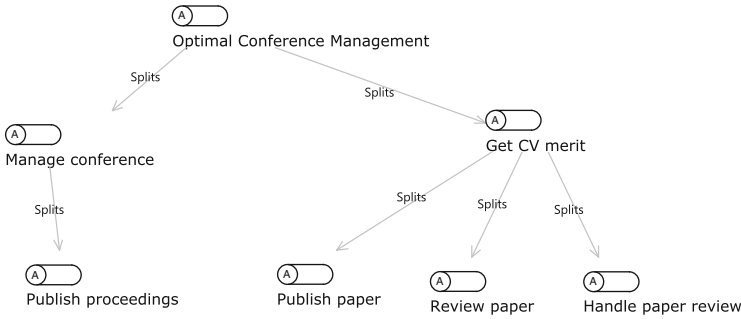


Fig. 17 Relationship between *mission* and *functional objectives* of the organization

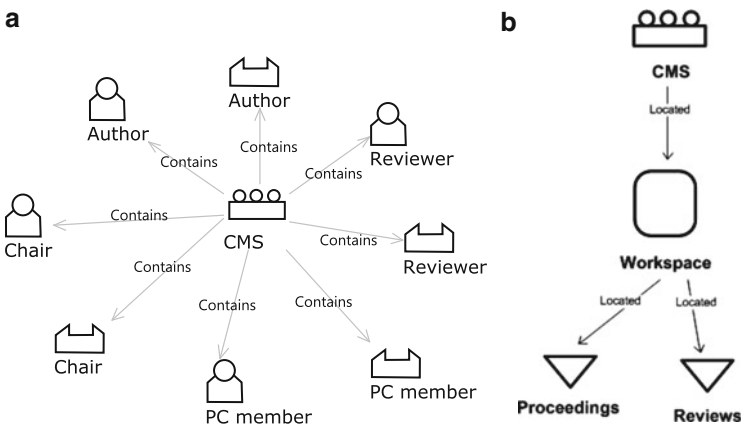


Fig. 18 Examples of (a) *Structural* and (b) *Environment Dimension* Models

2.3 Organizational Design Phase

In this phase (Fig. 19), the most suitable structure for the system organization is selected [15]. This structure will determine the relationships and pre-established restrictions that exist between the elements of the system, based on specific dimensions of the organization [16], which impose certain requirements on the types of work, on the structure of the system and on the interdependence between tasks.

For structure selection, a decision tree (Fig. 20) has been elaborated that enables the system designer to identify which is the structure that better adjusts to the conditions imposed by the organizational dimensions. This decision tree can be applied to the system as a whole or to each of its OUs, so then enabling structure combinations. At the end of this step of the methodology, context restrictions identified on Mission Analysis phase will be taken into account in order to modify the structure of the organization if necessary.

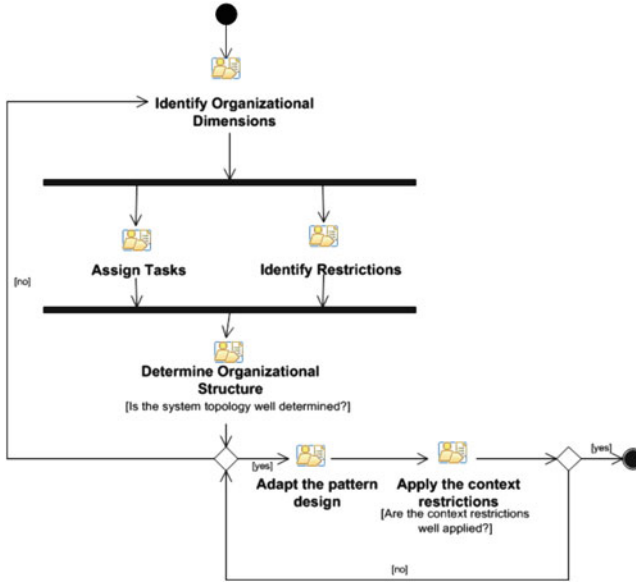


Fig. 19 Activity diagram of the *Organizational Design* phase

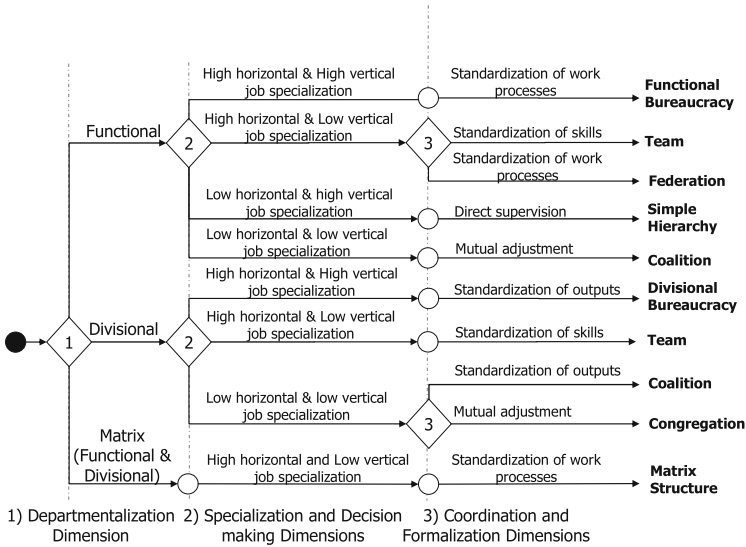


Fig. 20 Organizational Structure decision tree

Additionally (see Fig. 21), as a result of this phase, *Functional and Structural Dimension models* are updated. *Structural Dimension Model* is updated by adding new Organizational Units, Roles, Resources and Norms. *Functional Dimension Model* is updated in order to show new relationships between Roles and Organizational Roles and to describe the functionality of the services.

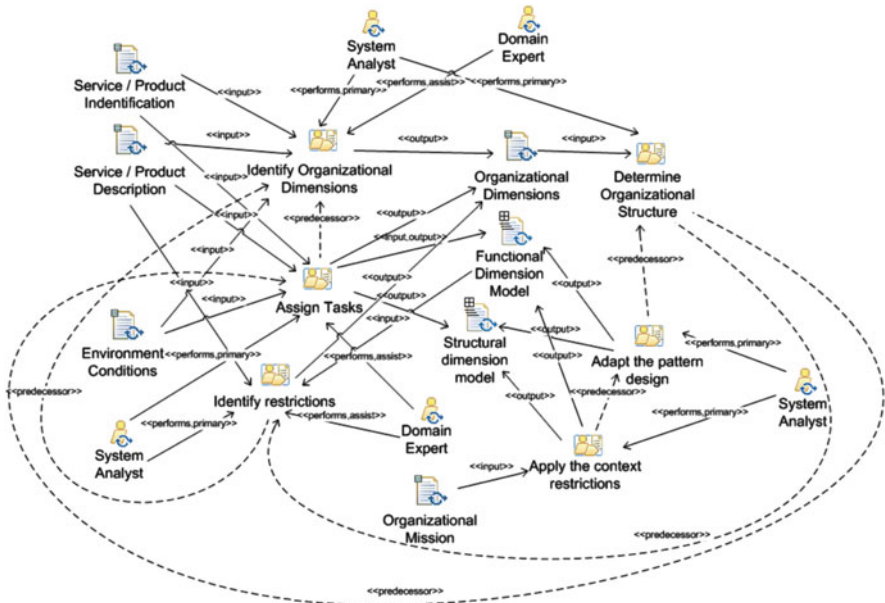


Fig. 21 Products of the *Organizational Design* phase

2.3.1 Process Roles

There are two roles involved in the Organizational Design phase:

- *System Analyst*. He/she is responsible for (1) defining the organizational dimensions of the system; (2) selecting the most suitable structure for the organization; and (3) adapting the selected structure to the organization.
- *Domain Expert*. He/she is responsible for supporting the system analyst during the Organizational Design phase, by giving him/her all the needed information about the organizational dimensions of the system.

2.3.2 Activity Details

As stated before, the first task to be accomplished in this phase is to identify the specific dimensions of the organizational structure. The dimensions of organizational structure [16] are:

- *Departmentalization*, which details the motivation of work group formation, that is, functional (on the basis of knowledge, skills or processes) or divisional (on the basis of the characteristics of the market, clients, products or services).
- *Specialization*, which indicates the degree of task division, based on the quantity and diversity of tasks (horizontal job specialization) and the control exercised on them (vertical job specialization).
- *Decision making*, which determines the degree of centralization of the organization, that is, the degree of concentration of authority and capture of decisions.
- *Formalization*, which specifies the degree in which tasks and positions are standardized, by means of norms and rules of behavior.

Table 18 *Organizational Design* activity

Task	Task description	Roles involved
Identify Organizational Dimensions	Define the organizational dimensions of the system by assigning tasks and identifying restrictions.	System Analyst and Domain Expert
Assign tasks	Tasks are grouped into Organizational Units and assigned to members of the organization.	System Analyst and Domain Expert
Identify restrictions	Existing restrictions about the organization members behavior are identified. Additionally, mechanisms to help coordination and cooperation between members are described	System Analyst and Domain Expert.
Determine Organizational Structure	Using the organizational dimensions previously identified, along with the decision tree (see Fig. 20), the best structure for the organization is identified. This activity can be applied not only to the whole organization, but also to some OUs.	System Analyst
Adapt the pattern design	The identified structure is adapted by modifying Functional and Structural Dimension Models	System Analyst
Apply the context restrictions	Context restrictions identified on Mission Analysis phase are applied to the structure of the organization.	System Analyst

- *Coordination Mechanism*, which indicates how individuals can coordinate their tasks, minimizing their interactions and maximizing their efficiency, using mutual adjustment, direct supervision or standardization.

These organizational dimensions are employed for determining the most suitable structure for the system specifications. Thus, it is carried out: (1) the *analysis of the organizational dimensions*, which allows specifying the functional granularity of the service, by means of grouping tasks and services together (defining more complex services) and assigning them to specific roles; and identifying general restrictions on the coordination and cooperation behavior of the members of the organization; (2) the selection of the most interesting *organizational structure* for the system; and (3) the *adoption of the selected structure* to the problem under study, using specific design patterns.

Along with the decision tree, a set of design patterns of different structures has been defined which include simple hierarchy, team, flat structure, bureaucracy, matrix, federation, coalition and congregation structures. These patterns describe their intrinsic structural roles, their social relationships, as well as their typical functionality. According to these design patterns, the diagrams of the structural and functional dimension models are updated, so then these intrinsic roles, relationships and functionality related to the design pattern are integrated inside the current problem.

At the end of this phase, the context restrictions identified on *Mission Analysis* phase (*Organizational Mission* document) are taken into account to provide a final design of the organization. Therefore, the structure obtained after integrating the design pattern must be modified to satisfy the context restrictions. This is an iterative process. First of all, textual restrictions are needed to be transformed into GORMAS entities. For example, if one of the restrictions specifies “There must be a manager”, a role called “Manager” is involved into the organization. Next, it is necessary to integrate these restrictions into the selected structure that must be adapted until an optimal structure for the organization is achieved. All tasks of this activity are detailed in Table 18.

Table 19 Products for *Organizational Design* phase

Name	Description	Work product kind
Functional Dimension Model	See Table 3.	Behavioral
Structural Dimension Model	See Table 11.	Structural
Organizational Dimensions	A document that describes the specific dimensions of the organization: <i>departmentalization, specialization, decision making, formalization and coordination mechanism.</i>	Structured text

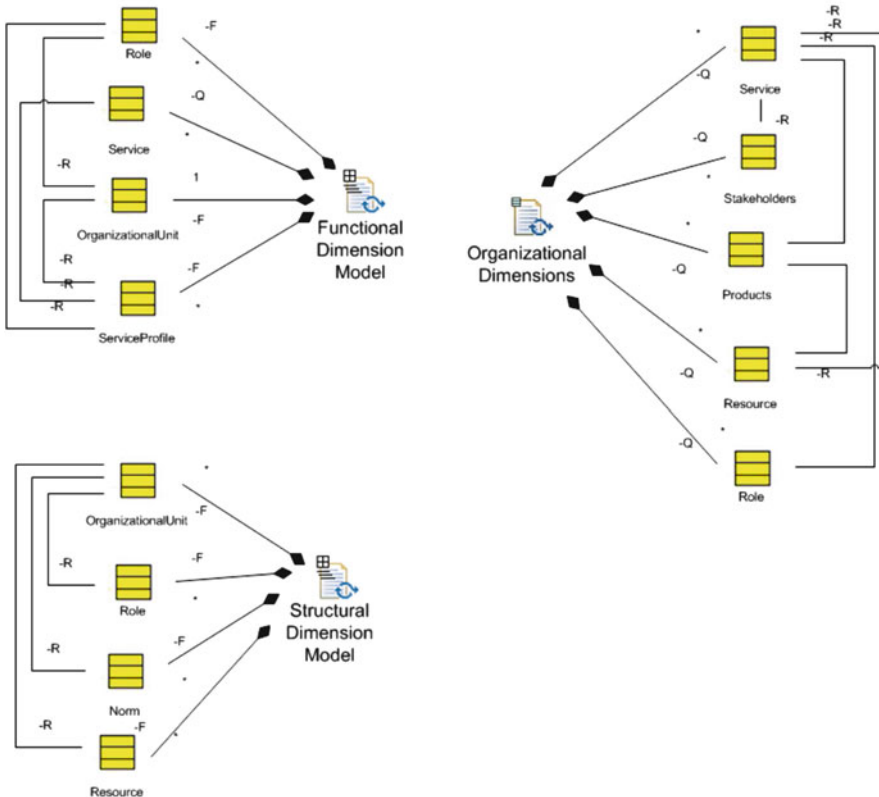


Fig. 22 Organizational Design phase. Relations between work products and metamodel elements. *F*: element refined; *Q*: element already defined; *R*: element related with another element

2.3.3 Work Products

This section describes the products generated on the organizational design phase. Two models are updated: the *Structural* and *Functional Dimension Model* diagrams. Moreover, the *Organizational Dimensions* structured text document is generated. It describes the specific dimensions of the organization structure that will be used to select the structure that best fits the system. Work products are described in Table 19. Figure 22 describes their relation with VOM elements.

Table 20 *Organizational Dimensions* document

Organizational Dimensions
Departmentalization
Is it necessary to act on the same type of resources? Does the offered functionality follow similar behavior patterns? Do the services require the same kind of inputs? Do they offer similar outputs? In case of affirmative answers, it is a functional departmentalization .
<ul style="list-style-type: none"> • <i>Tip:</i> Similar functionalities must be grouped and an Organizational Unit must be created for every group.
Do activities must be particularized by means of the clients or the offered products? Is the functionality different according to the geographical location? In case of affirmative answers, it is a divisional departmentalization .
<ul style="list-style-type: none"> • <i>Tip:</i> Functionalities aimed to the same client must be grouped and an Organizational Unit must be created for every group.
Are similar functionalities for different kinds of clients or products available? Are these functionalities related between them, following a specific order? In case of affirmative answers, it is a functional and divisional departmentalization .
Specialization and Decision Making
Have the roles got specialized tasks assigned? Do the roles make few types of tasks? Are the roles lacking of control over their own work? In case of affirmative answers, it is a high horizontal and high vertical job specialization .
<ul style="list-style-type: none"> • <i>Tip:</i> Agents' tasks control will be assumed by supervisor agents that are in charge of their Organizational Units.
Do the roles have assigned specialized tasks but exerting control over them? Can agents select the mechanism to carry out these tasks? In case of affirmative answers, it is a high horizontal and low vertical job specialization .
Are the roles in charge of different tasks not so related among them? Have these roles a low interdependence? Do the roles not offer control over their activities? In case of affirmative answers, it is a low horizontal and high vertical job specialization .
Are the roles in charge of some different tasks, assuming their control? In case of affirmative answers, it is a low horizontal and low vertical job specialization .
Is the environment simple? Is it necessary to process few quantities of information? In case of affirmative answers, centralization is recommended .
Is the environment complex? Is it necessary to process big amounts of information? In case of affirmative answers, centralization is not recommended .
Coordination and Formalization
Is the environment dynamic? Is the way to execute tasks flexible? Can tasks be carried out by different roles? Do roles present low vertical job specialization? Mutual adjustment (negotiation processes) must be applied.
Do the roles present high vertical job specialization? Are the control and management centralized in some points? Direct supervision must be applied.
Is the order of execution of tasks very important? Is it more indifferent to the resolution method by which the system has been obtained? Is it independent from whom is providing? Standardization of skills is applied.
Are the knowledge and skills that are available on certain tasks indispensable? Is there a predefined and globally assumed behavior for a particular skill? Standardization of skills must be applied.

Table 21 *Organizational Dimensions* document example

Organizational Dimensions
Departmentalization: Functional. Services are grouped by means of their functionality.
Specialization and Decision making: High horizontal and high vertical job specialization. Roles use and provide a reduced set of types of services.
Coordination and Formalization: Standardization of work processes. Order of tasks is very important and each task must be performed by a specific agent.

Organizational Dimensions. This is a structured text document that describes the dimensions that impose some requirements to the organizational structure. These dimensions are *departmentalization*, *specialization*, *decision making*, *formalization* and *coordination mechanism*. The template of this document can be found in Table 20 and an example is shown in Table 21 (for the CMS case study).

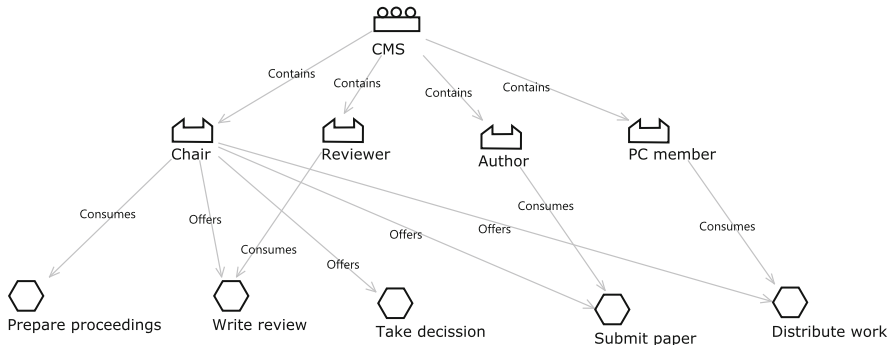


Fig. 23 Example of a *Functional Dimension Model* diagram updated on *Organizational Design* phase, for the CMS case study

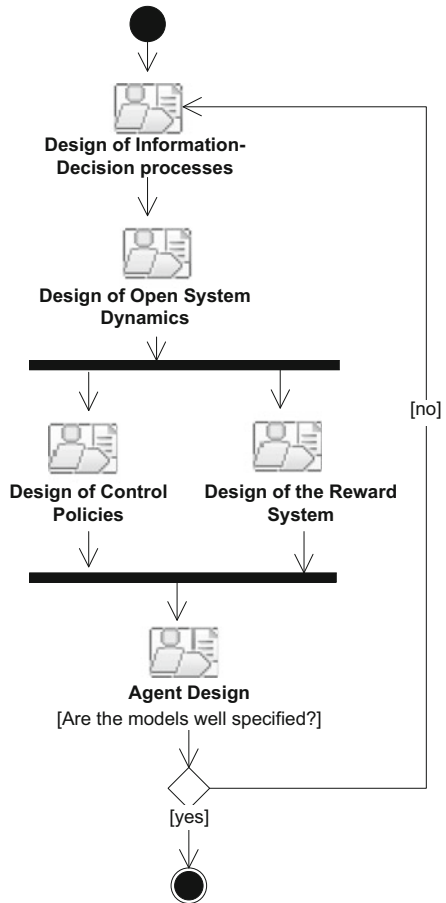
Functional Dimension Model. This diagram was previously instantiated on *Mission Analysis* phase (see Sect. 2.1.3) and updated on *Service Analysis* phase (see Sect. 2.2.3). In the *Organizational Design* phase, *Functional Dimension Model* is updated by adding relationships between roles and the services they provide/consume, along with the relationships between OUs and roles, and the services they provide. Also, services must be split into more specific services. An example of an updated *Functional Dimension Model* is shown in Fig. 23, which shows the OU (named CMS) and five services, being three of them (Write review, Take decision, and Distribute work) the ones that split from the previously defined *Review Paper* service. Additionally, there are represented the Contains relationships between OUs and Roles, showing which roles are consuming or offering each service.

Structural Dimension Model. This diagram was previously instantiated on *Service Analysis* phase (see Sect. 2.2.3). In the *Organizational Design* phase, *Structural Dimension Model* is updated by adding new OUs, roles and their relationships, and by relating OUs with the roles they contain. It is necessary to specify the inheritance of roles. Additionally, entities defined by the design pattern must be added to the diagram and the structure that best fit the organization that is adopted.

2.4 Organization Dynamics Design Phase

In this phase (Fig. 24), the detailed design of the system is carried out, which implies the following activities: design of the information decision processes; design of the system dynamics as an open system; design of the measuring, evaluation and control methods; definition of a suitable reward system; and design of the system agents, describing them by means of diagrams of the agent model. Moreover, it involves one process role (System Analyst) and seven work products (four model diagrams and three text documents), which are detailed next (see Fig. 25).

Fig. 24 Activity Diagram of the *Organization Dynamics Design* phase



2.4.1 Process Role

There is one role involved in the Organizational Design phase: System Analyst. He/she is responsible for (1) detailing services and related workflows; (2) identifying the operative goals; (3) defining the specific interactions between agents and their collaboration diagrams; (4) defining the ontology of the domain; (5) determining the services that have to be advertised; (6) determining the policies for role enactment; (7) identifying the internal and external agents; (8) considering standardization of work process, outputs and skills; (9) analyzing the types of behavior needed to promote; (10) selecting the type of reward system to be used; and (11) applying the selected reward system in the specific domain problem.

2.4.2 Activity Details

The activities that compose this phase of the methodology are detailed as follows.

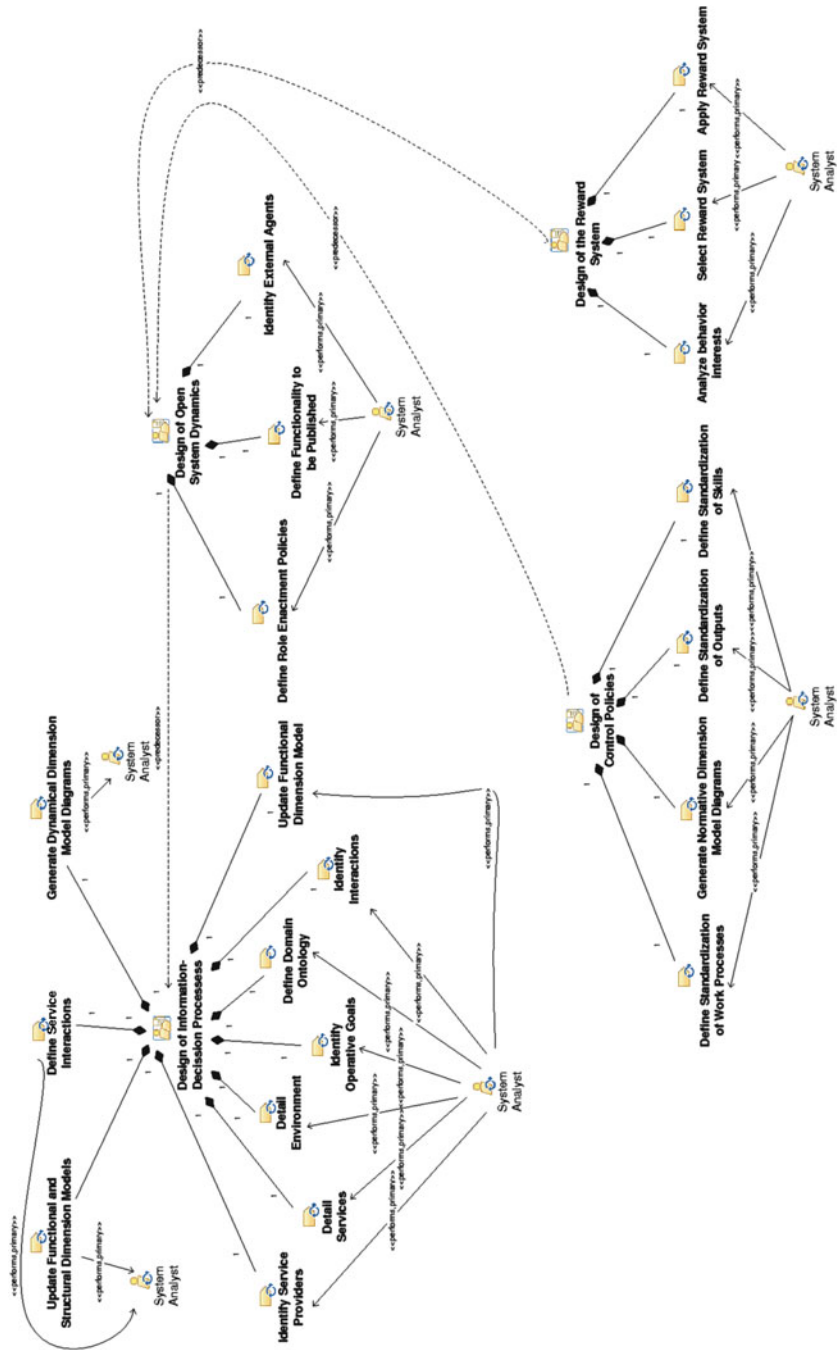


Fig. 25 Structural Diagram of the Organization Dynamics Design phase

Design of Information Decision Processes. The flows of information and adoption of decisions are described in order to determine how the information is processed and how agents work for obtaining the expected results (Fig. 26). More specifically:

- The specific functionality of services is specified, which implies (1) detailing services and related workflows for providing these services and splitting these workflows in concrete tasks; and (2) identifying the operative goals that represent the specific and measurable results that the members of an OU are expected to achieve. Thus, functional goals are split into operative goals, which are lately assigned to tasks.
- The flow of information of the system is specified, which implies: (1) defining the specific interactions between agents on the basis of services and their collaboration diagrams, in which agent messages are defined; and (2) defining the ontology of the domain, whose concepts will be used during interactions, taking into account the service inputs and outputs.

As a result, the diagrams of the *Dynamic Dimension Model* are defined. Moreover, *Structural* and *Functional Dimension Model* diagrams are updated. All tasks of this activity are detailed in Table 22.

Design of Open System Dynamics. In this activity (Fig. 27), the functionality offered by the agent organization as an open system is established, which includes the services that must be advertised as well as the policies of role enactment. In this sense, it is specified which is the functionality that must be implemented by the internal agents of the system and which functionality must be advertised in order to enable this functionality to be provided by external agents. Therefore, there are determined: (1) the services that have to be advertised; (2) the policies for role enactment, detailing the specific tasks for the *AcquireRole* and *LeaveRole* services of each OU; and (3) the identification of the internal and external agents.

All roles that need a control of their behaviors require a registration process inside the OU where they take part, so they are associated with external agents, who must request the *AcquireRole* service to play this role. On the contrary, roles like managers or supervisors are assigned to internal agents, since their functionality requires safety and efficiency. Figure 27 shows the products and services involved in this activity. Specific tasks of this activity are described in Table 23.

Design of Control Policies. In this activity (Fig. 28), the set of norms and restrictions needed for applying the Formalization Dimension of the organizational structure is defined. Three different types of standardization are considered: standardization of work processes, outputs, and skills.

The *standardization of work processes* implies specifying rules for controlling: (1) invocation and execution order of services; (2) precedence relationships between tasks; (3) deadline and activation conditions of services; and (4) service access to resources or applications. The *standardization of outputs* implies specifying norms for controlling the service products, based on minimum quality requirements,

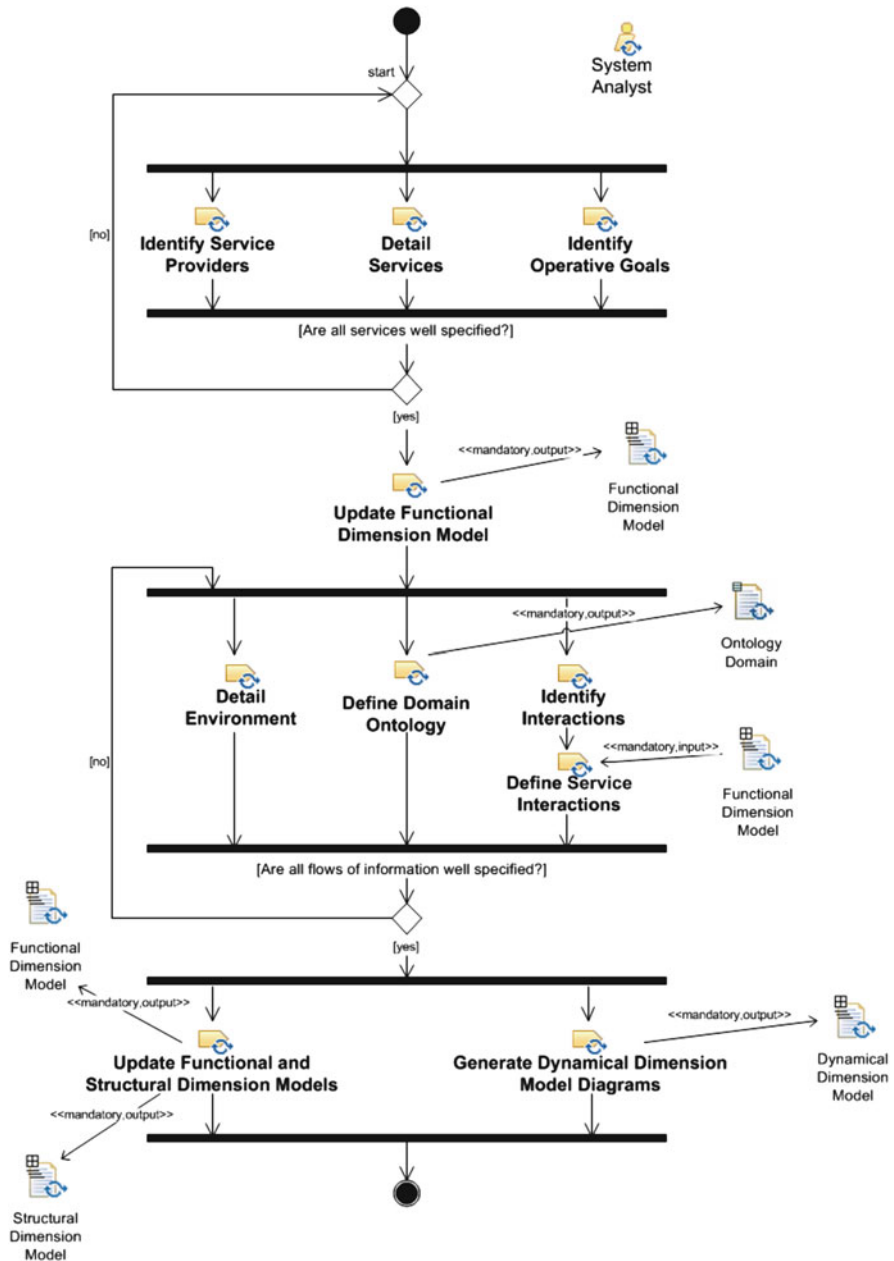


Fig. 26 Flow of tasks of the *Design of Information-Decision Processes* activity

perceived quality and previous established goals of productivity and performance. Finally, the *standardization of skills* is integrated in the role concept, which indicates the required knowledge and skills for an agent when playing this role. As a result,

Table 22 *Design of Information Decision Processes* activity description

Task	Task description
Identify Service Providers	<i>Functional Dimension Model</i> diagram is updated by adding the management roles for the OUs that offer services. Additionally, these roles must be added into the <i>Structural Dimension Model</i> diagram.
Detail Services	Every service that the design pattern contributed with must be split into tasks.
Identify Operative Goals	Functional objectives are split into operative goals that can be clearly specified.
Update Functional Dimension Model	After identifying service providers, detailing services and identifying operative goals, <i>Functional Dimension Model</i> diagram is updated.
Detail Environment	System interactions with clients and providers along resources, applications and agents are analyzed in order to determine their correction. New entities like resources and applications are defined to collect all the necessary information of the system.
Define Domain Ontology	System domain concepts are specified, taking into account the service inputs and outputs, and their relations. Additionally, existing ontologies can be reused in the system.
Identify Interactions	Relations between OUs are checked in order to create information flows that can facilitate their relationships and information exchange.
Define Services Interactions	An Interaction entity is defined for every Service, indicating its features and the participants of the interaction.
Update Functional and Structural Dimension Models	After identifying interactions, <i>Functional</i> and <i>Structural Dimension Model</i> diagrams are updated.
Generate Dynamical Dimension Model Diagrams	After defining service interactions, <i>Dynamical Dimension Model</i> diagrams are defined to represent them.

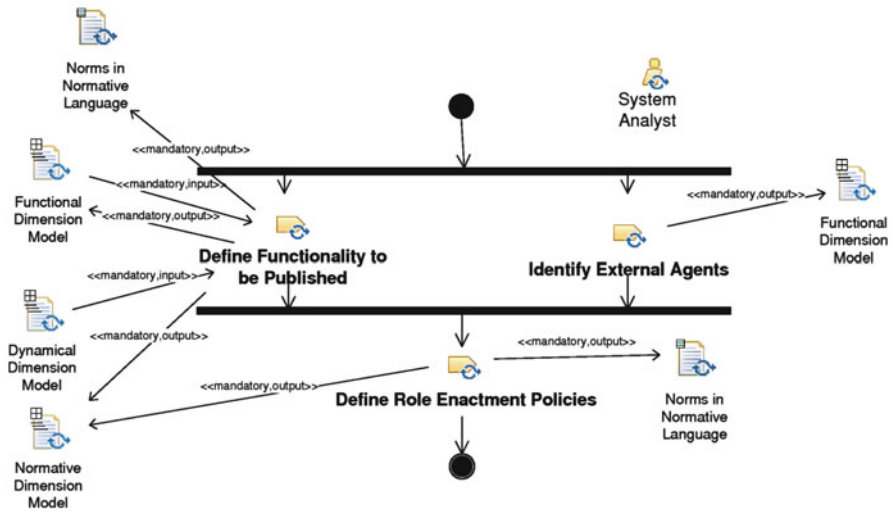


Fig. 27 Flow of tasks of the *Design of Open System Dynamics* activity

the diagrams of the *Normative Dimension Model* are generated, so the norms needed for controlling the behaviors of the members of the system are detailed. The specific tasks of this activity are described in Table 24.

Design of the Reward System. This activity defines the reward system needed for allowing the members of the organization to work toward its strategy (Fig. 29). Therefore, designer proceeds to:

Table 23 Design of Open System Dynamics activity description

Task	Task description
Determine Functionality to be published	Agents must be split into internal and external agents. Every service with the participation of external agents must be published by using a ServicePort entity.
Identify External Agents	There are two ways for identifying external agents: (i) every external agents has associated an internal agent who represents him, and it is familiar with the interaction protocols; and (ii) external agents act with the system by means of services, so we need to establish the service management and role enactment rules.
Define Role Enactment Policies	AcquireRole and LeaveRole services are split into tasks, taking in account activation conditions, preconditions and restrictions.

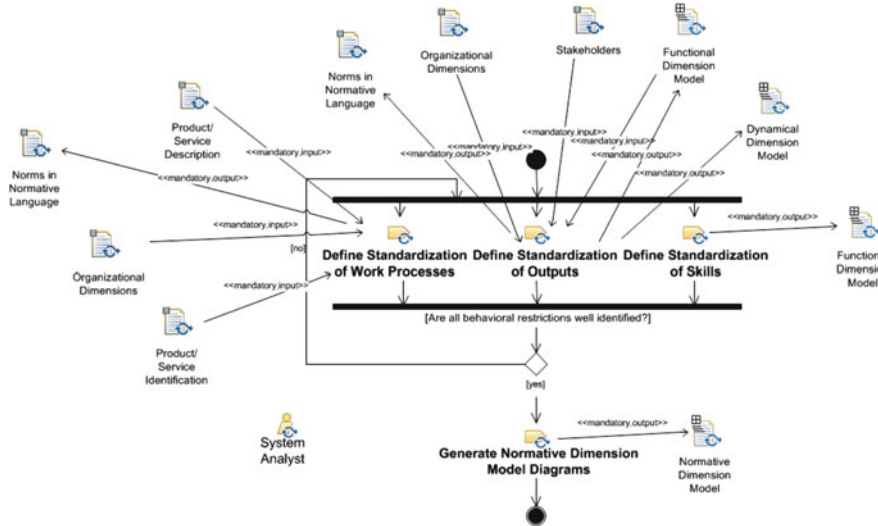


Fig. 28 Flow of tasks of the Design of Control Policies activity

Table 24 Design of Control Policies activity description

Task	Task description
Define Standardization of Work Processes	Implies specifying rules for regulating invocation and execution order of services, precedence relationships between tasks, required deadline for execution, access to resources, etc.
Define Standardization of Outputs	Implies establishing norms to control the results obtained by the service providers, by means of the minimum quality of the produced service or product; and the quality perceived by the stakeholders and to the previous established goals of productivity and performance.
Define Standardization of Skills	Implies defining the permissions, knowledge and aptitudes than an agent must acquire when taking a role.
Generate Normative Dimension Model Diagrams	Norms of the Organizational Units are specified, to allow them to establish an execution order between services; to define deadlines or activation/deactivation conditions for services; and to control the access to products and services, using ports.

- Analyze the types of behavior needed to promote: (1) the *willingness to join and remain* inside the system; (2) the *performance dependent on role*, so that the minimal levels of quality and quantity of work to execute are achieved; (3) the *effort on the minimal levels*, defining several measures of performance, efficiency

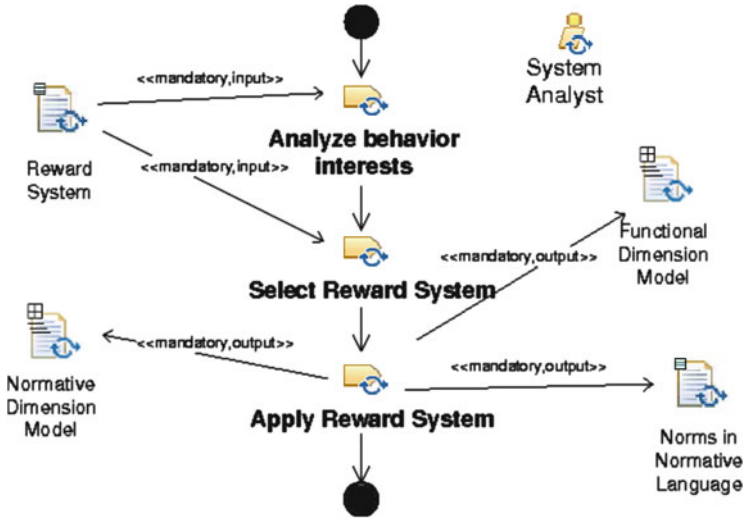


Fig. 29 Flow of tasks of the *Design of Reward System* activity

and productivity; and (4) the *cooperative behaviors* with the rest of members of the organization.

- Select the type of reward system to be used: (1) *individual rewards*, which establish several measures of behavior of unit members, allowing to promote their efforts over the average; (2) *group rewards* (competitive or cooperative), which establish group measures, rewarding unit members based on their specific performance inside the group; or (3) *system rewards*, which distribute certain gratifications (e.g., permissions, resource accesses) to all members of the system, trying to promote the participation inside the organization.
- Apply the selected reward system in the specific problem domain. Note that the methodological guideline enables the selection of the type of reward system that should be employed, but it does not detail the concrete mechanisms for implementing this system.

As a result, the diagrams of the *Normative Dimension Model* are updated, defining new norms and/or adding sanctions and rewards to the existing norms, according to the selected reward system. The specific tasks of this activity are described in Table 25.

Agent Design. In this activity, internal agents of the system are defined. Thus, each agent is related to the set of roles that it plays, specifying its functionality. Then, the *Structural* and *Functional Dimension Model* diagrams are updated.

2.4.3 Work Products

This section describes the products generated by the Organization Dynamics Design phase (see Table 26). Two models are generated: the *Dynamical* and *Normative*

Table 25 *Design of the Reward System* activity description

Task	Task description
Analyze behavior interests	Analyze the types of behavior needed to promote, like the willingness to join and remain inside the system; the performance dependent of roles; the effort on minimal levels; and the cooperative behaviors in order to create coalitions or groups formed by agents.
Select Reward System	Select the type of reward system to be used, by describing individual rewards (several measures of behavior of unit member, allowing to promote their efforts on the minimal levels), group rewards (can be cooperative or competitive and are several group measures, rewarding its members based on their specific performance inside the group) and system rewards (gratifications to all members of the system, just by belonging to this group, trying to promote the participation inside the organization).
Apply Reward System	The selected reward system is applied to the specific problem domain.

Table 26 Products for *Organization Dynamics Design* phase

Name	Description	Work product kind
Functional Dimension Model	See Table 3.	Behavioral
Structural Dimension Model	See Table 11.	Structural
Environment Dimension Model	See Table 11.	Behavioral
Dynamical Dimension Model	A diagram that uses the GORMAS notation that defines the role enactment process, the interactions between agents, as well as the mental states of the entities of the system.	Behavioral
Normative Dimension Model	A diagram that uses the GORMAS notation that describes normative restrictions over the behavior of the system entities, including organizational norms and normative goals that agents must follow, including sanctions or rewards.	Behavioral
Domain Ontology	A document that specifies the concepts of the organization domain, taking into account the service inputs and outputs.	Structured text
Norms in Normative Language	A document that contains a set of norms concerning entities of the system and written in normative language	Structured text
Reward System	A document containing a description of the reward system that our organization will apply.	Structured text

Dimension Model diagrams; whereas the *Functional* and *Structural Dimension Model* diagrams are updated. Three structured text documents are generated, regarding the domain ontology, the norms in normative language and the reward system of the organization. Figure 30 describes their relation with VOM elements.

Domain Ontology. To define this structured text document, system domain concepts are specified, taking into account the service inputs and outputs, and their relations. Moreover, if similar ontologies have already been proposed, correspondences between these ontologies and the specific one for the current problem must be established. GORMAS proposes an organization ontology that can be employed to describe a system. A detailed description of this ontology can be found in [17].

Norms in Normative Language. This structured text document describes the norms that regulate an organization. Norms are employed as mechanisms to limit the autonomy of the agents in complex systems in order to solve complex coordination

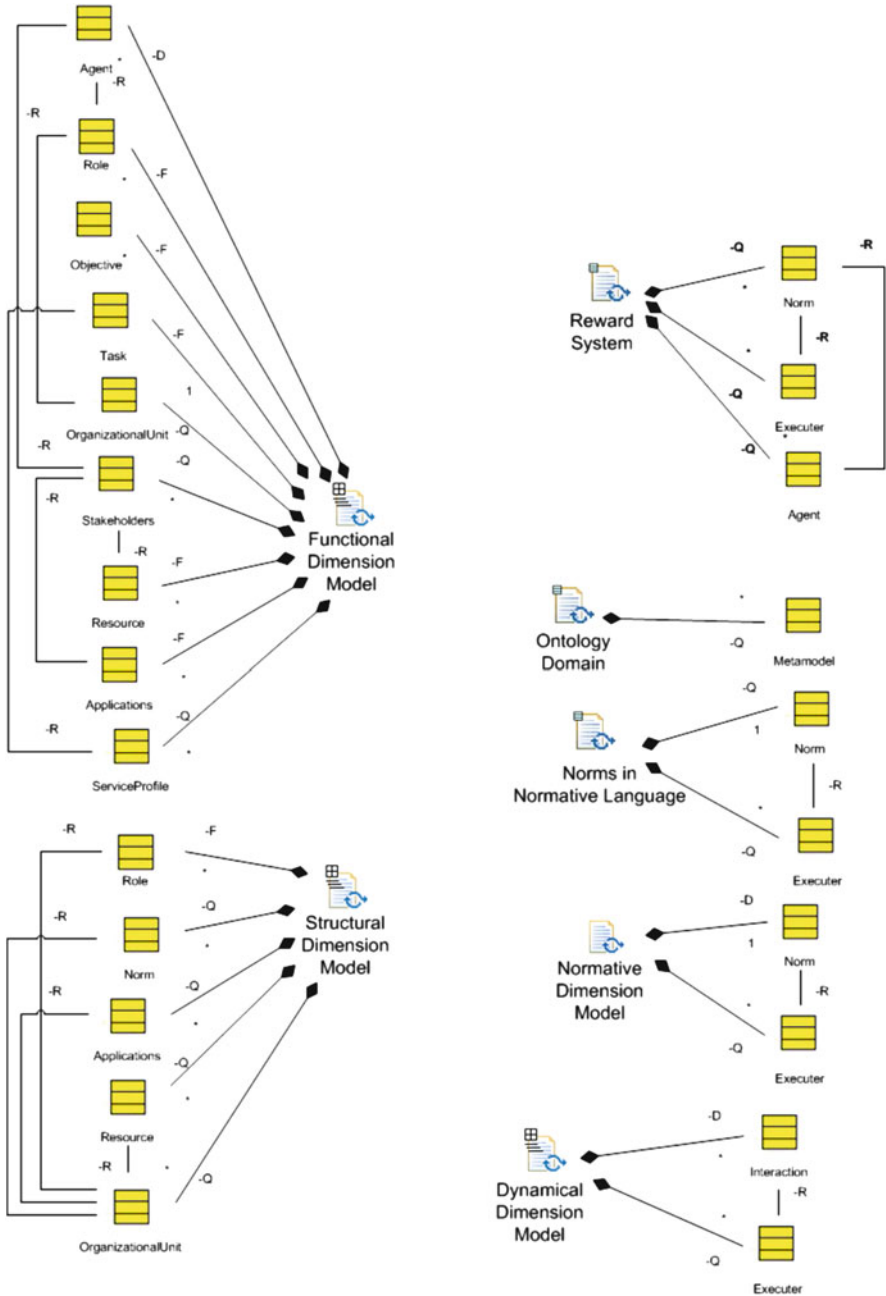


Fig. 30 *Organization Dynamics Design* phase. Relations between work products and metamodel elements. *D*: element introduced for first time; *F*: element refined; *Q*: element already defined; *R*: element related with another element



Fig. 31 Example of management roles identified on *Organizational Dynamics Design* phase, for the CMS case study

problems. Norms are presented using a normative language centered on requesting, serving and registering actions of a concrete service. A detailed description of this normative language can be found in [8].

Reward System. This document describes the behavior that should be promoted, specially when using an open system. Rewards can be individual, global or system rewards. GORMAS proposes an analysis about the kind of reward system that best fits the domain, but specific details of this system are left to the designer.

Functional Dimension Model. This diagram was previously instantiated on the *Mission Analysis* phase (see Sect. 2.1.3) and updated on *Service Analysis* (see Sect. 2.2.3) and *Organizational Design* phases (see Sect. 2.3.3). In the *Organizational Dynamics Design* phase, the *Functional Dimension Model* diagram is updated by adding management roles, new services that they can provide and operational objectives. As an example, Fig. 31 shows the added management roles of the CMS case study and relates them with the services they provide.

Structural Dimension Model. It was previously instantiated on the *Service Analysis* phase (see Sect. 2.2.3) and updated on the *Organizational Design* phase (see Sect. 2.3.3). In the *Organizational Dynamics Design* phase, the *Structural Dimension Model* is updated by adding new roles, resources, applications and their relationships, norms, agents and by relating OUs with the roles they contain. It is necessary to specify the inheritance of roles.

Dynamic Dimension Model. As stated before, this diagram defines the role enactment process, the interaction between agents, as well as the mental states of the system. In this phase of the methodology, interactions and their participant roles are identified, along with their relationships with objectives. Figure 32 shows an example for the CMS case study, including an interaction between two roles (Reviewer and Chair) using an interaction entity (Review Paper), and three interaction units (performatives) that uttered during the interaction. Also, it is depicted a condition (the reviewer is also an author) that will change the order to execute interaction units if it is fulfilled.

Normative Dimension Model. This diagram was previously defined as a description of normative restrictions over the behavior of the system entities, including organizational norms and normative goals that agents must follow. In this phase of the methodology, norms and their relations with roles, agents, services and objectives are described. Finally, the reward system is specified. Figure 33

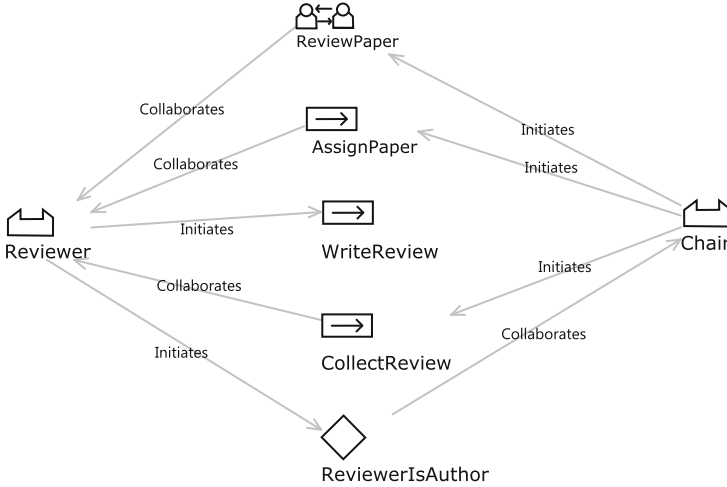


Fig. 32 Example of an interaction in the *Dynamical Dimension Model* for the CMS case study

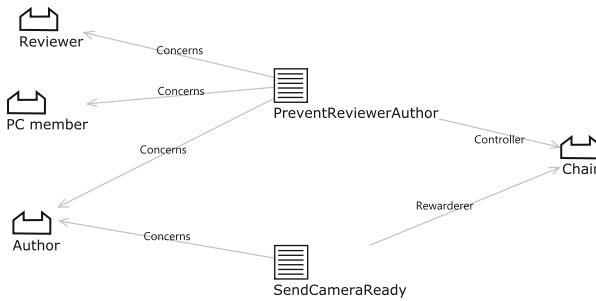


Fig. 33 Example of a *Normative Dimension Model* diagram for the CMS case study

shows an example of a *Normative Dimension Model* diagram, depicting two norms: (1) *PreventReviewerAuthor*, to avoid a PC member that is both a reviewer and an author to review his own papers, and it is supervised by the conference chair; and (2) *SendCameraReady*, a norm that authors have to follow to send a correct camera-ready version of their work prior to its final publication in the conference proceedings.

3 Work Product Dependencies

Figure 34 describes the dependencies among the different work products. For example, the *Organizational Mission* document is used to identify the stakeholders of the organization that will be described in the *Stakeholders* document. Additionally,

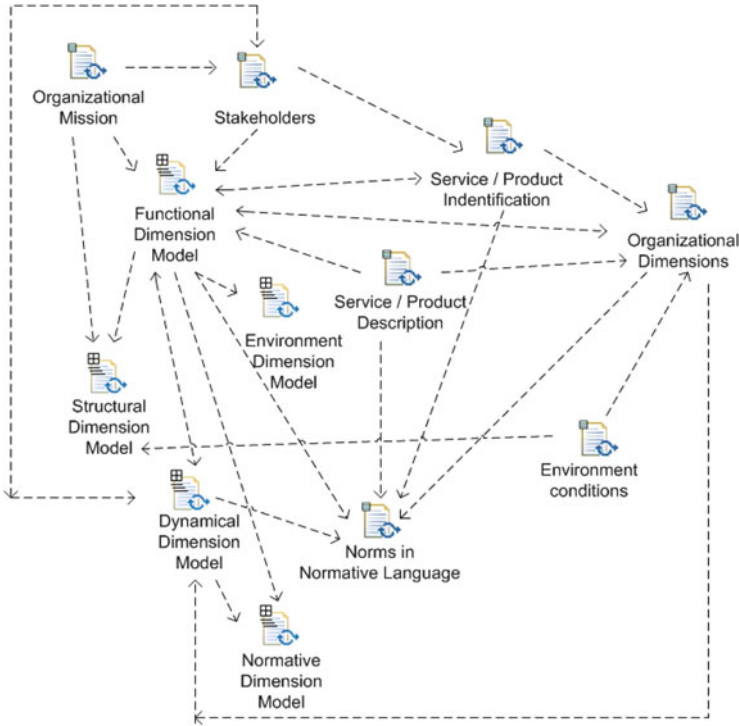


Fig. 34 Work product dependency diagram

the *Organizational Mission* document is used as a guidance to define the *Functional* and *Structural Dimension Model* diagrams.

Acknowledgements This work was supported by TIN2009-13839-C03-01 and TIN2012-36586-C03-01 projects of the Spanish government, and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

References

1. Lemaître, C., Excelente, C.B.: Multi-agent organization approach. In: Proceedings of II Iberoamerican Workshop on DAI and MAS (1998)
2. Argente, E., Julian, V., Botti, V.: MAS modeling based on organizations. In: Agent-Oriented Software Engineering IX, pp. 16–30. Springer, Berlin (2009)
3. Moreno-Luzón, M.D., Peris, F.J.: Strategic approaches, organizational design and quality management: integration in a fit and contingency model. *Int. J. Qual. Sci.* 3(4), 328–347 (1998)
4. Cossentino, M.: From requirements to code with the PASSI methodology. In: Agent-Oriented Methodologies, pp. 79–106. Idea Group Publishing (2005)
5. Omicini, A.: SODA: societies and infrastructures in the analysis and design of agent-based systems. In: Agent-Oriented Software Engineering, vol. 1957, pp. 185–193. Springer, Berlin (2001)

6. Pavón, J., Gómez-Sanz, J.: Agent oriented software engineering with INGENIAS. In: Multi-agent Systems and Applications III, pp. 394–403. Springer, Berlin (2003)
7. Garcia, E., Argente, E., Giret, A.: A modeling tool for service-oriented open multiagent systems modeling tool. In: 12th International Conference on Principles of Practice in Multi-agent Systems. PRIMA 2009. Lecture Notes in Computer Science, vol. 5925, pp. 345–360. Springer, Berlin (2009)
8. Criado, N., Julian, V., Botti, V., Argente, E.: A norm-based organization management system. In: Coordination, Organizations, Institutions, and Norms in Agent Systems. Springer, Berlin (2009)
9. Giret, A., Julian, V., Rebollo, M., Argente, E., Carrascosa, C., Botti, V.: An open architecture for service-oriented virtual organizations. In: PROMAS 2009 Post-Proceedings, pp. 1–15. Springer, Berlin (2010)
10. Cossentino, M.: Design Process Documentation Template. Technical report, FIPA (2011)
11. Seidita, V., Cossentino, M., Gaglio, S.: Using and extending the spem specifications to represent agent oriented methodologies. In: Agent-Oriented Software Engineering IX: 9th International Workshop, AOSE 2008 Estoril, Portugal, May 12–13, 2008 Revised Selected Papers, vol. 5386, p. 46. Springer, New York (2009)
12. Esparcia, S., Centeno, R., Hermoso, R., Argente, E.: Artifacting and regulating the environment of a virtual organization. In: 2011 23rd IEEE International Conference on Tools with Artificial Intelligence, pp. 547–554. IEEE, Boca Raton (2011)
13. Ricci, A., Viroli, M., Omicini, A.: Give agents their artifacts: the A&A approach for engineering working environments in MAS. In: Proceedings AAMAS, p. 150 (2007)
14. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: a fully configurable multi-user and multi-tool CASE and CAME environment. Lecture Notes in Computer Science, vol. 1080, pp. 1–21. Springer, Berlin (1996)
15. Argente, E., Julian, V., Botti, V.: Multi-agent system development based on organizations. Electron. Notes Theor. Comput. Sci. **150**(3), 55–71 (2006)
16. Mintzberg, H.: The Structuring of Organizations. Prentice-Hall (1979)
17. Criado, N., Argente, E., Julian, V., Botti, V.: Designing virtual organizations. In: 7th International Conference on Practical Applications of Agents and Multi-agent Systems (PAAMS2009). Advances in Soft Computing, vol. 55, pp. 440–449. Springer, Berlin (2009)

INGENIAS-Scrum

Juan C. González-Moreno, Alma Gómez-Rodríguez,
Rubén Fuentes-Fernández, and David Ramos-Valcárcel

Abstract

This chapter introduces the definition of an agile process for the INGENIAS methodology. It is based on a well-known development process: Scrum. The process adopts the iterative and fast plan presented originally by the methodology and uses some of the activities and most of the work products of the INGENIAS proposal with the Unified Development Process (UDP) (introduced in a previous chapter). The new approach is also based on the INGENIAS metamodel, but it is more focused on code development than on system specification. It takes advantage of the INGENIAS Agent Framework (IAF), which is part of the INGENIAS Development Kit (IDK). As this approach uses the same metamodels than the UDP based, there are not great differences in the models of the case study, but, instead, the organization of the work products and the time spent to get the final results is quite different.

1 Introduction

Agent-Oriented Software Engineering (AOSE) methodologies have adopted different development processes depending on their particular needs and what the prevalent processes in mainstream software engineering [1] were. From [2], the process models used in AOSE can be classified into the following groups:

- *Waterfall*. The waterfall-like process models prescribe a sequential, linear flow among phases. Although some waterfall models include some kind of return to

J.C. González-Moreno • A. Gómez-Rodríguez (✉) • D. Ramos-Valcárcel
Universidad de Vigo, Campus As Lagoas s/n, 32004 Ourense, Spain
e-mail: alma@uvigo.es; jcmoreno@uvigo.es; david@uvigo.es

R. Fuentes-Fernández
Universidad Complutense de Madrid, Avda. Complutense s/n, 28040 Madrid, Spain
e-mail: ruben@fdi.ucm.es

previous phases, in real projects this return occurs very late and the cost of any change in the initial specifications is unacceptable.

- *Evolutionary and Incremental*. The stages of this process model category consist of expanding increments of an operational software product, with the direction of evolution being determined by operational experience. Most of AOSE methodologies have process models within this category.
- *Transformation*. This software development may be seen as a sequence of steps that gradually transform a set of formal specifications into an implementation.
- *Spiral*. It organizes the development process in a cyclic way. Each cycle of the spiral consists of four phases: determining the objectives, evaluating the risks of these objectives, developing and verifying, and reviewing the previous stages.

Among all of them, agile processes [3] can be included in *Evolutionary and Incremental* class. This means that the development is organized to deliver functional increments of high value for the customer in short periods of time. These processes are code-oriented in the sense that they consider the main product of development is functional code.

Scrum [4] is an empirical agile project management framework. It relies on self-organizing, empowered teams to deliver the product increments, but also on a customer, or *Product Owner*, that must provide the development team with a list of desired features, using business value as the priority mechanism. In real life, *Scrum* is a mechanism in the sport of rugby for getting an out-of-play ball back into play. The term was adopted in 1987 by Ikujiro Nonaka and Hirotaka Takeuchi to describe *hyper-productive development*. Jeff Sutherland developed the Scrum process in 1993 while working at the Easel Corporation, and Ken Schwaber formalized the process in the first published paper on Scrum at OOPSLA 1995 [5].

The Scrum process framework is specially well suited for Knowledge Engineering Developments based on the use of multi-agent systems (MAS). This chapter addresses its integration with the INGENIAS methodology [6] using some specific tools. INGENIAS is a general purpose AOSE methodology that adopts a model-driven development (MDD) [7]. Its development is organized around the definition of models that are semi-automatically transformed into different products, for example, code, tests, and documentation.

The INGENIAS modeling language is defined through a metamodel, following common practices in MDD [8]. It is the basis to define the transformations that generate the code of support tools and MAS in INGENIAS. This metamodel has proved its capability and maturity in the development of MAS for different domains [6, 9, 10]. The main support tool is the INGENIAS Development Kit (IDK) [11], which includes a graphical editor for MAS specifications compliant with the INGENIAS metamodel, and integrates plug-ins to provide additional functionalities. Among these plug-ins, those to manage developments using the IAF are the most relevant ones.

The IAF [12] is a set of libraries that facilitate the implementation of INGENIAS agents built on top of the Java Agent DEvelopment Framework (JADE) [13]. The tool has been proposed from the experience in the application of the INGENIAS methodology over several years to enable a MDD.

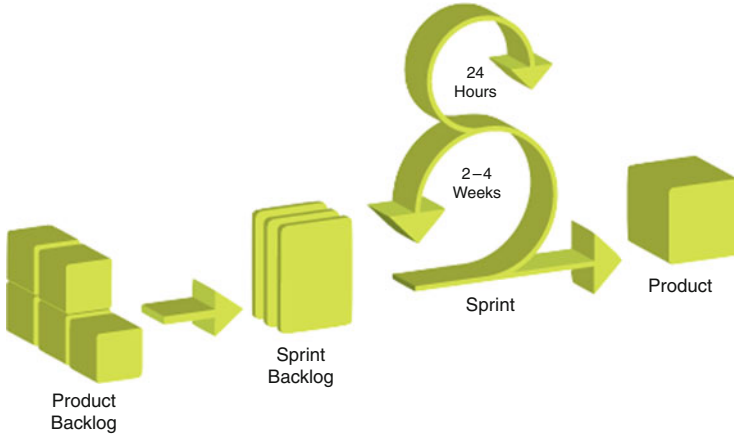


Fig. 1 Scrum life cycle

The IAF is fully integrated in the IDK [11] and provides facilities for project checking, code generation, debugging, and documentation. This means that following the guidelines of the IAF documentation, an experienced developer can focus most of its effort on specifying the system, converting a great deal of the implementation in a matter of transforming automatically specifications into code. This quick transition from specifications to code facilitates the adoption of agile processes such as Scrum [4].

The rest of this chapter fully specifies the development process based on the Scrum framework and the use of the IAF proposed in [1]. The *INGENIAS with the UDP chapter* in this book also considers the INGENIAS methodology, but with its original process based on the Unified Development Process (UDP) [14]. The interested reader can find additional discussion on the INGENIAS methodology in that chapter.

1.1 The INGENIAS-Scrum Process Life Cycle

Scrum [4] is a suitable process for contexts where developers consider short and incremental development cycles focused on running code. This scenario is enabled by the IAF [12] for the INGENIAS methodology (Fig. 1).

Scrum usually addresses the production of a software version (i.e., a *release*) every couple of months. Potential features to accomplish in a release are collected in the *product backlog* and prioritized. A *product owner*, who represents the customer, is in charge of updating this backlog. The release is produced in a number of iterations called *sprints*. Each sprint usually lasts from 2 to 4 weeks. The sprint content is defined by the *product owner* taking into consideration both priorities and team capabilities. The team defines the tasks required to develop the functionalities



Fig. 2 Scrum phases

selected for the sprint. Within a sprint, progress checkpoints are performed during the *daily scrums*. This enables the *scrum master* to assess work progress regarding the sprint goals, and to suggest adjustments to ensure the sprint success. At the end of each sprint, the team produces a potentially releasable *product increment*, whose evaluation drives the *backlog update* for the next sprint.

All this work is planned in two kinds of phases (see Fig. 2): the *Preparation* phase and the *Sprint phases*. The Preparation phase includes all the activities to be done before the first sprint, focused on the elaboration of the product backlog. The Sprint phases covers the execution of all the sprints required to release the final product. The following sections discuss the key issues of the integration of this process with INGENIAS through model-driven practices.

1.2 Metamodel

The INGENIAS metamodel describes the elements that constitute a MAS according to this methodology. This metamodel is explained in detail in the *INGENIAS with the UDP chapter*, so this chapter only offers a short overview. A detailed discussion of the metamodel can be found in [15, 16].

This metamodel considers several viewpoints (i.e., *models*) that correspond to different aspects of a MAS. Figure 3 (shared with the *INGENIAS with the UDP chapter*) shows them and their relationships. The viewpoints are

- The MAS organization, that is, the *Organization Model*.
- The definition, control and management of each agent mental state, that is, the *Agent Model*.
- The tasks and goals assigned to each agent, that is, the *Tasks and Goals Model*.
- The agent interactions, that is, the *Interaction Model*.
- The external environment where agents interact to satisfy their goals, that is, the *Environment Model*.

1.3 Guidelines and Techniques

The process proposed for INGENIAS [6] must be able to guide developers to obtain the products that meet the development goals. Being INGENIAS a MDD-oriented methodology, this means that once a specification conforming to the metamodels is defined, the products are generated through transformations and using, if needed, additional resources (e.g., code templates, external code, or scripts).

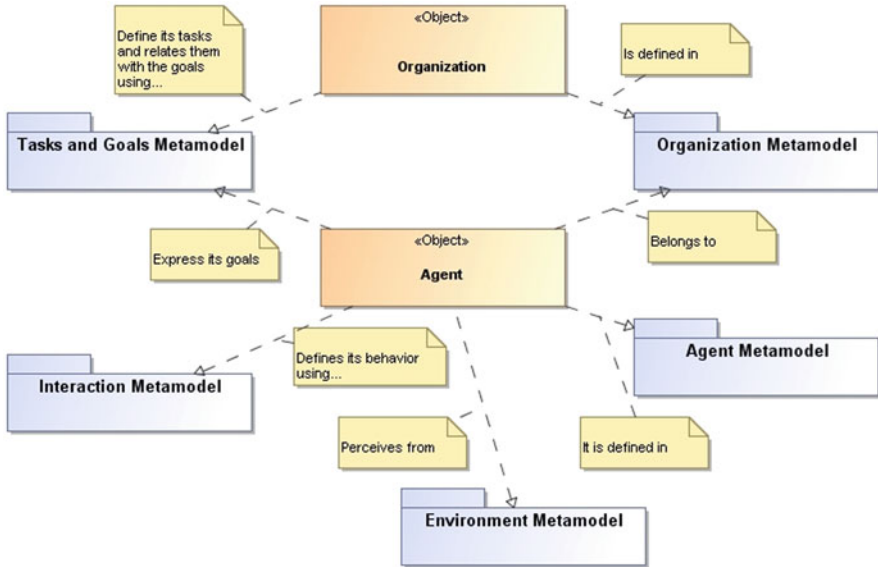


Fig. 3 Metamodel of the INGENIAS methodology

1.3.1 Scrum and MDD

Although Scrum [4] does not describe the engineering activities required for product development, the application of Scrum to INGENIAS does it. INGENIAS-Scrum relies on the use of the IAF [12], which allows to combine the classic approach for coding applications with the modern techniques of automatic code generation.

An IAF compliant specification must describe in a sound way the following aspects:

- *Agents*. It is the main building block in an INGENIAS MAS. An agent is defined completely when its perception, main tasks, and coordination means are described.
- *Tasks*. They are the basic units of agent’s behavior. A task modifies the agent mental state and performs actions over the applications.
- *Interactions*. They describe how agents coordinate in order to satisfy their goals. Coordination is defined in terms of transferred information units between agents.
- *Deployment*. It expresses how agents are instantiated and initialized in the final system.
- *Tests*. The test define the testing units that the MAS should pass through to ensure the system will achieve its goals and will work in a sound way along its life cycle.

This information is provided through several INGENIAS models (see Sect. 1.2). The core ones, in this case, are the *Agent Model*, the *Interaction Model*, and the *Tasks and Goals Model*. The *Environment Model* may also be considered, mainly to point out the relationships with external elements. Nevertheless, its elements can be

defined by using the rest of models, particularly using the Agent Model (as shown in Fig. 3). Regarding the *Organization Model*, it is not strictly necessary to use it, but it may help to have a global vision of the final solution in what refers to *agents and roles that participate*, and *responsibilities in the satisfaction of the main goals*. Moreover, it may be used to reflect many of the product items proposed in the original Scrum framework.

Note that although not all the INGENIAS models are strictly necessary to create an IAF compliant specification, this does not mean that the models do not provide information usable for code generation. For instance, the concepts of *Autonomous Entity* and *Organization* presented in Table 2 of the INGENIAS-UDP chapter could have a translation to code in the form of new capabilities or relationships.

1.3.2 IDK Considerations

The IDK [6, 11] is a graphical tool for MAS model creation. Although it is fully adapted to cover the INGENIAS metamodel, the editor offers several features that allow the adaptation to new metamodels or target platforms. At present many of such features could be used to accurately support the documentation provided for some non-standard artifacts like *user stories*, *cards*, *priorities*, *estimations*, etc. For instance, a good practice to cover users stories is to associate a package to each one and use the text description to introduce a full description of the story. This text description may be used also to include information about priority, velocity, tasks assignment, etc. After, this description can be refined and formalized by using the diagrams of the metamodel that is associated with such package. Another good complementary practice is to use a naming protocol for each package to document the version and level of development for such package. The use of that package naming convention also facilitates the implementation of an iterative and incremental process for any system. The capability of the tool, which allows to move diagrams from a package to another one, is also fundamental to facilitate such kind of process.

1.3.3 IAF Considerations

During the definition of the product backlog using the INGENIAS models, participants must take into account that an agent in the IAF performs a simple deliberation cycle:

- Identify new tasks to schedule and tasks to remove from the schedule
- Execute one scheduled task

This cycle omits the classic perception step as it is not needed. In fact, the agent can receive new information at any moment and this does not cause internal conflicts. The incorporation of new pieces of information does not imply a change in already scheduled tasks, though it may mean the incorporation of new tasks.

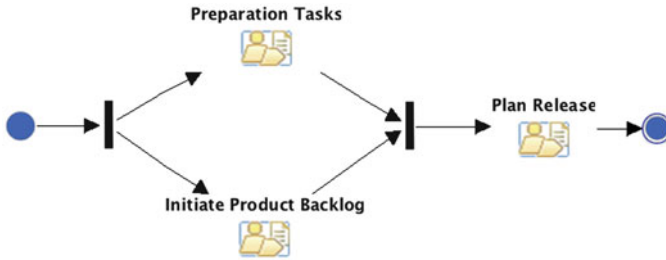


Fig. 4 The Preparation phase flow of activities

2 Phases of the INGENIAS-Scrum Process

2.1 Process Documentation: Preparation Phase

As the Scrum framework does not take into account all the aspects of project planning, the main goal of the Preparation phase is to define the *Plan Release*. This implies establishing the features to address in the release and the estimated tasks to achieve them.

Accordingly with the original Scrum framework, the *Preparation* phase comprises the following activities: *Initiate Product Backlog*, *Plan Release*, and *Preparation Tasks*. Figure 4 shows the workflow of these activities, and Fig. 5 shows the structure of the phase in terms of the tasks and work products to be accomplished. Section 2.1.1 provides information about the roles responsible of each task and the kind of responsibility they assume. Section 2.1.2 details the activities and their tasks, and a description of the work products related with those tasks appears in Sect. 2.1.3

It is interesting to note that several tasks of this phase have to be done using the IAF integrated in the IDK. The main guideline to apply correctly both INGENIAS tools is the technical report [12], which comes with the IDK distribution.

2.1.1 Process Roles

Following the Scrum approach, the roles implied in the Preparation phase are the *Product Owner*, the *Scrum Team*, and the *Stakeholder*. The following sections describe them in detail.

Product Owner

This role represents the *customer* during the development and is in charge of the *Initiate Product Backlog* activity. This activity produces a *backlog* containing enough features to allow the startup of a number of sprints. The role also participates in the *Plan Release* activity to set up the *release's* initial planning in order to launch the sprints.

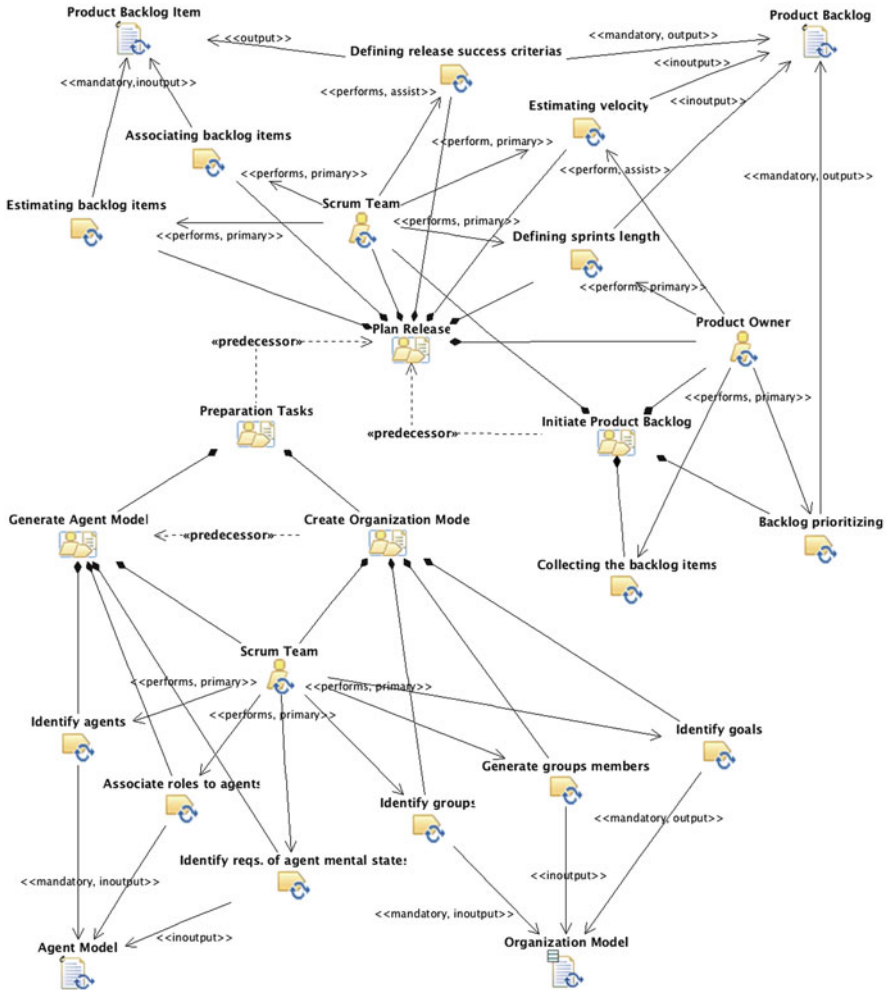


Fig. 5 The Preparation phase described in terms of activities and work products

To obtain the *backlog*, the Product Owner identifies a list of items, prioritizes them, and includes them in the *Product Backlog*. In a traditional approach, these items (on the preparation phase) consist of *User Stories* that details the requirements of the system. In the INGENIAS-Scrum approach, the Product Backlog will consist of an Agent Model and Organization Model (optionally). User stories are documented using the IDK by the Scrum Team and are stored on the description part (see Fig. 10) of the goals identified following the Product Owner indications. As this role is in charge of defining the functionality of the application, it is desirable that she/he could also determine the acceptance tests while prioritizing each item.

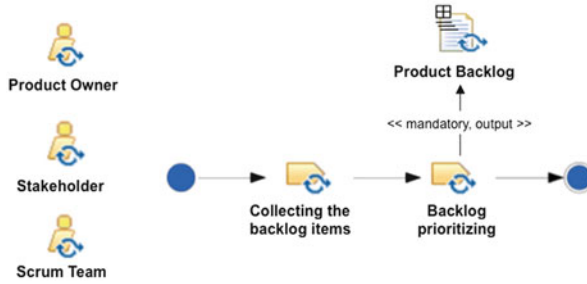


Fig. 6 Workflow of the *Initiate Product Backlog* activity

Scrum Team

It is mainly responsible of the *Preparation Tasks* activity that will be done jointly with *Initiate Product Backlog*. This role also participates actively in the rest of the activities of the *Preparation* phase. The main goal at this point of the life cycle is to specify an initial view of the system to be developed using the IDK and the INGENIAS metamodels (see Fig. 3). This view comprises the development of an *Organization Model* (optionally) and/or an initial *Agent Model* (mandatory) with several *Agent Diagrams*, including the *agents*, *roles*, and *goals* required by the Product Owner for the MAS. This initial view must include enough information to provide a planning for the first 2–3 sprints.

Stakeholder

The participation of this role is optional and represents, as usual, anyone that does not directly participate in the project but can influence the product under development.

2.1.2 Activity Details

Initiate Product Backlog

The product backlog is a prioritized list of features that contains short descriptions of all the functionality desired for the product. For INGENIAS, these requirements are described using the IDK, either by adapting a specification from a previous project or by defining a new one.

When using Scrum, it is not necessary to start a project with a lengthy, upfront effort to document all the requirements. Typically, a Scrum Team and its Product Owner begin by writing down everything they can think of easily, and this constitutes the first version of the product backlog. This is almost always more than enough for a first sprint (when documented accordingly with the IDK). The product backlog (i.e., the INGENIAS specification) is then allowed to grow and change as more information is learned about the product and the customers. The flow of tasks needed to perform this activity is described in Fig. 6 and detailed in Table 1.

Table 1 Description of the tasks of the *Initiate Product Backlog* activity

Activity	Task	Description	Involved roles
Initiate <i>Product Backlog</i>	Collecting the backlog items	The <i>Product Owner</i> establishes a first list of requirements (a requirement becomes a <i>Product Backlog Item</i>) that she/he wish to be implemented for the end of the release. After this, a workshop is organized in which the <i>Product Owner</i> participates, as well as, the whole team and the stakeholders (if necessary). The <i>Product Owner</i> introduces the backlog draft and the assistants may suggest additional items.	<i>Product Owner</i> <i>Scrum Team</i> <i>Stakeholder</i>
	Backlog prioritizing	The full list of backlog items is prioritized by the <i>Product Owner</i> . Items with higher priority must be addressed first in the release. If a backlog contains a lot of items, the definition of <i>Themes</i> , and the association of one of them to each item, is suggested. After this, the <i>Product Owner</i> prioritizes the <i>themes</i> .	<i>Product Owner</i> <i>Scrum Team</i> <i>Stakeholder</i>

Preparation Tasks

This activity is performed by the *Scrum Team* in order to prepare the next meetings with the *Product Owner* and the *Stakeholders*. It creates an ongoing *INGENIAS* specification intended to reorganize, formalize, and structure the preliminary specifications in the product backlog. This implies that this activity is more complex than the original *Scrum* one. It requires to perform several tasks which are common to the *INGENIAS* activities *Create the Organization Model* and *Create the Agent Model* that have been detailed on the *INGENIAS-UDP chapter*. Figure 7 shows the recommended flow of such tasks. As it can be easily seen, they differ a little from the ones proposed on the *INGENIAS-UDP chapter*.

At this stage of the process, the considerations introduced at Sect. 1.3.3 must be taken into account for creating and modifying the *Organization* and *Agent Models*.

Plan Release

The previous activities deliver an initial *INGENIAS* specification. With this information, the *Scrum Master*, jointly with the *Product Owner*, can establish the *Plan Release*, which defines the sprints required to meet the goals. Several tasks have to be performed, including the definition of the release success criteria, the estimation of the backlog items and the velocity of the sprint, the definition of the sprints length, and the association of the backlog items. Figure 8 shows the tasks workflow for this activity that Table 2 further explains.

2.1.3 Work Products

Work Product Kinds

The unique product produced by the Preparation phase is the initial *Product Backlog*. A *Product Backlog* is a set of items: the *Product Backlog Items* (PBI). A PBI in the *INGENIAS-Scrum* approach is a composed work product of type

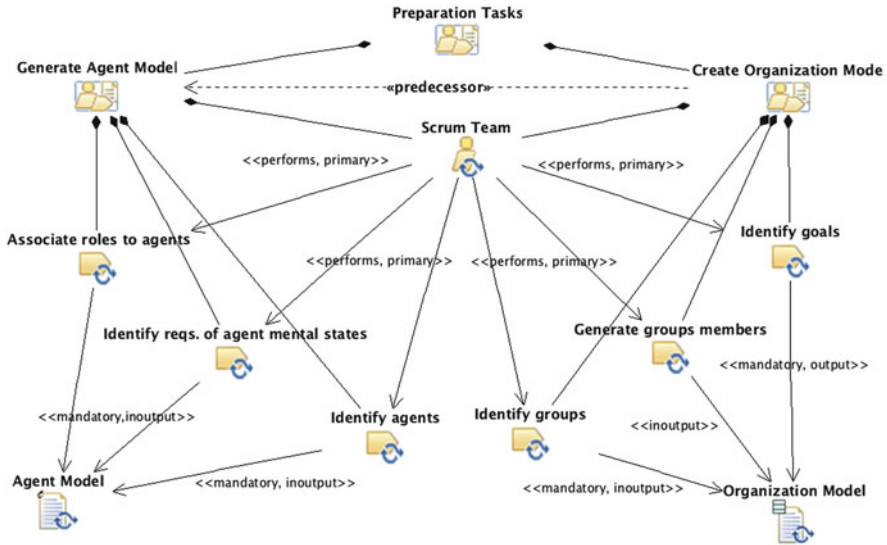


Fig. 7 Workflow of the *Preparation Tasks* activity

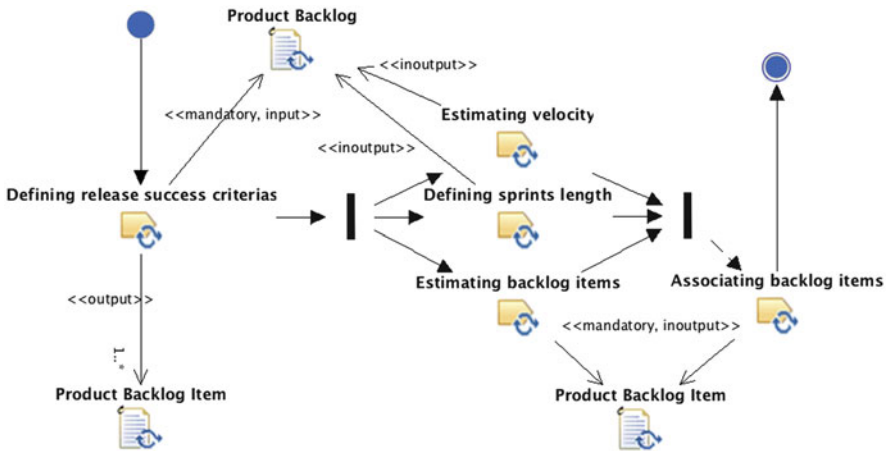


Fig. 8 Workflow of the *Plan Release* activity

Composite (Structured + Structural + Behavioural). In the original proposal, each PBI can be described as free text and refers to one of the following kind of items:

- *Features*. A feature usually is described with *User Stories* that comprise a short and simple description of the desired functionality from the user’s perspective. An example would be, “As an author, I can send a paper to the conference and I will be informed about its reception and the identification number assigned to it”

Table 2 Description of the tasks to be done during the *Plan Release* activity

Activity	Task	Description	Involved roles
<i>Plan Release</i>	Defining <i>release success criteria</i>	This task pursues to establish the criteria to consider a Release successfully finished by the Product Owner. The Scrum Team (the Scrum Master, in particular) must take into account that there are 2 types of releases: <i>End-Date Driven Release</i> , which must be available to the end users before a deadline; and <i>Feature Driven Release</i> , in which the release finishes when all of the requirements are implemented.	Product Owner Scrum Team
	Estimating <i>backlog items</i>	Estimation must be performed by the team item per item. It is a good idea to start by the highest priority items. In this activity, it is usual to use as range of values for estimations the <i>Fibonacci Numbers</i> : 1, 2, 3, 5, 8, and 13. Regarding the <i>estimation techniques</i> , the collective ones are preferred.	Scrum Team
	Defining <i>Sprint length</i>	Although historically the length of a sprint is 30 days, INGENIAS-Scrum recommends 15 days, although a week or 21 days are also acceptable lengths, according to the difficulty of the work and the human resources available.	Scrum Team
	Estimating <i>velocity</i>	The <i>velocity</i> (i.e., the number of items that could be finished) is calculated as the sum of all items checked and approved as fully implemented in a Sprint. This activity makes an estimation of the expected team velocity. It should be done automatically because it is supposed that the team has worked together in previous projects using the INGENIAS tools and the JADE platform. Nevertheless, the velocity can be manually estimated for the first time and adjusted in the next sprints.	Scrum Team
	Associating <i>backlog items to sprints</i>	This task takes into consideration the parameters obtained from the previous tasks. A slight adjustment of items prioritization may be done at this stage, so the velocity can be better adjusted to the team features. It is not required to make this association for all sprints of the release at the beginning, but it should be done for the 2 or 3 first ones.	Product Owner Scrum Team

- *Bugs*. In general, a bug does not present any real difference with a new feature (in the description), and it will be treated as if a new feature or a change is to be incorporated.

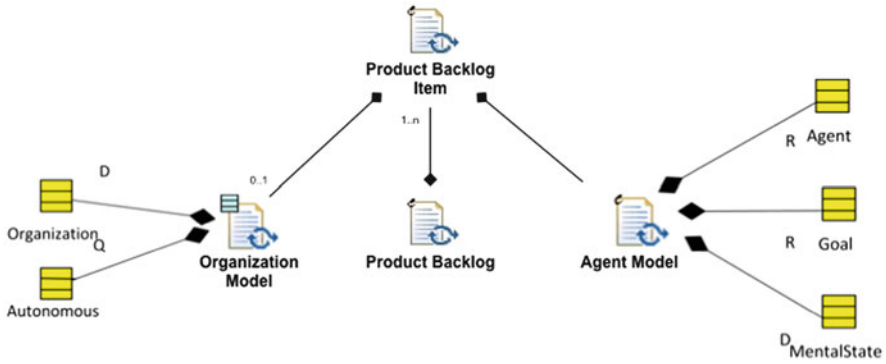


Fig. 9 The Preparation Work Product Model structure

- *Technical work.* This kind of item introduces detailed aspects related with the deployment of the system. An example of technical work would be: “Upgrade the Conference Server Operating System to OS X Mountain Lion” or “Use a version up to 2.0 of the Mail Manager system”.
- *Knowledge acquisition.* This item is related to obtain the skills and knowledge required for the project. The item reflects some kind of external knowledge that is necessary to better understand the problem or to apply a particular solution. An example could be asking the team members the study and/or the selection among several Java libraries to determine the most accurate one for the system or to find a solution to the problem restricted to the use of such library.

User stories usually determine several goals that must be satisfied by agents playing certain roles. Bugs must be documented by modifying the description of the related user story or adding a new one. Regarding technical works and knowledge acquisition items, a good practice is to document them by defining a test or proof for a part of the system, or by specifying the concrete API needed to solve the problem jointly with the agent that is in charge of applying such solution.

In INGENIAS-Scrum, a Product Backlog is described by a set of INGENIAS models as reflects Fig. 9. For a full description about the *Organization* and *Agent Model*, see *INGENIAS-UDP chapter*. At the Preparation Phase, each PBI, as described by the Product Owner and the Stakeholders, is specified as a Package description as pointed in Sect. 1.3.2, which is refined by the Scrum Team by using agent and/or organization diagrams to determine the Agent and/or Organization Models. Agent Model is mandatory at this phase, but could be complemented with an Organization Model to provide the responsibilities of groups, agents, and roles in order to satisfy the identified goals. Figure 9 shows the relationships between the MAS meta-model elements.

Product Backlog Item

In general, a PBI has to include the following general attributes:

- Estimated value, frequently calculated relatively to other items using priorities.
- Estimated development cost.

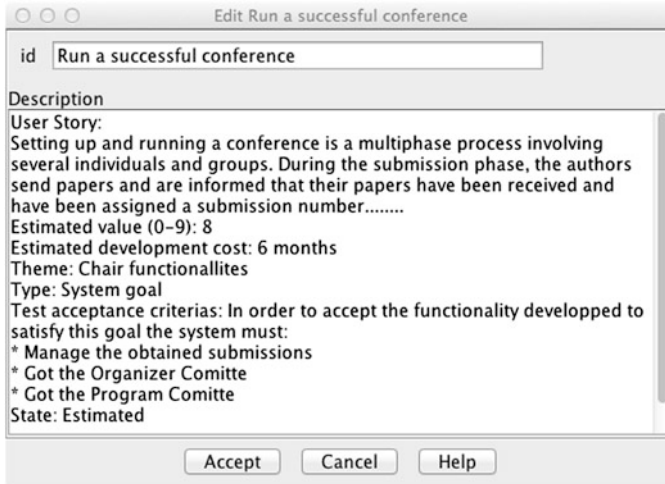


Fig. 10 Attributes of a Product Backlog Item specified in the description of an INGENIAS element

- Eventually the *theme* (i.e., functional domain) it belongs to.
- Eventually its type, which can be defined as: *system goal of the project*; *bug resulting from the process*; or *non-functional requirement*.
- The associated criteria for test acceptance.

Through this phase of the development process, a PBI can reach the following states: *created*, *estimated*, *planned* (associated to a future sprint), *associated to current sprint* (implementation is ongoing), and *done*.

At the end of the Preparation Phase, following the INGENIAS-SCRUM approach, all the PBI identified jointly with their state must have at least one *Goal* associated and its features must be documented in the description of the related package and detailed on the related goal description. Figure 10 shows a snapshot of how this information could be documented using the IDK.

2.2 Process Documentation: The Sprint Phases

The *Sprint* is the main phase of a Scrum project. The INGENIAS-Scrum process is iterative and incremental, which means that the project is split into consecutive sprints that will be done with the help of the IDK and IAF tools. In the original Scrum framework, each sprint is timeboxed between 1 week and a calendar month. In the INGENIAS-Scrum approach, no more than 21 days for a sprint are suggested because the most common sprint length for the Scrum framework is 2 weeks and the use of the INGENIAS model-driven tools promotes short cycles. The usual goal of each sprints is to implement and test JACE code covering at least one of the MAS goals. Figure 11 shows the workflow of the activities to be accomplished during the Sprint Phase.

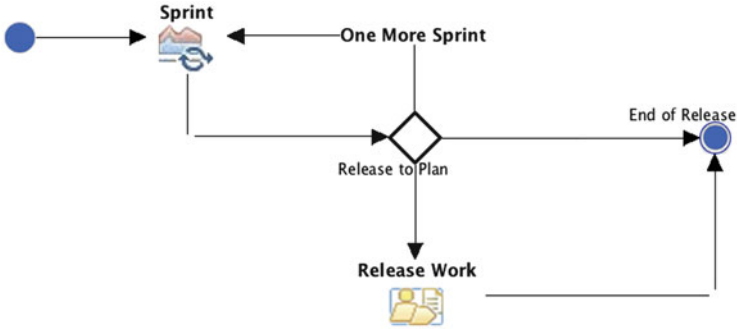


Fig. 11 The Sprint Phases flow of activities

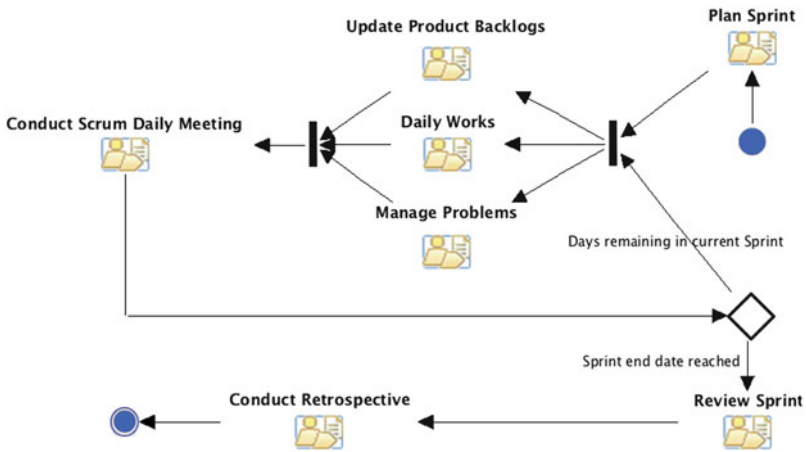


Fig. 12 The Sprint activity workflow

Current Phase (see Fig. 11) is composed of an iterative and complex phase, the proper *Sprint* and an optional activity the *Release Work*. The Sprint phase is composed of several activities whose workflows are shown in Fig. 12 and that is described in detail in Sect. 2.2.2. Section 2.2.1 introduces the roles involved on each Sprint. Finally, Sect. 2.2.3 details the work products of the *Sprint Phases*.

2.2.1 Process Roles

Following the Scrum approach, the roles that are implied in the Sprint Phases are described in the following sections.

The Product Owner

During each Sprint Phase, the Product Owner is responsible of analyzing the suggested changes in order to add them to the backlog and prioritize the added

items. This is done before the next *Sprint Planning*. Ideally, this work is done 1 or 2 days before the *Sprint Review* during a brainstorming session with the Scrum Team.

The Master Team

The iterative Sprint Phases perform the product development. The *Master Team* is the responsible for helping the Scrum Team to work autonomously and constantly improving itself by doing the following kinds of tasks:

- *Periodical tasks*, whose main goal is to organize and promote the collaborative work during the meetings: *Daily Scrum*, *Sprint planning*, *Sprint review*, and *Retrospective*.
- *Event tasks*, whose objective is to remove impediments. This task mainly relies on taking into consideration previous events to solve recurrent problems as soon as possible, while protecting the team from external distractions.
- *Background task*, in which the Master Team tries to ensure that the team remains productive and focused on the project goal: developing backlog items in close collaboration with the Product Owner.

The Scrum Team

In the Scrum framework, the teams are composed of 3–10 members. In the INGENIAS approach, this number should be between 3 and 5 people. The *Scrum Team* is the main responsible of the *Product Increment* and its participation is key in the activities: *Plan Sprint*, *Update Product Backlog*, *Daily Works*, and *Conduct Scrum Daily Meetings*. Also, its participation is important on the *Review Sprint* and *Conduct Retrospective* activity. While it performs a secondary role at the *Manage Problems* activity. By participating in these activities, it modifies the *Product Backlog* and the *Sprint Backlog*.

The Stakeholder

Like in the *Preparation* Phase, the participation of *Stakeholders* is not necessary in any of the Sprint Phases. They can participate playing a secondary role in the *Update Product Backlog*, *Conduct Retrospective*, and *Review Sprint* activities. Their participation is consultive, they make suggestions about the goals that the system must satisfy and the right solutions from the customer perspective.

2.2.2 Activity Details

Figure 12 shows the workflow of the activities to be performed at each Sprint. The full structure of this Phase in terms of activities, tasks, and workproducts is presented in four figures: Figs. 13, 14, 15, and 16.

Figure 13 shows the structure of the activities *Plan Sprint* and *Update Product Backlog*. The *Plan Sprint* is the first activity to be performed on each Sprint, while the *Update Product Backlog* is done in parallel with the *Daily Works* and the *Manage Problems* activities.

The structure presented in Fig. 14 refers to the most complex activity of the full process. It represents the *Daily Work* of the Scrum Team that must be accomplished using the IAF with the IDK tool.

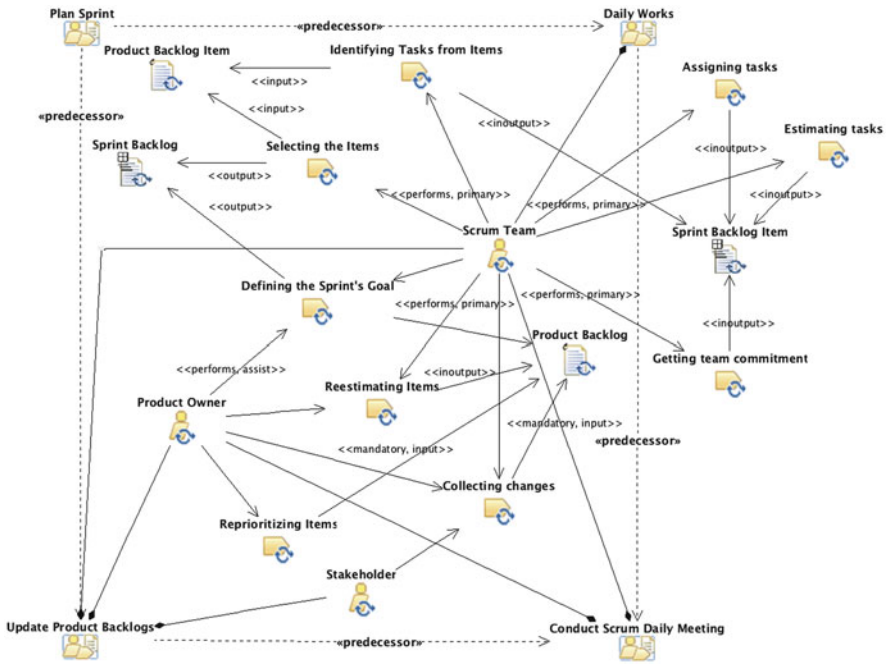


Fig. 13 Partial structure of the first activities done on each Sprint Phase

Figures 15 and 16 show the structure and the dependencies among the activities that control and determine the evolution of each sprint during the current release.

In the following subsections, all the activities jointly with their tasks are detailed and the workflow suggested to each one are showed.

Plan Sprint

The first activity of each sprint is to perform a *Sprint Planning Meeting*. During this meeting, the *Product Owner* and the *Scrum Team* discuss about the highest-priority items in the *Product Backlog*. External stakeholders may attend by invitation, although this is unusual in most cases. Team members determine the number of items they can commit to accomplish and then they create a sprint backlog, which is a list of the tasks to perform during the sprint. Figure 17 shows the workflow of the tasks with the work products and roles involved in this activity, and Table 3 describes in detail the tasks.

Update Product Backlog

The main goal of this activity is to update the backlog and adjust the planning, taking into account changes emerged since the last sprint. During a sprint, no one can modify the selection of backlog items done at the beginning of the sprint, that is, the sprint scope remains unchanged. However, anyone, even a external Stakeholder,

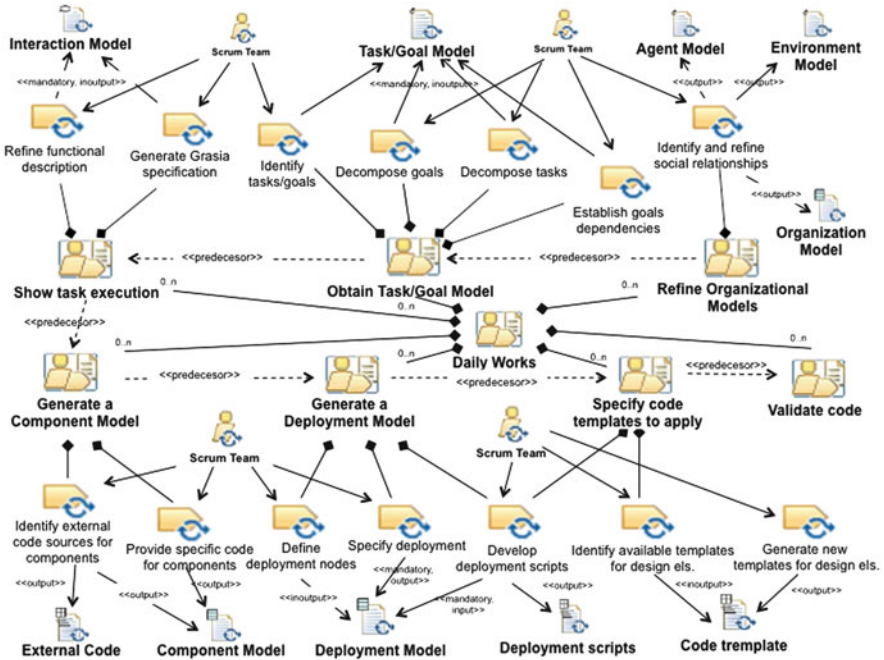


Fig. 14 Structure of the *Daily Work* activity on each Sprint Phase

can suggest new items for later sprints. These items are studied and prioritized by the Product Owner prior to the next sprint planning. Bugs and enhancement requests, coming from *partial product tests* done at the end of the previous sprint, can also be used to update the backlog. Figure 18 shows the relations between tasks, work products, and roles in this activity, and Table 4 describes its tasks in detail.

Daily Works

The team performs backlog tasks to reach the sprint goals. The original Scrum templates provide no information for technical design, coding, and test tasks. Tasks are not assigned by the Scrum Master, but chosen by the team members one each time. Team updates (when necessary) the estimation of their remaining work to do in the Sprint. The recommendation of the INGENIAS-Scrum process for this activity is to merge and distribute, accordingly to the needs, the tasks presented in the Elaboration and Construction subsections of the *INGENIAS-UDP chapter* for the following activities:

- Refine organizational models with social relationships, which includes refining the Organization, Agent, and Environment Models
- Create the Tasks and Goals Model
- Show tasks execution using the Interaction Model
- Create a Component Model

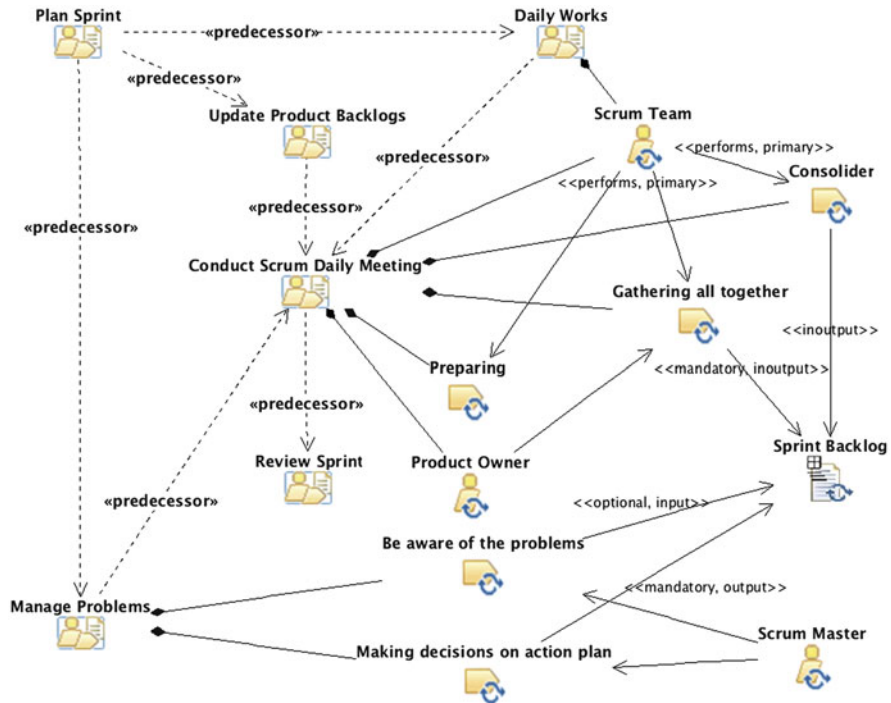


Fig. 15 Structure of the *Manage Problems* and *Conduct Scrum Daily Meeting* activities

- Create a Deployment Model
- Specify code templates to apply
- Validate code

Figure 14 shows the full structure of this activity, while the workflow and the details to accomplish such tasks are shown in the *INGENIAS-UDP* chapter.

Manage Problems

The Scrum Master takes into account the events that happen at any moment in a project. She/he tries to eliminate the problems experienced by the team members, so they can focus on their actual goals. The activity generates as output a Sprint Backlog and can use optionally as input a previous Sprint Backlog, as shown in Fig. 19. Table 5 describes the tasks to be done by the Scrum Master.

Conduct Scrum Daily Meeting

Every day of the Scrum sprint, all team members attend a *Scrum daily meeting*, including the Scrum Master and the Product Owner. This meeting is timeboxed to *no more* than 15 min. During the meeting, people participating in the development share their finished work, what they have to do that day, and identify any impediments or problems that could affect the team progress. This activity allows to synchronize

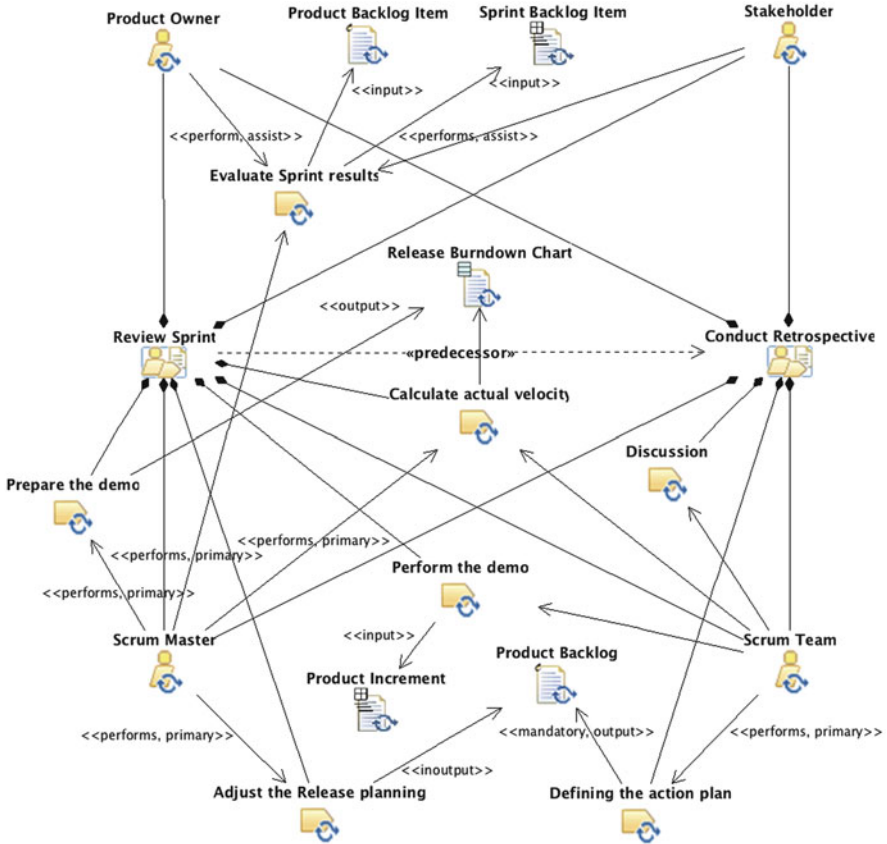


Fig. 16 Structure of the Review Sprints and Conduct Retrospective activities

ongoing work while discussing the progress of the current sprint. Figure 20 shows the relation between tasks, work products, and roles in this activity, and Table 6 details its tasks.

Review Sprint

At the end of a Scrum sprint, the team conducts a *Sprint Review* meeting. There, it demonstrates the functionality added during the current sprint using a demo. The main objective of this activity is to obtain feedback from the users invited to the review (such as the product owner or the stakeholders). The feedback can result in the acceptance of the work, the suggestion of changes to the delivered functionality, or even the revision or addition of backlog items. Figure 21 shows the relation between tasks, work products, and roles in this activity, and Table 7 details the tasks.

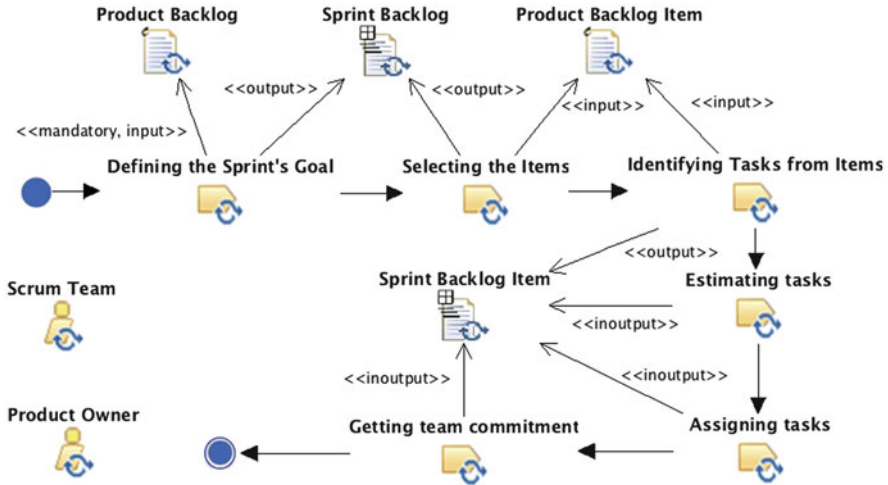


Fig. 17 Workflow of the *Plan Sprint* activity

Table 3 Description of the tasks of the *Sprint Planning* activity

Activity	Task	Description	Involved roles
<i>Plan Sprint</i>	Defining the <i>Sprint's Goal</i>	The goal of a sprint is proposed by the Product Owner. In the first sprints (2–3), it focuses on showing the feasibility of the potential architecture (i.e., the Organization and Agent Models). After architecture validation, the goals of sprints consist on satisfying a system <i>Goal</i> .	Product Owner Scrum Team
	Selecting the items	This task defines the scope of the sprint. The Scrum Team associates Product Backlog Items to the sprint. The team does it item by item, trying to balance the required effort with the team velocity. Then, the team collectively validates the subset of the backlog for the sprint.	Scrum Team
	Identifying tasks from items	This task is addressed in the second part of the meeting. Here, the team decides how to achieve the sprint goals. Each selected <i>Item</i> is decomposed into tasks to enable discussion and figure out solutions. The Product Owner can provide more details about the behavior of the selected item. The work planned in previous sprints, which has not been done, because of objectives reduction, becomes the priority for the next sprint.	Product Owner Scrum Team
	Estimating tasks	In order to distribute the development work between the team members, the duration of each task is estimated. Estimation is made in hours and taking into account that each task should be light	Scrum Team

(continued)

Table 3 (continued)

Activity	Task	Description	Involved roles
		enough (less than 16 hours). The team collectively addresses this task, reviewing the technical aspects during the discussion.	
	Assigning tasks	The team considers the number of persons required for each activity. All the activities must be studied, including <i>work meetings</i> , <i>coding</i> , and <i>document reviews</i> . It is desirable to delay the assignment of activities until the availability of the related team members is known.	Scrum Team
	Getting team commitment	This is a relevant task in which the team collectively decides what are the backlog items to implement in the current sprint. This decision constitutes the agreed and shared commitment of the sprint.	Scrum Team

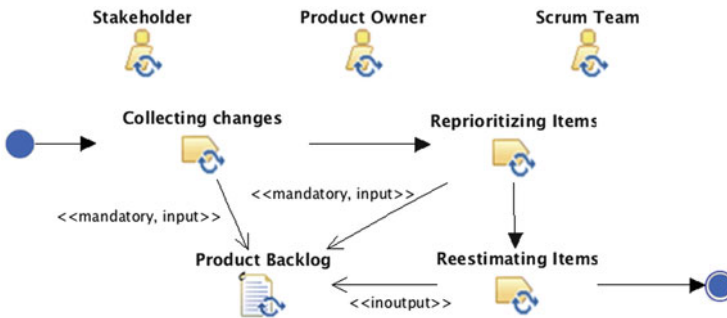


Fig. 18 Workflow of the *Update Product Backlog* activity

Table 4 Description of the tasks of the *Update Product Backlog* activity

Activity	Task	Description	Involved roles
Update <i>Product Backlog</i>	Collecting changes	Stakeholders or any member of the Scrum Team suggest new goals, functionalities, or changes to be added to the backlog. The bugs and enhancement requests obtained from product tests in the previous sprints are also incorporated into the collection of items that can be updated. The Product Owner decides whether to consider or not these items.	Product Owner Stakeholder Scrum Team
	Reprioritizing the items	New considered items must be prioritized.	Product Owner
	Reestimating items	New items added to the backlog have to be estimated by the Scrum Team, ideally during a brainstorming with the Product Owner previously to the next <i>Sprint Review</i> . The new estimation could imply a reestimation of existing items.	Scrum Team

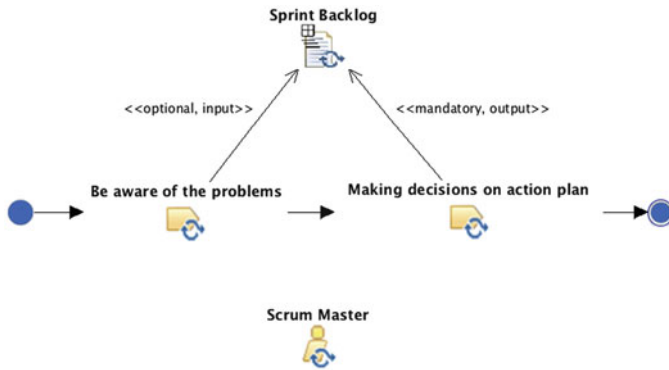


Fig. 19 Workflow of the *Manage Problems* activity

Table 5 Description of the tasks of the *Manage Problems* activity

Activity	Task	Description	Involved roles
Manage Problems	Be aware of the problem	The objective of the Scrum Master is to be informed about problems with the project. The daily meeting is the ideal moment to detect them. A good practice to follow up on the problem reporting is an informal chat initiated by the Scrum Master with the problem reporter.	Scrum Master
	Making decisions on the action plan	The Scrum Master must try to solve problems as soon as possible, in order to not disturb team progress. The master identifies possible solutions, and schedules meetings with owners if problems cannot be easily solved or need a management decision.	Scrum Master

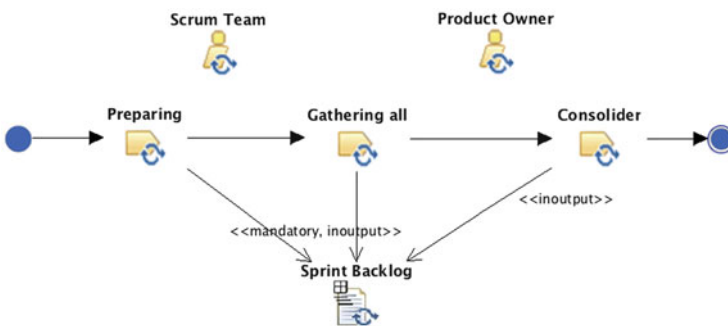


Fig. 20 Workflow for the *Conduct Scrum Daily Meeting* activity

Conduct Retrospective

The last activity of each sprint iteration is the *Sprint Retrospective*. This activity gives an opportunity to reflect at the end of the sprint and to identify the work to improve in the next sprint. The whole Scrum Team participates together with

Table 6 Description of the tasks of the *Conduct Scrum daily meeting* activity

Activity	Task	Description	Involved roles
Conduct Scrum daily meeting	Preparing	All the people update their planning (this sprint backlog) considering the remaining work for each task. As result of this task, an update or creation of a <i>Sprint Burndown Chart</i> is done.	Scrum Team
	Gathering all	Each team member reports her/his progress by answering the following 3 questions: <i>What has been done since the previous daily scrum? What will be done today? What problems have appeared?</i> Answers to the questions provide the update of the Sprint Backlog and changes in the Sprint Burndown Chart.	Scrum Team
	Consolidating	The resulting Sprint Backlog will help to take the right decisions on adjusting the Sprint Goal by removing forecasting content to be in time. Submitted problems are solved by the Scrum Master during the <i>Manage problems</i> activity.	Scrum Team

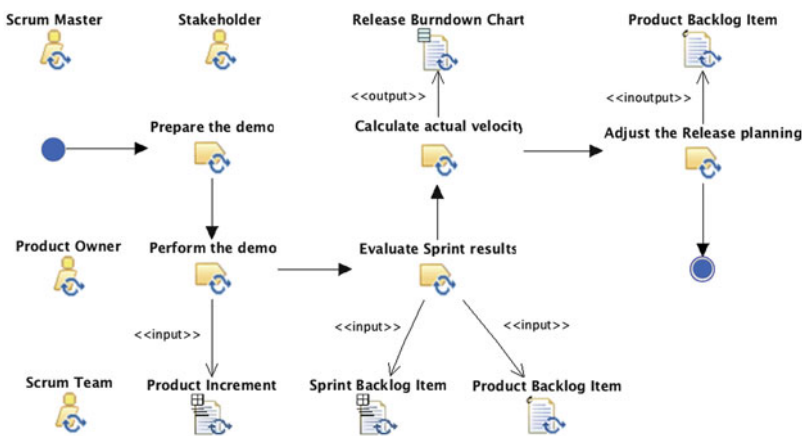


Fig. 21 Workflow for the *Review Sprint* activity

the Product Owner. The detailed tasks of the activity, its work products, and the collaborations between the roles are presented in Table 8 and Fig. 22. The complete activity has to be restricted to 2 h, though the average time is 1 h.

Release Work

As pointed previously, this activity is an optional one on the Sprint Phases. Its main objective is preparing a product release. Its actual execution depends on the way the product is made available to endusers and may vary from a project to another and

Table 7 Description of the tasks of the *Review Sprint* activity

Activity	Task	Description	Involved roles
<i>Review Sprint</i>	Preparing the demo	The schedule of the review is adjusted taking into account that the meeting must not exceed one hour.	Scrum Master
	Performing the demo	The team shows a demo of the product focused on the new goals satisfied during the actual sprint, allowing users to get a measurement of the work progress.	Scrum Team
	Evaluating the Sprint results	During this task, the Product Owner and the attending Stakeholders interview the team, provide impressions, make new proposals, and exchange requests. As a result, the Product Backlog is enhanced with the new items and with the bugs found.	Product Owner Stakeholders Scrum Master
	Calculating the actual velocity	A Release Burndown Chart is commonly used to compare the Sprint velocity with previous ones. The velocity is calculated as the sum of all items checked and approved as fully implemented.	Scrum Master
	Adjusting the <i>Release planning</i>	After reviewing the sprint, the team may realize that the initial conditions have changed from the last release planning. Then, it can be necessary to adjust it taking into account the items added, modified, or deleted, the changes in their priorities, the updated estimations, and of course, the average velocity of the team.	Scrum Team

Table 8 Description of the tasks of the *Conduct Retrospective* activity

Activity	Task	Description	Involved roles
Conduct retrospective	Discussion	Each team member is invited to express her/his bad and good results in the sprint providing solutions to the detected problems.	Scrum Team
	Defining the action plan	The team talks about potential improvements for the next sprint and the Scrum Master adds those approved to the product backlog. After that, the team sets up priorities with the help of the product owner and the stakeholder (if any).	Scrum Team Product Owner Stakeholder

among teams. Moreover, the team should rollout this optional activity with tasks that are not considered during “*normal*” sprints.

The only recommendation for this activity is trying not to make code changes because it is too late in the development process and it would imply a high risk of introducing bugs and errors in the final application. Some tasks that could be performed in this activity are:

- Hot deployment.
- Product packaging.
- Online download publishing.

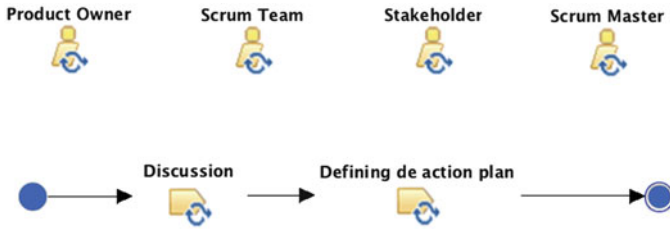


Fig. 22 Workflow for the *Conduct Retrospective* activity

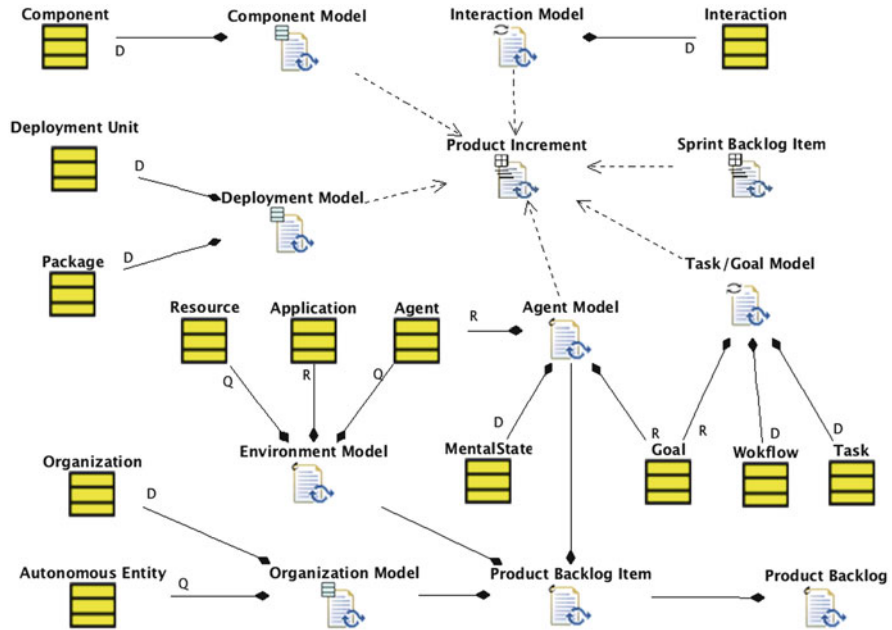


Fig. 23 The Sprint Work Product Model structure

- Technical documentation.
- User training.
- Product marketing.

2.2.3 Work Products

The Sprint Model generates several composed work products based on an IN-GENIAS specification. Their relationships with the MAS metamodel elements are described in Fig. 23.

Work Product Kinds

Table 9 describes the workproducts of each Sprint that are detailed in the next subsections.

Table 9 Work Product kinds for each Sprint phase

Name	Description	Work product kind
Product Increment	Source code for an increment of the product developed and finished on the previous or actual sprint	Structured
Product Backlog	A set of INGENIAS models that specify the product to obtain	Composited
Sprint Backlog	A set of PBI that the team must complete during the sprint	Composited
Burndown Chart	Graphical charts that show the evolution of the work	Structural
INGENIAS Model	See INGENIAS-UDP chapter	Structural,composite, behavioural

Product Increment

The primary work product of a Scrum project on each Sprint Phase is the *Product* itself. The Scrum Team is expected to bring the product or system to a potentially shippable state at the end of each Scrum Sprint. A product increment following the INGENIAS-Scrum approach is obtained from the IDK by generating code using the IAF. The code will be determined by the specified INGENIAS Models.

Product Backlog

During the iterations produced on the Sprints, each *Scrum Product Backlog* defined in the Preparation phase is completed and new ones are proposed and constructed. To complete the product backlog, the team must complete the life cycle of all its PBI reaching one after another the following states: *created*, *estimated*, *planned* (associated to a future sprint), *associated to current sprint* (implementation is ongoing), and *done*.

Sprint Backlog

As indicated in the previous sections, on the first day of a sprint and during the sprint planning meeting, team members create the *Sprint Backlog*. The sprint backlog can be thought of as the team's to-do list for the sprint. Whereas a product backlog is a list of features to be built, the sprint backlog is the list of tasks the team needs to perform in order to deliver the functionality they committed to deliver during the sprint.

Burndown Charts

Optional work products are the *Sprint Burndown Chart* and the *Release Burndown Chart*. Burndown charts show the amount of work remaining either in a Scrum sprint or a release. They are used for determining at a glance the evolution of a sprint or release, showing whether all the planned work will be finished by the desired date.

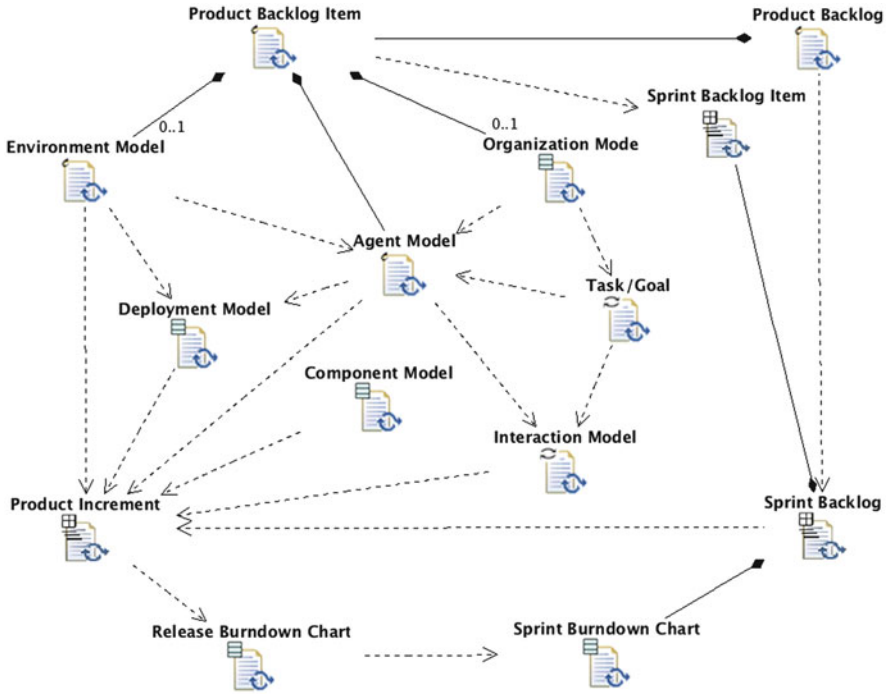


Fig. 24 Dependencies among INGENIAS-Scrum work products

INGENIAS Models

The INGENIAS-Scrum approach suggests additional work products for the process. The INGENIAS models are required in order to use the IDK and IAF tools for the automated code generation and test production. Nevertheless, part of the required code must be externally developed. For instance, some functionality related to resources and applications (e.g., graphical user interfaces or databases) and also concrete details of behavior (i.e., algorithms). INGENIAS does not provide modelling primitives to specify graphically these elements, though it has primitives that act as containers to embed this code in models. This allows that, afterwards, the IAF uses that information in the code generation. A developer may also find necessary or useful to modify the generated code.

2.3 Work Product Dependencies

Figure 24 shows an overview of the INGENIAS-Scrum work products, as well as their dependencies. As shown in the figure, the Agent Model depends on the Organization and Environment Models, and the Interaction Model has dependencies with the Agent and Tasks & Goals Models, among others. There are also dependencies

between product backlogs (what to get), sprint backlogs (what to do), and Burndown charts (how much is done).

Regarding the structure of the work products associated to INGENIAS, they are fully detailed in the *INGENIAS-UDP chapter*.

3 Case Study: Conference Management System

The Conference Management System (CMS) is an interesting case study that involves several aspects, from the main organization issues to paper submission and peer review. These are typically performed by a number of people distributed all over the world who exploit the Internet as the infrastructure for communication and cooperation.

This section presents a solution to this problem following the INGENIAS methodology, and applying an adaptation of the Scrum management framework to define an agile development process. The proposed process makes use of the INGENIAS metamodel to define the system through several models, and of support model-driven tools to generate automatically code from the specifications. Following this approach, the development of the CMS implies an incremental and iterative life cycle based on performing two phases: the *Preparation* phase and the *Sprint* phase.

The *INGENIAS with the UDP chapter* in this book presents a similar solution to this problem. It follows the original proposal based on the UDP [14] and the Rational Unified Process (RUP) [17]. Some aspects of this chapter are referred to the previous one, where the reader can find a more detailed description.

3.1 Preparation Phase

As pointed out in Sect. 2.1, this phase comprises the activities: *Initiate Product Backlog*, *Preparation Tasks*, and *Plan Release*. The latter is done after completion of the remaining two, which are developed in parallel.

The initial Product Backlog is a preliminary description of the product requirements. In this approach, it is a specification conforming to the INGENIAS metamodel and generated with its tools. It can be developed from scratch or taking as basis examples provided with the IDK distribution.

In order to apply the first strategy, the team has to be familiarized with the IDK tool and know what examples the IAF provides with its distribution: *the hello world*, *a GUI agent example*, and *an interaction example*. Any of these examples can be used as starting point for this initial backlog, although they do not cover the complexity of the CMS.

The second approach is the one adopted here, as the CMS is a well-known and documented case study in the literature. This implies creating an initial *Organization Model* applying the three tasks offered by the original INGENIAS process: *Identify groups*, *Generate group members*, and *Identify goals*. Figure 25 shows the resulting

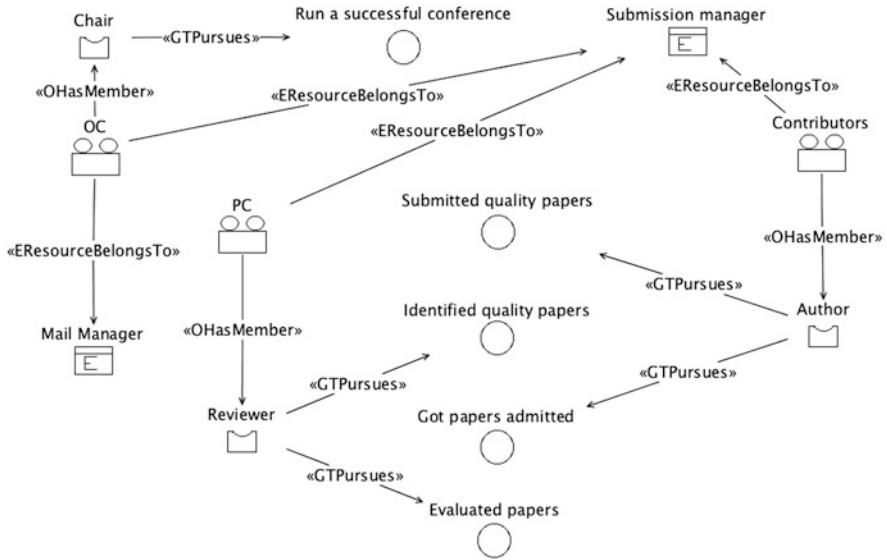


Fig. 25 Organization diagram with the main user groups of the CMS and their access to applications

model. Note that the result is just the same than the one obtained using the original INGENIAS development process, but without generating use cases in order to identify the functionality perceived by the user.

This first activity identifies three groups in the CMS case study. The *OC* (Organizing Committee) includes the *Chairs*, the *PC* (Program Committee) for the *Reviewers*, and the *Contributors* with the *Authors*. These groups perform tasks related to certain goals. The main goal that the *OC* must satisfy is *Run a successful conference*. Regarding the *PC*, it has to satisfy the goals: *Identified quality papers* and *Evaluated papers*. Finally, *Contributors* have to satisfy the goals *Submitted quality papers* and *Got papers admitted*. There are also two external applications, that is, the *Submission manager* and the *Mail manager*. Group members use them to get some services, according to certain use constraints. Every group can access the *Submission manager*, though not for the same tasks, but only the *OC* can use the *Mail manager*.

The Product Owner prioritizes the goals to be addressed in the sprints in order to fix the *Plan Release*. In this case, the choice is to consider in the first sprints the behavior associated to the *Run a successful conference* goal, which is associated to the *OC* group. The assistants to the meeting agree that the goals associated with the roles *Reviewer* and *Author* can be done in later sprints because the associated risk is lower than the selected one.

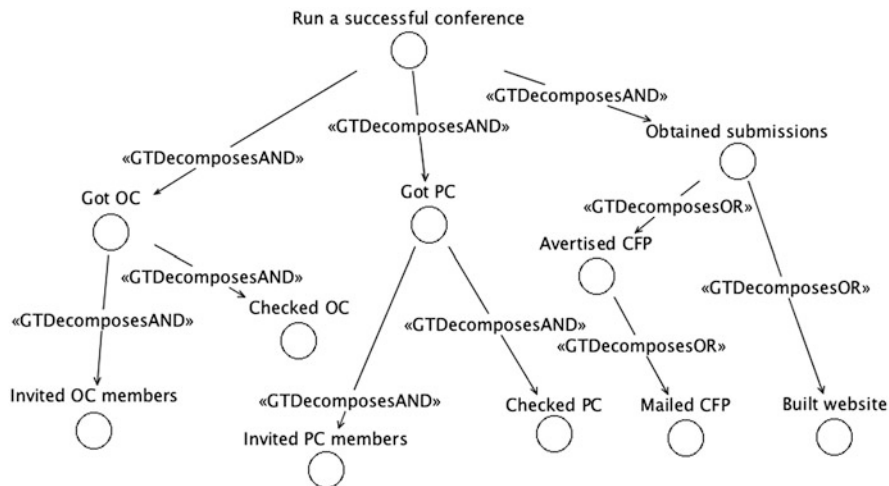


Fig. 26 Initial decomposition of the *Run a successful conference* goal

3.2 Sprint Phase

In the first sprint, the Scrum team performs the *Daily Works* activity related with the *Run a successful conference* goal. Its first results are the diagrams in Figs. 26 and 27 for the Tasks and Goals Model, and Fig. 28 for the Agent Model. Figure 26 shows the refinement of the selected goal in different subgoals. Figure 27 introduces the tasks needed to achieve those goals, that is, their products are potentially able to satisfy the conditions related to those goals. Finally, Fig. 28 shows the initial Agent Model obtained from the first refinement of the Organization Model, assigning the identified tasks to the *Chair* role.

The *Daily Works* activity follows with the INGENIAS activity cycle that includes the tasks: *Show task execution*, *Generate a Component Model*, *Generate a Deployment Model*, *Specify code templates to apply*, and *Validate code*. Regarding these tasks, the team follows short iteration cycles that give a similar result to those explained in Sect. 3 in the *INGENIAS chapter for the Construction Phase*. Anyway, the last activities are beyond the scope of this case study, which is focused on the development process adopted and not on the application obtained. Nevertheless, the facilities included in the IDK [11], and in particular the IAF [12], facilitate generation of testing code for these specifications. These tests will implement the acceptance tests suggested for each BPI associated with the specified goals.

The product obtained is used in the *Scrum Daily Meeting* and *Review Sprint* activities. At the end of each sprint, the *Conduct Retrospective* activity is used to better accomplish the remaining PBI in the future sprints through their revision regarding the results of the current sprint.

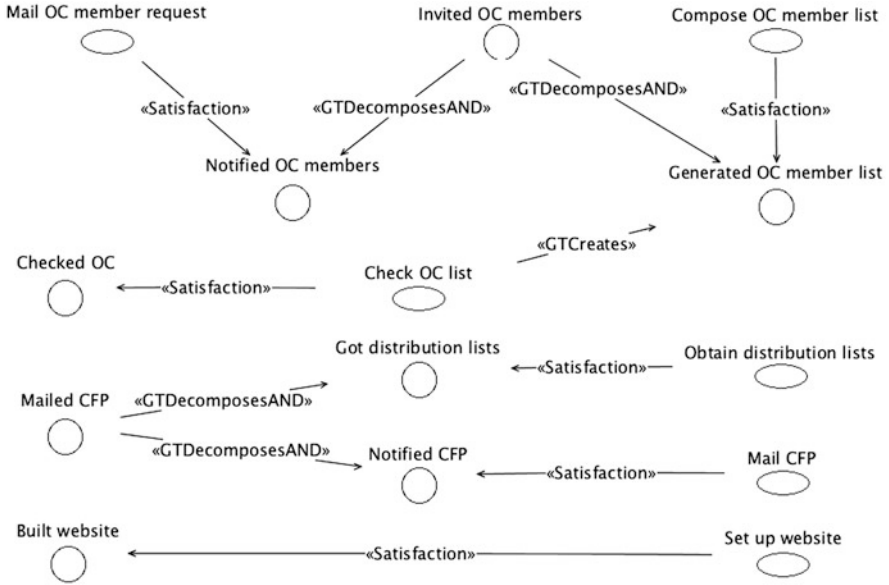


Fig. 27 Initial assignment of tasks related to the *Run a successful conference* goal

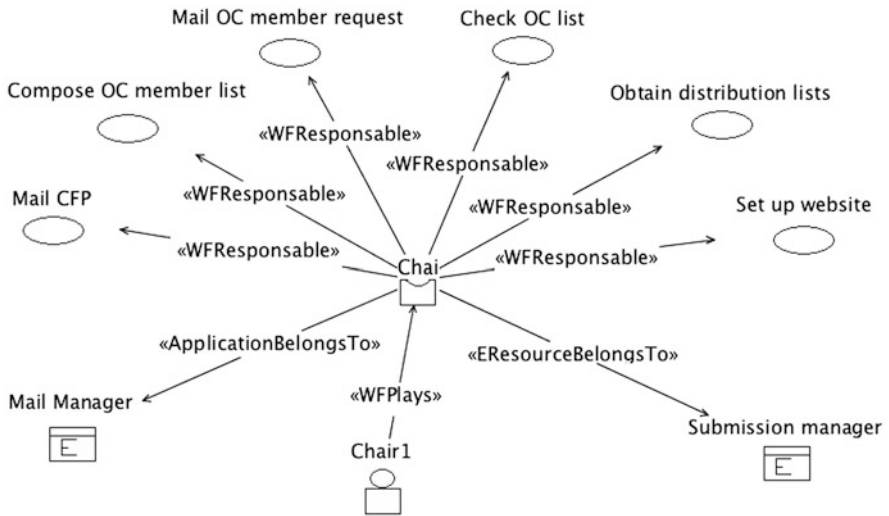


Fig. 28 Initial assignment of tasks to the *Chair* role

References

1. García-Magariño, I., Gómez-Rodríguez, A., Gómez-Sanz, J., González-Moreno, J.C.: *Adv. Soft Comput.* **50**, 108 (2009)
2. Cernuzzi, L., Cossentino, M., Zambonelli, F.: *Eng. Appl. Artif. Intell.* **18**(2), 205 (2005)
3. Martin, R.: *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River (2003)
4. Schwaber, K., Beedle, M.: *Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River (2001)
5. Schwaber, K.: In: 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 1995), pp. 117–134 (1995)
6. Pavón, J., Gómez-Sanz, J.J., Fuentes-Fernández, R.: In: Henderson-Sellers, B., Giorgini, P. (eds.) *Agent-Oriented Methodologies*. Article IX, pp. 236–276. Idea Group Publishing, Hershey (2005)
7. France, R., Rumpe, B.: In: 2007 Future of Software Engineering (FOSE 2007), pp. 37–54. IEEE Computer Society, Minneapolis (2007)
8. García-Magariño, I., Fuentes-Fernández, R., Gómez-Sanz, J.: *Inf. Softw. Technol.* **51**(8), 1217 (2009)
9. Fuentes-Fernández, R., Gómez-Sanz, J.J., Pavón, J.: *IEICE Trans. Inf. Syst.* **E90-D**(8), 1243 (2007)
10. Fuentes-Fernández, R., Gómez-Sanz, J.J., Pavón, J.: *Int. J. Agent Oriented Softw. Eng.* **1**(1), 2 (2007)
11. Gómez-Sanz, J.J., Pavón, J., Fuentes-Fernández, R., García-Magariño, I., Rodríguez-Fernández, C.: *INGENIAS Development Kit, V. 2.8*. Tech. rep., Universidad Complutense de Madrid (2008)
12. Gómez-Sanz, J.: *INGENIAS Agent Framework. Development Guide V. 1.0*. Tech. rep., Universidad Complutense de Madrid (2008)
13. Bellifemine, F., Poggi, A., Rimassa, G.: In: 5th International Conference on Autonomous Agents (AGENTS 2001), pp. 216–217. ACM, Montreal (2001)
14. Booch, G., Jacobson, I., Rumbaugh, J.: *The Unified Software Development Process*. Addison-Wesley, Reading (1999)
15. Gómez-Sanz, J.: *Modelado de sistemas multi-agente*. Ph.D. thesis, Universidad Complutense de Madrid, Facultad de Informática (2002)
16. Grupo de Investigación en Agentes Software: *Ingeniería y Aplicaciones*. INGENIAS Section. <http://grasia.fdi.ucm.es/main/?q=es/node/61> (2010)
17. Rational Software. *Rational Unified Process: White Paper* (1998)

The O-MASE Methodology

Scott A. DeLoach and Juan C. Garcia-Ojeda

Abstract

Today's software industry is tasked with building evermore complex software applications, and multiagent system technology is a promising approach capable of meeting these new demands. Unfortunately, multiagent systems have not been widely adopted in industry for reasons that include lack of industrial strength methods and tools to support multiagent development. Method engineering, an approach to constructing processes from a set of existing method fragments, has been suggested as a solution to this problem. This chapter presents the Organization-based Multiagent Software Engineering (O-MaSE) methodology framework, which integrates a set of concrete technologies aimed at facilitating industrial acceptance. Specifically, O-MaSE is a customizable agent-oriented methodology based on consistent, well-defined concepts supported by plug-ins to an industrial strength development environment, agentTool III. O-MaSE is defined, and demonstrations of customizing O-MaSE for the CMS problem as well as applying the customized process to the CMS design are presented.

1 Introduction

Organization-based Multiagent Software Engineering (O-MaSE) [4] is a new approach in the analysis and design of agent-based systems, being designed from the start as a set of method fragments to be used in a method engineering

S.A. DeLoach (✉)
Kansas State University, 234 Nichols Hall, Manhattan, KS 66506, USA
e-mail: sdeloach@ksu.edu; sdeloach@k-state.edu

J.C. Garcia-Ojeda
Facultad de Ingenieria de Sistemas, Universidad Autonoma de Bucaramanga,
Avenida 42 No 48-11, El Jardin. Bucaramanga, Santander, Colombia
e-mail: jgarciao@unab.edu.co

framework [1, 2, 12]. The goal of O-MaSE is to allow designers to create customized agent-oriented software development processes. O-MaSE consists of three basic structures: (1) a metamodel, (2) a set of method fragments, and (3) a set of guidelines. The O-MaSE metamodel defines the key concepts needed to design and implement multiagent systems. The method fragments are tasks that are executed to produce a set of work products, which may include models, documentation, or code. The guidelines define how the method fragments are related to one another.

The aT3 Process Editor (APE) shown in Fig. 1 is a tool that supports the creation of custom O-MaSE-compliant processes [10]. APE is part of the agentTool III tool-set, which provides tool support to developing multiagent systems using O-MaSE [11]. There are five key elements of APE: a Method Fragment Library, the Process Editor, a set of Task Constraints, a Process Consistency checker, and a Process Management tool. The Library is a repository of O-MaSE method fragments, which can be extended by APE users. The Process Editor allows users to create and maintain O-MaSE-compliant processes. The Task Constraints view helps process engineers specify Process Construction Guidelines to constrain how tasks can be assembled, while the Process Consistency mechanism verifies the consistency of custom processes against those constraints. Finally, the Process Management tool provides a way to measure project progress using the custom process.

O-MaSE also provides a set of Method Construction Guidelines that states how O-MaSE method fragments may be combined to form O-MaSE-compliant processes. Table 1 shows the Method Construction Guidelines for the O-MaSE Tasks. These Method Construction Guidelines are defined in terms of a precondition and post-condition. The precondition specifies the set of Work Products that must be available prior to the Task being undertaken while the post-conditions specify the Work Products produced by the task. For example, for the Model Goals task, either a Requirements Spec must be available or a Goal Model/GMoDS and a Role Model must be available. The Requirements Spec is used when the Model Goals task is used to model system-level goals, while the Goal Model/GMoDS and Role Model are used when the task is used to model role-level goals. Disjunctive preconditions generally specify alternative ways the Task can be used. However, it does not limit what information can be used in the definition of a model. For instance, the Model Domain task only requires a Requirements Spec as input; however, that does not mean that other Work Products such as Goal Models cannot be used in the Task. This additional information is generally documented in the individual task definitions.

Useful references related to O-MaSE include the following:

- Scott A. DeLoach and Juan Carlos Garcia-Ojeda. O-MaSE: a customizable approach to designing and building complex, adaptive multiagent systems. *International Journal of Agent-Oriented Software Engineering*. Volume 4, no. 3, 2010, pp. 244–280.
- Juan C. Garcia-Ojeda, Scott A. DeLoach, and Robby. agentTool Process Editor: Supporting the Design of Tailored Agent-based Processes. *Proceedings of the 24th Annual ACM Symposium on Applied Computing to be held at the Hilton Hawaiian Village Beach Resort and Spa Waikiki Beach, Honolulu, Hawaii, USA, March 8–12, 2009*.

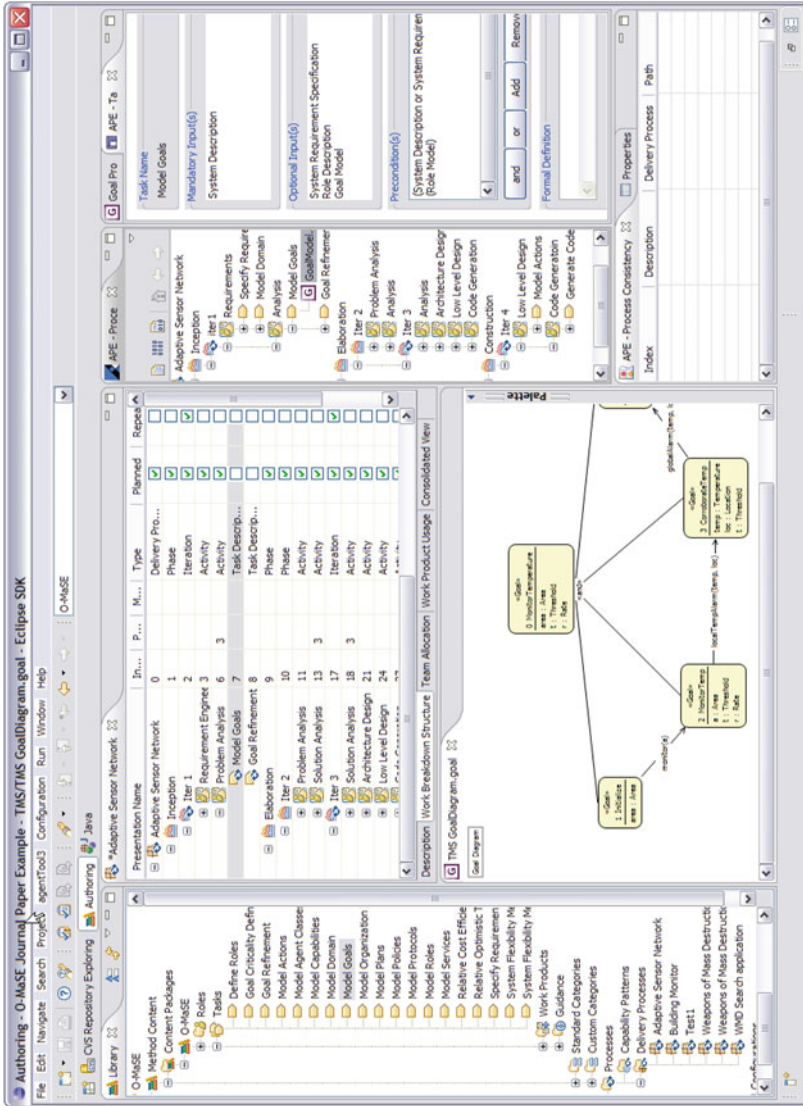


Fig. 1 agentTool III Process Editor

Table 1 Method construction guidelines

Task	Pre-condition	Post-condition
Requirements Specification	True	Requirements Spec
Model Goals	Requirements Spec \vee ((Goal Model \vee GMoDS) \wedge Role Model)	Goal Model
Refine Goals	Goal Model	GMoDS
Model Domain	Requirements Spec	Domain Model
Model Organization Interfaces	Requirements Spec \wedge GMoDS	Organization Model
Model Roles	GMoDS \wedge Organization Model	Role Model
Define Roles	Role Model	Role Description
Model Agent Classes	GMoDS \vee Role Model \vee Organization Model	Agent Class Model
Model Protocols	Role Model \vee Agent Class Model	Protocol Model
Model Policies	GMoDS \vee Organization Model \vee Role Description \vee Agent Class Model	Policy Model
Model Plans	(GMoDS \wedge Role Model) \vee (GMoDS \wedge Agent Class Model)	Plan Model
Model Capabilities	Role Model \wedge Agent Class Model \vee Domain Model	Capability Model
Model Actions	Capability Model \wedge Domain Model	Action Model
Code Generation	(Plan Model \vee Protocol Model) \wedge (Capability Model \vee Action Model)	Source Code

- Scott DeLoach, Lin Padgham, Anna Perini, Angelo Susi, and John Thangarajah. Using Three AOSE Toolkits to Develop a Sample Design. *International Journal of Agent Oriented Software Engineering*. Volume 3, no. 4, 2009, 2009, pp 416–476.
- Scott A. DeLoach. Organizational Model for Adaptive Complex Systems. in Virginia Dignum (ed.) *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global: Hershey, PA. ISBN: 1-60566-256-9 (March 2009). This chapter copyright 2008, IGI Global, www.igi-pub.com. Posted by permission of the publisher.
- Lin Padgham, Michael Winikoff, Scott DeLoach, and Massimo Cossentino. A Unified Graphical Notation for AOSE. *Proceedings of the 9th International Workshop on Agent Oriented Software Engineering*, Estoril Portugal, May 2008.
- Scott A. DeLoach. Developing a Multiagent Conference Management System Using the O-MaSE Process Framework. *Proceedings of the 8th International Workshop on Agent Oriented Software Engineering*, May 14, 2007, Honolulu, Hawaii.
- Juan C. Garcia-Ojeda, Scott A. DeLoach, Robby, Walamitien H. Oyenand and Jorge Valenzuela. O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. *Proceedings of the 8th International Workshop on Agent Oriented Software Engineering*, Honolulu HI, May 2007.

Table 2 O-MaSE overview

Entity	Task	Work Product	Role
Requirements Gathering	Requirements Specification	Requirements Spec	Requirements Engineer
Problem Analysis	Model Goals	Goal Model	Goal Modeler
	Refine Goals Model Domain	Domain Model	Domain Modeler
Solution Analysis	Model Organization	Organization Model	Organization Modeler
	Interfaces		
	Model Roles	Role Model	Role Modeler
	Define Roles Define Role Goals	Role Description Document Role Goal Model	
Architecture Design	Model Agent Classes	Agent Class Model	Agent Class Modeler
	Model Protocols	Protocol Model	Protocol Modeler
	Model Policies	Policy Model	Policy Modeler
Low Level Design	Model Plans	Agent Plan Model	Plan Modeler
	Model Capabilities	Capabilities Model	Capabilities Modeler
	Model Actions	Action Model	Action Modeler
Code Generation	Generate Code	Source code	Programmer

- Scott A. DeLoach and Jorge L. Valenzuela. An Agent-Environment Interaction Model. in L. Padgham and F. Zambonelli (Eds.): AOSE 2006, LNCS 4405, pp. 1–18, 2007. Springer-Verlag, Berlin Heidelberg 2007.
- Scott A. DeLoach. Multiagent Systems Engineering of Organization-based Multiagent Systems. 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05). May 15–16, 2005, St. Louis, MO. Springer LNCS Vol 3914, Apr 2006, pp 109–125.

1.1 The O-MaSE Life Cycle

O-MaSE was designed from scratch as a set of fragments that could be assembled by developers to meet the specific requirements of their project. While SPEM [15] uses Phases to organize the various Activities of a development method, O-MaSE makes no commitments to a predefined set of Phases. Instead, O-MaSE explicitly defines Activities and Tasks (see an overview in Table 2) and allows method engineers to organize Activities in different ways based on project need. For instance, O-MaSE has been used to support modern iterative, incremental approaches as well as much simpler waterfall-based approaches. The fact that O-MaSE does not commit to any specific set of phases causes a minor problem when trying to map O-MaSE directly to the DPDT. To alleviate this problem, we assume that we follow a traditional waterfall approach when describing O-MaSE as shown in Fig. 2. As shown, there are three main Phases: Requirement Analysis, Design, and Implementation, with

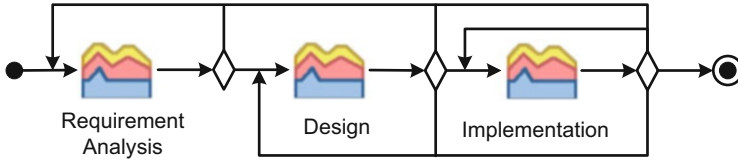


Fig. 2 Using waterfall phases with O-MaSE

the main Activities allocated as shown. When using O-MaSE on a real project, the process designer is free to define their own set of phases and iterations and to assign Activities and Tasks to those phases and iterations as appropriate. As this will be unique for each system being developed, there are no hard-and-fast rules on what activities should be placed in which phases.

1.2 The O-MaSE Metamodel

The O-MaSE metamodel defines the main concepts and relationships used to define multiagent systems. The O-MaSE metamodel is based on an organizational approach and includes notions that allow for hierarchical, holonic, and team-based decomposition of organizations. The O-MaSE metamodel was derived from the Organization Model for Adaptive Computational Systems (OMACS), which captures the knowledge required of a system's organizational structure and capabilities to allow it to organize and reorganize at runtime [5]. The key decision in OMACS-based systems is which agent to assign to which role in order to achieve which goal. As shown in Fig. 3, an Organization is composed of six entity types: Goals, Roles, Agents, Organizational Agents, a Domain Model, and Policies. Each of these entities is discussed below, and a concise definition is given in Table 3.

While a variety of subtle interpretations of goals exist in the artificial intelligence and agent communities, O-MaSE defines a Goal as an objective of the organization, which is generally described in terms of some desired state of the world. A Role defines a position within an organization whose behavior is expected to achieve a particular goal or set of goals. (Due to the naming conflict between O-MaSE Roles and SPEM roles, the term *method role* is used to refer to SPEM roles throughout the remainder of this chapter.) Agents are assigned to play those roles and perform the behavior expected of those roles. Agents are autonomous entities that can perceive and act upon their environment [19]. To carry out perception and action, an agent possesses a set of capabilities. Capabilities can be used to capture soft abilities (i.e., algorithms) or hard abilities (i.e., physical sensors or effectors). An agent that possesses all the capabilities required to play a role may be assigned that role in the organization. Capabilities can be defined as (1) a set of sub-capabilities, (2) a set of actions that may interact with the environment, or (3) a plan that uses actions in specific ways.

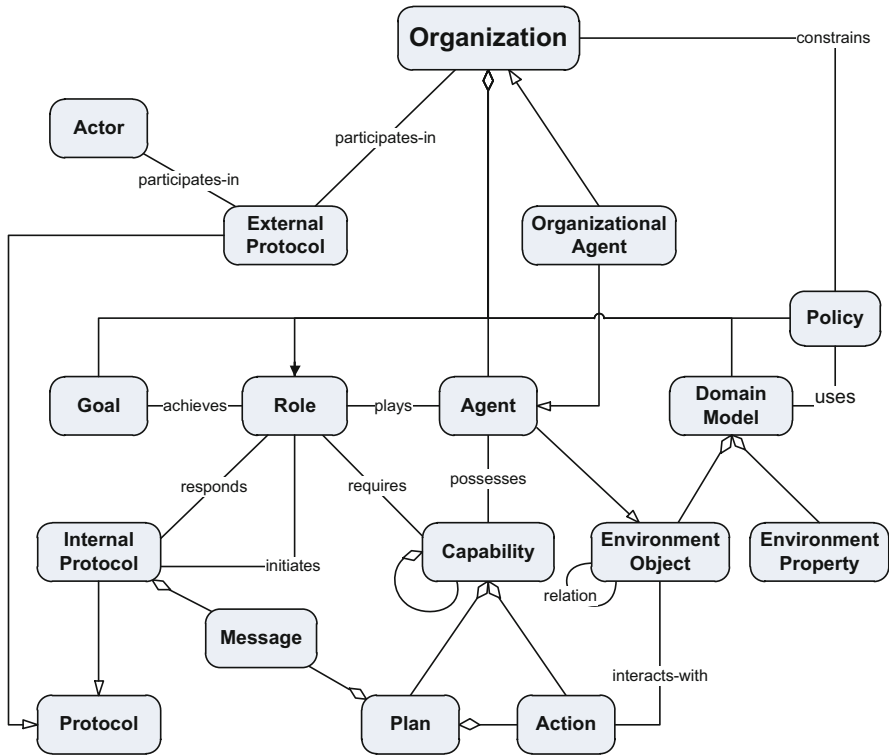


Fig. 3 O-MaSE metamodel

Table 3 Metamodel entities

Entity	Definition
Goal	A desirable state; goals capture organizational objectives
Role	Capture behavior that achieves a particular goal or set of goals
Agent	Autonomous entities that perceive and act upon their environment; agents play roles in the organization
Organizational Agent	A sub-organization that functions as an agent in a higher-level organization
Capability	Soft abilities (algorithms) or hard abilities of agents
Domain model	Captures the environment including objects and general properties describing how objects behave and interact
Policy	Constrain organization behavior often in the form of liveness and safety properties
Protocol	Define interaction between agents, roles, or external Actors; they may be internal or external
Actor	Actors that exist outside the system and interact with the system
Plan	Abstractions of algorithms used by agents; plans are specified in terms of actions with the environment and messages in protocols

Organizational Agents (OAs) are organizations that act as agents in a higher level organization and thus capture the notion of organizational hierarchy. As agents, OAs may possess capabilities, coordinate with other agents, and be assigned to play roles. OAs are similar to the notion of non-atomic holons in the ASPECS methodology [3]. Therefore, OAs represent an extension to the traditional Agent–Group–Role (AGR) model [8, 9] and the organizational metamodel proposed by Odell et al. in [17].

The Domain Model is used to capture the key elements of the environment in which agents will operate. These elements are captured as Domain Object Types from the environment, which includes agents, and the relationships between those object types. It can also be used to capture general Environment Properties that describe how the objects behave and interact [6]. A designer may use entities defined in the O-MaSE model (goals, roles, agents, etc.) along with entities defined in the Domain Model to specify organizational Policies to constrain how an organization may behave in a particular situation. Policies are often used to specify liveness and safety properties of the system being designed.

Protocols define interactions between roles or between the organization and external Actors. Protocols are generally defined as patterns of communication between such entities [16]. A protocol can be of two types, External or Internal. External Protocols specify interactions between the organization and external actors (i.e., humans or other software applications), while Internal Protocols specify interactions between agents playing specific roles in the organization. Either messages or actions can be used to define protocols. Messages are typically used for communications; however, actions may be used to modify the environment as a means of communication [14].

2 Phases

The first step in using O-MaSE to define a system is to define an O-MaSE compliant process. There may be several ways to define an O-MaSE compliant process; however, the simplest approach is to perform a bottom-up analysis of the work products required to produce the desired system. In a bottom-up approach, the key decision is what type of system we need to develop and what are the final work products that are needed to support the implementation of such a system. From there we work backwards to determine which other work products are required to produce the final work products. The example given here for the CMS is an appropriate O-MaSE compliant process [12].

Ultimately, the CMS is a centralized system with which a variety of humans interact. The roles of the system and humans are well defined, and, outside of major system shutdown, there is little chance of failure that would require that various agents might need to be reassigned goals in order for the system to work efficiently and robustly. Therefore, there is no requirement for an autonomously adaptive system such as produced by OMACS [7]. Thus, the definition of individual capabilities of the roles and agents is not required. Therefore, we can implement the system by defining a set of agent classes, the protocols between those classes, a set of

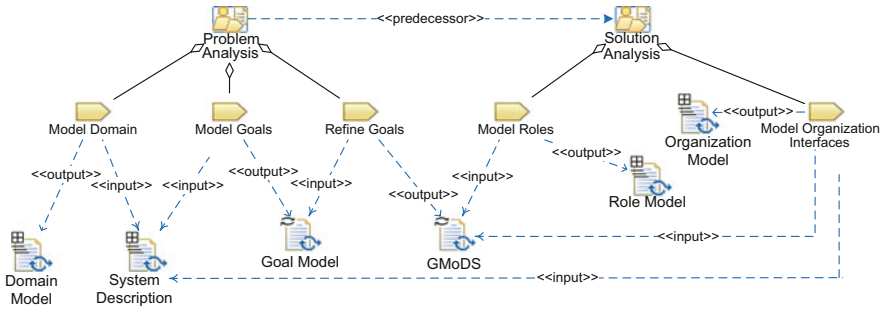


Fig. 4 CMS O-MaSE-compliant process—analysis phase

policies to constrain agent behavior and interaction, and a set of plans to implement the agent behavior. This information is provided by the Agent Class Model, Protocol Model, Policy Model, and Plan Model, which are defined using the Model Agent Classes, Model Protocols, Model Policies, and Model Plans tasks.

In order to provide the appropriate inputs to the tasks, we need to define a set of roles and goals for the system. Also, to allow us to define the parameters of the goal model, the protocols between the agents, and the policies and plans, we need to have a valid domain model. Thus, the work products we need to create to support the design phase include a Domain Model, a Goal Model, a Goal Model for Dynamic Systems (GMoDS) model, an Organizational Model, and a Role Model. These work products are defined using the Model Domain, Model Goals, Refine Goals, Model Organizational Interfaces, and Model Roles tasks.

The final step is to define the phases and possible iterations for our process. Since this is a fairly simple system, we choose a simple waterfall approach. The input to the process is the system specification of the CMS, while the output is the design models discussed above. We do not consider an implementation phase as the system can be implemented in a number of ways depending on where and how the final system is to be used.

Thus, the final process chosen for designing the CMS is a basic waterfall approach as shown in Fig. 2. However, because the system does not require adaptivity in terms of assigning agents to roles, we have simplified the Requirement Analysis and Design phases as shown in Figs. 4 and 5. Requirement Analysis begins by using the existing system requirements to define a Domain Model in the Model Domain task. Next, we define the basic Goal Model and refining it into a GMoDS Goal Model via the Model Goals and Goal Refinement tasks. Once the Goal Model is complete, the GMoDS Goal Model is used to create the initial Role Model.

The Design Phase begins by creating an Agent Class Model based on the Role Model and GMoDS model created during the Requirement Analysis phase. The details of the protocols identified in the Agent Class Model are further refined into several Protocol Models. While we chose to define the Protocol Models based on the Agent Class Model, we could have also defined the protocols after creating the Role

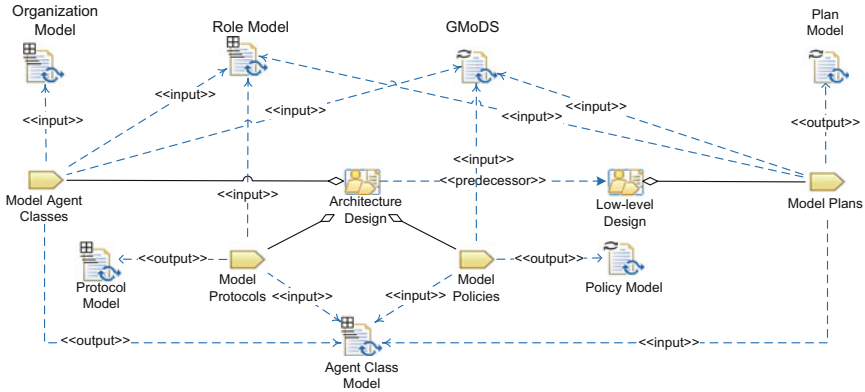


Fig. 5 CMS O-MaSE-compliant process—design phase

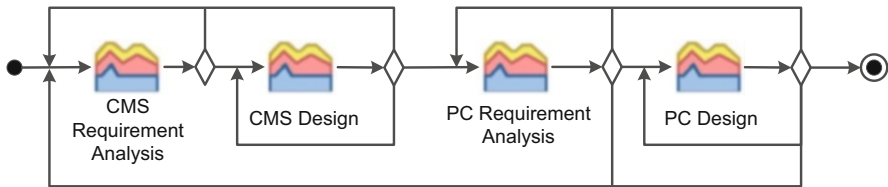


Fig. 6 O-MaSE-compliant process for CMS and PC organizations

Model as it also identifies protocols. Next, we model the policies that the agents and their protocols must adhere to. Finally, the plans of the agents are defined in the Model Plans task and produce a set of plans that implement the agent behavior.

However, to manage the complexity of the Conference Management System (CMS) provided earlier in the book, we decided to make use of the decomposition and abstraction mechanisms available in O-MaSE. Specifically, we decided that since the Program Committee (PC) (including the PC Chair, Vice-Chairs, and PC Members) operated as a single entity in relation to the other system actors (Authors, Publishers, and Reviewers), we would treat the PC as a separate entity in the design process and use the O-MaSE notion of an Organizational Agent to capture the PC. Thus, at the top-level description of the system, the PC is a single entity that is further decomposed in terms of its own organization. This actually requires a slight modification to the waterfall model. In actuality, this approach simply requires one iteration of the Requirement Analysis and Design phases for the CMS organization and a second iteration for the PC organization. However, to clarify the situation, we show an extended version of the CMS process in Fig. 6.

Again, as a reminder, the phases used to define O-MaSE as presented below are not actually part of the O-MaSE definition but only included to help define O-MaSE according to the DPDT. The process shown in Figs. 6, 4, and 5 is actually a subset

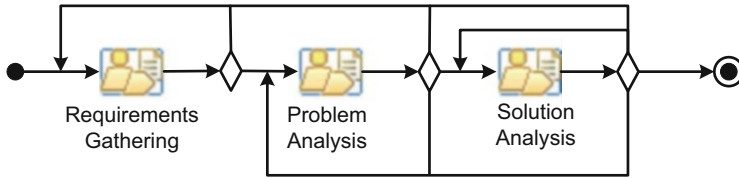


Fig. 7 Requirement Analysis phase flow of activities

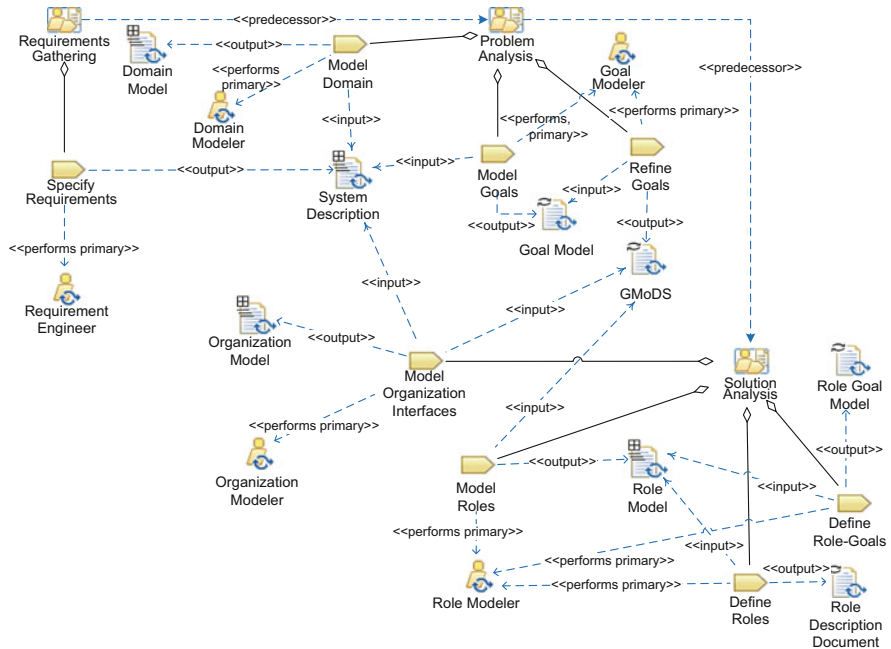


Fig. 8 Requirement Analysis phase in terms of activities and work products

of the phases, tasks, and activities discussed below, and examples of work products produced from this process are presented where appropriate.

2.1 Requirement Analysis

In traditional software engineering practice, the requirement analysis phase attempts to define and validate requirements for a new or modified software product, taking into account the views of all major stakeholders. A generic example of an O-MaSE requirement analysis phase is shown in Fig. 7 in terms of process flow and Fig. 8 in terms of process roles, work products, and tasks.

2.1.1 Process Roles

This phase uses five roles: Requirement Engineer, Goal Modeler, Domain Modeler, Organization Modeler, and Role Modeler.

Requirement Engineer

The Requirement Engineer captures and validates the requirements of the system. Thus, the person in this role must be able to think abstractly, work at high levels of abstraction, and collaborate with stakeholders, domain modelers, and project managers.

Goal Modeler

The Goal Modeler is responsible for the generation of the GMoDS goal model. Thus, Goal Modeler must understand the system description/SRS, be able to interact openly with various domain experts and customers, and be proficient in GMoDS AND/OR Decomposition and ATP Analysis [5].

Domain Modeler

The Domain Modeler captures the key concepts and vocabulary in the current and envisioned environment of the system, helping to further refine and validate requirements.

Organization Modeler

The Organization Modeler is responsible for documenting the Organization Model. Thus, the Organization Modeler must understand the system requirements, Goal Model, and Domain Model and be skilled in organizational modeling techniques.

Role Modeler

The Role Modeler creates the Role Model and the Role Description work products, which requires knowledge of the role model specification and a general knowledge of the system.

2.1.2 Activity Details

In the Requirement Analysis phase, there are three activities: Requirement Gathering, Problem Analysis, and Solution Analysis.

Requirement Gathering

Requirement Gathering is the process of identifying software requirements from a variety of sources. Typically, requirements are either functional requirements, which define the functions required by the software, or nonfunctional requirements, which specify traits of the software such as performance quality and usability. An overview of the Requirement Gathering tasks and work products used is shown in Fig. 9.

Problem Analysis

Problem Analysis captures the purpose of the product and documents the environment in which it will be deployed using three tasks: Model Domain, Model Goals,

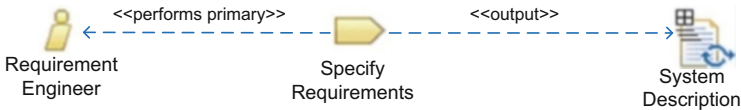


Fig. 9 Requirement Gathering activity diagram

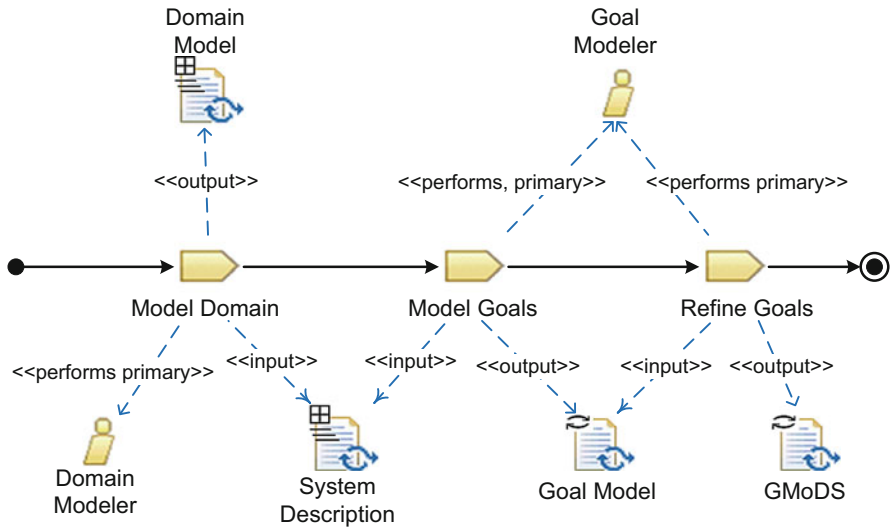


Fig. 10 Problem Analysis activity diagram

and Refine Goals. The Model Domain task captures the object types, relationships, and behaviors within the domain in which system will operate. The Domain Model captures the environment as a set of Object Types and Agents that are situated in the environment. Object types are defined by a name and a set of attributes. In O-MaSE, domain object types are similar to object classes rather than instances. The Model Goals task transforms the initial system requirements into a set of structured goals for the system. The deliverable of the Model Goals task is an initial Goal Model. The Refine Goals task captures the dynamic aspects of the Goal Model and further defines each goal using a technique called Attribute–Precede–Trigger Analysis. The result is a refined version of the Goal Model called a GMoDS goal model [5]. An overview of the Problem Analysis tasks and work products used is shown in Fig. 10.

Solution Analysis

Solution Analysis defines the required system behavior based on the goal and domain models. The end result is a set of roles and interactions in the Organization Model. Solution Analysis is decomposed into four tasks: Model Organizational Interfaces, Model Roles, and either Define Roles or Define Role Goals. The Model Organization Interfaces task identifies the organization’s interfaces with external entities, which can include other agents, organizations, or external actors.

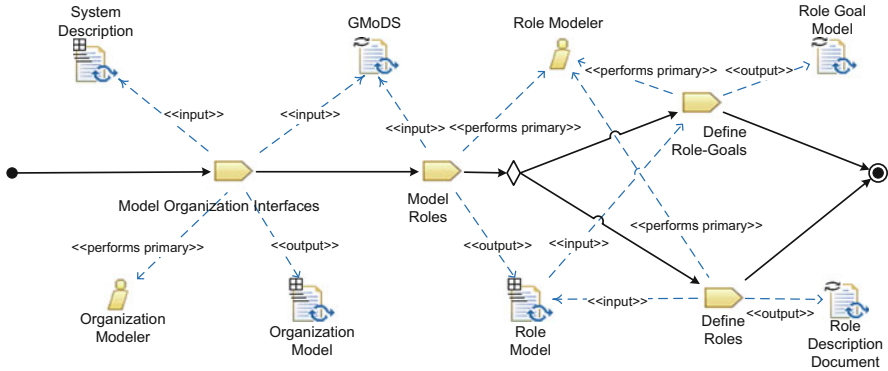


Fig. 11 Solution Analysis activity diagram

The Model Roles task identifies all the roles in the organization as well as their interactions with each other and with external actors, resulting in a Role Model. The goal of role modeling is to assign each leaf goal from the organization Goal Model to a specific role and to identify interactions between roles as well as with external actors. Interactions with external actors should be consistent with the Organization Model. The internal behavior of roles can be defined either through the Define Roles or Define Role Goal tasks and both types of definitions can be used within the same system. In the Define Roles task, the designer specifies the capabilities required by a role, the goals the role is able to achieve, constraints associated with the role, and the plan(s) that implements the role, which are defined via the Model Plan task as described below. In the Define Role Goals task, role behavior is defined by a role-level Goal Model. The top-level goal in a role-level Goal Model is the leaf goal from the organization that is to be achieved by the role. An overview of the Solution Analysis tasks and work products used is shown in Fig. 11.

2.1.3 Work Products

The Requirement Analysis phase produces eight work products. One is a pure text document, the System Description. The other seven work products are models that create various elements in the O-MaSE metamodel. The relationships between these models and the O-MaSE metamodel are documented in Fig. 12. Each model is defined in terms of elements from the O-MaSE metamodel, which are represented with UML class icons. Within each model, each metamodel element may be Defined, reFined, Quoted, Related, or Relationship Quoted.

Work Product Kinds

There are six possible work products produced in the Requirement Analysis phase: System Description Specification, Goal Model, GMoDS Model, Domain Model, Organization Model, and Role Model as defined in Table 4. Examples of most of these models are provided in the sections below. Each work product is specified in

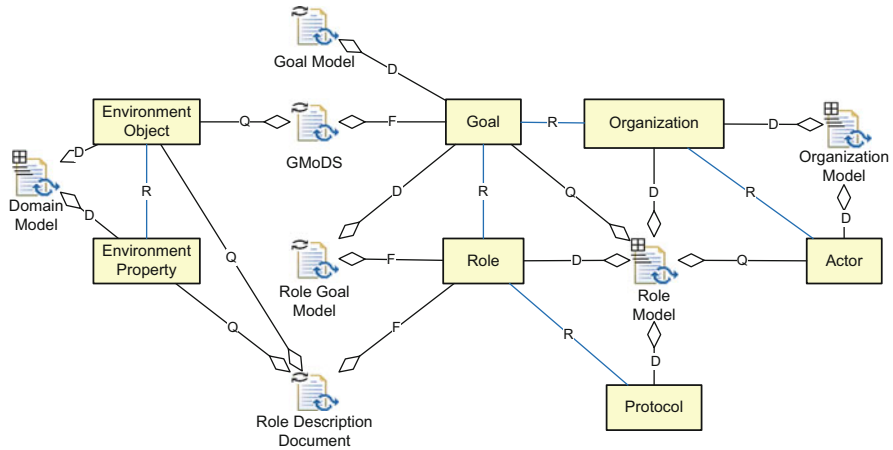


Fig. 12 Requirement Analysis document structure

Table 4 Requirement Analysis work products

Name	Description	Kind
System Description Specification	describes the technical requirements for a particular agent-oriented software	Structural
Goal Model	captures the purpose of the organization as a goal tree; includes goal attributes, precedence and triggering relationships	Behavioral
Domain Model	defines the language that can be used when defining the operation of the system	Structural
Organizational Model	documents the interaction between the organization and the external actors	Structural
Role Model	depicts organization roles, the goals they achieve and interactions between roles/external actors	Structural

terms of the kind of model, information, or data documented. A *structural* work product is used to model static aspects of the system, a *behavioral* work product is used to model dynamic aspects of the system, and a *composite* work product is used to model both static and dynamic aspects of the system. For further details on the differences between types of work products see [20].

System Description

There are many ways to capture and categorize requirements for use in systems. O-MaSE assumes that either traditional or multiagent-focused requirement gathering techniques are sufficient and thus does not stipulate a specific document structure.

Goal Model

The top-level CMS GMoDS goal model is shown in Fig. 13. (The initial version of the Goal Model from the Model Goals task is simply the GMoDS model with

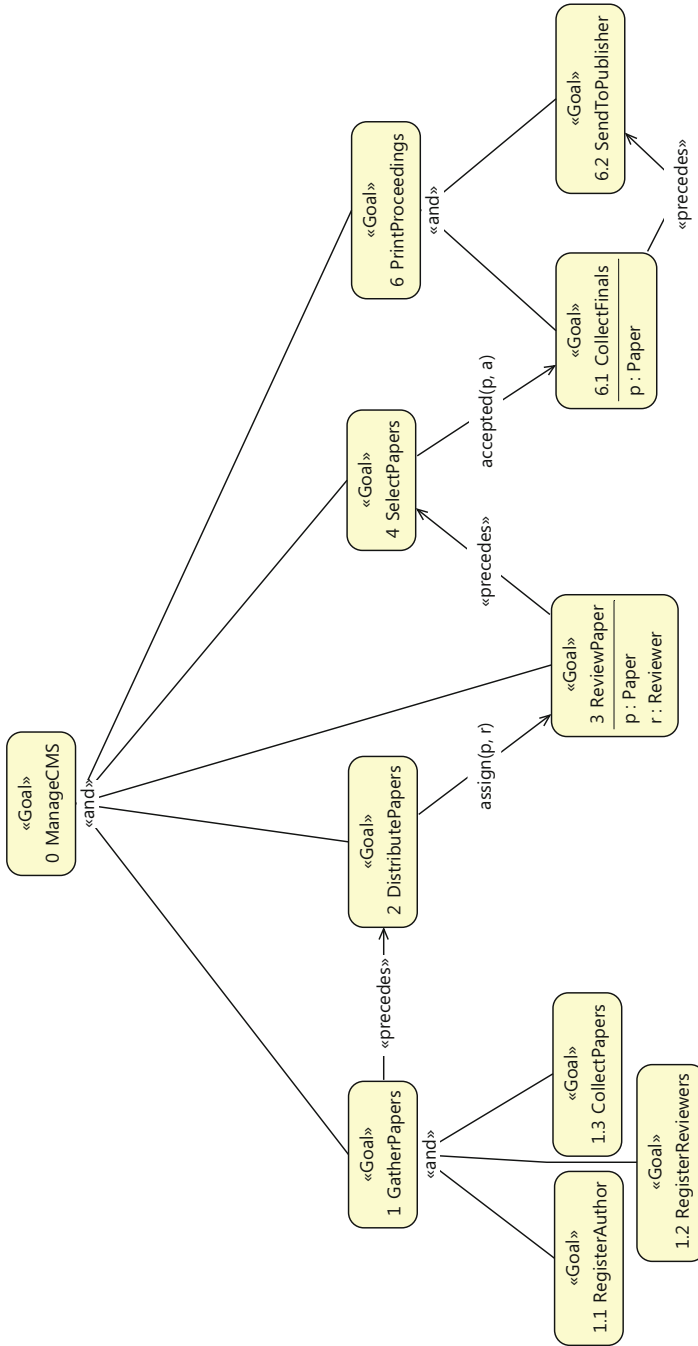


Fig. 13 Top-level GMoDS Goal Model for conference management system

the precede and trigger relations removed.) The overall goal of ManageCMS is broken down into five sub-goals: GatherPapers, DistributePapers, ReviewPaper, SelectPapers, and PrintProceedings. The GatherPapers goal is further decomposed into two sub-goals, RegisterAuthor and CollectPapers, while the PrintProceedings goal is also decomposed into the CollectFinals and SendToPublisher sub-goals. GMoDS allows new goal instances to be created via an *event trigger*, which is denoted by an arrow between two goals labeled by an event signature such as assign(p,r) on the arrow between the DistributePapers and ReviewPaper goals. In this case, it means that when an assign event occurs during pursuit of the DistributePapers goal, a new ReviewPaper goal is created. We use this to create a ReviewPaper goal for each reviewer assigned to review a given paper. Similarly, the accepted event during pursuit of the SelectPapers goal will create a new CollectFinals goal to collect each paper accepted for the proceedings.

The GMoDS precedence relation (denoted by an arrow labeled with «precedes») is used in the goal model to ensure proper sequence of actions in the system. Thus, since the papers must be gathered before they can be distributed for review, the GatherPapers goal precedes the DistributePapers goal (i.e., GatherPapers must be achieved before DistributePapers can begin to be pursued). Likewise, all the reviews must be performed (ReviewPapers goal) before papers can be selected (SelectPapers) for the conference, and all CollectFinals goals must be achieved before the SendToPublisher goal is attempted.

The PC organization is actually designed to achieve three top level goals: SendToPublisher, DistributePapers, and SelectPapers goals. The DistributePapers and SelectPapers goals are further decomposed as shown in Fig. 14 while the SendToPublisher is not.

The DistributePapers goal is decomposed in the AssignPapers and DisseminatePapers goals. While the description of the CMS provides for several ways to assign papers, there is only a single goal that drives that assignment process. How the PC actually achieves the assignment goal is defined by the process they use. Therefore, this aspect of the CMS definition should be captured as a process, which in the case of O-MaSE would be captured as variations to a plan.

The SelectPapers goal is decomposed into several sub-goals: CollectReviews, MakeDecision, and InformAuthors. Since all the reviews should be collected prior to making a decision, a precedence relation exists between the CollectReviews and MakeDecision goals. As a decision is made on each paper, a *declined* or an *accepted* event triggers either an InformDeclined or an InformAccepted goal for that paper. In addition, an *accepted* event triggers a CollectFinals goal in the CMS goal model as shown in Fig. 13.

Domain Model

The Domain Model is an essential part of problem analysis and is very important to the O-MaSE approach in general. The Domain Model defines the language that can be used by designer to ensure that everyone is talking about this same thing. It is also essential in formally defining the operation of the system from the attributes in the goal model to the information passed via system protocols to the policies of the system. The Domain Model for the CMS is shown in Fig. 15.

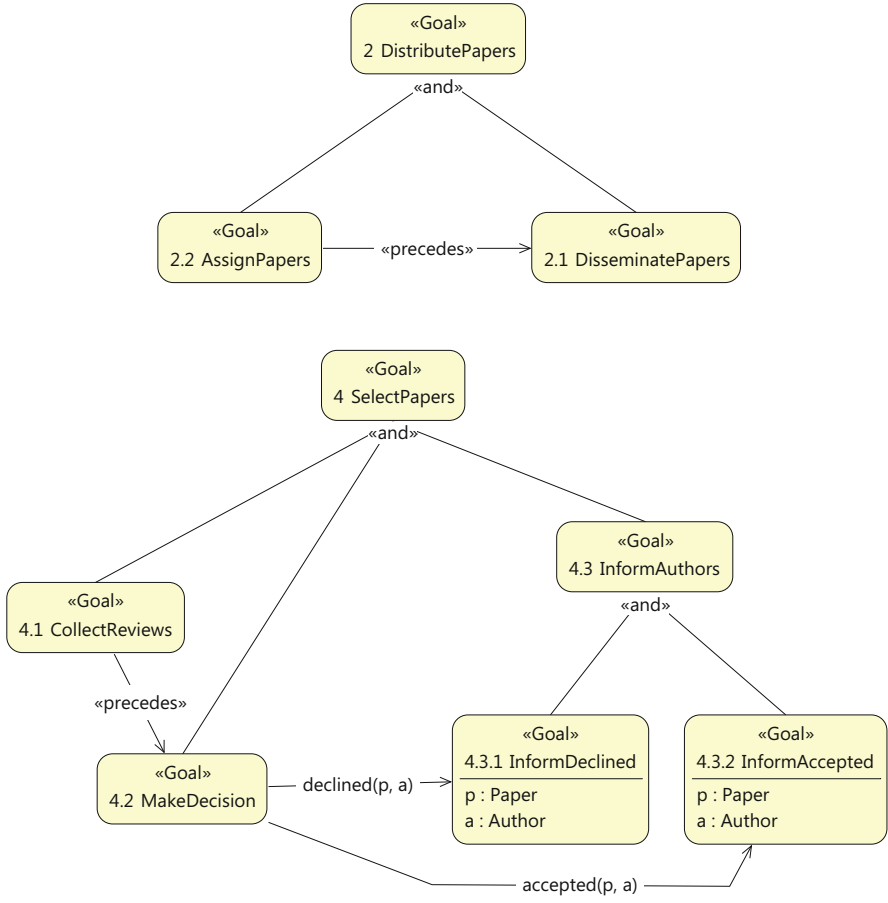


Fig. 14 Program committee Goal Model

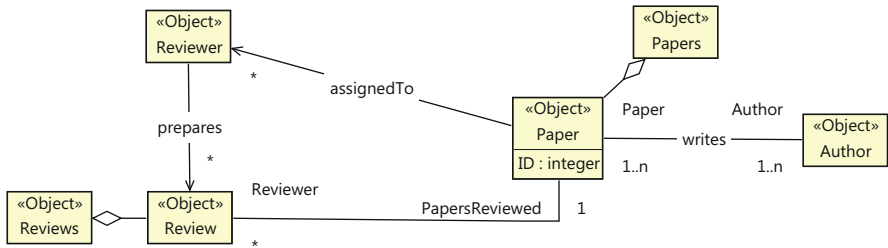


Fig. 15 Domain Model for conference management system

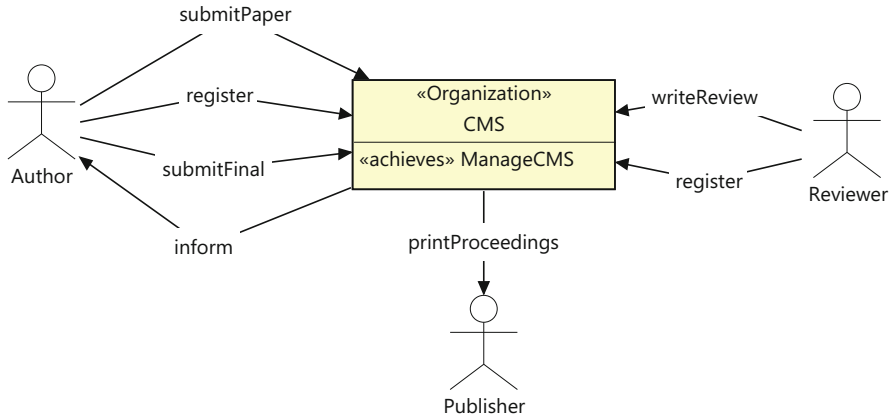


Fig. 16 Organization Model for conference management system

There are four main objects of interest in the CMS domain: Author, Paper, Reviewer, and Review. Each of these and their relationships are shown explicitly in the Domain Model. In addition, an aggregation of individual papers (Papers) and reviews (Reviews) is also defined. The model uses standard UML notation to define the multiplicities that are allowed between Objects. For instance, an Author must have at least one Author, and to be valid, an Author must have at least one Paper. It also shows that a Reviewer prepares a Review, and each Review has exactly one Paper that it can be written over. In addition, the Domain Model allows the definition of Object attributes. As shown, each Paper object has an ID.

Organizational Model

The Organization Model for the CMS is shown in Fig. 16. Here, the decision to make the PC a sub-organization shows up in the absence of the PC as an external actor. As shown, since the PC is a sub-organization, we are essentially considering it to be part of the system at this level. The three external actors shown, Author, Publisher, and Reviewer, all interact with the system through the given protocols. As the organization is refined into a Role and Agent Class Models, these external actors and protocols should show up in a consistent manner.

Role Model

The role model for the top-level CMS is shown in Fig. 17. There are six roles defined to carry out the goals defined in Fig. 13. Each role is designed to achieve a single goal as denoted by the «achieves» attribute in each role. Two exceptions are the PaperCollector role, which is designed to collect the initial and final copies of the papers, and the Registrar role, which is designed to register both authors and reviewers.

The role model also shows the external actors that interact with each of the roles. (As discussed above, the PC will actually be a sub-organization and thus the PC

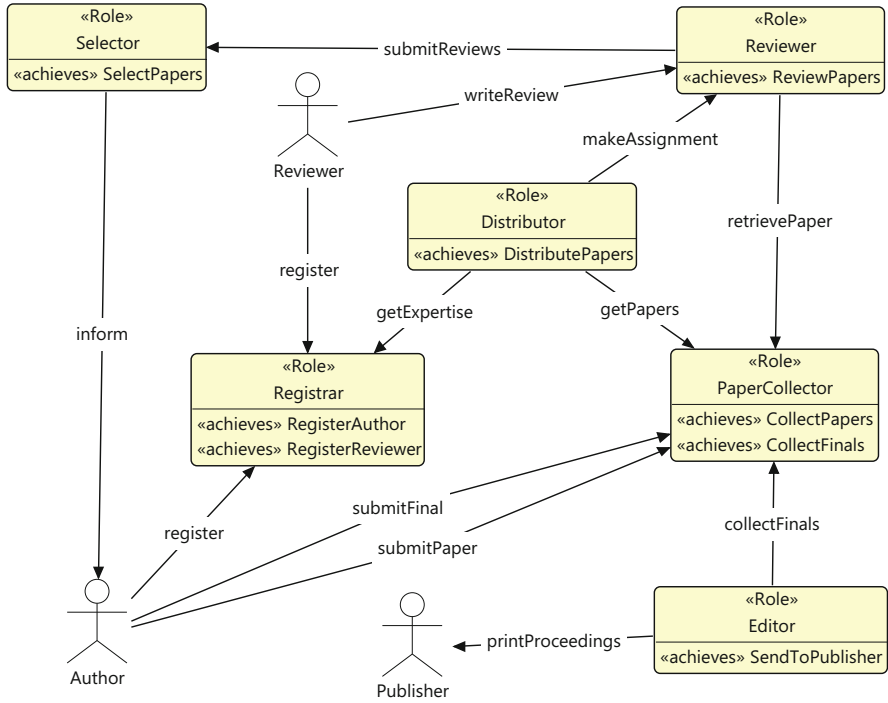


Fig. 17 Role Model for conference management system

Chair, Vice-Chairs, and PC Member external actors show up at a lower level of the design shown later.) The arrows between roles and between actors and roles represent protocols that support the passing of information. The direction of the arrow denotes who initiates the protocol and not the flow of information itself, which may occur in both directions. Each protocol is defined by a Protocol Model that shows the details of the messages and data passed. However, by studying both the goal model and role model, one can begin to understand the overall flow of the system.

The system begins with Author and Reviewer actors registering with the Registrar role and the Authors submitting papers to the PaperCollector role. Next, the Distributor role (which will eventually performed by the PC) gets the papers from the PaperCollector and the reviewer expertise from the Registrar and then assigns those papers to the reviewers. After all the Reviewers have submitted their reviews to the Selector, the Selector decides which papers are accepted and declined and informs the appropriate Authors. Once all the Authors have submitted the final version of their paper to the PaperCollector, the Editor takes those papers and sends them to the Publisher for publication.

The Role Model for the PC organization is shown in Fig. 18. Since the PC organization is designed to achieve three different CMS goals, the role model must

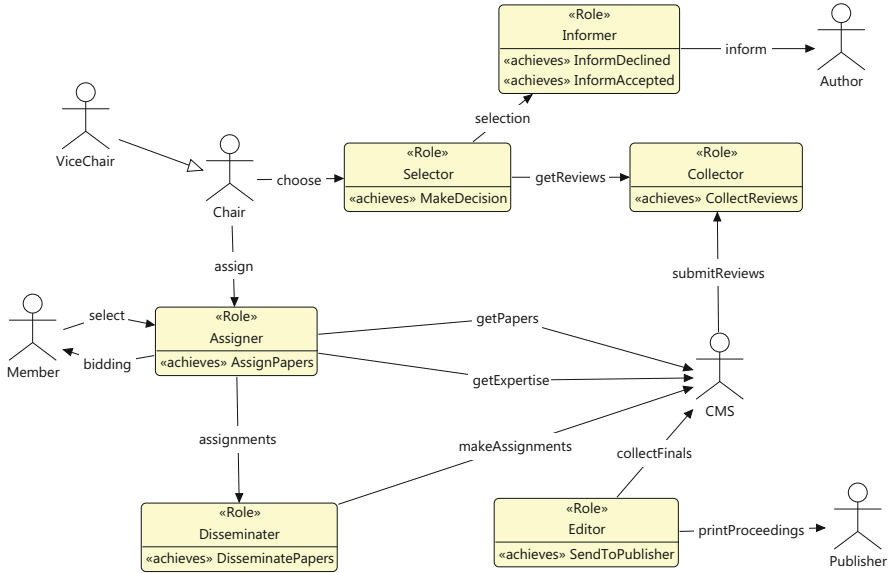


Fig. 18 Program Committee Role Model

accommodate all three. Notice that the actors of the PC organization include the PC Chair, Vice-chairs, and Members. In addition, the CMS is modeled as an external actor since its functionality lies outside the PC organization. As shown in the CMS Agent Class Model in Fig. 17, the PC organization must play the Editor, Distributor, and Selector roles defined in the CMS Role Model (Fig. 17).

The key to the correct decomposition of the PC organization is ensuring that the interfaces defined in the CMS Agent Class and Role models are consistently implemented in the PC organization. The easiest example of this is the CMS Editor role, which is implemented directly as a single Editor role in the PC organization. Notice that the protocols from the Editor to the Publisher and PaperCollector in the CMS Role Model are implemented as protocols to the CMS and Publisher actors in the PC Role Model. Thus, the CMS’s PaperCollector role is captured as part of the CMS actor in the PC Role Model.

The CMS Distributor role is implemented as two separate roles in the PC organization: Assigner and Disseminator. The Assigner role is used to encapsulate the various approaches to assigning papers to PC Members as defined in the CMS description. While the approaches are not defined in the Role Model, the protocols required for the various approaches are. For instance, the PC Chair can assign them directly and thus assign protocol from the PC Chair to the Assigner role. The PC Members can also be involved in selecting papers or can be part of a bidding process; these options require separate protocols: select and bidding. The process can also be carried out automatically by the Assigner role. Once the assignments have been made the Assigner role sends the assignments to the Disseminator role

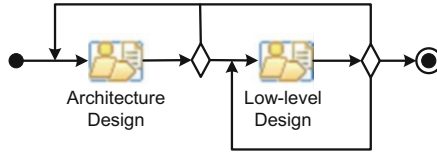


Fig. 19 Design-Phase flow of activities

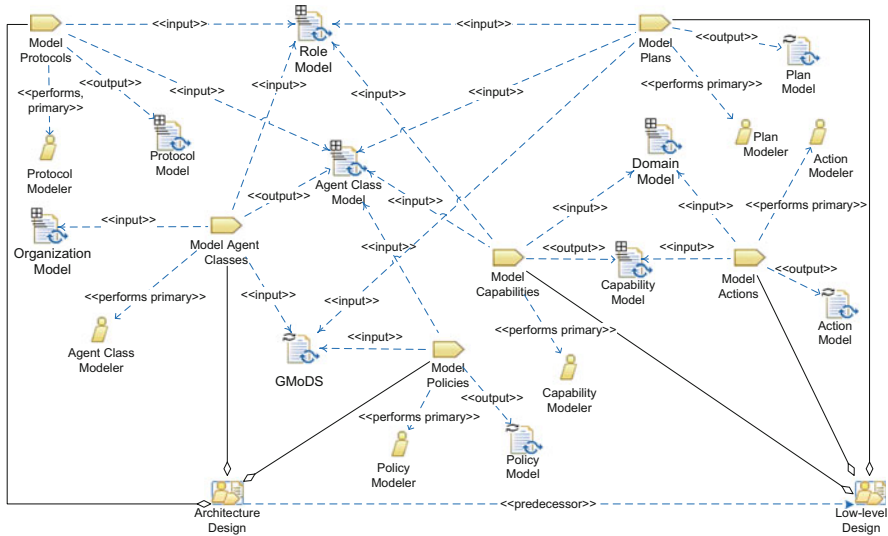


Fig. 20 Design Phase in terms of activities and work products

who is responsible for making assignments. These assignments are sent to the CMS actor, which includes the Reviewer role as defined in the CMS Role Model. We assume that if PC Members are Reviewers, they are registered in the system as both and are thus included as both an external PC Member actor and a CMS Reviewer.

2.2 Design

The design phase consists of two activities: Architecture Design and Low-Level Design. Once the goals, environment, behavior, and interactions of the system are known, Architecture Design is used to create a high-level description of the main system components and their interactions. This high-level description is then used to drive Low-Level Design, where the detailed specification of the internal agent behavior is defined. This low-level specification is then used to implement the individual agents during the Implementation phase. A generic example of an O-MaSE design phase is shown in Fig. 19 in terms of process flow and Fig. 20 in terms of process roles, work products, and tasks.

2.2.1 Process Roles

There are six roles in the design phase: Agent Class Modeler, Protocol Modeler, Policy Modeler, Capability Modeler, Plan Modeler, and Action Modeler.

Agent Class Modeler

The Agent Class Modeler is responsible for creating the Agent Class Model and requires general modeling skills and knowledge of the O-MaSE Agent Class Model specification.

Protocol Modeler

The Protocol Modeler designs the protocols required between agents, roles, and external actors and requires protocol modeling skills.

Policy Modeler

The Policy Modeler is responsible for designing the policies that govern the organization.

Capability Modeler

The Capability Modeler is responsible for defining the Capability Model and requires modeling skills and O-MaSE Capability Model specification knowledge.

Plan Modeler

The Plan Modeler designs the plans necessary to play a role; required skills include understanding of Finite State Automata and O-MaSE Plan Model specification knowledge.

Action Modeler

The Action Modeler documents the Action Model, which requires the ability to specify appropriate pre- and post-conditions for capability actions.

2.2.2 Activity Details

The Design phase has two activities: Architecture Design and Low-level Design. In the Architecture Design we focus on documenting the different agents, protocols, and policies using three tasks: Model Agent Classes, Model Protocols, and Model Policies. In the low-level design we focus on the capabilities possessed by, actions performed by, and plans followed by agents. The tasks of low-level design include Model Capabilities, Model Plans, and Model Actions.

Architecture Design

Architecture Design consists of three tasks as shown in Fig. 21. The Model Agent Classes task identifies the types of agents in the organization and the protocols between them. Agent classes may be defined by the roles they play or the capabilities they possess, which implicitly defines the roles they can play. Thus, an Agent Class provides a template for a type of agent in the system.

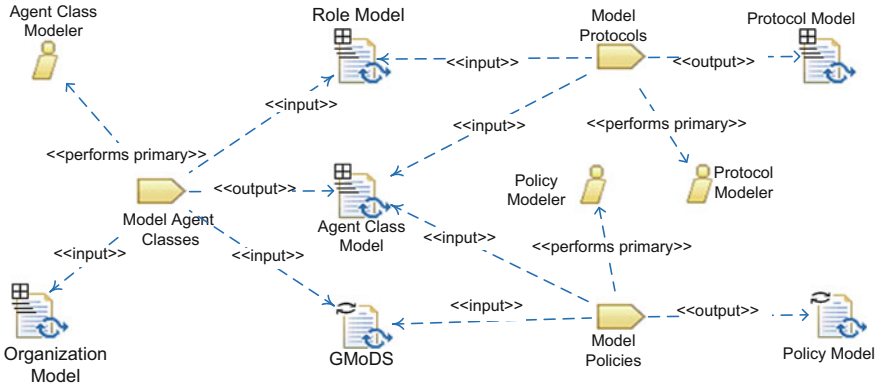


Fig. 21 Architecture Design activity diagram

The Model Protocols task specifies the details of the interactions between agents or roles. Since protocols can be specified in Organization Models, Role Models, and Agent Class Models, the method engineer may decide which set of protocols to define. If the Role Model protocols are defined via Protocol Models, agent classes playing those roles should inherit those protocols. The Protocol Model defines the types of messages sent between the two entities and is similar to UML interaction models [18].

The Model Policies task defines a set of rules that describe how an organization should behave. In general, policies are used to restrict agent behavior and may be enforced at design time or at runtime. How policies are enforced is a critical decision that affects the way the Policy Model is used during development. If there is no runtime mechanism designed or provided by the runtime environment, designs and implementations must be evaluated to ensure that they conform to the policies.

Low-Level Design

Low-level Design consists of three tasks as shown in Fig. 22. The Model Capabilities task defines the internal structure of the capabilities possessed by agents in the organization, which may be modeled as an Action or a Plan. An action is an atomic functionality possessed by an Agent and defined using the Model Actions task. A plan is an algorithmic definition of a capability and is defined using the Model Plans task.

The Model Plans task captures how an agent can achieve a specific type of goal using a set of actions specified as a Plan Model (a Finite State Machine). The Model Actions task defines the low-level actions used by agents to perform plans and achieve goals. Actions are typically defined as a function with a signature and a set of pre- and post-conditions. In some cases, actions may be modeled by providing detailed algorithmic information.

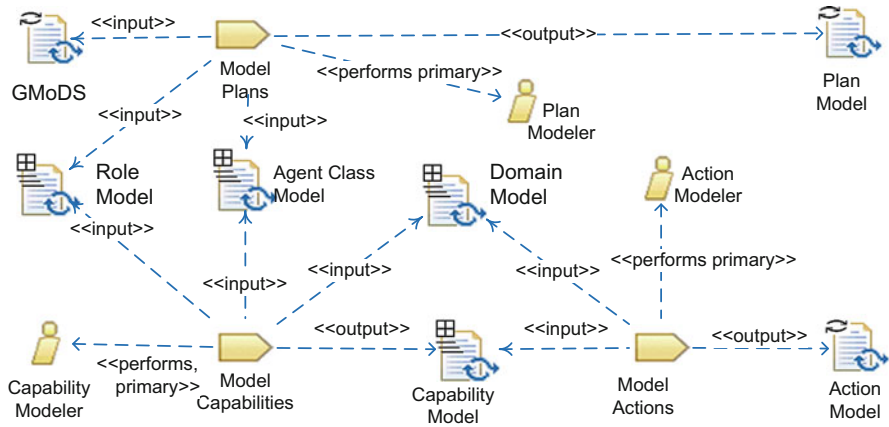


Fig. 22 Low-level Design activity diagram

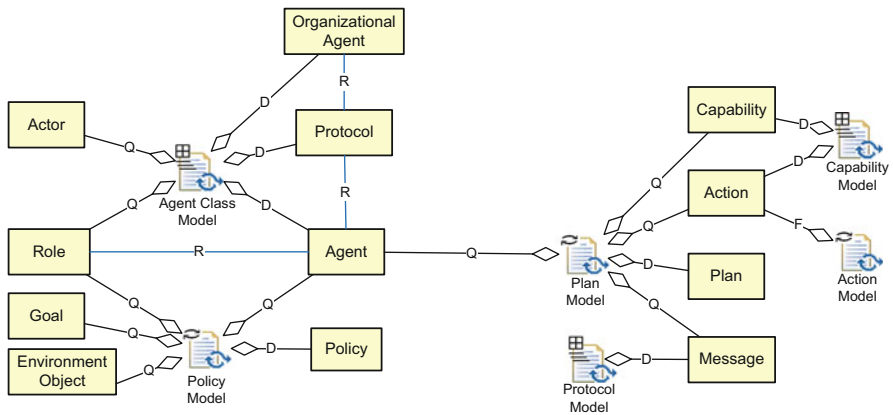


Fig. 23 Design document structure

2.2.3 Work Products

The Design phase produces six work products, each of which is a model that creates various elements in the O-MaSE metamodel. The relationships between these models and the O-MaSE metamodel are documented in Fig. 23. Each model is defined in terms of elements from the O-MaSE metamodel, which are represented with UML class icons. Within each model, each metamodel element may be Defined, reFined, Quoted, Related, or Relationship Quoted.

Work Product Kinds

There are six work products produced in the Design phase: Agent Class Model, Protocol Model, Policy model, Capability Model, Plan Model, and Action Model as defined in Table 5.

Table 5 Design work products

Name	Description	Kind
Agent Class Model	defines the agent classes and sub-organizations that will populate the organization.	Structural
Protocol Model	represents the different relations/interaction between external actors and agents/roles.	Structural
Policy Model	describes all the rules/constraints of the system	Behavioral
Capability Model	defines the internal structure of the capabilities possessed by agents in the organization.	Structural
Plan Model	captures how an agent can achieve a specific type of goal using a set of actions (which includes sending and receiving messages).	Behavioral
Action Model	defines the low-level actions used by agents to perform plans and achieve goals.	Behavioral

Agent Class Model

The Agent Class Model is shown in Fig. 24. There are three agents and one sub-organization (an organizational agent) defined to implement the six roles defined in the role model. The Database agent plays the PaperCollector role, the Registration agent plays the Registrar role, and the Reviewer agent plays the Reviewer role. The protocols defined in the role model are each inherited by the agent class model based on the roles assigned to the various agents.

A unique aspect of this design is the use of an organizational agent to capture the PC. In this design, the PC organization plays the Distributor, Selector, and Editor roles within the CMS. A further decomposition of this organizational agent is given below.

The Agent Class Model for the PC organization is shown in Fig. 25. The six roles from the PC Role Model result in four separate agents in the PC organization. Two agents simply implement single roles: the ReviewCollector agent plays the Collector role, while the Editor agent plays the Editor role. However, four other roles are combined into two agents. The reason for combining these roles is the fact that in both cases, there were two roles that communicated directly with each other and operated in a basically sequential manner. Therefore, the Disseminator and Assigner roles were combined into the PaperAssigner agent, while the Informer and Selector roles were combined into the PaperSelector role. The protocols and external agents were inherited directly from the Role Model, and no new protocols or external agents were added.

Protocol Model

The Model Protocols activity defines the internal details of each protocol identified in the Role and Agent Class models. At this point, all of the protocols in the CMS or the PC organization are modeled. One of the more interesting examples of a Protocol Model is shown in Fig. 26, which shows the *bidding* protocol between the Member actor and the Assigner agent. In reality, this protocol would likely be more

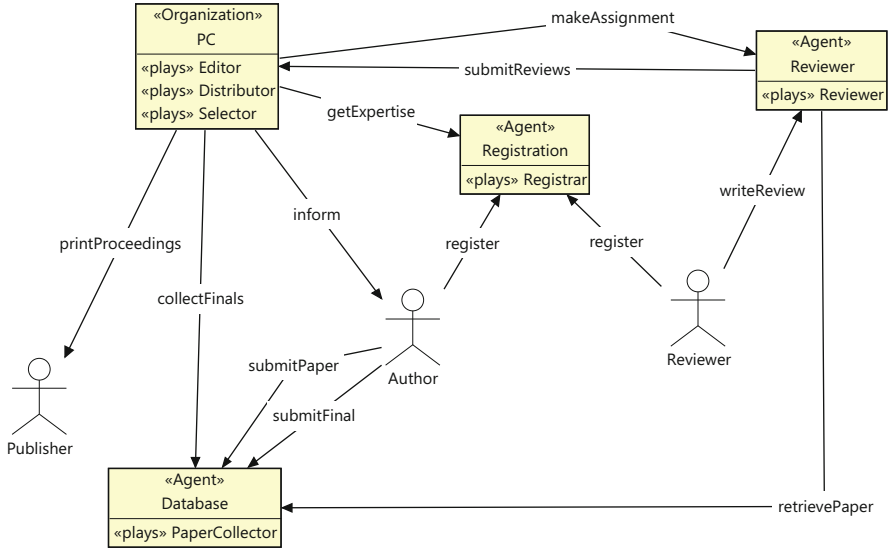


Fig. 24 Agent Class Model for conference management system

complex, possibly allowing the member to change bids or to protest an assignment; however, for the purposes of this example, we will take a simple approach. When the PC distributes papers via a bidding process, this protocol is used between the Assigner role and each Member of the PC committee. The protocol only captures the interactions between the Assigner and one Member—this protocol is repeated for each member. First, the PC Assigner role sends the *callForBids* message to the Member with a list of available papers. Next, the Member decides what bid to place on each of the papers in the set of papers received. For each paper the Member would like to bid on, a *bid* message is sent back to the Assigner. When the Member has completed bidding on papers, the Member sends a *done* message to the Assigner. Once the Assigner receives all the bids from all the Members, the Assigner decides the final assignments for each Member and a *assignment* message is sent to the Member and the protocol is complete.

Policy Model

As defined in [13], we use a language that includes temporal formula with quantification. For simplicity, we limit our examples to first-order predicate logic in this example. The language used to specify policies comes from entities defined in the various models, for example, objects defined in the Domain Model, the Roles defined in the Role Model, and the Agents defined in the Agent Class Model. While most of the interesting policies apply the PC organization, we do want each paper to have a unique ID. Thus, a policy to ensure that each paper has a unique ID can be stated as

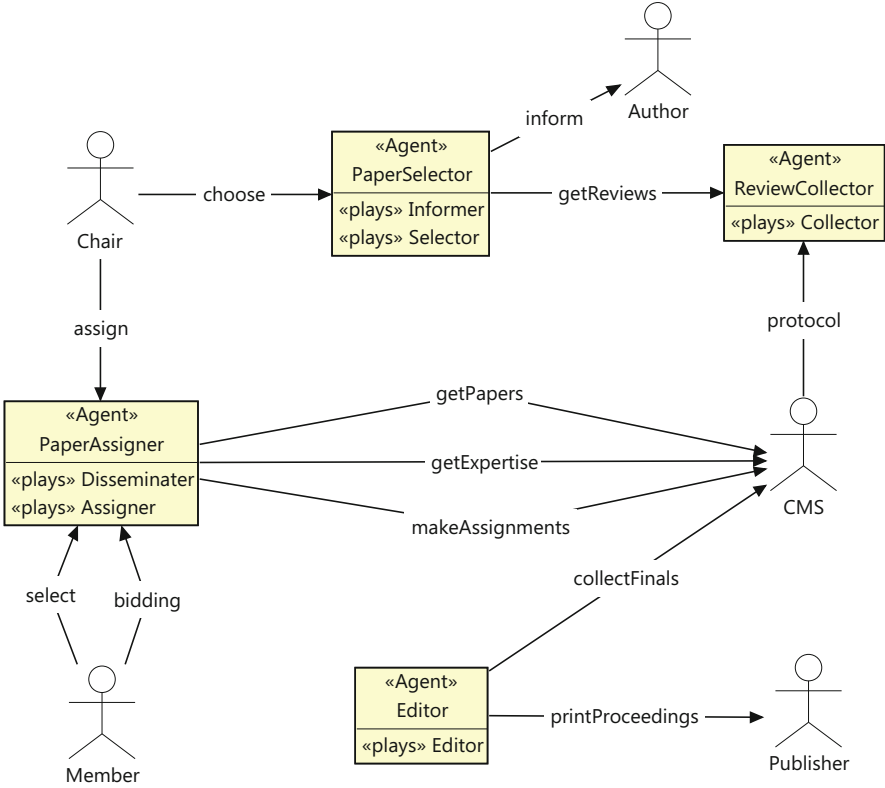


Fig. 25 Program Committee Agent Class Model

$$\forall p1, p2 : Paper \ p1.ID = p2.ID \Rightarrow p1 = p2$$

The CMS requires that a PC Member or Reviewer may not see or infer information about their own submissions. Essentially, this requires that PC Members would not be able to see any reviews or decisions made about their papers except through the normal inform protocol with Authors. We assume here that the design of the system only allows a PC Member (not the Chair) to see submitted reviews through the submitReview protocol with the Collector role. We also assume that the Member can only see reviews related to papers that the member has submitted a review for. Thus to specify that a member cannot review their own paper and that a Member may not view reviews related to their papers, we can specify the following policies:

$$\forall p : Paper, m : Member, a : Author \ a = m \wedge writes(a, p) \Rightarrow m \notin assignedTo(p, m)$$

$$\forall p : Paper, m : Member \ submitReview(m, p) \Rightarrow m \in assignedTo(p, m)$$

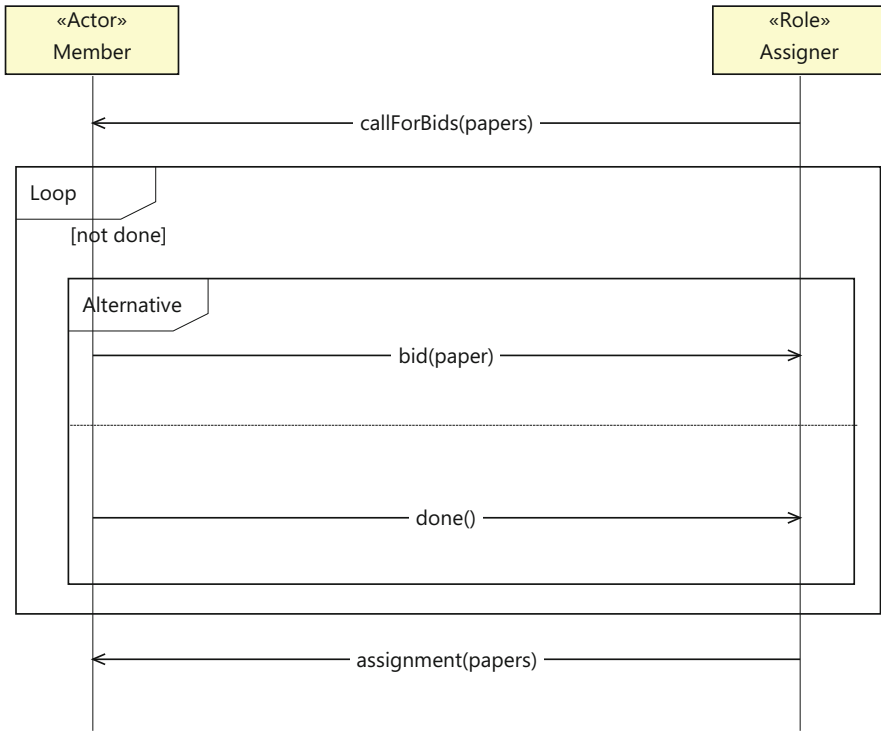


Fig. 26 Program Committee bidding protocol

The first policy states that if a Member is also an Author and writes a paper, that Member cannot be in the set of Reviewers in the assignedTo association with that paper. The second policy states that if a Member submits a review of a paper, then that Member must be in the set of assigned reviewers in the assignedTo association with that paper.

Capability Model

Capability Model captures the internal structure of the capabilities possessed by agents in the organization. Each capability may be modeled as an Action or a Plan. An action is an atomic functionality possessed by an Agent and defined using an Action Model as described in section “Action Model”. A plan is an algorithmic definition (defined via a state machine) of a capability that uses actions and implements protocols. Each plan is defined using a Plan Model as presented in section “Plan Model”.

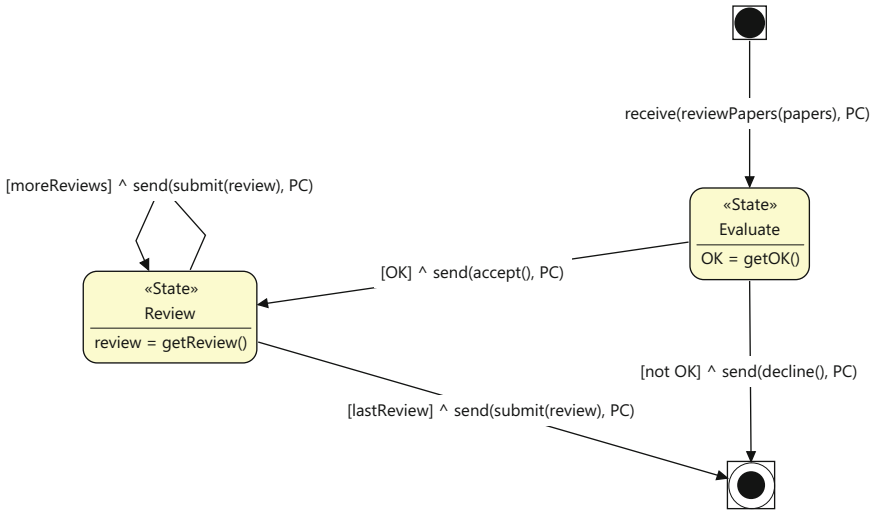


Fig. 27 Program Committee bidding protocol

Plan Model

Typically, a plan is required for each type of goal that an agent can achieve. Thus, since agents are defined by the roles they play, one must look at the goals that can be achieved by each role the agent can play. We illustrate this process with the Reviewer agent. Since the Agent Class Model in Fig. 24 defines that Reviewer agent can only play the Reviewer role, we only need to look at the goals that can be achieved by the Reviewer role. In the Role Model of Fig. 17 we can see that the Reviewer role is designed to only achieve the ReviewPapers goal. Therefore, we only need to define a single plan to fully define the behavior of the Reviewer agent.

The ReviewPapers plan for the Review agent is shown in Fig. 27. It is defined by a simple-state machine that starts when the agent receives a *reviewPapers* message from the PC organization (denoted by the label on the transition from the start state to the Evaluate state). The Reviewer has the right to accept or reject the papers presented. If accepted, the Reviewer sends an *accept* message to the PC and enters the Review state. Here the Reviewer agent waits for the actual human reviewer to enter reviews. As each review is received, the Reviewer sends the review to the PC via a *review* message. When all reviews have been received, the Review plans end.

Action Model

The Action Model defines the low-level actions used by agents to perform plans and achieve goals. Actions belong to capabilities possessed by agents. Actions are typically defined as a function with a signature and a set of pre- and post-conditions. In some cases, actions may be modeled by providing detailed algorithmic information. If using automatic code generation techniques, this information is generally captured as a function or an operation in the language being generated. In either case, the Action Model is usually just a textual document.

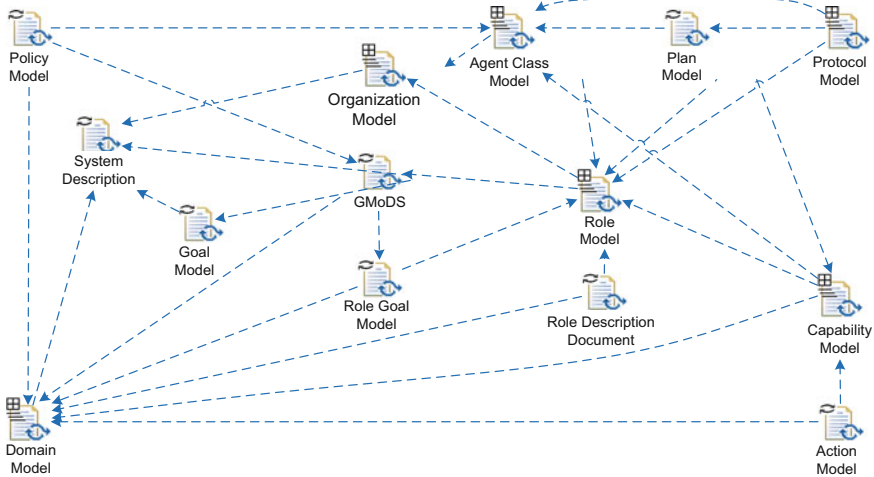


Fig. 28 Work Product Dependencies

2.3 Implementation

Finally, the design is translated to code. The purpose of this phase is to take all the design models created during the design and convert them into code that correctly implements the models. Obviously, there are numerous approaches to code generation based on the runtime platform and implementation language chosen. In this phase there is a single Role, the Programmer who is responsible for writing code based on the various models produced during the Design phase. The output of the Generate Code task is the source code of the application. While not currently covered in the process, system creation ends with testing, evaluation, and deployment of the systems.

3 Work Product Dependencies

Figure 28 identifies the dependencies between all the work products in O-MaSE. These dependencies characterize different pieces of information produced during the different stages of the development process and serve as inputs to and outputs of work units (i.e., either activities or tasks).

References

1. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. *Inf. Softw. Technol.* **38**, 275–280 (1996)
2. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardization to research. *Int. J. Agent Oriented Softw. Eng.* **1**, 91–121 (2007)

3. Cossentino, M., Gaud, N., Hilaire, V., Galland, S., Koukam, A.: ASPECS: an agent-oriented software process for engineering complex systems. *J. Auton. Agent Multi Agent Syst.* **20**, 260–304 (2009)
4. DeLoach, S.A., Garcia-Ojeda, J.C.: O-MaSE: a customizable approach to designing and building complex, adaptive multiagent systems. *Int. J. Agent Oriented Softw. Eng.* **4**, 244–280 (2010)
5. DeLoach, S.A., Miller, M.: A goal model for adaptive complex systems. *Int. J. Comput. Intell. Theory Pract.* **5**, 83–92 (2010)
6. DeLoach, S.A., Valenzuela Jorge, L.: An agent-environment interaction model. In: Padgham, L., Zambonelli, F. (eds.) *Agent-Oriented Software Engineering VII: 7th International Workshop, AOSE 2006*. Lecture Notes in Computer Science, vol. 4405, pp. 1–18. Springer, Heidelberg (2006)
7. DeLoach, S.A., Oyanan, W., Matson, E.T.: A capabilities based model for artificial organizations. *Auton. Agent Multi Agent Syst.* **16**, 13–56 (2008)
8. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Proceedings of the 3rd International Conference on Multi Agent Systems, pp. 128–135. IEEE Computer Society, Washington (1998)
9. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Muller, J.P., Odell, J. (eds.) *Agent-Oriented Software Engineering IV*. Lecture Notes in Computer Science, vol. 2935, pp. 214–230. Springer, Berlin (2003)
10. Garcia-Ojeda, J.C., DeLoach, S.A.: Robby: agentTool process editor: supporting the design of tailored agent-based processes. In: Proceedings of the 2009 ACM Symposium on Applied Computing (SAC '09), pp. 707–714. ACM, New York (2009)
11. Garcia-Ojeda, J.C., DeLoach, S.A.: Robby: agentTool III: from process definition to code generation. In: Decker, K., Sichman, J., Sierra, G., Castelfranchi, C. (eds.) Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), vol. 2, pp. 1393–1394. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2009)
12. Garcia-Ojeda, J.C., DeLoach, S.A., Robby, Oyanan, W.H., Valenzuela, J.: O-MaSE: a customizable approach to developing multiagent development processes. In: Luck, M., Padgham, L. (eds.) Proceedings of the 8th International Conference on Agent-oriented Software Engineering VIII (AOSE'07). Lecture Notes in Computer Science, vol. 4951, pp. 1–15. Springer, Berlin (2007)
13. Harmon, S., DeLoach, S.A., Robby: trace-based specification of law and guidance policies for multiagent systems. In: Artikis, A., O'Hare, G.M., Stathis, K., Vouros, G. (eds.) *Engineering Societies in the Agents World VIII*. Lecture Notes in Artificial Intelligence, vol. 4995, pp. 333–349. Springer, Berlin (2008)
14. Holland, O., Melhuish, C.: Sigmergy, self-organization, and sorting in collective robotics. *Artif. Life* **5**, 173–202 (1999)
15. Object Management Group. Software and systems process engineering meta-Model specification, v2.0. Object Management Group. <http://www.omg.org/spec/SPEM/2.0/PDF> (2008). Accessed 14 May 2012
16. Odell, J., Parunak, H., Bauer, B.: Representing agent interaction protocols in UML. In: Wooldridge, M.J., Ciancarini, P. (eds.) *First International Workshop, AOSE 2000 on Agent-oriented Software Engineering*, pp. 121–140. Springer, New York (2001)
17. Odell, J., Nodine, M., Levy, R.: A metamodel for agents, roles, and groups. In: Giorgini, P., Muller, J. (eds.) Proceedings of the 5th International Conference on Agent-Oriented Software Engineering (AOSE'04). Lecture Notes in Computer Science, vol. 3382, pp. 78–92. Springer, Berlin (2005)
18. Rumbaugh, J., Jacobson, I., Booch, G.: *The Unified Modeling Language Reference Manual*, 2nd edn. Addison-Wesley, Upper Saddle River (2004)

-
19. Russell, S.J., Norvig, P.: *Artificial Intelligence: A Modern Approach*, 2nd edn. Prentice Hall, Upper Saddle River (2002)
 20. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent systems design. In: *Proceedings of the 7th Workshop from Objects to Agents (WOA 2006)*, pp. 130–137 (2006)

PASSI: Process for Agent Societies Specification and Implementation

Massimo Cossentino and Valeria Seidita

Abstract

PASSI (a Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology for designing and developing multiagent societies, integrating design models and concepts from both Object-Oriented software engineering and artificial intelligence approaches using the UML notation. The models and phases of PASSI encompass anthropomorphic representation of system requirements, social viewpoint, solution architecture, code production and reuse, and deployment configuration supporting mobility of agents. PASSI is made up of five models, concerning different design levels, and 12 activities performed to build multiagent systems. In PASSI, the UML notation is used as the modeling language, since it is widely accepted both in the academic and industrial environments.

1 Introduction

PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology for designing and developing multiagent societies. The methodology integrates design models and concepts from both Object-Oriented software engineering and artificial intelligence approaches. PASSI

M. Cossentino

Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, Palermo, Italy

e-mail: cossentino@pa.icar.cnr.it

V. Seidita (✉)

Dipartimento di Ingegneria Chimica, Gestionale, Informatica, Meccanica, University of Palermo, Palermo, Italy

e-mail: valeria.seidita@unipa.it

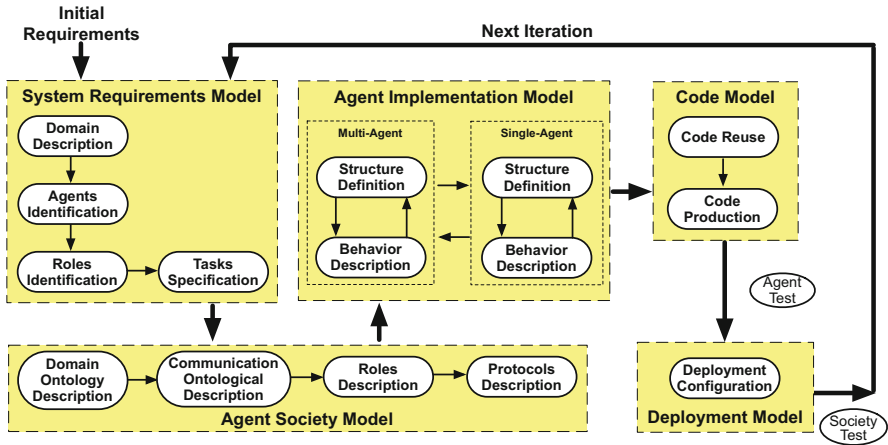


Fig. 1 The PASSI design process

has been conceived in order to design FIPA-compliant agent-based systems, initially for robotics and information systems applications. Systems designed by using the PASSI process are usually composed of peer agents (although social structures can be defined). According to FIPA specifications, agents are supposed to be mobile, and they can interact by using semantic communications, referring to an ontology and interaction protocol. PASSI is suitable for the production of medium–large MASs distributed in several execution nodes. The adoption of patterns and the support of specific CASE tools (PTK) allow a quick and affordable production of code for the JADE platform. This encourages the use of this process even in time/cost-constrained projects or where high quality standards have to be met.

The design process is composed of five models (see Fig. 1): the System Requirements Model is a model of the system requirements; the Agent Society Model is a model of the agents involved in the solution in terms of their roles, social interactions, dependencies, and ontology; the Agent Implementation Model is a model of the solution architecture in terms of classes and methods (at two different levels of abstraction: multi- and single agent); the Code Model is a model of the solution at the code level; and the Deployment Model is a model of the distribution of the parts of the system (i.e., agents) across hardware processing units and their movements across the different available platforms.

In the following, the PASSI process will be described by initially considering its whole process and then its five components, each of them representing a phase, a portion of work for which a specific outcome and milestones can be identified and represented in the following diagram.

The Conference Management System (CMS) case study is used for better illustrating the notation used in all the PASSI work products; the case study is not completely reported because of space concerns, portions of diagrams are shown, and we have selected elements with the intent to give a complete view on the used notation. Useful references about the PASSI process are as follows:

- M. Cossentino. From Requirements to Code with the PASSI Methodology. In *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini (Editors). Idea Group Inc., Hershey, PA, USA. 2005.
- M. Cossentino, S. Gaglio, L. Sabatucci, and V. Seidita. The PASSI and Agile PASSI MAS Meta-models Compared with a Unifying Proposal. *Lecture Notes in Computer Science*, vol. 3690. Springer-Verlag GmbH. 2005. pp. 183–192.
- M. Cossentino and L. Sabatucci. *Agent System Implementation in Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance*. CRC Press, April 2004.
- M. Cossentino, L. Sabatucci, and A. Chella. Patterns reuse in the PASSI methodology. In *Engineering Societies in the Agents World IV, 4th International Workshop, ESAW 2003, Revised Selected and Invited Papers*, volume 3071 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2004. pp. 294–310
- M. Cossentino, L. Sabatucci, A. Chella - A Possible Approach to the Development of Robotic Multi-Agent Systems - *IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03)*. October, 13–17, 2003. Halifax (Canada)
- Chella, M. Cossentino, and L. Sabatucci. Designing JADE systems with the support of case tools and patterns. *Exp Journal*, 3(3):86–95, Sept 2003.

Useful references about PASSI extensions are as follows:

- M. Cossentino, N. Gaud, V. Hilaire, S. Galland, A. Koukam. ASPECS: an Agent-oriented Software Process for Engineering Complex Systems. *International Journal of Autonomous Agents and Multi-Agent Systems (IJAAMAS)*. 20(2). 2010.
- M. Cossentino, G. Fortino, A. Garro, S. Mascillaro and W. Russo. PASSIM: a simulation-based process for the development of MASs. *International Journal on Agent Oriented Software Engineering (IJAOSE)*, 2(2). 2009.
- V. Seidita, M. Cossentino, S. Gaglio. Adapting PASSI to Support a Goal Oriented Approach: a Situational Method Engineering Experiment. *Proc. of the Fifth European workshop on Multi-Agent Systems (EUMAS'07)*. 13–14 December, 2007. Hammameth (Tunisia).
- A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. Agile PASSI: An Agile Process for Designing Agents. *International Journal of Computer Systems Science & Engineering*, 21(2). March 2006.

In the following sections, all the aspects of PASSI are described by using SPEM 2.0 [4] and the extensions proposed by Seidita et al. [6]. Figure 2 shows the SPEM 2.0 icons that the reader can find in the following figures.

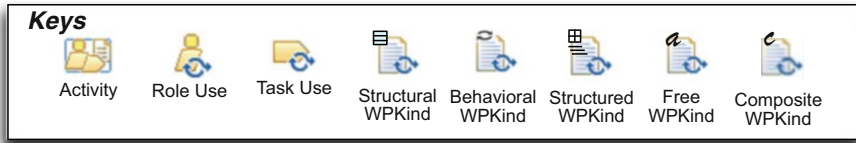


Fig. 2 The SPEM 2.0 icons



Fig. 3 The PASSI process phases

1.1 The PASSI Process Life Cycle

PASSI includes five phases arranged in an iterative/incremental process model (see Fig. 3):

- **System Requirements:** It covers all the phases related to requirements elicitation, requirements analysis, and agents/roles identification
- **Agent Society:** All the aspects of the agent society are addressed: ontology, communications, roles description, and interaction protocols
- **Agent Implementation:** A view on the system's architecture in terms of classes and methods to describe the structure and the behavior of single agents.
- **Code:** A library of class and activity diagrams with associated reusable code and source code for the target system.
- **Deployment:** How the agents are deployed and which constraints are defined/identified for their migration and mobility.

Each phase produces a document that is usually composed by aggregating UML models and work products produced during the related activities. Each phase is composed of one or more subphases, each one responsible for designing or refining one or more artifacts that are part of the corresponding model. For instance, the System Requirements model includes an agent identification diagram that is a kind of UML use case diagram but also with some text documents like a glossary and the system usage scenarios. The details of each phase will be discussed in the following sections.

Problem Domain

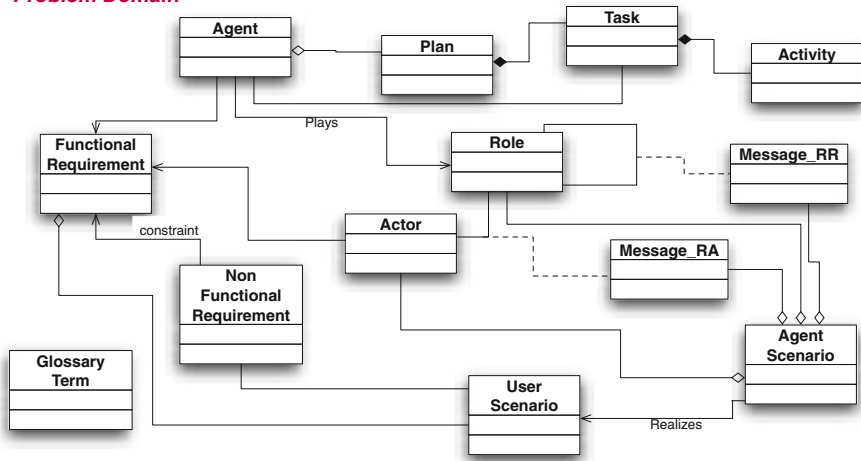


Fig. 4 The PASSI system metamodel—problem domain

1.2 The PASSI System Metamodel

The description of the PASSI MAS metamodel (MMM) addresses three logical areas: (1) the problem domain (Fig. 4), (2) the agency domain (Fig. 5), and (3) the solution domain (Fig. 6).

In the problem domain, the designer includes components describing the requirements that the system is going to accomplish: these are directly related to the requirements analysis phase of the PASSI process.

Then the designer introduces the agency domain components, which are used to define an agent solution for the problem. Finally, in the PASSI MMM solution domain, agency-level components are mapped to the adopted FIPA-compliant implementation platform elements. Here, it is supposed that the platform supports at least the concepts of agent and task. This represents the code-level part of the solution and the last refinement step. Going into the details of the model, the Problem Domain deals with the user’s problem in terms of scenarios, requirements, ontology, and resources. Scenarios describe a sequence of interactions among actors and the system to be built. The ontological description of the domain is composed of concepts (categories of the domain), actions (performed in the domain and effecting the status of concepts), and predicates (asserting something about a portion of the domain, i.e., the status of concepts). Resources are the last element of the problem domain. They can be accessed/shared/manipulated by

agents. A resource could be a repository of data (like a relational database), an image/video file, or also a product to be sold/bought. The Agency Domain contains the components of the agent-based solution. None of these components is directly implemented; they are converted to the correspondent object-oriented entity that constitutes the real code-level implementation. The key concept of this domain is the agent. An agent is responsible for realizing some functionalities descending from one or more functional requirements. It has also to respect some nonfunctional requirement constraints (like, for instance, performance prescriptions). The agent is a situated entity that lives in an environment from which it receives perceptions (the related knowledge is structured according to the designed domain ontology). Sometimes an agent, has access to available resources, and it is capable of actions in order to pursue its own objectives or to offer services to the community. The functionality of an agent is organized through roles. Each agent during its life plays some roles. A role is a peculiarity of the social behavior of an agent. When playing a role, an agent may provide a service to other agents executing tasks and processing messages. A task specifies the computation that generates the effects of a specific agent behavioral feature. This means that an agent's behavior can be composed by assembling its tasks, and the list of actions that are executed within each task cannot be influenced by the behavior planning. Tasks are structural internal components of an agent, and they contribute to define the agent's abilities. An agent's tasks cannot be directly accessed by other agents unless the agent itself offers them as a set of services; this is an obvious consequence of considering agents as autonomous entities. A communication is an interaction between two agents and it is composed of one or more messages seen as speech acts. The information exchanged during a communication is composed of concepts, predicates, or actions defined in the ontology. The flow of messages and the semantics of each message are ruled by an agent interaction protocol (AIP). The last Agency Domain element is the service. It describes a set of coherent functionalities exported by the agent for the community. The Implementation Domain describes the structure of the code solution in the chosen FIPA-compliant implementation platform (for instance, JADE). It is essentially composed of three elements: (1) the FIPA-Platform Agent, which is the base class catching the implementation of the Agent entity represented in the Agency domain; (2) the FIPA-Platform Task, which is the implementation of the agent's Task; and (3) the ServiceDescription component, which is the implementation-level description (for instance, an OWL-S file) of each service specified in the Agent Domain.

1.2.1 Definition of the System Metamodel Elements (Table 1)

Table 1 Definition of the system metamodel elements

Concept	Definition	Domain
Functional Requirement	Functional requirements describe the functions that the software is to execute. (from IEEE SEBOK 2004)	Problem
Nonfunctional Requirement	Nonfunctional requirements constrain the solution and are sometimes known as constraints or quality requirements. (from IEEE SEBOK 2004)	Problem
Actor	An external entity (human or system) interacting with the multi agent system.	Problem
Glossary Term	Terms used in the project so that everyone has a common understanding of them.	Problem
User Scenario	“A narrative description of what people do and experience as they try to make use of computer systems and applications” [M. Carrol, Scenario-based Design, Wiley, 1995]	Problem
Agent	We consider two different aspects of the agent: during the initial steps of the design, it is seen (this is the Agency Domain Agent) as an autonomous entity capable of pursuing an objective through its autonomous decisions, actions, and social relationships. This helps in preparing a solution that is later implemented, referring to the agent as a significant software unit (this is the Solution Domain FIPA-Platform Agent). In detail, an Agent is an entity that <ul style="list-style-type: none"> • Is capable of action in an environment • Can communicate directly with other agents typically using an Agent Communication Language • Is driven by a set of functionalities it has to accomplish • Possesses resources of its own • Is capable of perceiving its environment • Has only a partial representation of this environment in the form of an instantiation of the domain ontology (knowledge) • Can offer services • Can play several different (and sometimes concurrent or mutually exclusive) roles 	Problem
Role	A portion of the social behavior of an agent that is characterized by a goal (accomplishing some specific functionality) and/or provides a service.	Problem
Message_RR	A message exchanged between two roles.	Problem
Message_RA	An interaction between an agent and a role. This can be a message as well as another kind of interaction (for instance, GUI-based).	Problem
Plan	The behavior of an Agent is specified within its plan. It is the description of how to combine and order Tasks and interactions to fulfill a (part of a) requirement.	Problem
Task	A task specifies the computation that generates the effects of the behavioral feature. It is a nondecomposable group of atomic actions that cannot be directly addressed without referring to their belonging task.	Problem
Activity	The composing unit of a task. An activity takes a set of inputs and converts them into a set of outputs, though either or both sets may be empty. An Activity can either be decomposable into other activities or atomic.	Problem
Message_Type	The portion of communication related to the communicative act.	Problem
Ontology Element	Element (abstract class). An ontology is composed of concepts, actions, and predicates. An Ontology element is an abstract class used as a placeholder for the ontology constituting elements (concepts, predicates, or actions). Concept —Description of a certain identifiable entity of the domain. Action —It expresses an activity carried out by an agent. Predicate —Description of a property of an entity of the domain.	Problem
Service	A service is a single, coherent block of activity in which an agent will engage. A set of services can be associated with each agent role.	Agency

(continued)

Table 1 (continued)

Concept	Definition	Domain
Agency_Agent	An autonomous entity capable of pursuing an objective through its autonomous decisions, actions, and social relationships. It is capable of performing actions in the environment it lives; it can communicate directly with other agents, typically using an Agent Communication Language; it possesses resources of its own; it is capable of perceiving its environment; it has a (partial) representation of this environment in the form of an instantiation of the domain ontology (knowledge); it can offer services; it can play several, different (and sometimes concurrent or mutually exclusive) <i>agency_roles</i> . Each agent may be refined by adding knowledge items necessary to store/manage communication contents.	Agency
Agency_Task	Each task is an entity that aims to reach a subgoal (e.g., dealing with a communication or executing some transformations on a specific resource). The term <i>task</i> can be used as a synonym of behavior but with the significance of atomic part of the overall agent behavior.	Agency
Agency_Role	A portion of the behavior of an agent that is characterized by an objective (accomplishing some specific functionality) and/or that provides a service.	Agency
Resource	A entity existing in the environment.	Agency
Communication	An interaction among two agents, referring to an Agent Interaction Protocol and a piece of the domain ontology (knowledge exchanged during the interaction). Usually it is composed of several messages, each one associated with one Performative.	Agency
Content Language	A language with a precisely defined syntax semantics and pragmatics, which is the basis of communication between independently designed and developed agents.	Agency
Communication Role	Agent role used in the definition of the agent interaction protocol (see FIPA specification).	Agency
Performative	The message's performative indicates the adopted FIPA Interaction Protocol.	Agency
Agent Interaction Protocol	It is a pattern specifying the sequence of message types within a communication. Usually message types are identified by the performative (or speech act) associated to the message.	Agency
Artifact	An element of the world that an agent may perceive or modify.	Solution
Task Action	It is an implementation operation to be used in the implementation task.	Solution
Agent Action	It is an implementation operation to be used in the implementation agent.	Solution
Platform Task	The code in the implementation platform of the agency task.	Solution
Platform Agent	The agent type defined in the chosen agent implementation platform.	Solution
Task Attribute	A specialization of an implementation attribute to be used inside an implementation task.	Solution
Agent Attribute	A specialization of an implementation attribute to be used inside an implementation agent.	Solution
Implementation Agent	It is the detailed design abstraction used for implementing the requirements analysis agent. It is a significant portion of the software and it usually includes some Implementation Tasks and Ontology Elements (representing the agent's knowledge). It may belong to social structures.	Solution
Implementation Task	It is the detailed design abstraction used for implementing the analysis-level task. It may include several atomic actions that can be composed in a plan.	Solution
Task Code	The code of the task.	Solution
Agent Code	The code of the agent.	Solution
Pattern	Reusable solution to agent design problems.	Solution
Node	A computational unit of the system where agent is deployed.	Solution
Network Connection	The type of connection between two nodes.	Solution
State	The state of the agent intended as the configuration of its knowledge, values of concepts, and so on.	Solution

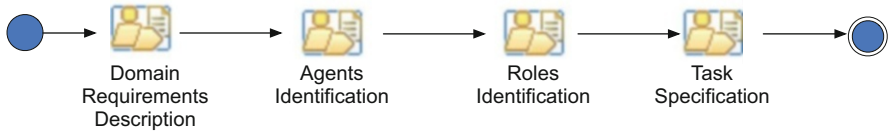


Fig. 7 The System Requirements phase flow of activities

2 Phases of the PASSI Process

2.1 The System Requirements Phase

The process flow at the level of activities is reported in Fig. 7. The process flow inside each activity will be detailed in the following sections (after the description of process roles). The System Requirements phase involves two different process roles and seven work products as described in Fig. 8. The phase is composed of four activities (i.e., Domain Requirements Description (DRD), Agents Identification, Roles Identification, and Task Specification), each of them composed of one or more tasks (for instance, Identify Use Cases and Refine Use Cases).

2.1.1 Process Roles

Two roles are involved in the System Requirements phase: the System analyst and the Domain expert. They are described in the following sections.

System Analyst

She or he is responsible for the following:

1. Use case identification during the DRD activity. Use cases are used to represent system requirements.
2. Use case refinement during the DRD activity. Use cases are refined with the help of a Domain Expert.
3. Use case clustering during the Agent Identification (AID) activity. The System Analyst analyzes the use case diagrams resulting from the previous phase and attempts to cluster them in a set of packages.
4. Naming agents during the AID activity. After grouping the use cases in a convenient set of packages, the last activity of this phase consists in designing these packages with the names that will distinguish the different agents throughout the project.
5. Roles identification during the Role Identification (RID) activity. The System Analyst studies (textual) scenarios and system requirements (as defined in the previous phase) and identifies the roles played by agents.
6. Designing scenarios during the RID activity. Each scenario is designed in the form of sequence diagrams, thus depicting the details of agent interactions.
7. Tasks identification during the Task Specification (TSP) activity. It consists in the identification of the behavioral capabilities that each agent needs to perform the specified roles and fulfill the requirements that are under its responsibility.

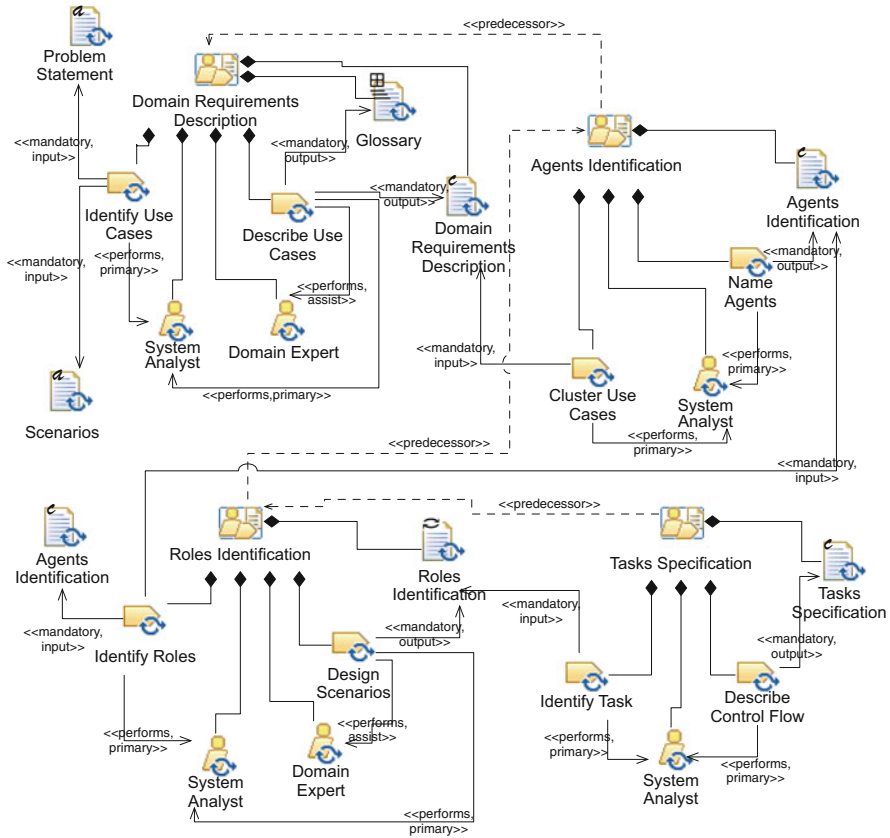


Fig. 8 The System Requirements phase described in terms of activities and work products

8. Description of the control flow during the TSP activity. It consists in introducing the communication relationships among tasks of different agents and the control flow among tasks of the same agent.

Domain Expert

She or he supports the system analyst during the description of the domain requirements.

2.1.2 Activity Details

Domain Requirements Description

The DRD aims at representing system functional and nonfunctional requirements. The flow of tasks inside this activity is reported in Fig. 9 and the tasks are detailed in the following table (Table 2).

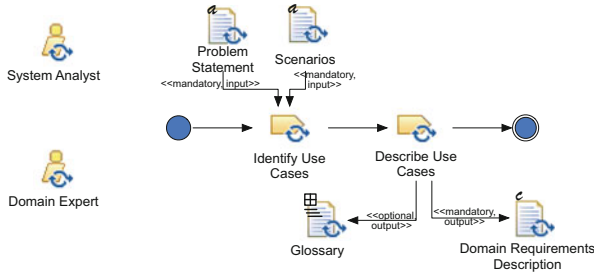


Fig. 9 The flow of tasks of the Domain Requirements Description (DRD) activity

Table 2 Domain Requirements Description—the task description

Activity	Task	Task Description	Roles Involved
Domain Requirements Description	Identify Use Cases	Use cases are used to represent system requirements.	System Analyst (perform)
Domain Requirements Description	Refine Use Cases	Use cases are refined with the help of a Domain Expert.	System Analyst (perform), Domain Expert (assist)

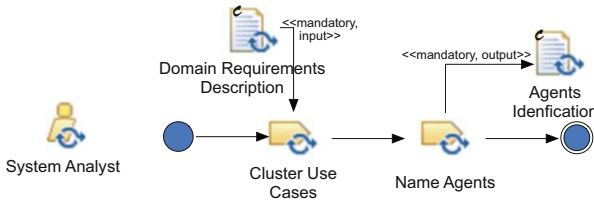


Fig. 10 The flow of tasks of the Agents Identification (AID) activity

Table 3 Agents Identification—the task description

Activity	Task	Task Description	Roles Involved
Agents Identification	Cluster Use Cases	The System Analyst analyzes the use case diagrams resulting from the previous phase and attempts their clustering in a set of packages.	System Analyst (perform)
Agents Identification	Name Agents	After grouping the use cases in a convenient set of packages, the last activity of this phase consists in identifying these packages with the names that will distinguish the different agents throughout the project.	System Analyst (perform)

Agents Identification

The Agents Identification activity aims at assigning system functionalities to the responsibility of newly defined agents.

The flow of tasks inside this activity is reported in Fig. 10 and the tasks are detailed in the table (Table 3).

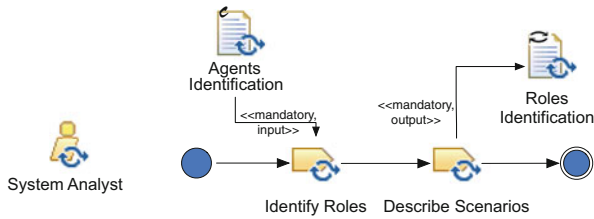


Fig. 11 The flow of tasks of the Roles Identification (RID) activity

Table 4 Roles Identification—the task description

Activity	Task	Task Description	Roles Involved
Roles Identification	Identify Roles	The System Analyst studies (textual) scenarios and system requirements (as defined in the previous phase) and identifies the roles played by agents.	System Analyst (perform)
Roles Identification	Design Scenarios	Each scenario is designed in the form of a sequence diagram thus depicting the details of agent interactions.	System Analyst (perform), Domain Expert (assist)

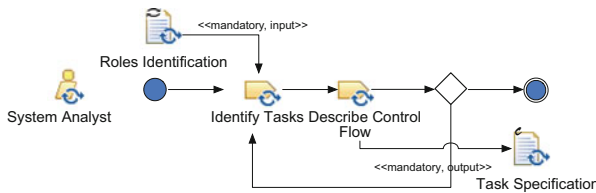


Fig. 12 The flow of tasks of the Task Specification (TSP) activity

Roles Identification

This activity aims at identifying the roles played by agents and their main interactions.

The flow of tasks inside this activity is reported in Fig. 11 and the tasks are detailed in the table (Table 4).

Task Specification

This activity aims at describing the behavior of each agent (agent’s plan) by considering its activities and communications.

The flow of tasks inside this activity is reported in Fig. 12 and the tasks are detailed in the following table (Table 5).

2.1.3 Work Products

The System Requirements Model generates four composed work products (documents including text and diagrams) and one structured text document. Their relationships with the MAS metamodel elements are described in Fig. 13. This figure represents the System Requirements model in terms of Work Products. Each

Table 5 Tasks Specification—the task description

Activity	Task	Task Description	Roles Involved
Tasks Specification	Identify Tasks	It consists in identifying the (agent) roles involved in fulfilling the requirements that are under the responsibility of each agent. It also includes the identification of the activities that each agent performs while playing a role.	System Analyst (perform)
Tasks Specification	Describe the control flow	It consists in introducing the communication relationships among tasks of different agents and the control flow among tasks of the same agent.	System Analyst (perform)

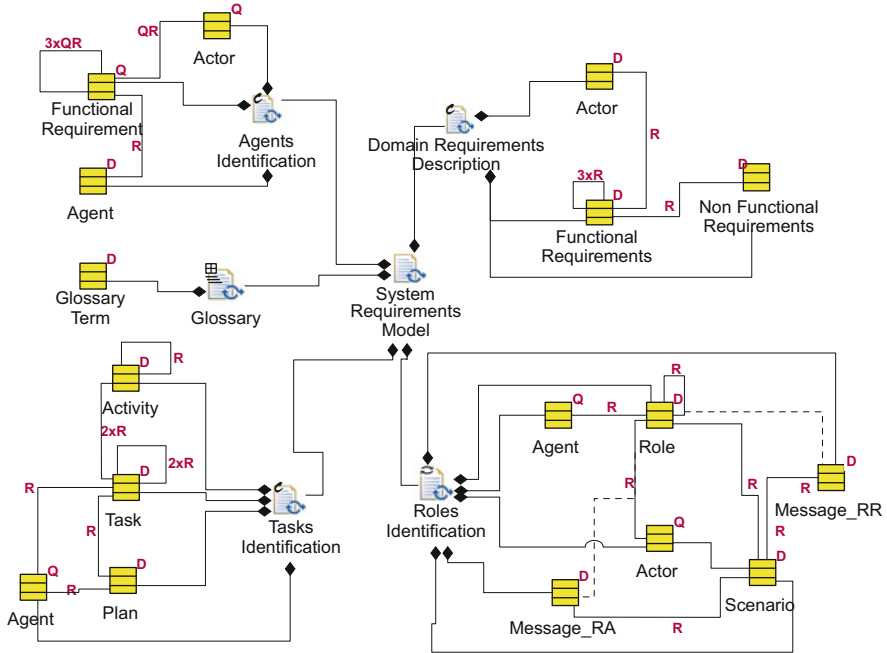


Fig. 13 The System Requirements Model documents structure

of them depicts one or more elements from the PASSI MAS metamodel; each MAS metamodel element is represented using a UML class icon (yellow filled) and, in the documents, such elements can be Defined **D**, Related **R**, Quoted **Q**, or Refined (a refining action corresponds to defining and quoting elements; see [1]).

Work Product Kinds

Table 6 describes the work products of the System Requirements phase according to their kinds.

Domain Requirements Description

This activity produces a composite document composed of use case diagrams and portions of (structured) text containing the complete documentation of use cases in

Table 6 System Requirements Phase—Work Product Kinds

Name	Description	Work Product Kind
Problem Statement	A description of the problem to be solved with the system. It is complemented by the Scenario document.	Free Text
Scenarios	Textual description of the scenarios in which the system to be developed is involved.	Free Text
Domain Description	Composed of the Domain Description diagram, a documentation of use cases reported in it, and the nonfunctional requirements of the system.	Composite (Structured + Behavioral)
Agent Identification	A document composed of (1) a use case diagram representing agents and the functionalities assigned to them and (2) a structured text description of the agents.	Composite (Structured + Behavioral)
Roles Identification	A document composed of several sequence diagrams (one for each scenario) and the roles description text.	Composite (Structured + Behavioral)
Task Specification	A document composed of several task specification diagrams (one for each agent) and a structured text description of each task.	Composite (Structured + Behavioral)
Glossary	A glossary of terms.	Structured Text

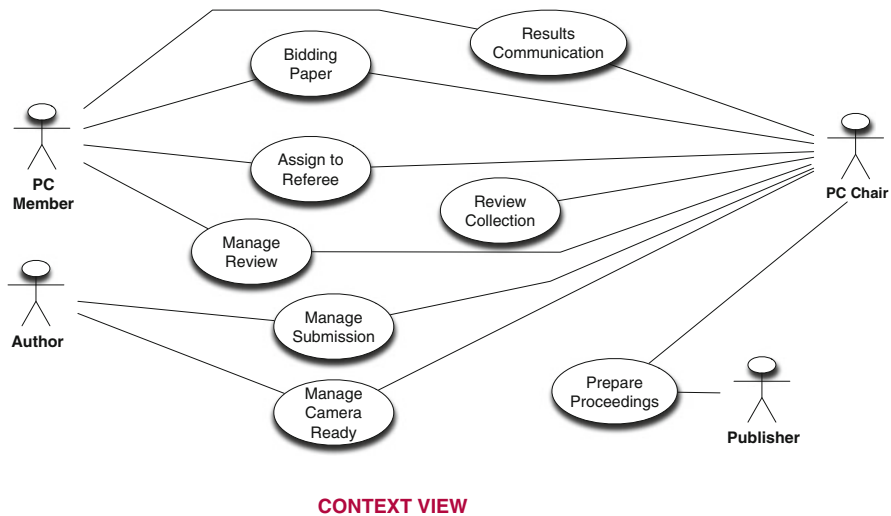


Fig. 14 Domain Requirements Description diagram—a portion of the CMS case study

terms of name, participating actors, entry condition, flow of events, exit condition, exceptions, and special requirements. It also reports the nonfunctional requirements identified for the system and associated to each use case. Common UML use case diagrams are used to represent the system requirements.

Figures 14 and 15 show an example of the notation of the DRD diagram for the CMS case study. Some guidelines for enacting this work product can be found in the *OpenUP website*, *Guideline: Identify and Outline Actors and Use Cases*, online at: http://epf.eclipse.org/wikis/openup/practice.tech.use_case_driven_dev.base/guidances/guidelines/identify_and_outline_actors_and_ucs_BB5516A9.html.

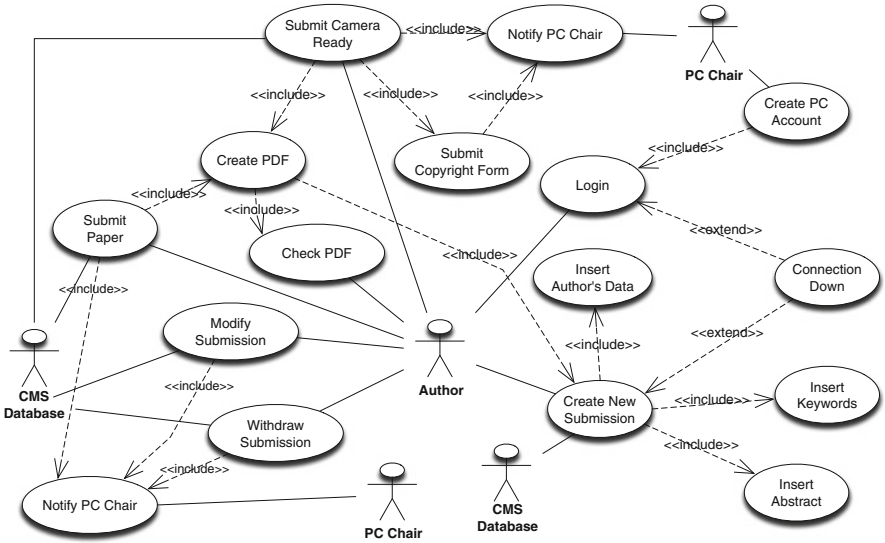


Fig. 15 Domain Requirements Description diagram—a portion of the CMS case study

Agents Identification

The resulting work product of this phase is a composite document including the following:

- A use case diagram (Agent Identification diagram) reporting use cases now clustered inside a set of packages, each one representing one agent.
- A table describing agents' main features (requirements, constraints, ...). As it is common, we represent external entities interacting with our system (people, devices, conventional software systems) as actors.

Relationships between use cases of the same agent follow the usual UML syntax and stereotypes, whereas relationships between use cases of different agents are stereotyped as communication (see below).

Figure 16 shows an example of the notation of the Agent Identification diagram for the CMS case study. In order to enact this diagram starting from a use case diagram, packages are used to group functionalities that will be assigned to an agent (whose name is the name of the package). Interactions between agents and external actors consist of communication acts; this implies that if some kind of include/extend relationship exists between two use cases belonging to different agents, this stereotype is to be changed to communication since a conversation is a unique way of interaction for agents. This is a necessary extension of the UML specifications that allow communication relationships only among use cases and actors. The direction of the relationships goes from the initiator of the conversation to the participant.

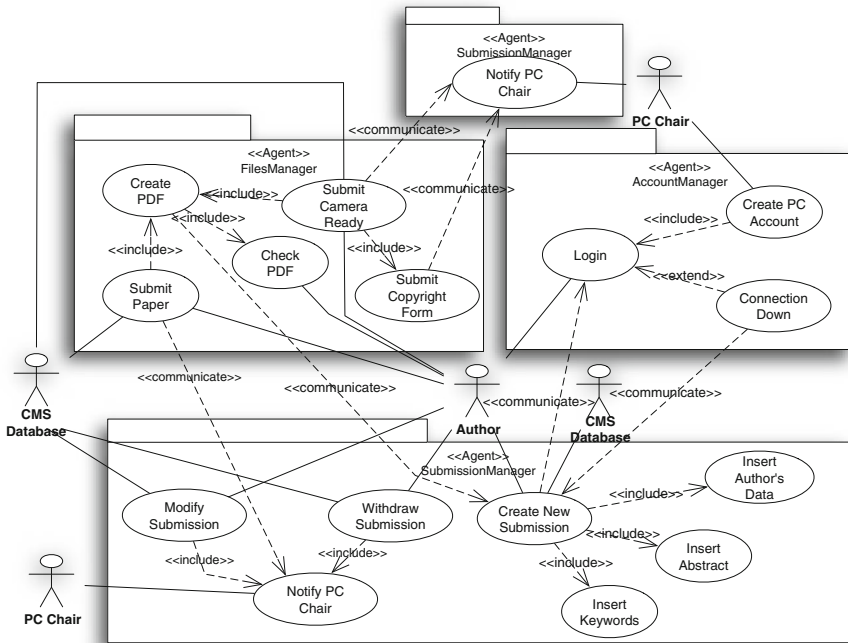


Fig. 16 Agents Identification diagram—a portion of the CMS case study

Roles Identification

This activity delivers one composite document. It is composed of one or more Roles Identification Diagram, a text description of the scenario reported in each diagram, and one or more tables describing role features. Usually diagrams are UML sequence diagrams reporting roles played by agents, actors, and the message they exchange.

Sequence diagrams describe all the possible communication paths between agents. A path describes a scenario of interacting agents, working to achieve a required behavior of the system. Each agent may belong to several scenarios, which are drawn by means of sequence diagrams in which objects are used to symbolize roles.

Figure 17 shows an example of the notation of the Roles Identification diagram for the CMS case study. Common UML sequence diagrams are used to identify roles. The name of each class is in the form: <role name>:<agent name> Role names are not underlined because they are not instances of agent classes but rather they represent a part of the agent behavior.

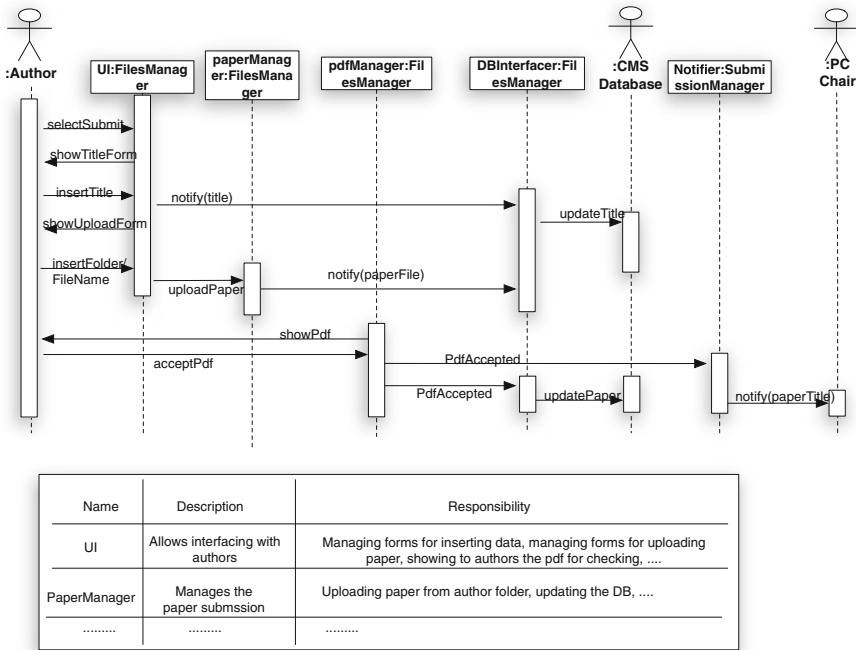


Fig. 17 Roles Identification diagram—a portion of the CMS case study

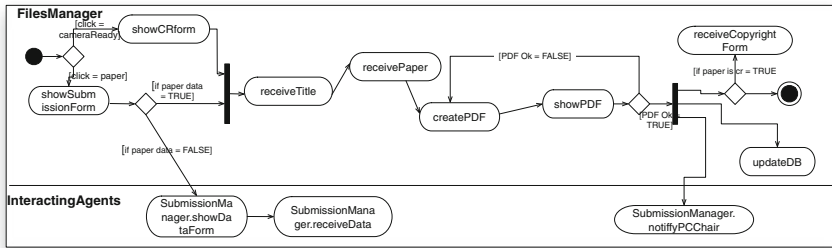
Task Specification

This activity delivers a Task Specification composite document. This is composed of the following:

- One or more Task Specification Diagrams, each diagram is used to specify the plan of a different agent. Usually these diagrams are UML activity diagrams reporting two swim lanes: the leftmost contains tasks belonging to external agents, the rightmost contains tasks belonging to the agent whose plan is defined in the diagram.
- A table describing each task in terms of list of activities, data, and internal plan.

A Task Specification diagram represents the plan of the agent behavior. It shows the relationships among the external stimuli received by the agent and its behavior (expressed in terms of fired tasks). Relationships between activities represent communications between tasks of different agents (these communications cross the border separating the two swim lanes) or invocation messages triggering task execution within a single agent (these messages are all located within the rightmost swim lane).

Figure 18 shows an example of the notation of the Task Specification diagram for the CMS case study. Common UML activity diagrams are used to specify an agent’s plan. In the PASSI methodology, tasks can be identified by looking at lifelines



Task: *showSubmissionForm*
 Description This task deals with managing paper submission form.
 Activities Shows form and check paper data
 Data authorName, authorAff, authorAddr, abstract, keyword
 Behavior manages the paper data check, the communication with the SubmissionManager agent if the author has not completed the insertion of his data and starts paper files receiving activities.
 Task: *receiveTitle*

Fig. 18 Task Specification diagram—a portion of the CMS case study



Fig. 19 The Agent Society phase flow of activities

of roles in PASSI RID sequence diagrams. Plans designed by using this activity are static, and therefore they are not suitable for agents exploiting dynamic plan composition capabilities. The Communicate relationship is identified by studying the Message_RR relationships of the RID (Roles Identification Document) work product.

2.2 The Agent Society Phase

The Agent Society phase provides a model of the social interactions and dependencies among the agents involved in the solution. The process flow at the level of activities is reported in Fig. 19. The process flow inside each activity will be detailed in the following sections (after the description of process roles). The Agent Society phase involves three different process roles and five work products as described in Fig. 20. The phase is composed of four activities (i.e., Domain Ontology Description, Communication Ontological Description (COD), Protocol Description, and Roles Description), each of them composed of one or more tasks (for instance, Ontology Revision or Performative Identification).

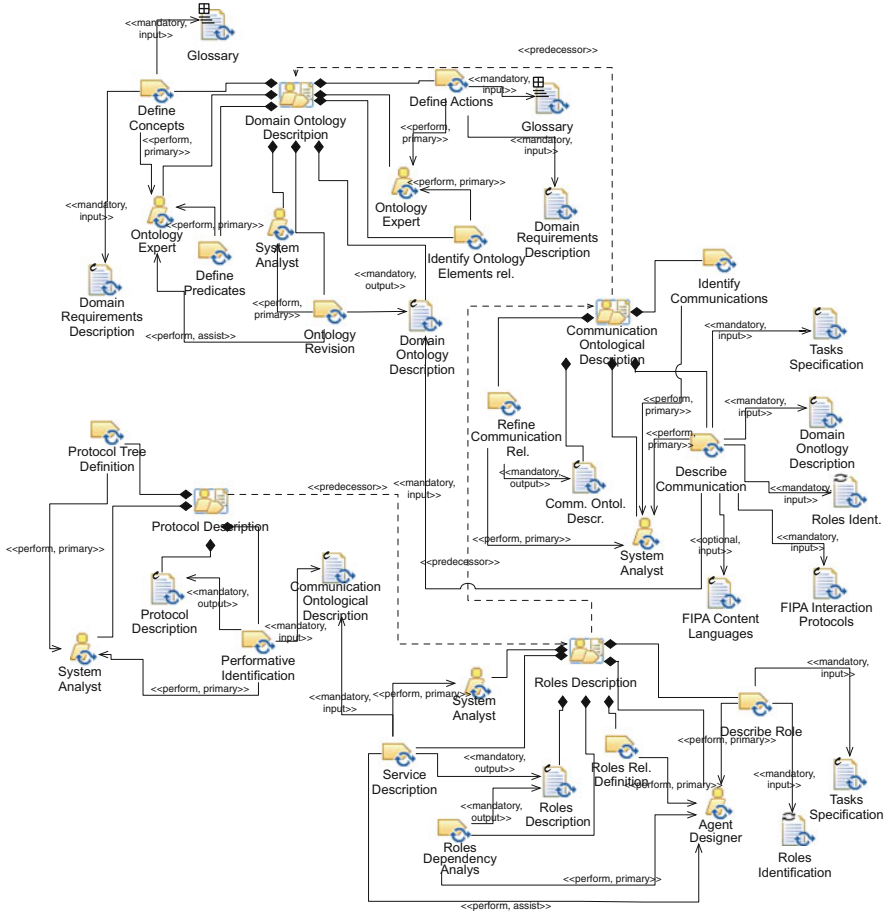


Fig. 20 The Agent Society phase described in terms of activities and work products

2.2.1 Process Roles

Three roles are involved in the Agent Society phase: the System Analyst, the Ontology Expert, and the Agent Designer. They are described in the following sections.

System Analyst

She or he is responsible for the following:

1. Ontology revision. The revision of ontological elements in order to define the pieces of knowledge of each agent and their communication ontology.
2. Communications identification. It consists in introducing an association for each communication between two agents, looking at exchanged messages in the scenario.

3. Communications definition. The description of agents' communication in terms of ontology, content language, and interaction protocol.
4. Communication relationships refinement. The identification of association classes in order to link each communication to the three fundamental elements of communication itself (ontology, language, and protocol).
5. Generalize the roles listed in the COD in order to identify the Communication_Roles.
6. Analyze scenarios in order to identify message types and performatives.

Ontology Expert

She or he is responsible for the following:

1. Concepts definition. It consists in the identification of concepts describing the system domain.
2. Predicates definition. The identification of predicates (assertions about concepts of the system domain).
3. Actions definition. The identification of activities an agent may perform.
4. Ontology relationships refinement. It consist in relating the previous three elements with ontological relationships.

Agent Designer

She or he is responsible for the following:

1. Roles description. It consists in the description of role classes arranged in packages, each package representing one agent, whose behavior is represented by tasks in the operation compartment.
2. Roles relationships definition. The definition of associations between roles (communications, dependencies, role changes). Communications derive from communication ontological description (COD), changes of role from RID, and dependencies from both COD and RID.
3. Collaboration rules definition. It consists in the introduction of agent society rules and the analysis of collaborations an agent needs. These depend on the society model the designer wants to achieve. These rules prevent some roles from providing services to other roles if they do not satisfy the required condition (actually reported in the services description document).
4. Knowledge description. It consists in listing agents, specifying their knowledge represented as attributes.

2.2.2 Activity Details

Domain Ontology Description

The Domain Ontology Description aims at describing the agent environment in terms of an ontology composed of concepts, predicates, and actions.

The flow of tasks inside this activity is reported in Fig. 21 and the tasks are detailed in Table 7.

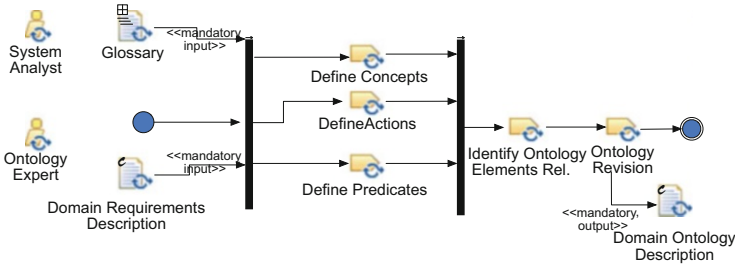


Fig. 21 The flow of tasks of the Domain Ontology Description (DOD) activity

Table 7 Domain Ontology Description—the task description

Activity	Task	Task Description	Roles Involved
Domain Ontology Description	Define Concepts	The identification of concepts describing the system domain.	Ontology Expert (perform)
Domain Ontology Description	Define Predicates	The identification of predicates and the assertions about concepts of the system domain.	Ontology Expert (perform)
Domain Ontology Description	Define Actions	The identification of the activities an agent may perform.	Ontology Expert (perform)
Domain Ontology Description	Identify Ont. Elem. Relationships	It consists in relating the previous three elements with ontological relationships.	Ontology Expert (perform)
Domain Ontology Description	Ontology Revision	The revision of ontological elements in order to define the pieces of knowledge of each agent and their communication ontology.	System Analyst (perform), Ontology Expert (assist)

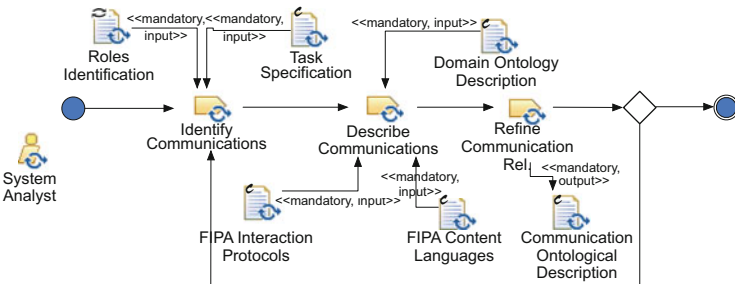


Fig. 22 The flow of tasks of the Communication Ontological Description (COD) activity

Communication Ontological Description

The COD aims at describing semantic agent communications in terms of exchanged knowledge (referred to as an ontology), content language, and interaction protocol. The flow of tasks inside this activity is reported in Fig. 22, and the tasks are detailed in Table 8.

Table 8 Communication Ontological Description—the task description

Activity	Task	Task Description	Roles Involved
Communication Ontological Description	Identify Communications	It consists in defining communications among agents, looking at exchanged messages in the scenario.	System Analyst (Performs)
Communication Ontological Description	Describe Communications	It consists in the description of agents' communications in terms of ontology, content language, and interaction protocol. Agents' knowledge structures necessary to deal with communication contents have to be introduced in the agents.	System Analyst (Performs)
Communication Ontological Description	Refine Communication Relationships	The identification of general communication association classes in order to enhance reuse and improve the architecture.	System Analyst (Performs)

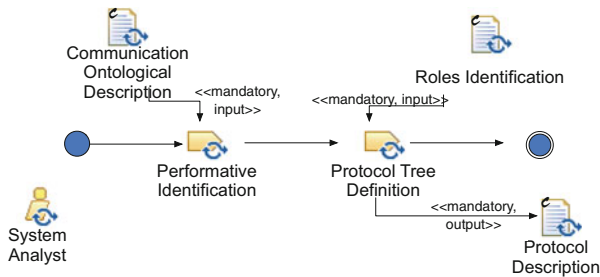


Fig. 23 The flow of tasks of the Protocol Description (PD) activity

Table 9 Protocol Description—the task description

Activity	Task	Task Description	Roles Involved
Protocol Description	Performative Identification	For each communication, the System Analyst identifies the available standard FIPA performatives.	System Analyst (Perform)
Protocol Description	Protocol Tree Definition	System Analyst designs some dedicated protocols by using the same FIPA documentation style.	System Analyst (Perform)

Protocol Description

The Protocol Description aims at designing new protocols for communication among agents when an existing one may not be reused due to FIPA specifications. The new protocol is documented by using the FIPA approach.

The flow of tasks inside this activity is reported in Fig. 23, and the tasks are detailed in Table 9.

Roles Description

The Roles Description aims at modeling the life cycle of each agent, looking at the roles it can play, at the collaboration it needs, and the communications in which it participates.

The flow of tasks inside this activity is reported in Fig. 24, and the tasks are detailed in Table 10.

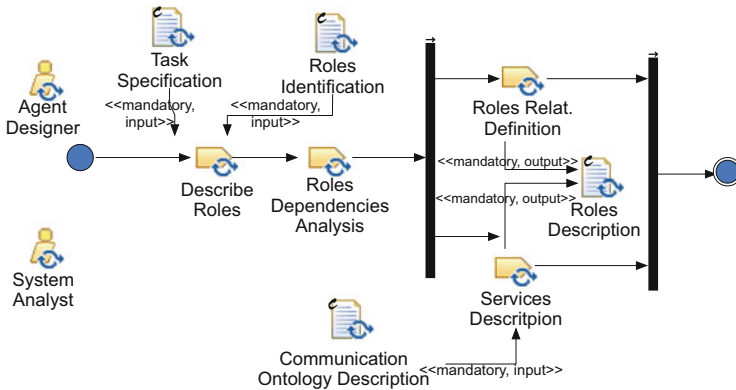


Fig. 24 The flow of tasks of the Role Description (RD) activity

Table 10 Roles Description—the task description

Activity	Task	Task Description	Roles Involved
Roles Description	Describe Roles	It consists in the description of role classes arranged in packages, each package representing one agent, whose behavior is represented by tasks in the operation compartment.	Agent Designer (perform)
Roles Description	Roles Dependencies Analysis	It consists in the introduction of agent society rules and the analysis of collaborations that an agent needs. These depend on the society model that the designer wants to introduce. These rules prevent some roles from providing services to other roles if they do not satisfy the required condition (actually reported in the services description document). Goals of roles are defined in this activity too and describe the objective of the specific role (when this is not involved in providing some service or sharing resources).	Agent Designer (perform)
Roles Description	Roles Relationships Definition	The definition of associations between roles (communications, dependencies, role changes). Communications derive from COD, changes of role from RID and dependencies from both COD and RID.	Agent Designer (perform)
Roles Description	Service Description	The description of service dependencies between two roles (service, resource, soft service, and soft resource). A resource dependency is seen as a resource providing service and is modeled in the services description document.	System Analyst (perform), Agent Designer (assist)

2.2.3 Work Products

The Agent Society phase results in five work products, four of them are composed and the last one is a free text. The relationships among the phase delivered work products and the MAS metamodel constructs are reported in Fig. 25.

Work Product Kinds

Table 11 describes the work products of the Agent Society phase according to their kinds.

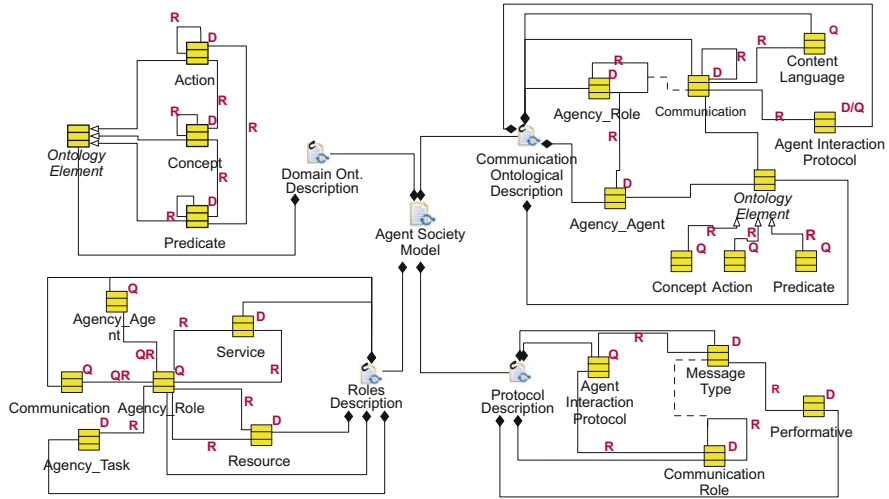


Fig. 25 The Agent Society phase documents structure

Table 11 Agent Society Phase—Work Product kind

Name	Description	Work Product Kind
Domain Ontology Description	A composite document composed of one class diagram (whose classes represent concepts, actions, and predicates) and a text document describing the <i>concepts</i> , <i>predicate</i> , and <i>action</i> elements.	Composite (Structured + Structural)
Communication Ontological Description	A composite document composed of one class diagram (whose classes represent agents and communications) and a text document describing the elements reported in the diagram.	Composite (Structural + Structured)
Roles Description	A composite document composed of one class diagram (whose classes represent agents and roles) and a text document describing the elements reported in the diagram.	Composite (Structured + Structural)
Protocol Description	A composite document composed of one sequence diagram and a text document describing the protocol.	Composite (Structured + Structural)
Services	A textual description of services.	Free Text

Domain Ontology Description

This activity produces a composite document composed of a class diagram (whose classes represent concepts, actions, and predicates) and a text document describing the elements reported in the diagram with the following details:

- Concepts are described in terms of their attributes
- The returned type is specified for predicates
- Actions have an Actor (that is responsible to do the job), a ResultReceiver (that is to be notified of the action results), and an Act that describes the action to be done with the required input and prescribed outcome

Information described in the class diagram is (optionally) completed by a text document, reporting the following data for each element (concept, action, predicate) of the ontology.

Figure 26 shows an example of the notation of the Domain Ontology Description diagram for the CMS case study.

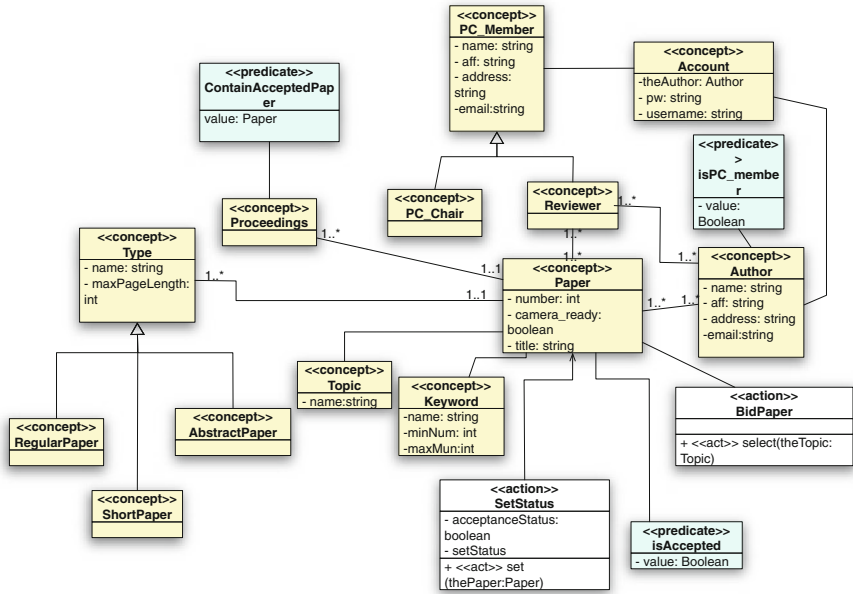


Fig. 26 Domain Ontology Description diagram—a portion of the CMS case study

Information described in the class diagram is (optionally) completed by a text document, reporting the detailed data, such as type and description, for each element (concept, action, and predicate) of the ontology.

Communication Ontological Description

This activity produces a composite document composed of a class diagram (whose classes represent agents and communications) and a text document describing the elements reported in the diagram. The COD diagram is a representation of agents' (social) interactions; this is a structural diagram (for instance, a class diagram) that shows all agents and all their interactions (lines connecting agents). According to FIPA standards, communications consist of speech acts [5] and are grouped by FIPA in several interaction protocols [2] that define the sequence of expected messages. As a consequence, each communication is characterized by three attributes, which we group into an association class. The attributes are: ontology (a piece of the ontology defined in the PASSI DOD activity), content language (see [3]) and interaction protocol. This is the characterization of the communication itself (a communication with different ontology, content language, or interaction protocol is certainly different from this one), and its name is used to uniquely refer to this communication (which can have, obviously, several instances at runtime since it may be enacted more than once).

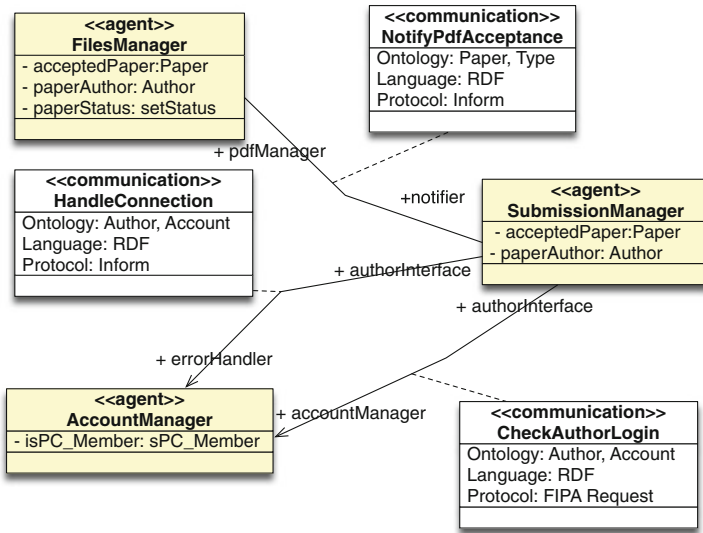


Fig. 27 Communication Ontological Description diagram—a portion of the CMS case study

Figure 27 shows an example of the notation of the COD diagram for the CMS case study; this portion of communication ontology only reports the agents shown in Fig. 16 and refers to the messages exchanged as seen in Fig. 17.

Roles Description

This activity produces a composite document composed of a class diagram where roles are classes grouped in packages representing the agents and a text document describing the elements reported in the diagram. Roles can be connected by relationships, representing changes of role, dependencies for a service, or the availability of a resource and communications. Each role is obtained composing several tasks, for this reason we specify the tasks involved in the role using the operation compartment of each class. Figure 28 shows an example of the notation of the Roles Description diagram for the CMS case study.

Roles come from RID diagrams or from COD diagram, their tasks (Agency_task) from the TSP diagrams of the agent, agents playing the designed roles come from RID diagrams, communications come from COD, dependencies come from the type of communications specified in the COD (a Request means a Service or SoftService dependency, a Query means a Resource or softResource dependency, . . .), role_change dependencies come from the analysis of the COD diagram against the RID when the same agent plays different roles in a sequence. Figure 28 is completed by a text document, listing the expected messages within each communication and detailing dependencies.

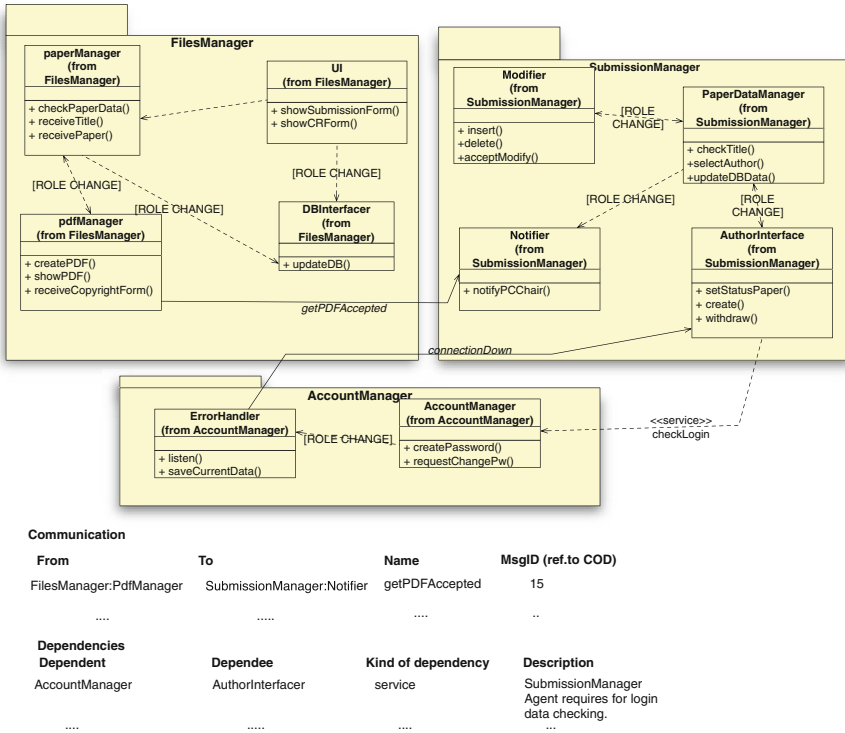


Fig. 28 Roles Description diagram—a portion of the CMS case study

Protocol Description

It is not necessary to design a new protocol if it is possible to reuse a standard FIPA protocol in the Role Description. If a new protocol has been chosen, this can be designed using the same approach of the FIPA documents (AUML sequence diagrams). Figure 29 shows an example of FIPA-Request protocol.

2.3 The Agent Implementation Phase

The Agent Implementation phase provides a model of the solution architecture in terms of classes and methods. The process flow at the level of activities is reported in Fig. 30. The process flow inside each activity will be detailed in the following sections (after the description of process roles). The Agent Implementation phase involves three different process roles and four work products (four UML models) as described in Fig. 31. The phase is composed of four activities (i.e., the Multiagent Structure Definition, the Multiagent Behavior Description, the Single-Agent Structure Definition, and the Single-Agent Behavior Description), each of them

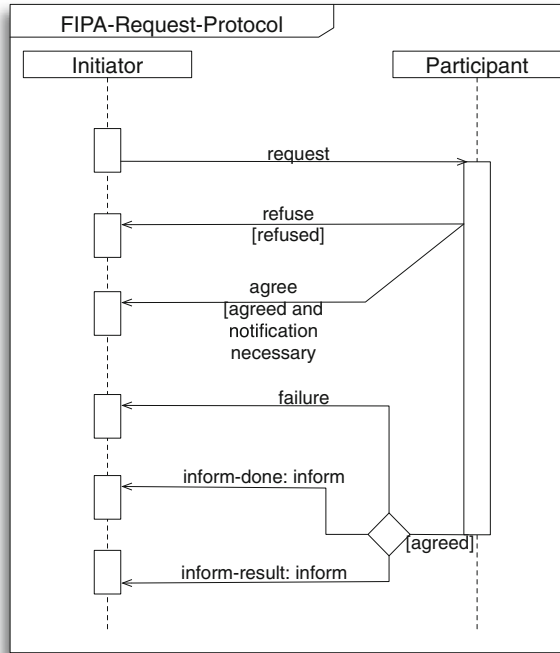


Fig. 29 Protocol Description diagram—a portion of the CMS case study

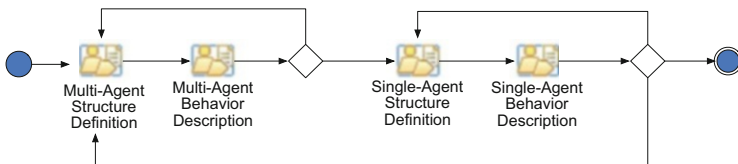


Fig. 30 The Agent Implementation phase flow of activities

composed of one or more tasks (for instance, List Agent Communication and Set Up Attributes and Methods).

2.3.1 Process Roles

Three roles are involved in the Agent Implementation phase: the System Analyst, the Ontology Expert, and the Agent Designer. They are described in the following sections.

System Analyst

She or he is responsible for

1. Listing agents in the implementation model
2. Identifying Artifacts and their interactions with agents

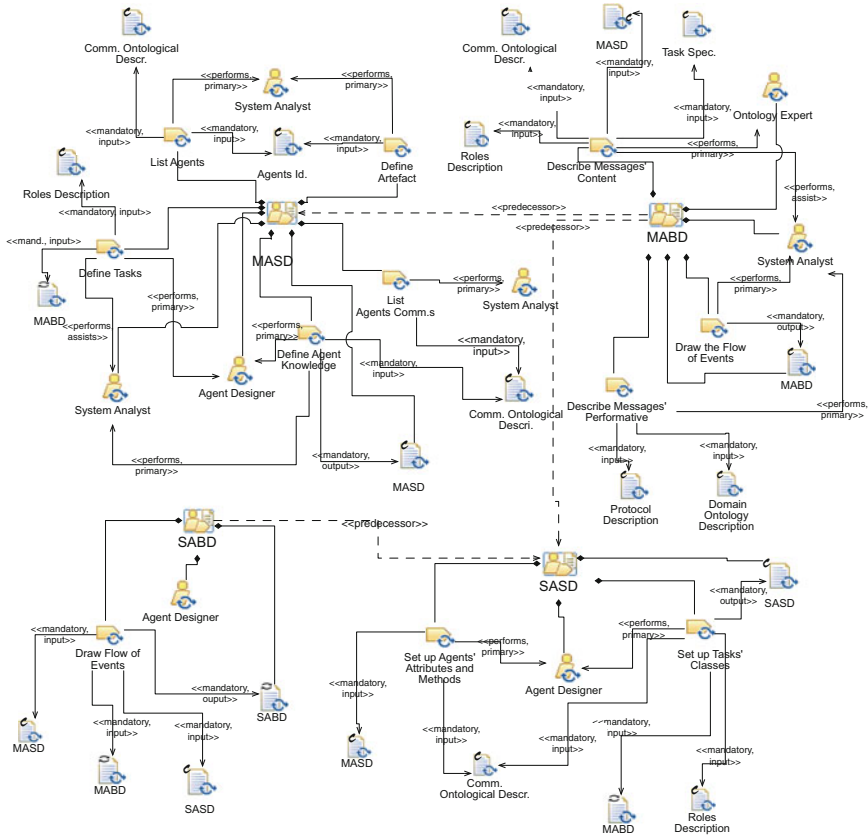


Fig. 31 The Agent Implementation phase described in terms of activities and work products

3. Identifying agents' communications
4. Assisting in defining tasks and knowledge of agents
5. Assisting in describing messages' contents
6. Drawing the flow of events

Ontology Expert

She or he is responsible for

1. Describing messages' contents
2. Describing messages' performatives

Agent Designer

She or he is responsible for

1. Analyzing the flow of event in order to describe the behavior of agents
2. Defining the tasks of each agent
3. Defining the knowledge each agent needs

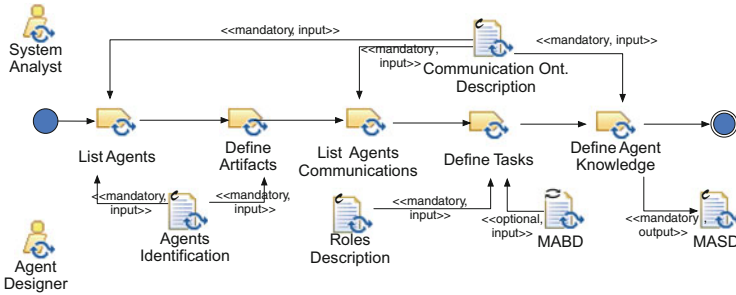


Fig. 32 The flow of tasks of the Multiagent Structure Definition (MASD) activity

Table 12 Multiagent Structure Definition—the task description

Activity	Task	Task Description	Roles Involved
Multiagent Structure Definition	List Agents	The System Analyst identifies and reports the agents that will constitute the implementation model.	System Analyst (perform)
Multiagent Structure Definition	Define Artifacts	The System Analyst defines the Artifacts that the agent interacts with.	System Analyst (perform)
Multiagent Structure Definition	List Agents' Communications	The System Analyst identifies and reports the agent communications that will be used in the implementation model.	System Analyst (perform)
Multiagent Structure Definition	Define Tasks	The Agent Designer defines the tasks of each single agent in the implementation model.	Agent Designer (perform) System Analyst (assist)
Multiagent Structure Definition	Define Agent Knowledge	The Agent Designer defines the knowledge each agent needs in order to accomplish its duties and to cope with the communications it is involved in.	Agent Designer (perform) System Analyst (assist)

4. Representing knowledge. The description of attributes for each class (agent) in order to represent a piece of knowledge on the domain
5. Describing method implementation
6. Setting up attributes, methods, and task classes

2.3.2 Activity Details
Multiagent Structure Definition

The Multiagent Structure Definition aims at describing the multi-agent system structure in terms of tasks, knowledge, and communications. Interactions with external entities (actors, artifacts) are described as well.

The flow of tasks inside this activity is reported in Fig. 32, and the tasks are detailed in Table 12.

Multiagent Behavior Description

The Multiagent Behavior Description (MABD) aims at describing the multiagent system behavior in terms of tasks, actions, the flow of control internal to each agent, and the messages exchanged among agents.

The flow of tasks inside this activity is reported in Fig. 33, and the tasks are detailed in Table 13.

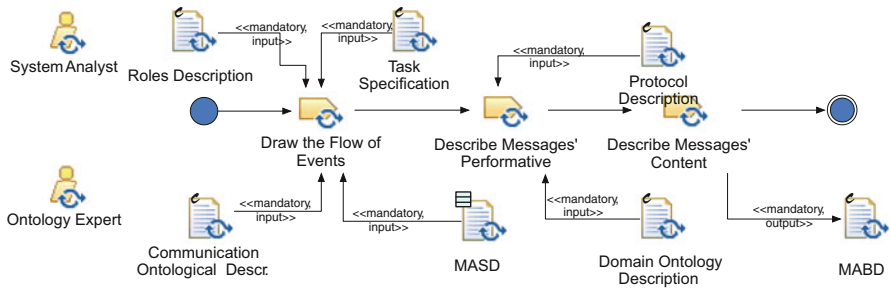


Fig. 33 The flow of tasks of the Multiagent Behavior Description (MABD) activity

Table 13 Multiagent Behavior Description—the task description

Activity	Task	Task Description	Roles Involved
Multiagent Behavior Description	Draw the Flow of Events	The System Analyst describes and completes the agents' behavior by analyzing the tasks related to communication among agents.	System Analyst (perform)
Multiagent Behavior Description	Describe Messages' Performative	The System Analyst assigns Performatives to messages.	System Analyst (perform)
Multiagent Behavior Description	Describe Messages' Content	The Ontology Expert assigns Ontology Elements to messages.	Ontology Expert (perform) System Analyst (assist)

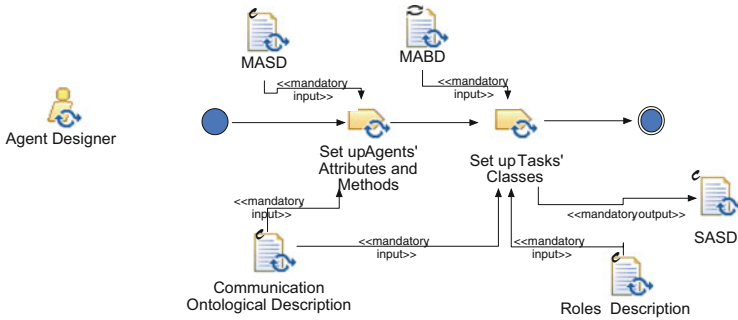


Fig. 34 The flow of tasks of the Single-Agent Structure Definition (SASD) activity

Single-Agent Structure Definition

The Single-Agent Structure Definition aims at defining the agent's interior structure through all of the classes making up the agent, which are the agent's main class and the inner classes identifying its tasks.

The flow of tasks inside this activity is reported in Fig. 34, and the tasks are detailed in Table 14.

Single-Agent Behavior Description

The Single-Agent Behavior Description aims at defining the behavior of an Implementation Agent class or an Implementation Task class.

Table 14 Single-Agent Structure Definition—the task description

Activity	Task	Task Description	Roles Involved
Single-Agent Structure Definition	Set up Attributes and Methods	The Agent Designer illustrates the agent’s interior structure through all of the classes making up the agent and sets up attributes and methods of the agent class.	Agent Designer (perform)
Single-Agent Structure Definition	Set up Task’s Classes	The Agent Designer sets up tasks’ classes (e.g., methods required to deal with communication events such as the handleRequest method of the IdleTask invoked when the agent gets a request communicative act).	Agent Designer (perform)

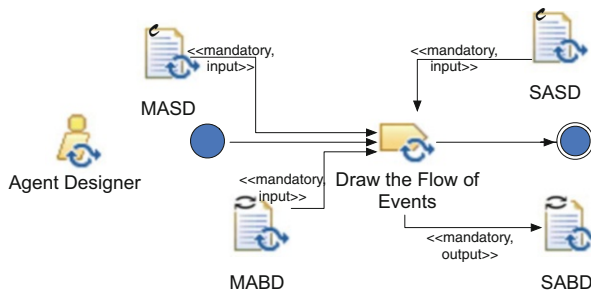


Fig. 35 The flow of tasks of the Single-Agent Behavior Description (SABD) activity

Table 15 Single-Agent Behavior Description—the task description

Activity	Task	Task Description	Roles Involved
Single-Agent Behavior Description	Draw the Flow of Events	The Agent Designer describes the method implementation in the way she or he considers appropriate, for instance, using flow charts, state diagrams, or semi-formal text descriptions.	Agent Designer (perform)

The flow of tasks inside this activity is reported in Fig. 35, and the tasks are detailed in Table 15.

2.3.3 Work Products

The Agent Implementation phase results in four composed work products, one of which is both behavioral and structural. The relationships among the phase-delivered work products and the MAS metamodel constructs are reported in Fig. 36.

Work Product Kinds

Table 16 describes the work products of the Agent Implementation phase according to their kinds.

Multiagent Structure Definition

This activity results in a composed work product where one class diagram represents the whole system with agents and actors involved. Each agent is represented by a class; its tasks are shown as operations and detailed only by name. An association can represent a communication or a dependency. The class diagram is completed

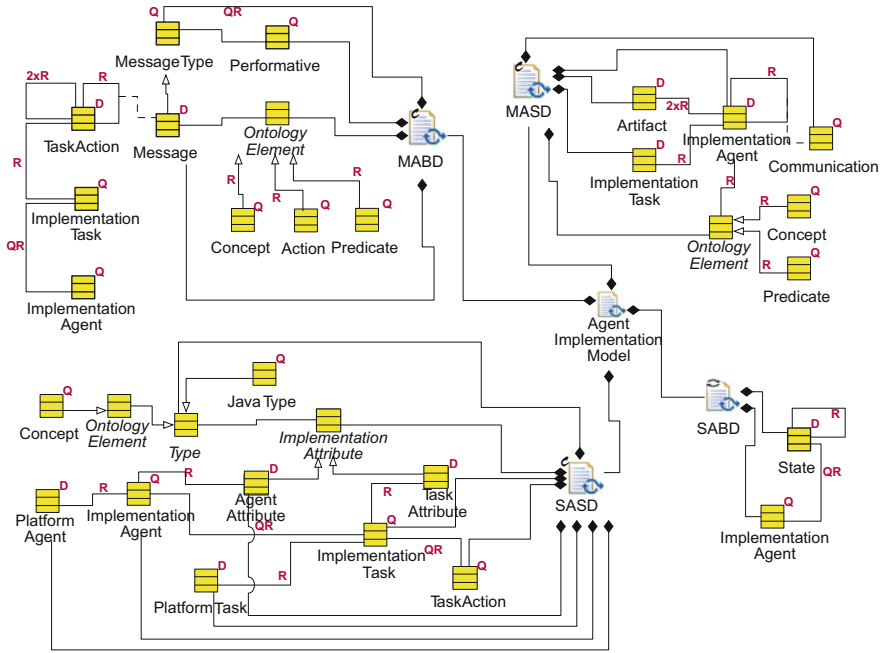


Fig. 36 The Agent Implementation phase documents structure

Table 16 Agent Implementation phase—Work Product kind

Name	Description	Work Product Kind
Multiagent Structure Definition	A composite document made of one class diagram (one class for each agent) and one structured text document (in the form of a table) describing the agent's tasks.	Composite (Structural + Structured)
Multiagent Behavior Description	A composite document made of one or more activity diagrams, showing the flow of events between and within both the main agent classes and their inner classes (representing their tasks) and one structured text document (in the form of a table) describing tasks, methods, and events.	Composite (Structural + Structured)
Single-Agent Structure Definition	A composite document made of several class diagrams, describing the internal structure (in terms of classes and methods) of agents involved in the process and one structured text document (in the form of a table) describing agents' tasks, methods, and parameters.	Composite (Structural + Structured)
Single-Agent Behavior Description	A behavioral diagram (activity diagram, sequence diagram, state chart,...) specifying the implementation of each method.	Behavioral

by a table where each task is described. Figure 37 shows an example of MASD for CMS case study.

Implementation Agents come from the transformation (1:1) of agents (Agency_Agent) reported in the COD activity; Implementation Tasks come from the

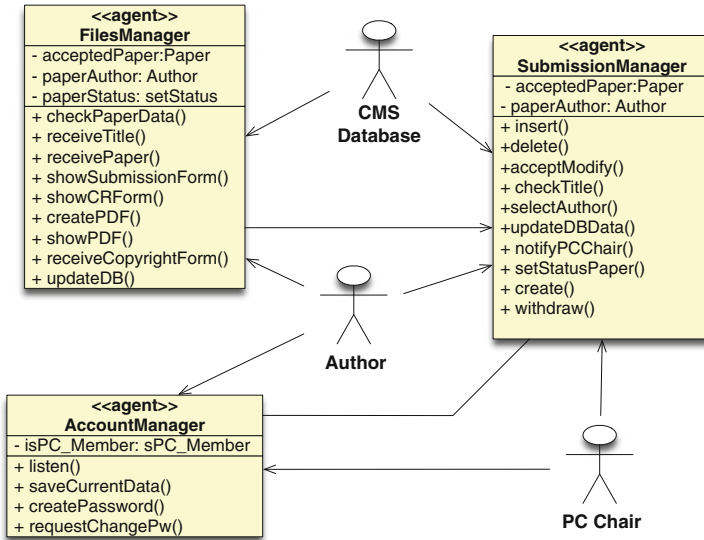


Fig. 37 Multiagent Structure Definition diagram—a portion of the CMS case study

transformation (1:1) of tasks (Agency_Tasks) reported in the RD activity. New Implementation Tasks are likely to be introduced as a consequence of the finer description of the agent behavior that is done in the loop between this activity and the MABD activity. Communications are reported from the COD activity, Artifacts of the real world that are perceived and effected by agents (Implementation Agents) are newly defined looking both at resources the agent controls and external elements the agent perceives (this also includes human interaction with the agent using a GUI). The consideration of external actors (reported in the Agent Identification work product) helps in identifying artifacts. Concept and Predicate are the knowledge that the agent needs to perform its duty. They are reported from ontology aggregated to communications in COD. Actions from COD are used to define new Implementation Tasks.

Multiagent Behavior Description

This activity delivers a composite work product where a sequence diagram shows the behavior of the multiagent system. Figure 38 shows an example of MABD for CMS case study. Implementation Agents and Implementation Tasks are reported from the MASD activity. TaskActions come from the study of actions related to communications in the COD activity as well as the Activities specified in the Task Specification. They are related as follows: *Invocation* and *Done* relationships among Task Actions come from the study of Activity Invocation relationships of the TSP or they are defined from scratch in order to complete the agent behavior, and *New Task* relationships come from the Task Invocation relationships of the TSP or they are

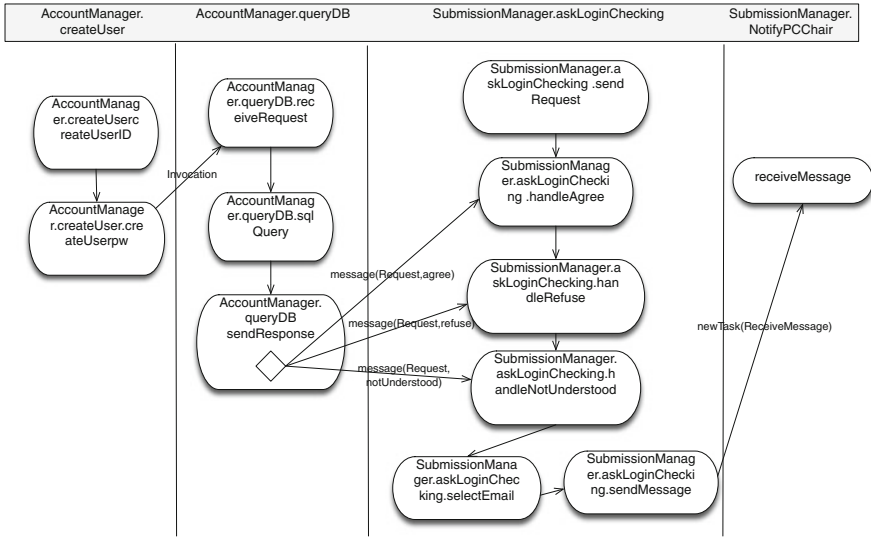


Fig. 38 Multiagent Behavior Description diagram—a portion of the CMS case study

defined from scratch in order to complete the agent behavior. Message relationships are deduced from the communicate relationships of the TSP. Ontology elements are assigned to messages according to what is prescribed by communications defined in the COD diagram. Performatives are assigned to messages according to the Interaction Protocols reported in the FIPA Agent Interaction Protocol specifications or the PD document.

Single-Agent Structure Definition

This activity is devoted to deliver a composite work product, several class diagrams (one for each agent), and the textual description of attributes and methods. Figure 39 shows an example of SASD for CMS case study.

Implementation Agent, Implementation Task, Invocation, and Task Action are defined by refining what is specified in the MASD–MABD activities with the addition of attributes and methods and by specifying the inheritance relationship with the implementation platforms’ main classes of agents/tasks.

Single-Agent Behavior Description

In this activity, Implementation Agent, Implementation Task, Invocation, and Task Action are studied from MABD (or even from SASD) in order to understand the behavior of the agent/task that should be designed. The new behavior is designed starting from this input in the form of an activity or state diagram. Figure 40 shows a very simple portion of SABD for CMS case study.

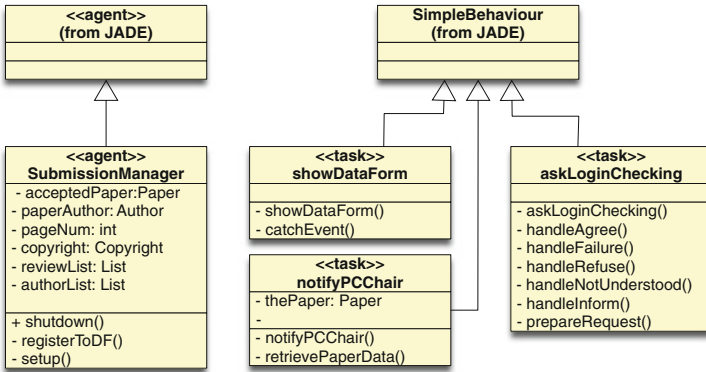


Fig. 39 Single-Agent Structure Definition diagram—a portion of the CMS case study

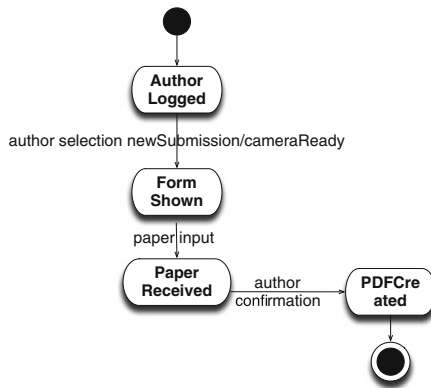


Fig. 40 Single-Agent Behavior Description diagram—a portion of the CMS case study

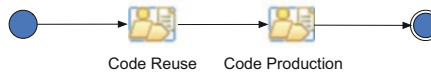


Fig. 41 The Code phase flow of activities

2.4 The Code Phase

The Code phase provides the model of the solution at the code level. The process flow at the level of activities is reported in Fig. 41. The process flow inside each activity will be detailed in the following sections (after the description of process roles). The Code phase involves two different process roles and two work products as described in the following Fig. 42. The phase is composed of two activities: the Code Reuse and Code Production, each of them composed of one or more tasks.

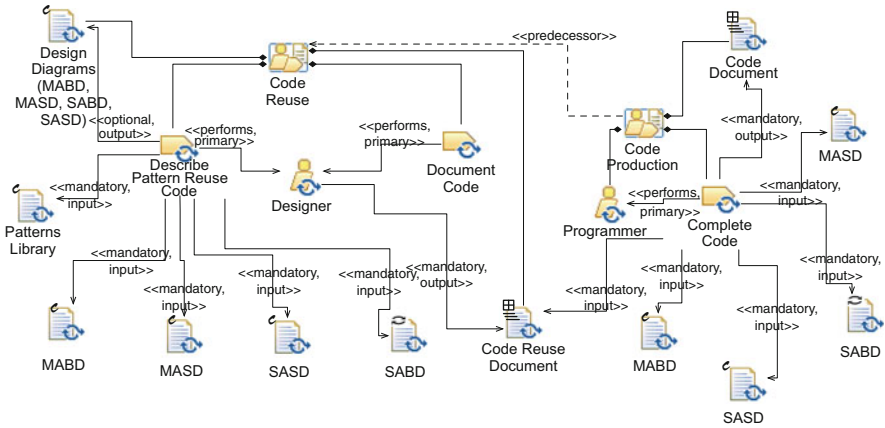


Fig. 42 The Code phase described in terms of activities and work products

2.4.1 Process Roles

Two roles are involved in the Code phase: the Agent Designer and the Programmer. They are described in the following sections.

Agent Designer

She or he is responsible for

1. Analyzing and reporting patterns from a library
2. Analyzing communication in order to select patterns

Programmer

She or he is responsible for

1. Writing the system code

2.4.2 Activity Details

Code Reuse

The Code Reuse aims at reusing existing patterns of agents and tasks. The flow of tasks inside this activity is reported in Fig. 43, and the tasks are detailed in Table 17.

Code Production

The Code Production aims at completing the code of the application. The flow of tasks inside this activity is reported in Fig. 44, and the tasks are detailed in Table 18.

2.4.3 Work Products

The Code phase results in two structured work products, one of which reports the code resulting from the reuse of patterns in the solution and the other is the final solution code. The relationships among the phase-delivered work products and the MAS metamodel constructs are reported in Fig. 45.

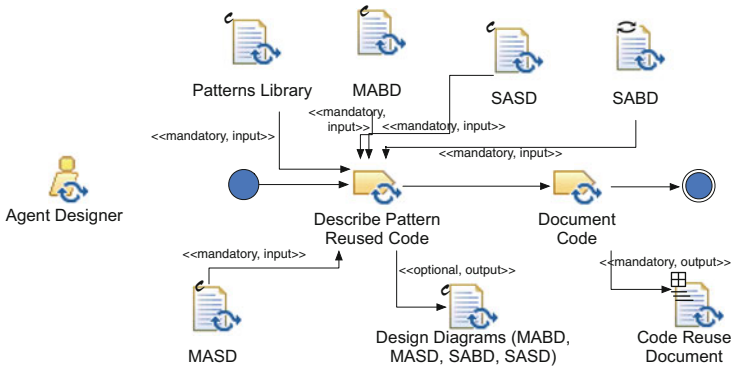


Fig. 43 The flow of tasks of the Code Reuse activity

Table 17 Code Reuse—the task description

Activity	Task	Task Description	Roles Involved
Code Reuse	Describe Pattern Reused Code	The Agent Designer iteratively selects elements to be implemented (an agent, a task, . . .), selects and applies patterns for his specific needs.	Agent Designer (perform)
Code Reuse	Document Code	The Agent Designer documents the code.	Agent Designer (perform)

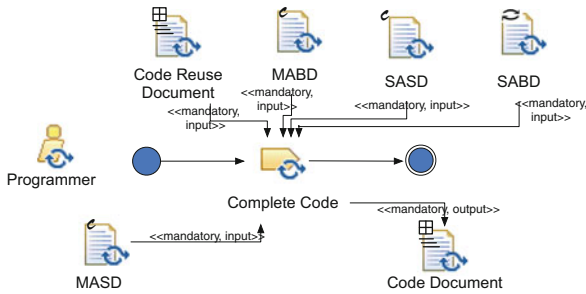


Fig. 44 The flow of tasks of the Code Production activity

Table 18 Code Production—the task description

Activity	Task	Task Description	Roles Involved
Code Production	Complete Code	The Programmer completes the code of the application starting from the design, the skeleton produced by the CASE tool, and reused patterns	Programmer (perform)

Work Product Kinds

Table 19 describes the work products of the Code phase according to their kinds.

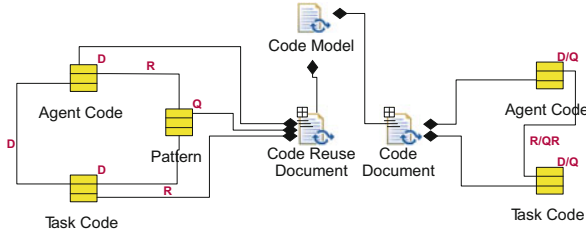


Fig. 45 The Code phase document structure

Table 19 Code phase—Work Product kind

Name	Description	Work Product Kind
Code Reuse Document	A document describing the code resulting from the adoption of patterns of agents and tasks in the solution.	Structured
Code Document	A document describing the code of the multi-agent system application.	Structured

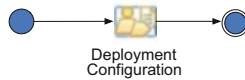


Fig. 46 The Deployment phase flow of activities

Code Reuse

This activity delivers the Code Reuse Document, reporting the code produced by the adoption of patterns of agents and tasks in the solution.

Code Production

This activity delivers the complete code of the application. This is rather a conventional activity, and for this reason, here, we do not show any example of the document.

2.5 The Deployment Phase

The Deployment phase provides details about the position of the agents in distributed systems or in mobile-agent contexts. The process flow at the level of activities is reported in Fig.46. The process flow inside each activity will be detailed in the following sections (after the description of process roles). The Deployment phase involves one process role and one work product as described in Fig.47. The phase is composed of one activity: the Deployment Configuration composed of one task (Describe Deployment Configuration).

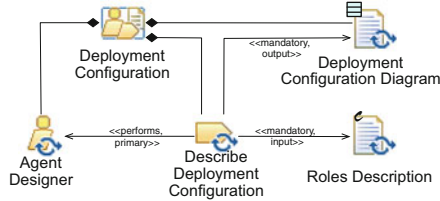


Fig. 47 The Deployment phase described in terms of activities and work products

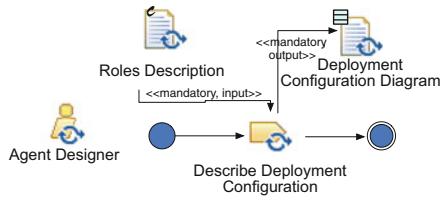


Fig. 48 The flow of tasks of the Deployment Configuration activity

Table 20 Deployment Configuration—the task description

Activity	Task	Task Description	Roles Involved
Deployment Configuration	Describe Deployment Configuration	The Agent Designer identifies the number of nodes by looking at the communications among Agency Agents	Agent Designer (perform)

2.5.1 Process Roles

One role is involved in the Deployment phase: the Agent Designer. It is described in the following sections.

Agent Designer

She or he is responsible for:

1. Identifying the number of nodes to be implemented
2. Specifying the communication channels to be made available among nodes hosting the different Agency Agents

2.5.2 Activity Details

Deployment Configuration

The Deployment Configuration aims at completing the code of the application. The flow of tasks inside this activity is reported in Fig. 48, and the tasks are detailed in Table 20.

2.5.3 Work Products

The Deployment phase results in one work product. The relationships among the phase-delivered work product and the MAS metamodel constructs are reported in Fig. 49.

Fig. 49 The Deployment phase document structure

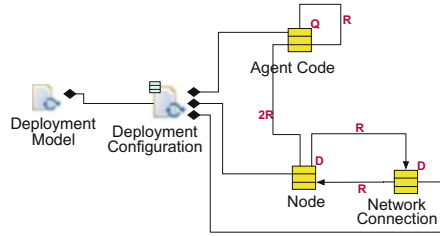
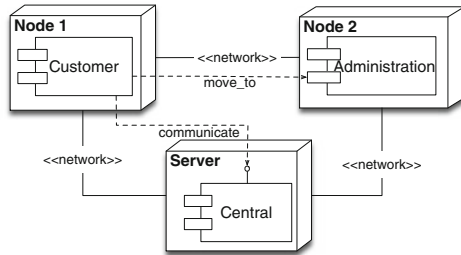


Table 21 Deployment phase—Work Product kind

Name	Description	Work Product Kind
Deployment Configuration	One UML diagram representing where the agents are located and which different elaborating units and communication channels they need	Structural

Fig. 50 Deployment Configuration Diagram—a portion of the CMS case study



Work Product Kinds

Table 21 describes the work products of the Deployment phase according to their kinds.

Deployment Configuration

This activity delivers the Deployment Configuration diagram that illustrates the location of the agents (the processing units where they live), their movements, and their communication support. The standard UML notation is useful for representing processing units (by boxes), agents (by components), and the like. What is not supported by UML is the representation of the agent’s mobility, which we have done by means of a syntax extension consisting of a dashed line with a *move_to* stereotype.

Figure 50 shows an example of notation for the Deployment Configuration Diagram. This diagram does not refer to the CMS case study since, due to the specificity of this kind of case study, we could not extract an example showing all the elements used in PASSI for the Deployment Diagram. The number of nodes to be implemented are deduced from the Agency Role by RD and by the available Agent Code that is here reported. Communication among nodes are deduced for Communication between Agency Agents, again from RD.

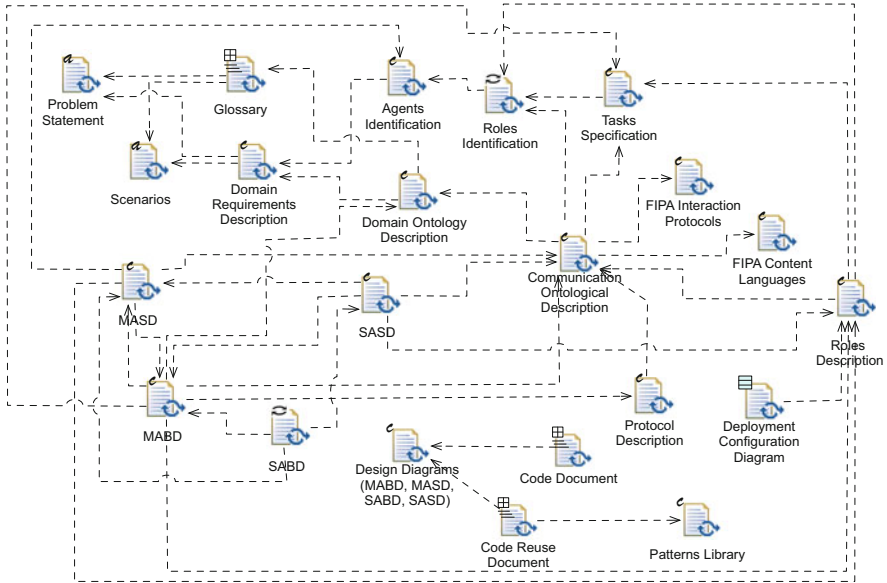


Fig. 51 Dependencies diagram

3 Work Product Dependencies

The diagram in Fig.51 describes the dependencies among the different work products.

References

1. Cossentino, M., Seidita, V.: Metamodeling: representing and modeling system knowledge in design processes. In: Proceedings of the Ninth European Workshop on Multi-Agent Systems (EUMAS'12), 17–19 December 2012
2. Foundation for Intelligent Physical Agents. FIPA Interaction Protocol Library Specification (2000)
3. Foundation for Intelligent Physical Agents. FIPA Content Languages Specification (2001)
4. Software process engineering metamodel. Version 2.0. Final Adopted Specification ptc/07 03-03. Object management group (omg), March 2007
5. Searle, J.R.: Speech Acts. Cambridge University Press, Cambridge (1969)
6. Seidita, V., Cossentino, M., Gaglio, S.: Using and extending the SPEM specifications to represent agent oriented methodologies. In: Luck, M., Gomez-Sanz, J.J. (eds.) Agent-Oriented Software Engineering IX. Lecture Notes in Computer Science, vol. 5386, pp. 46–59. Springer, Berlin (2008). doi:10.1007/978-3-642-01338-6_4

ROMAS Methodology

Emilia Garcia, Adriana Giret, and Vicente Botti

Abstract

This chapter presents the ROMAS methodology. ROMAS is an agent-oriented methodology that guides developers on the analysis and design of *regulated open multiagent systems*. These kinds of systems are composed of heterogeneous and autonomous agents and institutions, which may need to coexist in a complex social and legal framework that can evolve to address the different and often conflicting objectives of the many stakeholders involved. Contracts and norms are used to formalize the normative context and to establish restrictions on the entities' behaviors.

1 Introduction

ROMAS methodology defines a set of activities for the analysis and design of *regulated open multiagent systems*. The most relevant characteristics of systems of this kind are that they are *social*, *open*, and *regulated*. First, they are social in the sense that autonomous and heterogeneous entities interact between themselves to achieve global and individual objectives. Besides, the entities of the system can be structured as institutions or groups of agents with similar characteristics, functionality, or that interact with the rest of the system as a single entity. Second, they are open in the sense that, dynamically at runtime, external parties can interact and become part of the system. Third, they are regulated in the sense that every entity or institution in the system can have associated a set of norms that must fulfill. Besides, the expected behavior of each entity should be clearly specified by means of defining its rights and duties inside the system. In ROMAS, we consider the *normative context* of a system to be the set of norms that regulates the behavior of

E. Garcia • A. Giret • V. Botti (✉)
Universitat Politècnica de Valencia, Valencia, Spain
e-mail: mgarcia@dsic.upv.es; agiret@dsic.upv.es; vbotti@dsic.upv.es

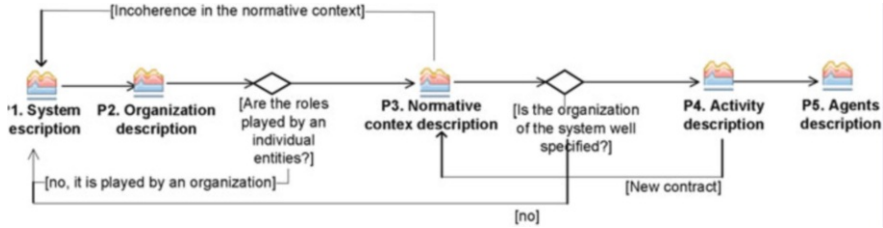


Fig. 1 ROMAS design process

each entity and the set of contracts that formalizes the relationships between these entities. The *normative context* of each entity is specified by the set of norms that directly affects the behavior of a particular entity. ROMAS catches the requirements of the design of systems from the global system's purposes to the specification of the behavior of each individual entity. The rest of the chapter presents the ROMAS metamodel and phases. Following some useful references about ROMAS:

- Emilia Garcia, A. Giret and V. Botti *Regulated Open multiagent Systems based on contracts* The 19th International Conference on Information Systems Development (ISD 2010) pp. 235–246. (2010)
- Emilia Garcia, A. Giret and V. Botti *A Model-Driven CASE tool for Developing and Verifying Regulated Open MAS* Journal Science of Computer Programming (2011)
- Emilia Garcia, G. Tyson, S. Miles, M. Luck, A. Taweel, T. Van Staa and B. Delaney *An Analysis of Agent-Oriented Engineering of e-Health Systems* 13th International Workshop on Agent-Oriented Software Engineering (AOSE–AAMAS) (2012)

1.1 The ROMAS Process Lifecycle

ROMAS methodology is composed of five phases that help developers to analyze and design the system from the highest level of abstraction to the definition of individual entities. As shown in Fig. 1, this is not a linear process but an iterative one. The identification of a new element of functionality during one phase may imply the revision of previous phases. For example, during the second phase, when a role that can be played by an organization as a whole is detected, it is necessary to go back to the first phase of the methodology to analyze the characteristics, global objectives, and structure of this organization.

1.2 ROMAS Metamodel

The analysis and design of these systems are formalized by means of several diagrams that are instances of the ROMAS metamodel. Table 1 describes the ROMAS metamodel elements. In order to facilitate the modeling tasks, this unified metamodel can be instantiated by means of four different views that are described below:

Organizational View

In this view, the global goals of organizations and the functionality that organizations provide and require from their environment are defined (Fig. 2). The static components of the organization, that is, all elements that are independent of the final executing entities, are defined too. More specifically, it defines the following:

- The entities of the system (*Executer*): *AAgents* and *Roles*. The classes *Executer* and *AAgents* are abstractions used to specify the metamodel, but neither of them are used by designers to model systems.
- An *Agent* is an abstract entity that represents an atomic entity (*Agent*) or a group of members of the organization (*Organizational Unit*), which is seen as a unique entity from outside.
- The *Organizational Units* (OUs) of the system can also include other units in a recursive way as well as single agents. The *Contains relationships* include conditions for enabling a dynamical registration/deregistration of the elements of an OU through its lifetime.
- The global *Objectives* of the main organization. The objectives defined in this view are nonfunctional requirements (soft goals) that are defined to describe the global behavior of the organization.
- The *Roles* defined inside the OUs. In the *contains* relationship, a minimum and maximum quantity of entities that can acquire a particular role can be specified. For each role, the *Accessibility* attribute indicates whether a role can be adopted by an entity on demand (external) or it is always predefined by design (internal). The *Visibility* attribute indicates whether entities can obtain information from this role on demand. This attribute can take the value “public” if anyone can obtain information of this role, and it takes the value “private” if only members of this organizational unit (i.e., private role). A hierarchy of roles can also be defined with the *InheritanceOf* relationship.
- The organization’s social relationships (*RelSocialRelationship*). The type of social relationship between two entities is related to their position in the structure of the organization (i.e., information, monitoring, supervision), but other types are also possible. Some social relationships can have a *ContractTemplate* associated with them, which can formalize some predefined commitments and rights that must be accepted or negotiated during the execution time. Each *Contract Template* is defined using the Contract Template view.
- The *Stakeholders* interact with the organization by means of publishing offers and demands of *Products* and *Services* on the *Bulletin Board*.
- The *Bulletin Board* can be considered as an information artifact for Open MAS. This artifact allows the designer to define the interaction with external entities and

Table 1 Definition of ROMAS metamodel elements

Concept	Definition	Metamodel views
Objective	An objective is a specific goal that agents or roles have to fulfill. It can be refined into other objectives.	Organizational, internal view
Organizational Unit (OU)	A set of agents that carry out some specific and differentiated activities or tasks by following a predefined pattern of cooperation and communication. An OU is formed by different entities throughout its lifecycle, which can be either single agents or other organizational units that are viewed as a single entity.	Organizational, internal, contract template
Role	An entity representing part of the functionality of the system. Any entity that plays a role within an organization acquires a set of rights and duties.	Organizational, internal, contract template, activity
Agent	An entity capable of perceiving and acting into an environment, communicating with other agents, providing and requesting services/resources, and playing several roles.	Organizational, internal, contract template, activity
Norm	A restriction on the behavior of one or more entities.	Organizational, internal, contract template, activity
Contract template	A set of predefined features and restrictions that all final contract of a specific type must fulfill. A contract represents a set of rights and duties that are accepted by the parties.	Organizational, internal, contract template, activity
Bulletin Board	A service publication point that offers the chance of registering and searching for services by their profile.	Organizational, internal, contract template, activity
Product	An application or a resource.	Organizational, internal, contract template, activity
Service Profile	The description of a service that the agent might offer to other entities.	Organizational, internal, activity
Service Implementation	A service-specific functionality that describes a concrete implementation of a service profile.	Internal, activity
Task	An entity that represents a basic functionality that consumes resources and produces changes in the agent's mental state.	Organizational, internal, contract template, activity
Stakeholder	A group that the organization is oriented toward and interacts with the OUs.	Organizational
Believe	A claim that an agent thinks that it is true.	Internal
Fact	A claim that is true at the system's domain.	Internal
Event	The result of an action that changes the state of the system when it occurs.	Internal
Interaction	An entity defining an interaction between agents.	Activity
Interaction Unit	A performative employed during the interaction.	Activity
Translation Condition	An artifact that allows to define the sequence of tasks depending on a condition.	Activity
Executer	A participant in an interaction. It can be an Organization, an Agent, or a Role.	Organizational, internal, contract template, activity

facilitates trading processes. When an agent wants to trade, they can consult or publish their offer on the Bulletin Board. Each offer or demand can be associated with a Contract Template. It means that this offer or demand has some predefined restrictions that are specified in this Contract Template view.

Internal View

This view allows defining the internal functionality, capabilities, belief, and objectives of each entity (organizations, agents, and roles) by means of different instances of this model (Fig. 3). More specifically, it defines the following features of each entity:

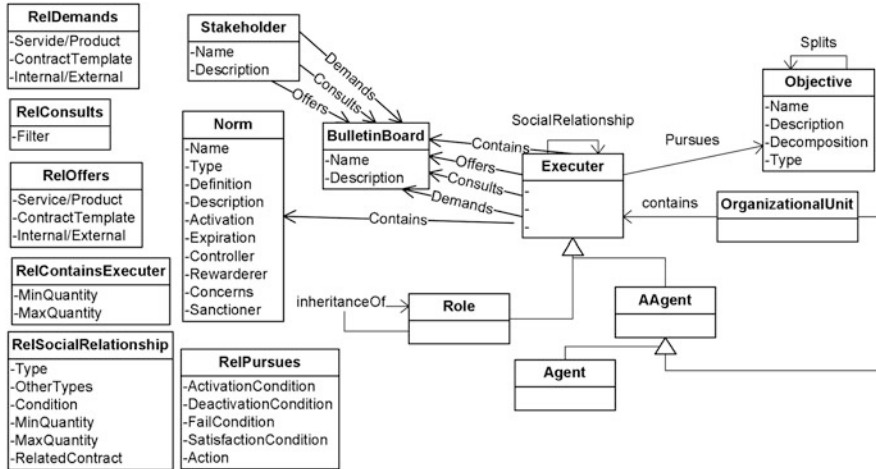


Fig. 2 Organizational view (the class *RelXXX* represents the attributes of the relationship *XXX*)

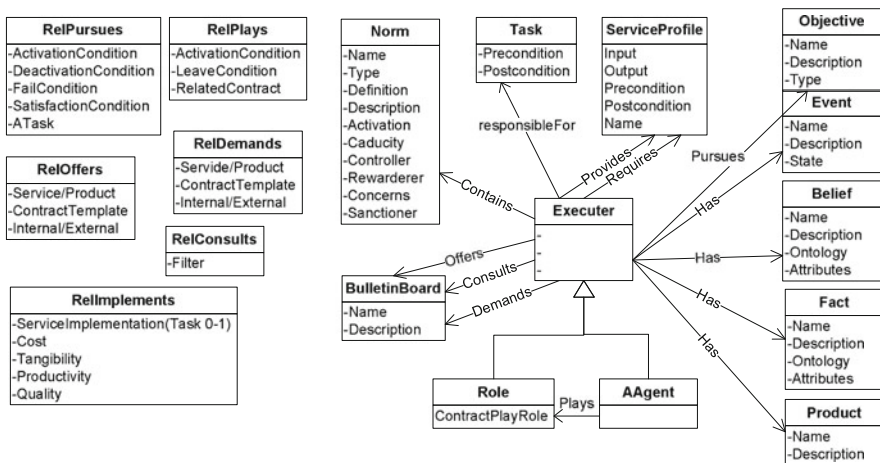


Fig. 3 Internal view (the class *RelXXX* represents the attributes of the relationship *XXX*)

- The *Objectives* represent the operational goals, that is, the specific goals that agents or roles have to fulfill. They can also be refined into more specific objectives. They might be related to a Task or Interaction that is needed for satisfying this objective.
- The *Mental States* of the agent, using belief, events, and facts.
- The *products* (resources/applications) available by an OU.
- The *tasks* that the agent is responsible for, that is, the set of tasks that the agent is capable of carrying out. A task is an entity that represents a basic functionality that consumes resources and produces changes in the agent's Mental State.

- the *Implements Service Profile*
- Internal entities can publish offers and demands in a *BulletinBoard*, as external stakeholders can do by means of the organizational view. This publication can also have an associated Contract Template to describe some predefined specifications.
- the *roles* that an agent or an organizational unit may play inside other organizational units (*Plays* relationship). *ActivationCondition* and *LeaveCondition* attributes of this relationship indicate in which situation an OU acquires or leaves a role.
- the *roles* played by each agent. *ActivationCondition* and *LeaveCondition* attributes of this *play* relationship indicate in which situation an agent can acquire or leave a role.
- the *Norms* specify restrictions on the behavior of the system entities. The relationship *Contains Norm* allows defining the rules of an organization and which norms are applied to each agent or role. *Norms* control the global behavior of the members of the OU.

Contract Template View

This view allows defining *Contract Templates*. Contracts are inherently defined at runtime. Despite this, designers represent some predefined restrictions that all final contracts of a specific type should follow by means of a *contract template*. Contract templates can be used at runtime as an initial point for the negotiation of contracts and to verify if the final contract is coherent with the legal context. The syntax of a contract template is defined in Fig. 4. More specifically, it defines the following:

- The relationship *Signants* indicates who is allowed to sign this type of contracts. It could be a specific agent, an agent who plays a specific role, or an organization. A *ThirdPart* could be anyone who participates in the negotiation protocol or who is affected by the final execution of the Contract.
- The relationship *Protocol* indicates which protocols are recommended to negotiate this type of contract.
- After the negotiation, the *Notary* is responsible for verifying the correctness and coherence of the final contract definition. He should check if any term of a contract violates any norm of the regulated environment.
- Each type of contract can define which *Receipts* will be generated during the execution time. Receipts are proof of facts; for example, a receipt can be generated when an agent successfully provides a service.
- In case of conflict, the *Judge* has to evaluate the Complaints and the generated Receipts following the *ConflictResolution* protocol. If he decides that there has been a violation of a norm, the *RegulationAuthority*, who is the main authority in the context of a contract, can punish or reward the agent behaviors.
- The relationship *Hard clause* indicates that any instance of this type of contract has to include this norm. *Soft clauses* are recommendations, so during the negotiation stage, *Signants* will decide whether this norm will be included or not in the final contract.

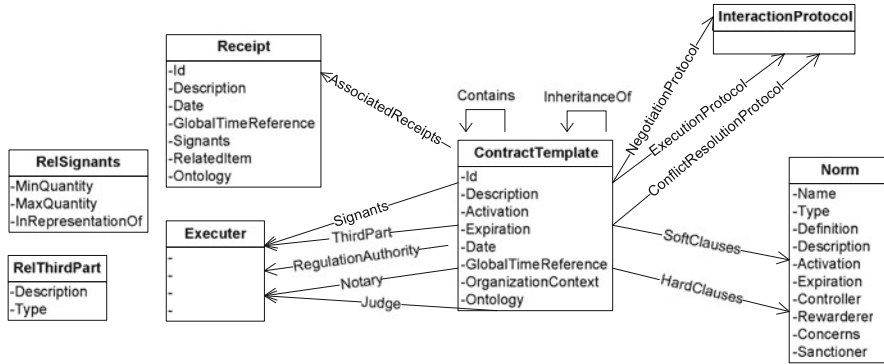


Fig. 4 Contract template view (the class *RelXXX* represents the attributes of the relationship *XXX*)

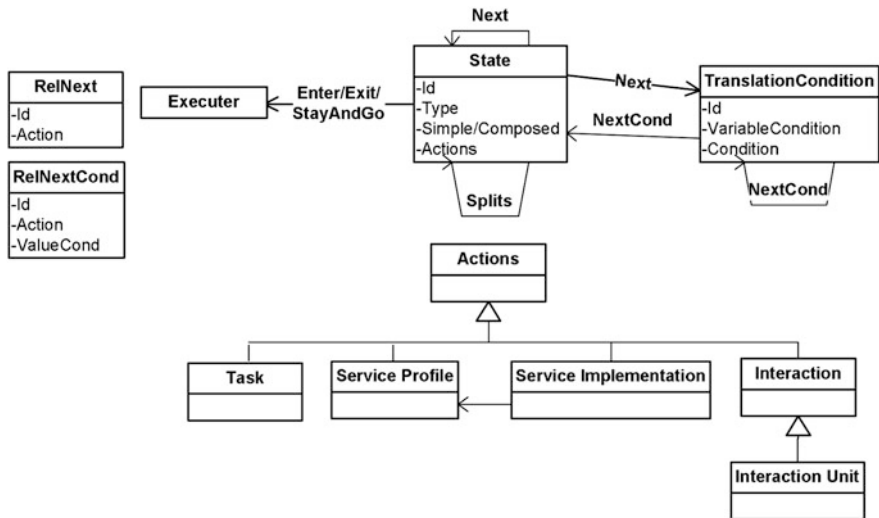


Fig. 5 Activity view (the class *RelXXX* represents the attributes of the relationship *XXX*)

Activity View

This view allows defining the sequence of actions in which a task, a service, or a protocol can be decomposed (Fig. 5). Each *state* represents an action or a set of actions that must be executed. An *action* is a first-order formula that indicates which task or service is executed or which message is interchanged between the agents that participate in this state. The relationship *next* indicates the sequence of states. These sequences can be affected by a *translation condition* that indicates under which circumstances the a state is going to be the next step of the process.

1.2.1 ROMAS Metamodel Notation

ROMAS diagrams use an UML-like graphical notation called GOPPR [6]. A caption to understand the graphical elements of the diagram is shown in Fig. 6. This notation

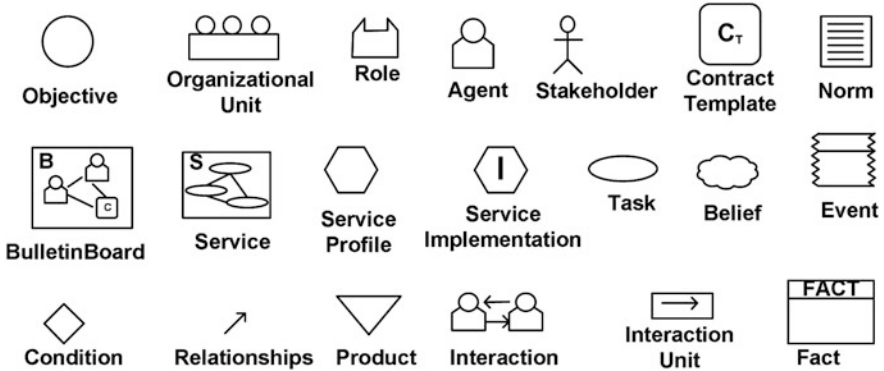


Fig. 6 Entities from the ROMAS graphical notation

is also used to define diagrams on INGENIAS, GORMAS, and ANEMONA methodologies. Some metamodel constructions proposed by ROMAS, such as *contract templates*, have been added to the notation.

2 Phases of the ROMAS Process

In this section, the phases that compose the ROMAS methodology are described. ROMAS offers a set of guidelines to analyze and formalize the requirements of the system, its functionality, the social relationship between the entities, and their normative context. These guidelines also guide designers to specify the interactions and service interchanges between the entities of an organization and between external entities.

2.1 PHASE 1: System Description

During this phase, the analysis of the system requirements, global goals of the system, and the identification of use cases are carried out. Besides, the global goals of the organization are refined into more specific goals, which represent both functional and nonfunctional requirements that should be achieved. Finally, the suitability of the ROMAS methodology for the specific system to develop is analyzed.

2.1.1 Process Roles

There are two roles involved in this phase: the system analyst and the domain expert. The *domain expert* is responsible for: (1) describing the system requirements, by means of identifying the system's main objectives, the stakeholders, the environment of the organization, and its restrictions; (2) supporting the system analyst in

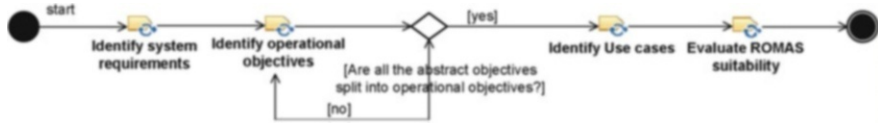


Fig. 7 Phase 1: activity tasks

Table 2 Phase 1: activity tasks

ID.task	Task	Description	Roles involved
1.1	Identify system requirements	Following the guideline <i>system description document</i> , the requirements of the system are analyzed, including global objectives of the system, stakeholders that interact with the system, products and services are offered and demands to/from stakeholders, external events that the system handles and normative documents such governmental laws attached to the system.	Domain expert
1.2	Identify Operational Objectives	Following the guideline <i>objective description document</i> , the global objectives of the system are analyzed and split into operational objectives, i.e., into more low level objectives that can be achieved by means of the execution of a task or a protocol.	System analyst and domain expert
1.3	Identify use cases	Using the information obtained in the previous task, the use cases of the system regarding the tasks and protocols associated to the operational objectives identified are defined.	System analyst and domain expert
1.4	Evaluate ROMAS suitability	Following the guideline <i>ROMAS suitability guideline</i> , the suitability of the ROMAS methodology for the development of the system to be developed regarding its specific features is evaluated.	System analyst

the analysis of the objectives of the system; and (3) supporting the system analyst in the description of the use cases of the system. The *system analyst* is responsible for: (1) analyzing the objectives of the system; (2) identifying the use cases; and (3) evaluating the suitability of the ROMAS methodology for the system to be developed regarding its requirements.

2.1.2 Activity Details

The flow of tasks of this phase is reported in Fig. 7, and each task is detailed in Table 2.

2.1.3 Work Products

The following section describes the products generated on the *System definition* phase and the guidelines used to define them: (1) *System definition document*; (2) *Objectives description document*, (3) *Use case diagram*, (4) *ROMAS suitability guideline*. Figure 8 shows graphically the products used and produced by each task. ***System Description Document***

This document is employed to identify the main features of the system and its relationship with the environment. It is a structured text document whose template is shown in Table 3. Table 4 presents the analysis of CMS case study using this document.

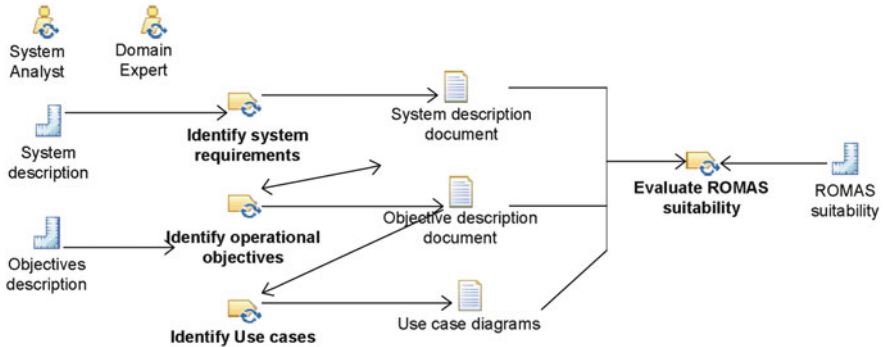


Fig. 8 Phase 1: resources and products used

Objectives Description Document and Objective Decomposition Diagram

This document analyzes the global objectives of the system and decomposes them into operational objectives. It is a structured text document whose template is shown in Table 5. Every global objective specified in the *system description document* is described using this document. The global objectives of the systems are refined into more specific ones that should also be described using this document. The document will be completed when all the global objectives are decomposed into operational objectives, that is they are associated to tasks, protocols, or restrictions that must be fulfilled in order to achieve these objectives. It is recommended to create one table for each global objective. The first column of each table will contain the property's name, the second the description of the global objective, and the following columns the descriptions of the objectives in which this global objective has been decomposed.

It is recommended to graphically represent the decomposition of the objectives by means of a diagram in order to provide a general overview of the purpose of the system that can be easily understood by domain experts. The graphical overview of the CMS case study objectives is shown in Fig. 9, where A means abstract objective and O means operational objective. As an example of the decomposition of a global objective into operational ones, Table 6 shows the decomposition of the global objective *Conference registration*.

Use Case Diagram

These diagrams are UML graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency. The actions identified in the analysis of the operational objectives are related, forming activity diagrams in order to clarify the sequence of actions that will be performed in the system. Figure 10 shows the sequence of actions that can be performed in the CMS case study.

ROMAS Suitability Guideline

After analyzing the requirements of the system, it is recommended to use this guideline in order to evaluate the suitability of the ROMAS methodology for the

Table 3 Template for system description document

Property	Description	Guideline	
System identifier	General name of the system to be developed.	It is recommended to select a short name or an abbreviation.	
System description	Informal description of the system.	There is no limitation on the length of this text. - What is the motivation for developing such a system? -Is there any system requirement that specify if the system must be centralized or decentralized? - Which is the main objective of this system?	
Domain	Domain or domains of application.	If this system must be able to be applied in different domains, it is recommended to add a text that explains each domain and whether it is necessary to adapt the system to each domain.	
Kind of environment	Identify and specify the kind of environment of the system.	- Can the functionality of the system be distributed between different entities? - Are the resources of the system distributed in different locations? - Are here external events that affect the internal state and behavior of the system? Is it a reactive system? - Is it a physical or a virtual environment? Is there any physical agent or robot that plays a role in the system? - Is there any human interaction with the system? - Should the results of the system be presented graphically? Is there any graphical environment?	
Global objectives	Functional and non-functional requirements (softgoals) that specify the desired-global behavior of the system.	- Which are the purposes of the system? - Which results should provide the system? - Should the system keep any parameter of the system between an specific threshold? (ex. the temperature of the room, the quantity of money in an account and so on)	
Stakeholders	Identifier	An identifier for the stakeholder.	Are there external entities or applications that are able to interact with the system?
	Description	Informal description of the stakeholder.	
	Type	Indicate if the stakeholder is a client, a provider or a regulator.	
	Contribution	To point out what the organization obtains from its relationship with the stakeholder.	
	Requires	A set of products and/or services that the stakeholder consumes.	
	Provides	A set of products and/or services that the stakeholder offers to the organization.	
	Frequency	To point out whether this stakeholder contacts with the organization frequently, occasionally or in an established period of time.	
Resources	Resources and applications available by the system.	- Is there any application or resource available by the system? - Is this resource physical or virtual?	
Events	External events that produce a system response.	Which events can produce an effect on the system? How the system capture these events and how response to them?	
Offers	A set of products or services offered by the organization to its clients.	Is there any product or service that the system should provide to an external or internal stakeholder?	
Demands	A set of products or services demanded by the organization to its clients.	Are there any requirements that the system cannot provide itself? Is it important who provide this service or product?	

(continued)

Table 3 (continued)

Property	Description	Guideline
Restrictions	An overview about which types of restrictions the system should imposed on its entities.	<ul style="list-style-type: none"> - Behavioral restrictions: Is there any system requirement that specify limits on the behavior of the members of the system? - Critical restrictions: Is there any action whose inadequate usage could be dangerous for the system? - Usage restrictions: Is there any restriction on the usage of the system resources? Is there any restriction on the usage of the services and products offered by the system? Is there any restriction on who is an appropriate stakeholder to provide a service or product to the system? - Legal restrictions: Is there any normative document, such as governmental law or institutional internal regulations, that affects the system's entities behavior?

Table 4 Phase 1—Case study: system description document

Case study: <i>System description document</i>	
System identifier	CMS (Conference Management System)
System description	This system should support the management of scientific conferences. This system involves several aspects from the main organization issues to paper submission and peer review, which are typically performed by a number of people distributed all over the world.
Domain	Research
Kind of environment	Virtual and distributed environment with established policies and norms that should be followed.
Global objectives	<ul style="list-style-type: none"> - Management of user registration - Management of conference registration - Management of the submission process - Management of the review process - Management of the publication process
Stakeholders	There is no external entity that interact with the system. Every entity that wants to interact with the system should be registered and logged in the system.
Resources	Database: it should include personal information and affiliation and information about which users are registered as authors, reviewers or publishers for each conference. Also it should include information about each conference, i.e., its status, its submitted papers and reviews,...
Events	Non external events are handled by the system.
Offers	<ul style="list-style-type: none"> - NewUsers_registration(); - Log_in();
Demands	
Restrictions	<ul style="list-style-type: none"> - The system should follow the legal documentation about the storage of personal data. - Each conference should describe its internal normative.

development of the analyzed system. Table 7 shows the criteria used to evaluate whether ROMAS is suitable.

ROMAS is focused on the development of regulated multiagent systems based on contracts. ROMAS is appropriate for the development of distributed systems, with autonomous entities, with a social structure, with the need of interoperability standards, regulation, and trustworthiness between entities and organizations. ROMAS is not suitable for the development of centralized systems or non-multiagent systems. Although non-normative systems could be analyzed using ROMAS, it is not recommended.

The analysis of the CMS case study features following this guideline shows that ROMAS is suitable for the development of this system. It is a distributed system, composed of intelligent systems with social relationships between them. The entities of the system should behave following the regulations of the system.

Table 5 Phase 1: Objectives description

Property	Description	Guideline	
Identifier	Objective name identifier.	It is recommended to select a short name or an abbreviation.	
Description	Informal description of the objective that is pursued.	There is no length limitation on this text. It should clearly describe this objective.	
Activation Condition	First order formula that indicates under which circumstances this objective begins being pursued.	<ul style="list-style-type: none"> - Does the organization pursue this objective from the initialization of the system? - Is there any situation that activates this objective? Common circumstances that can activate objectives are: when an event is captured, the failure of other objective, the violation of a restriction, when an agent plays a specific role, and so on. - If this objective is deactivated, is there any situation that forces the objective to be pursued again? 	
Deactivation Condition	First order formula that indicates under which circumstances this objective stops being pursued.	<ul style="list-style-type: none"> - Is this objective pursued during the whole lifecycle of the system? - Is there any situation that deactivates this objective? Common circumstances that deactivate an objective are: when it is satisfied, when other objective is satisfied, when some restriction has been violated, and so on. 	
Satisfaction Condition	First order formula that indicates in which situation this objective is satisfied.	<ul style="list-style-type: none"> - Is the satisfaction of this objective measurable? - What results should be produced to claim that this objective has been satisfied? 	
Fail Condition	First order formula that indicates in which situation this objective has failed.	<ul style="list-style-type: none"> - Is there any situation that is contrary to this objective and that will invalidate it? - Is there any threshold that should not be exceeded? 	
Type	Objectives can be abstract or operational. An <i>abstract objective</i> is a non-functional requirement that could be defined to describe the global behavior of the organization. An <i>operational objective</i> is a specific goal that agents or roles have to fulfill.	If there is a task that can be executed in order to satisfy this objective, it is an operational objective, in other cases it is an abstract objective. Abstract objectives can be refined into other abstracts or operational objectives.	
Decomposition	First order formula that specified how this objective is decomposed.	If this is an abstract objective it should be decomposed in several operational objectives which indicates which tasks should be executed in order to achieve this objective. Operational objectives can also be decomposed in order to obtain different subobjectives that can be pursued by different members of the organization. This fact simplifies the programming task and facilitates the distribution of responsibilities.	
Related Action/Restriction	Objectives can be related to a restriction on the behavior of the system, or to an action that must be executed in order to achieve this objective. Actions can be tasks, services or protocols.	The difference between a task and a protocol is that a task can be executed by one single agent, however a protocol is a set of tasks and interactions between two or more agents. Services are pieces of functionality that an entity of the system offers to the others, so the main difference between services and tasks or protocols is that they are executed when an entity request this functionality. At this phase it is not necessary to detail all the parameters of the task. You should describe in a high abstraction level what actions and activities are necessary to achieve this objective.	
	Type		Task, service or protocol.
	Identifier		An identifier for the task, service or protocol.
	Description		Informal description of the action.
	Resources		Which applications or products are necessary to execute this task (for example, access to a database). This feature can be known at this analysis phase due to requirement specifications.

(continued)

Table 5 (continued)

Property	Description	Guideline
		However, if there is no specification, the specific implementation of each task should be defined in following steps of the methodology.
Activation condition	First order formula that indicates under which circumstances this action will be activate.	
Inputs	Information that must be supplied.	
Precondition	A set of the input conditions and environment values that must occur before executing the action in order to perform a correct execution.	
Outputs	Information returned by this action and tangible results obtained.	
Postconditions	Final states of the parameters of the environment, by means of the different kinds of outputs.	

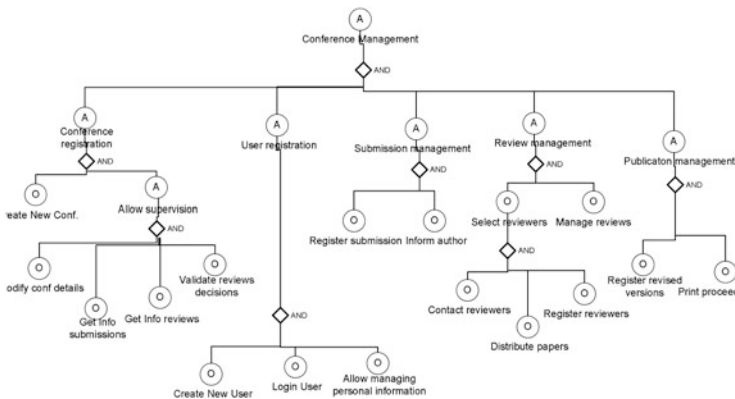


Fig. 9 Case study: objective decomposition diagram

The rights and duties that an entity acquires when it participates in the system should be formalized. For example, reviewers should know, before they acquire the commitment of reviewing a paper, when its revision must be provided. Therefore, a contract-based approach is recommendable.

Table 6 Phase 1—Case study: objective description document

<i>Case study: Conference registration objective decomposition</i>						
Identifier	Create new conference	Allow supervision	Modify conf details	Get Info submissions	Get Info reviews	Validate reviews decision
Description	The system should allow the registration of new conferences. The entity who registers the conference should be the chair of it.	The authorized entities should be able to supervise the status of the conference and modify its details.	The description details such as deadlines, topics and general description can be modified by the chair or vice-chair of the conference.	The system should provide information about the submitted papers.	The system should provide information about the reviews that has been uploaded in the system.	The chair should validate the decisions about acceptance or rejection of papers performed by the reviewers.
Activation Condition	True (always activated)	Conference_status= activated	Conference_status= activated	Conference_status= activated	Conference_status= activated	Conference_status= revision
Deactivation Condition	False	Conference_status= canceled	Conference_status= canceled	Conference_status= canceled	Conference_status= canceled	Conference_status= canceled
Satisfaction Condition						
Fail Condition						
Type	Operational	Abstract	Operational	Operational	Operational	Operational
Decomposition		Modify conf details AND Get info submissions AND Get info reviews AND Validate reviews decision				
Related Action/Restriction						
Identifier	Create_new_conference()		Modify_conf_details()	Get_Info_submissions()	Get_Info_reviews()	Validate_reviews_decision()
Type	Service		Service	Service	Service	Task/Protocol

(continued)

Table 6 (continued)

Case study: Conference registration objective decomposition	
Description	<p>The registration must be performed by means of a graphical online application.</p> <p>After checking that the user that is trying to modify the conference details is authorized to do that, the system will provide a graphical online application to update the details. The information is shown by means of a graphical online application.</p> <p>Only pc members can access to the information about submissions. The information is shown by means of a graphical online application</p> <p>Only pc members that are not authors of the paper can access to the reviews of a specific paper. The information is shown by means of a graphical online application</p> <p>The chair should validate one per one the decision for each paper. If the chair performs the action by itself this objective would be pursued by means of a task. If the final decision is negotiated between the PC members this objective should be pursued by means of a protocol.</p>
Resources	<p>Access to the conferences database</p> <p>Access to submitted papers database</p> <p>Access to reviews database.</p>
Activation condition	<p>Registered user demand</p>
Inputs	<p>Deadlines, topics of interests and general information</p>
Precondition	<p>The entity that executes the task should be a registered user.</p>
Outputs	<p>After the validation the decision is considered final and authors should be notified.</p>
Postconditions	<p>The user that executes the task becomes the chair of the conference.</p>
	<p>After the review deadline is finished</p>

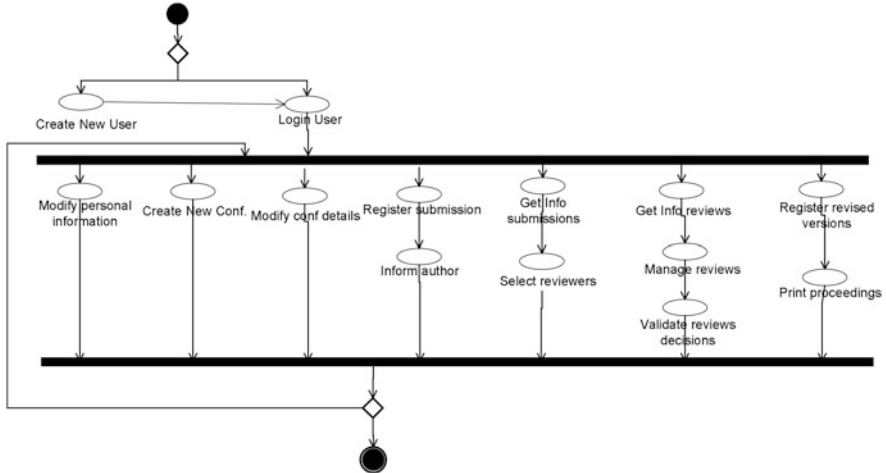


Fig. 10 Case study: use case

2.2 PHASE 2: Organization Description

During this phase, the analysis of the structure of the organization is carried out. In the previous phase of the methodology, the operational objectives are associated to specific actions or restrictions. In this phase, these actions and restrictions are analyzed in order to identify the roles of the system. A *role* represents part of the functionality of the system, and the relationships between roles specify the structure of the system.

2.2.1 Process Roles

The roles involved in this phase are the same as in the previous phase: the *system analyst* and the *domain expert*. The domain expert is in charge of supporting the system analyst facilitating information about domain requirements and restrictions.

2.2.2 Activity Details

The flow of tasks of this phase is reported in Fig. 11, and each task is detailed in Table 8.

2.2.3 Work Products

This phase uses the work products produced in the previous phase (*System definition*), and it produces the following work products: one *role description document* for each role; one *internal view diagram* for each role; and one *organizational view diagram*. Figure 12 shows graphically the products used and produced by each task.

Some of the work products generated are instances of the ROMAS metamodel. Figure 13 describes the relation between these work products and the metamodel elements in terms of which elements are *defined* (D), *refined* (F), *quoted* (Q),

Table 7 ROMAS suitability guideline

DISTRIBUTION: It is recommendable to use a distributed approach to develop the system if any of these questions is affirmative.

- Composed system: Is the system to be developed formed by different entities that interact between them to achieve global objectives? Are there different institutions involved in the system?
- Subsystems: Is the system composed by existing subsystems that must be integrated?
- Distributed data: Is the required data spread widely in different locations and databases? Are there any resources that the system uses distributed in different locations?

INTELLIGENT ENTITIES: It is recommendable to use an agent approach to develop the system if any of these questions is affirmative.

- Personal objectives: Do the entities involved in the system have individual and potentially different objectives?
- Heterogenous: Is possible that entities of the same type had been implemented with different individual objectives and implementations?
- Proactivity: Are the entities of the system able to react to events and also able to act motivated only by their own objectives?
- Adaptability: Should be the system able to handle dynamic changes in its requirements and conditions?

SOCIAL STRUCTURE: It is recommendable to use an organizational approach to develop the system if any of these questions is affirmative.

- Systems of systems: Does the system needs the interaction of existing institutions between which exist a social relationship in the real-world that must be taken into account?
- Social relationships: Do the entities of the system have social relationships, such as hierarchy or submission, between them?
- Departments: Is the functionality of the system distributed in departments with their own objectives but that interact between them to achieve common objectives?
- Regulations: Are there different regulations for different parts of the system, i.e., is there any regulation that should be applied to a group of different entities but not to the rest of them?
- Domain-like concepts: Is the domain of the system in the real-world structured by means of independent organizations?

INTEROPERABILITY: The system must implement interoperable mechanism to communicate entities if any of these questions answers is affirmative.

- Technical Interoperability: Is possible that different entities of the system use different (potentially incompatible) technologies?
- Process Interoperability: Is possible that different entities of the system employ divergent (potentially incompatible) processes to achieve their goals?
- Semantic Interoperability: Is possible that different entities of the system utilize different vocabularies and coding schemes, making it difficult to understand the data of others?

REGULATIONS: If the system has regulations associated it is recommended to apply a normative approach to develop the system. Only in the unlikely possibility that the norms of the system were static (no possibility of changing over time) and all the entities of the system are implemented by a trustworthy institution taking into account the restrictions of the system a non normative approach could be used.

- Normative documents: Is the system or part of it under any law or institutional regulation?
- Resources restrictions: Are there specific regulations about who or how system resources can be accessed?
- Dynamic regulations: Should the system be adapted to changes in the regulations?
- Openness: Is the system open to external entities that interact and participate in the system and these entities should follow the regulations of the system?
- Risky activities: Is there any action that if it is performed the stability of the system would be in danger?

TRUSTWORTHINESS: It is recommended to use a contract-based approach if any of these questions is affirmative.

- Formal interactions: Are there entities that depend on the behavior of the others to achieve their objectives and whose interactions terms should be formalized?
 - Contractual commitments: Should the entities of the system be able to negotiate terms of the interchanges of products and services and formalize the results of these negotiations?
 - Social commitments: Are the entities of the system able to negotiate their rights and duties when they acquire a specific role? Could the social relationships between agents be negotiated between them?
 - Control system: Is the system responsible of controlling the effective interchange of products between entities?
 - Openness: Is the system open to external entities that interact and participate in the system acquiring a set of rights and duties?
-

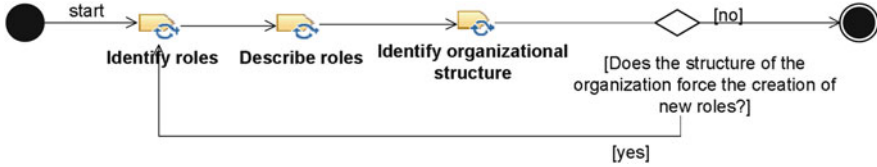


Fig. 11 Phase 2: activity tasks

Table 8 Phase 2: activity tasks

ID.task	Task	Description	Roles involved
2.1	Identify roles	Following the guideline <i>Role identification guideline</i> the roles of the system are identified and associated to different parts of the system functionality.	System analyst and domain expert
2.2	Describe roles	Following the guideline <i>Role description document</i> each identified role is analyzed. The details about each role are graphically represented by means of instances of the <i>internal view diagram</i> .	System analyst and domain expert
2.3	Identify organizational structure	Identify how the members of the organization interact between them, i.e., which social structure has the organization and graphically represent that using an <i>organizational view diagram</i>	System analyst and domain expert

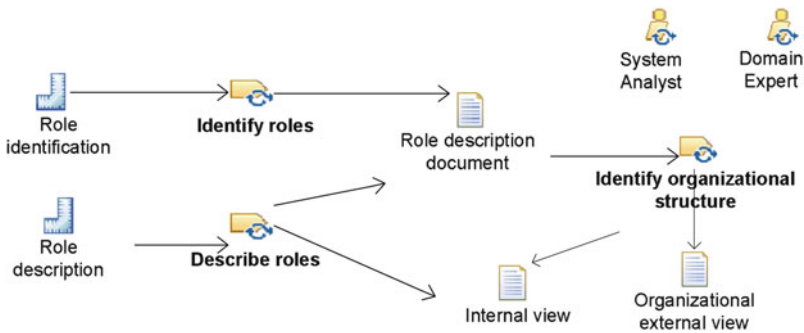


Fig. 12 Phase 2: resources and products used

related (R), or *relationship-quoted (RQ)*. These terms are described in the specification of the FIPA template [2].

Role Identification Guideline

A role is an entity representing a set of goals and obligations, defining the services that an agent or an organization could provide and consume. The set of roles represents the functionality of the system; therefore, the roles that a system should have are defined by the objectives of the system and should also take into account previous system requirements. The relationships and interactions between

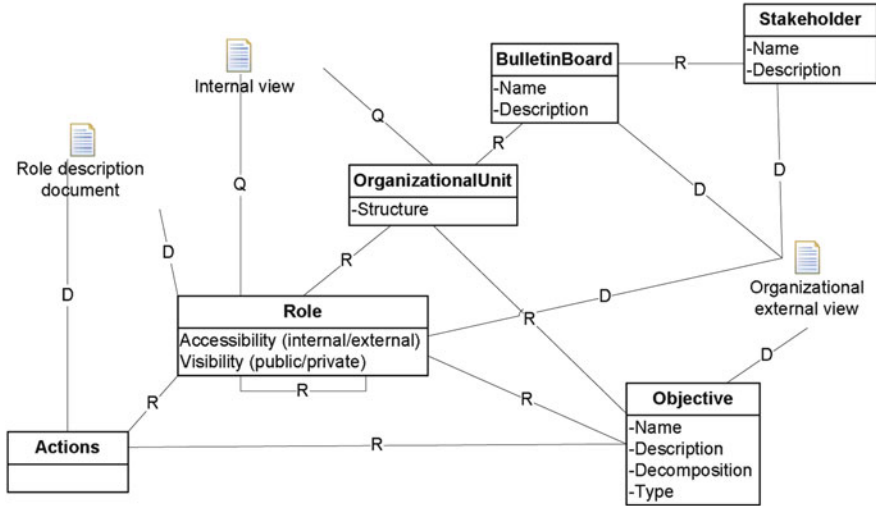


Fig. 13 Phase 2: relations between work products and metamodel elements

roles are the basis to define the structure of the organization. This guideline is designed to help the system analyst to identify the roles that are necessary in the system. Figure 14 represents the sequence of activities to do.

The first step of the process consists in asking the *domain expert* and checking in the *system description* document whether there is any pre-established role defined in the requirements of the system.

After that, every operational objective described in the *Objective description document* should be analyzed. It is recommended to analyze all the operational objectives obtained by the decomposition of an abstract objective before analyzing the next abstract objective.

If this operational objective is associated to a restriction, it would add a norm in the organization that pursues this objective. Besides, if this restriction is associated to an external event or a threshold, there must be an entity responsible for handling this event or measuring this threshold variable.

If this operational objective is associated to a protocol, the system analyst should revise the sequence of actions necessary to perform this protocol in order to obtain all the entities that participate in this protocol.

Usually, each task and functionality are associated to a role in order to create a flexible and distributed system. However, decomposing the system into too many entities can increase the number of messages between entities, the number of restrictions, and the complexity of each activity. Although the system analyst is responsible for finding the balance taking into account the specific features of the domain, here we present some general guidelines:

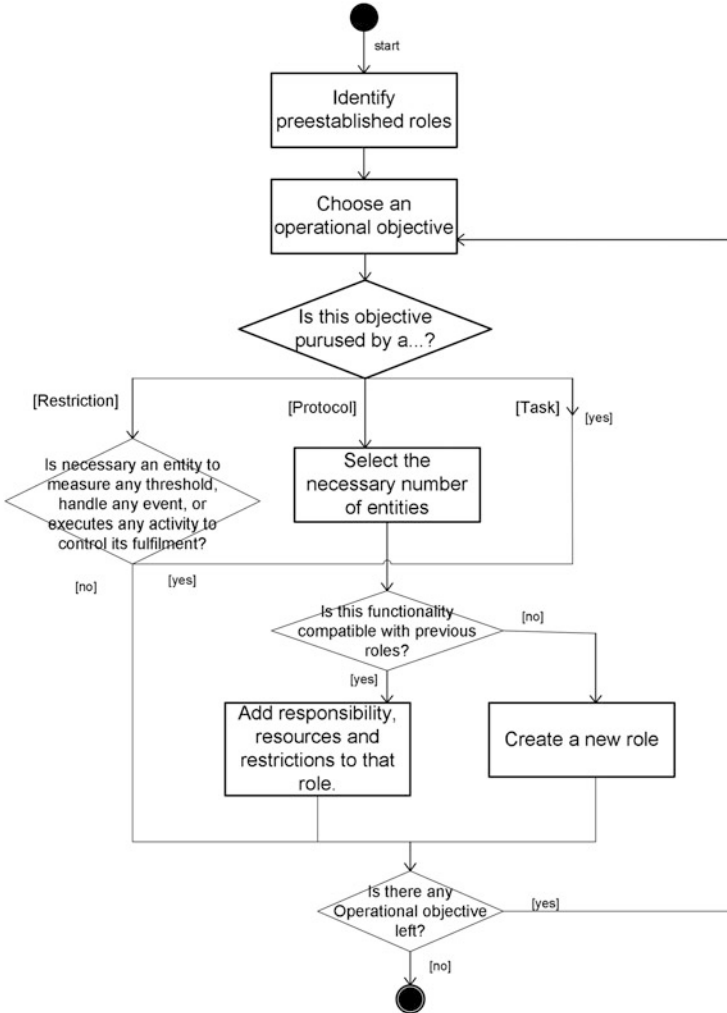


Fig. 14 Phase 2: role identification guideline

It is not recommended to assign two functionalities to the same role when

- These functionalities have different physical restrictions, that is, they must be performed in different places.
- These functionalities have temporary incompatible restrictions, that is, when they cannot be executed at the same time by the same agent. For example, it is usual that an entity was able to buy and sell, but as far as he is not able to sell and buy the same item at the same time, it is recommended to create one role Buyer and one role Seller. Remember that roles represent functionalities, so any final entity of the system could be able to play several roles at the same time.

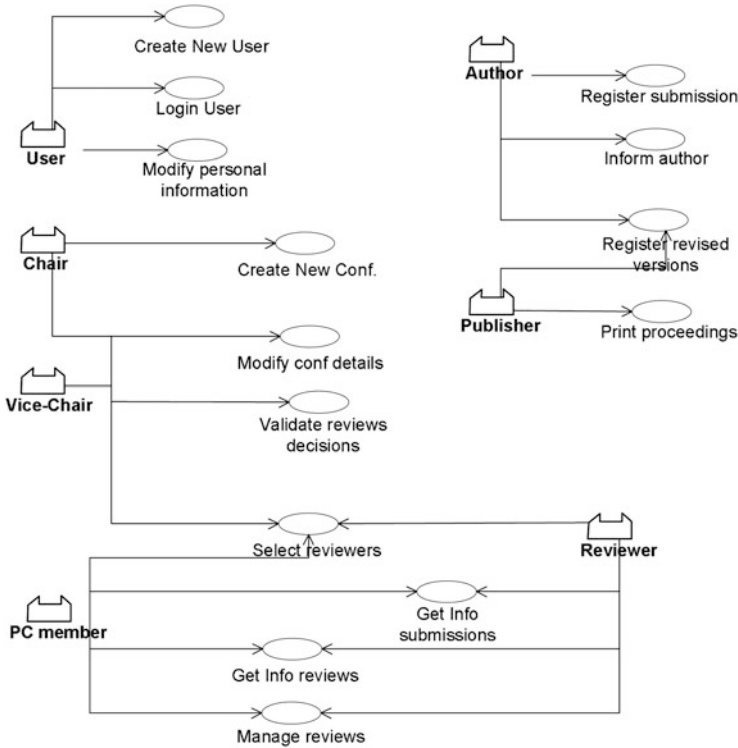


Fig. 15 Case study: roles overview

- These functionalities involve the management of resources that are incompatible. For example, the functionality of validating who is able to access a database should not be joint to the functionality of accessing the database. The reason is that if the entity who is accessing the database is responsible for validating its own access, the security can be compromised.

It is recommended to assign two functionalities to the same role when

- These functionalities cannot be executed concurrently and they are part of a sequence.
- These functionalities access the same resources and have the same requirements.
- These functionalities can be considered together as a general functionality.

In order to provide a fast and general overview, it is recommended to create a graphical representation of the relationships between the identified roles and the tasks and protocols. A relationship between a role and an action in this diagram means that the role is responsible for this action, participates in it, or is affected by its results. Figure 15 gives an overview of the results obtained when applying this guideline to the CMS case study. As is shown, seven roles have been identified:

Table 9 Phase 2: role description document

Property	Description	
Identifier	General name of role. It is recommended to select a short name or an abbreviation.	
Description	Informal description of the role. There is no length limitation on this text.	
List of objectives in which the role participates	Objective's identifier	The identifier of the objective that this role is going to contribute to its satisfaction
	Description of its contribution	An informal text describing how this role contributes to the satisfaction of this objective.
	Task/Protocol/Service	Which task is responsibility of this role or in which protocol this role participates. If this task should be activated as a reaction of a petition of other entity, this task should be published as a service.
	Responsibility shared with	A text explaining if this role pursue this objective alone or if he collaborates with others to achieve it.
Resources: Used	A list of the resources (products, services and applications) that this role requires to develop its functionality. This text should specify which type of access the role needs (reading, executing, writing, partial or full access)	
Resources: Provided	A list of the resources (products, services and applications) that this role provides.	
Events	A list of the events that this role handles and with which task.	
Restrictions	A list of the restrictions that are inherent to the functionality that this role executes. These restrictions are mainly derived from the information provided by the Domain Expert.	
Other memberships	A text explaining if this role in order to executes a task inside the organization it must be part of other different organization. If it is know, its rights and duties in the other organization must be detailed in order to ensure the coherence its objectives, rights and duties. However, due to privacy restrictions it is probable that many details cannot be shared between organizations.	
Personal objectives	A role can pursue an objective not directly related to the functionality required by the organization. For example, it can pursue an objective in order to maintain its integrity.	
Who plays the role	Is this role played by a single entity or by an organization? If it is played by an organization this organization must be analyzed following each step of the methodology.	

- The *User* role is an entity of the system that must be registered in order to access the system. On the contrary of the rest of the roles, this role is not related to any specific conference.
- The *Author* role is an entity attached to a specific conference in which this role can submit papers and receive information about the status of its papers.
- The *Chair* role is the main role responsible for a conference. This role is able to create a conference and share the responsibility of selecting the reviewers, validate the revisions, and update the conference details with the *Vice-Chair* role.
- The *PC member* role is responsible for managing the reviews, can participate in the selection of the reviewers, and have access to the information about submissions and reviews for a specific conference.
- The *Reviewer* role is responsible for submitting the reviews to the system.
- The *Publisher* role is responsible for managing the revised versions of the papers and printing the proceedings.

Role Description Document and Internal View Diagram

Each role should be described by means of the guideline offered in Table 9. This guideline allows the analysis of the roles and also the analysis of the relationships between them. After this analysis, this information is graphically represented by means of an *internal view diagram* for each role.

Table 10 Phase 2: Case study—reviewer role description document

Property	Description		
Identifier	Reviewer		
Description	This role is responsible from submit the reviews to the system. It is attached to an specific conference. It is responsible from submit a review from a paper within the established deadline and in the specific format that the conference specifies.		
List of objectives in which the role participates	Objective's identifier	Select reviewers	Manage reviews
	Description of its contribution	Reviewers should negotiate with <i>PC members</i> which papers are they going to review, when they are supposed to provide the reviews and which specific format these reviews must have.	Reviewers should send to the system their reviews. PC members would validate the reviews and contact the reviewers if there is any doubt in the information supplied
	Task/Protocol/Service	Reviewers participate in the protocol <i>Select_reviewers()</i>	Reviewers are responsible from the protocol <i>Manage_reviews()</i> and they offer the service <i>Execute_review()</i>
	Responsibility shared with	PC members	
Resources: Used	- Reviews and papers database. -They use the service <i>Get_info_submissions()</i>		
Resources: Provided			
Events	Conference details modification event		
Restrictions	- The same entity cannot be the author and the reviewer of the same paper. - Reviewers only have access to the information about their own reviews. They do not have access to other reviews or to the authors's personal details.		
Other memberships	Any entity that wants to play the role reviewer should be previously registered in the system as a <i>user</i> .		
Personal objectives	In general, there is no personal objectives for reviewers in the system. However, some conferences can encourage the productivity of their reviewers by offering rewards for each revised paper or for presenting the reviews before a specific date.		
Who plays the role	This role is played by a single entity.		

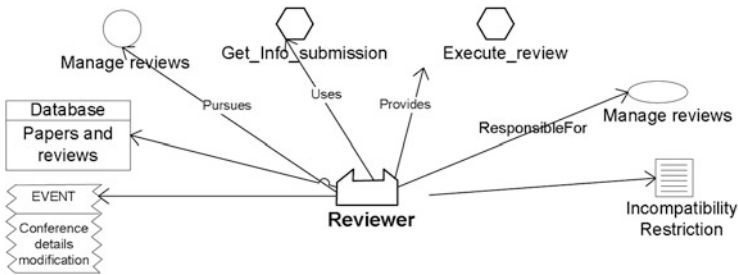


Fig. 16 Case study: reviewer role diagram

As an example, Table 10 shows the description of the role *reviewer* from the case study. Figure 16 shows its graphical representation using a ROMAS internal view diagram.

Organizational View Diagram

One organizational view diagram is created to graphically represent the structure of the system. Besides, this diagram also describes the overview of the system by

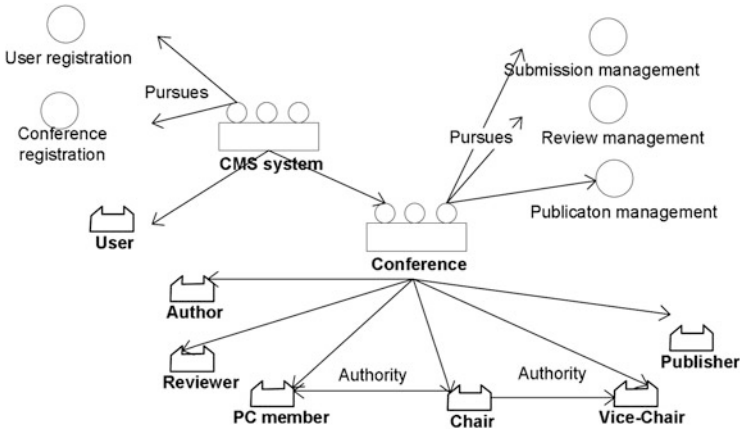


Fig. 17 Case study: organizational diagram

means of its global *objectives* and how the system interacts with the environment of the system (which *services* offers and demands to/from the *stakeholders* and which *events* the system is able to handle). The necessary information to fulfill these diagrams is obtained from the *System description document*. Due to the fact that in the literature, there are several well-defined guidelines to identify the organizational structure of a system, ROMAS does not offer any new guideline. Instead, the use of the guideline defined by the GORMAS methodology in [1] is recommended.

Figure 17 shows the organizational view diagram of the CMS case study. Inside the main system, the *Conference organization* represents each conference that is managed by the system. Each conference is represented as an organization because using this abstraction, each one can define its own internal legislation and can refine the functionality assigned to each entity of the system.

2.3 PHASE 3: Normative Context Description

During this phase, the normative context of the system is analyzed by means of identifying and formalizing the norms and the social contracts that regulate the entities' behavior inside the system. Norms are formalized using the following syntax: *(normID, Deontic, Target, Activation, Expiration, Action, Sanction, Reward)*.

2.3.1 Process Roles

The system analyst is responsible for performing the activities of this phase. The domain expert will support the system analyst during the identification of the norms that regulate the system.



Fig. 18 Phase 3: activity tasks

Table 11 Phase 3: activity tasks

ID.task	Task	Description	Roles involved
3.1	Identify restrictions from requirements	Following the guideline <i>Organizational norms</i> , the system analyst formalizes the norms described in the requirements that regulate the agent behavior. This norms refine the <i>organizational view diagram</i> of the organization associated to these norms.	System analyst and domain expert
3.2	Identify social contracts	Following the guideline <i>Social contracts</i> , the social contracts of the system are identified and formalized by means of the <i>contract template view diagram</i> .	System analyst and domain expert
3.3	Validate normative context	Following the guideline <i>Normative context validation</i> , the coherence among system’s norms and between them and the social contracts of the system is validated.	System analyst

2.3.2 Activity Details

The flow of tasks inside this activity is reported in Fig. 18, and the tasks are detailed in Table 11. Figure 19 describes the relation between these work products and the metamodel elements in terms of which elements are *defined* (D), *refined* (F), *quoted* (Q), *related* (R), or *relationship-quoted* (RQ).

2.3.3 Work Products

Figure 20 shows graphically the products used and produced by each task. The remainder of this section details the following work products: (1) *Organizational norms guideline*, (2) *Social contracts guideline*, (3) *Normative context validation guideline*, (4) *Contract template view diagram*. The *organizational view diagram* of the organization associated to the identified norms is refined by means of adding these norms to the diagram, in the same way that the *internal view diagram* of each role is refined by adding the social contracts and norms attached to this role.

Organizational Norms Guideline

This guideline specifies a process to identify and formalize restrictions on the behavior of entities gained from the analysis of system requirements. These normative restrictions are associated with specific features of the system and are usually well known by domain experts but not formally expressed in any document. This guideline helps the system analyst identify these restrictions with the support of the domain expert.

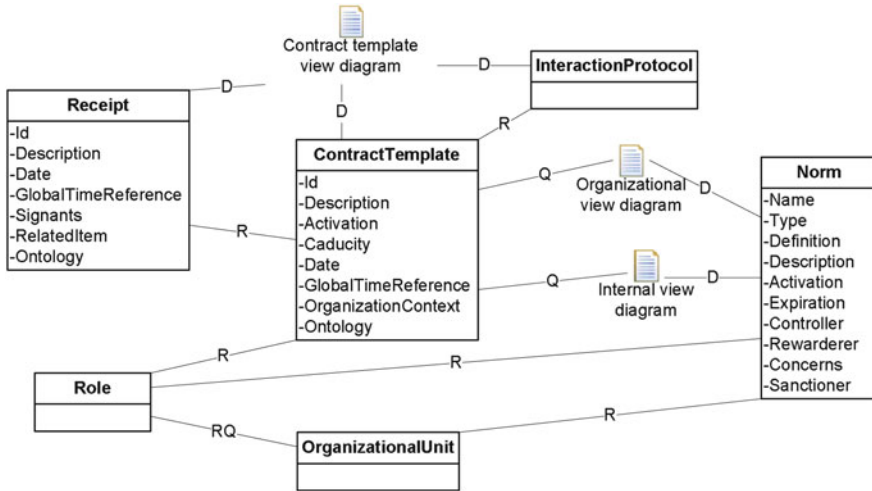


Fig. 19 Phase 3: relations between work products and metamodel elements

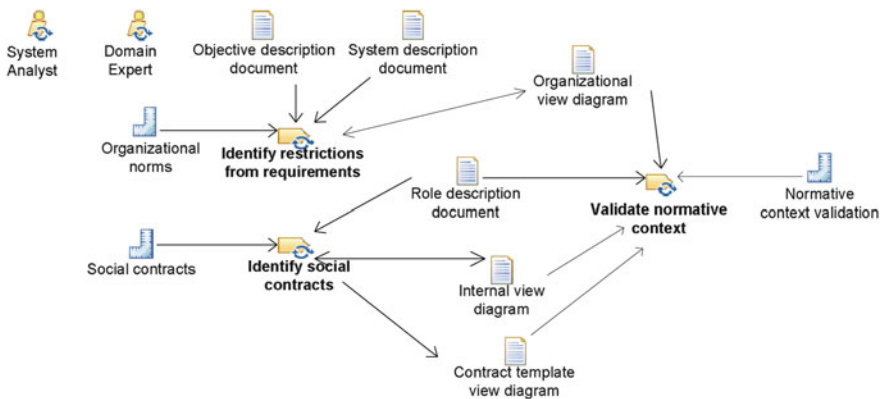


Fig. 20 Phase 3: resources and products used

Step 1 Analysis of system description documents These documents contains in plain text the requirements of the system, and if the system is composed of several organizations, there will be one system description document for each organization. The norms that arise from a document only affect the entities inside the organization that this document describes. The following steps are executed:

1.1 Analysis of the resources For each resource of the system, such as a database or an application, it is analyzed who has access to the resource, who cannot access it, and who is responsible for its maintenance. Therefore, permission, prohibition, and obligation norms are associated with these resources. For example, the analysis of the *Conference* database highlights the norm that only the *chair* of

the conference can modify the details of the conference (NModifyDetails, FORBIDDEN, !Chair, Modify(ConferenceDB),-,-,-,-).

1.2 Analysis of the events: For each event that the organization must handle, an obligation norm to detect this event is created. If the organization should react to this event by executing a task, another obligation norm is specified. The activation condition of this norm is the event itself.

1.3 Analysis of the offers/demands: External stakeholders can interact with the organization, offering and demanding services or resources. If the system is obliged to offer any specific service, an obligation norm is created. If there are specific entities that are allowed to offer, demand, or use a service, a related permission norm is created. On the other hand, if there are specific entities that are not allowed to offer, demand, or use it, a related prohibition norm is created.

1.4 Analysis of domain restrictions The last attribute of the system description document analyzes if there are normative documents attached to the organization or if there are specific domain restrictions that must be taken into account. In both cases, the restrictions are specified in plain text, so the system analyst should analyze and formalize these restrictions using a formal syntax. For example, in the CMS case study, the domain expert has claimed that “*Each conference should describe its internal normative*”. This restriction is formalized as (confNormative, OBLIGED, Conference, Define(Normative),-,-,-,-). This norm will be attached to every conference; therefore, the task of defining the internal normative should be added to a role inside the conference. In this case, this task has been added to the *chair* responsibilities.

Step 2 Analysis of the objectives description document We can differentiate two types of objectives: the objectives attached to restrictions and the objectives attached to specific actions. First, for each objective that pursues the stability of any variable of the system in a threshold, a forbidden norm should be created to ensure that the threshold is not exceeded. A variable of a system is anything that the system is interested in measuring; for example, the temperature of a room or the quantity of money that a seller earns. Second, for each objective that is attached to an action, an obligation norm is created in order to ensure that there is an entity inside the system that pursues this objective. The activation and expiration conditions of the created norms are determined by the activation and expiration conditions of the related objective.

Social Contracts Guideline

This guideline specifies a process to identify and formalize social contracts inside a specific organization regarding the information detailed in the *role description document*, the roles’ *internal view diagrams*, and the structure of the organization. Social contracts are used to formalize two kinds of social relationships: (1) *play role contract template*, which specifies the relationship between an agent playing a role and its host organization; and (2) *social relationship contract template*, which specifies the relationship between two agents playing specific roles. Social order thus emerges from the negotiation of contracts over the rights and duties of participants.

One *play role contract template* should be defined for each role of the organization in order to establish the rights and duties that any agent playing this role

should fulfill. Therefore, in the CMS case study, seven *play role contract templates* should be formalized: one for role *user* of the main organization and six for each role described inside the *Conference* organization (*author*, *reviewer*, *PC member*, *Chair*, *Vice-chair*, *Publisher*). That means that the rights and duties of an agent that tries to play a role inside a conference can be different, depending on how each conference negotiates these contracts. For example, one conference can establish that a PC member cannot submit a paper to this conference while other conferences do not add any restriction like that. Since every agent that intends to play a specific role inside the system must sign a *play role contract*, every agent will be aware of its rights and duties inside the organization in advance.

One *social relationship contract template* should be defined for each pair of roles that must interchange services and products as part of the social structure of the organization. Contracts of this kind should be negotiated and signed by the related entities and not by the organization as a whole. However, if the terms of the contract are not negotiated by the entities and the relationship between these agents is determined by their organization, it is not necessary to create a social relationship contract. Instead, the rights and duties of each role over the other are included in their respective *play role contracts*. In the CMS case study, there is an authority relationship between the *chair* role and the *vice-chair* role. The terms of this relationship are specified by each conference. Therefore, the rights and duties from one entity to the other are formalized in their respective *play role contracts*, and no social relationship contract is created.

Below, each step of the guideline that should be applied to each role of the system is described.

- *Adding identified norms*: Every restriction or norm identified during the application of the *Organizational norms guideline* that affects the role should be added to the contract. The norms that are attached to several roles but that include this specific role should be added. This can increase the size of the contract, so it is the responsibility of the domain expert to specify which norms should be communicated. For example, in the case of CMS case study, not all governmental norms related to the storage of personal data are included in the contracts; only a norm that specifies that any agent inside the system should follow this regulation is specified in the contracts.
- *Analysis of the organizational objectives*: In previous phases of the ROMAS methodology, the requirements of the system are analyzed by means of the analysis and decomposition of the objectives of the system. Each objective is associated with an action that must be performed in order to achieve it, and these actions are associated with specific roles that become responsible for executing them. Therefore, for each objective related to a role, an obligation norm must be created that ensures the execution of this action. If the action related to the objective is a *task*, the role is obliged to execute this task. If it is related to a *service*, the role is obliged to offer and register this service. The activation and expiration of the norm match the activation and expiration of the objective.
- *Analysis of offers/demands*: The description of each role should specify which resources and services this role must offer and which ones it can use. For each

resource and service that this role is able to use, a permission norm is added. For each resource or service that this role cannot have access to, a prohibition norm is created. Also, for each resource and service that this role is supposed to provide, an obligation norm is added. In this sense, an agent would not be able to play a role unless it were able to provide all the services and resources that are specified in the *play role contract*.

- *Analysis of the events*: For each event that the role must handle, a norm that forces any agent that plays this role to detect this event is created. If the role should react to that event by executing a task, an obligation norm is created whose activation condition is that event and indicates that the role should execute this action.
- *Analysis of the relationships*: As is discussed earlier, the norms derived from the social relationships between roles should be included in the *play role contract template* when they cannot be negotiated by the entities playing these roles, that is, they are rigidly specified by the organization. In other cases, a *social relationship contract* should be created and these norms included in it. The norms that are derived from the social relationship should be activated only when the social relationship is active and their deontic attribute depends on the type of relationship between the parties. If two roles are incompatible, a prohibition norm is added specifying this fact. In the same way, if any agent playing one role is required to play another, an obligation norm is included in the contract. Usually, a social collaboration appears when several roles should interact to achieve a global goal of the organization. In such cases, a set of obligation norms specifies which actions and services are the responsibility of each entity. If the collaboration relationship indicates *information*, it means that one role is obliged to inform another when some conditions occur. An *authority/submission* relationship requires the specification of: (1) which services should provide the submitted party, (2) which actions the authority can force the other agent to do, and (3) which actions the submitted party cannot perform without the consent of the authority.
- *Analysis of personal objectives*: A personal objective of a role is a goal that is not directly related to the main goals of the system, but that all the agents that play this role will pursue. The system as an entity can establish some restrictions on the performance of personal objectives. An example of a personal objective in the *CMS* case study is that although the agents that play the role *author* pursue the objective of *Submitting as many papers as possible*, each conference can establish limits on the quantity of papers that an author can submit to the same conference.

Normative Context Validation Guideline

The validation of the normative context is understood as the verification that there are no norms in conflict, that is, that the normative context is coherent. As is presented in [3], conflicts in norms can arise for four different reasons: (1) the obligation and prohibition to perform the same action; (2) the permission and prohibition to perform the same action; (3) obligations of contradictory actions; (4) permissions and obligations of contradictory actions. Therefore, after the specification of the organizational norms and the social contract templates that

define the structure of the organization, it is necessary to verify that the normative context as a whole is coherent.

Each organization can define its own normative context independently of the other organizations that constitute the system. The first step is analyzing the normative context of the most simple organizations, that is, the organizations that are not composed of other organizations. After that, we will analyze the coherence between this simple organization and the organization in which it is. This process will continue until analyzing the coherence of the main organization of the system.

In order to analyze the coherence of a specific organization, it is necessary to verify that: (1) There is no state of the system in which two or more organizational norms in conflict are active. (2) There is no norm that avoids the satisfaction of an organizational objective, that is, there is no norm that is active in the same states as the objective is pursued and whose restriction precludes the satisfaction of this objective. (3) There is no social contract that specifies clauses that are in conflict with the organizational norms. (4) There is no pair of social contracts whose clauses are in conflict between them and, therefore, the execution of one contract would preclude the satisfactory execution of the other one. (5) There is no social contract in which a role participates whose clauses preclude the satisfaction of the roles objective.

The validation task can be performed manually or by means of automatic techniques such as model checking. In [4], we present a plug-in integrated in our case tool that allows a simple verification of the coherence between the organizational norms and the contracts by means of the SPIN model checker [5].

Contract Template View Diagram

One contract template diagram is created for each identified social contract. The recommended steps to specify a contract template are

- *Identify signants*: If it is a *play role contract template*, the signants are the entity that tries to pursue this role and the organization as a whole. If there is a specific role in charge of controlling the access to the organization, the entity playing this role will sign the contract on behalf of the organization. If it is a *social relationship contract template*, the signants are the entities playing the roles that have the relationship.
- *Attach clauses*: The norms that have been identified by means of the *social contract guideline* are included in the contract. If the norm to be included in the contract must be in any contract of this type, this norm is defined as a *hard clause*. On the contrary, if the norm to be included in the contract is a recommendation, this norm is defined as a *soft clause*.
- *Define receipts*: In order to monitor the correct execution of the contract, it is recommended to define specific artifacts that entities participating in the contract should provide in order to prove the fact that they have executed their required actions.
- *Define authorities*: Optionally, the designer can define who is responsible for verifying the coherence of the final contract (*notary*), for controlling the correct execution of the contract (*regulation authority*), and for acting in case of dispute between the signant parties (*Judge*).

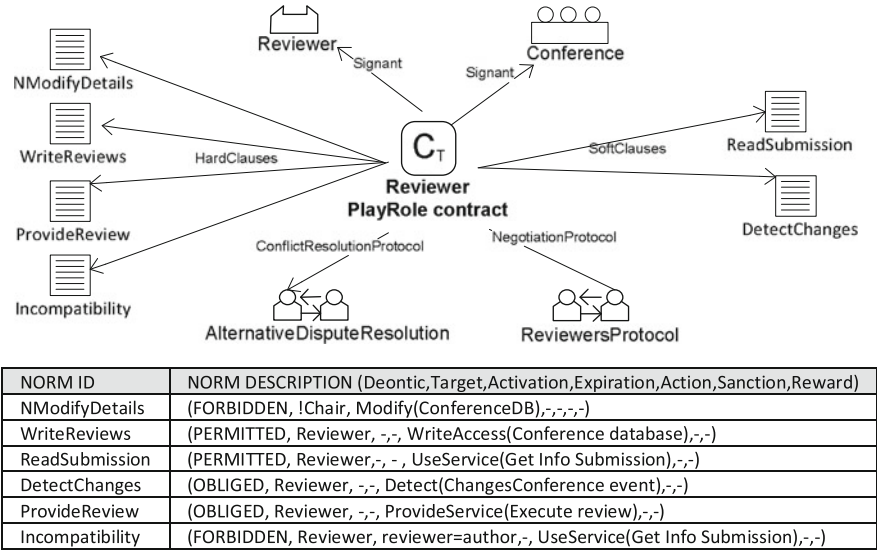


Fig. 21 Phase 3: Case study—reviewer play role contract template

– *Identify protocols*: Optionally, the designer can define specific negotiation, execution, and conflict resolution protocols. At this phase, only a general description of these protocols is provided. They will be completely specified in the next phase of the methodology.

Figure 21 shows the *play role contract template* that any entity that wants to play the role reviewer should sign. It is signed by the role that wants to play the role *reviewer* and by the *conference* in which the entity wants to participate. There are six clauses attached to this contract template that specify that an entity playing this role is not allowed to modify the details about a conference unless it is also the chair of this conference (*NModifyDetails* norm), neither to access the submission information about a paper in which he is also author (*Incompatibility* norm). This entity would have permission to access the reviews database (*WriteReviews* norm) and to use the service *Get Info Submission* (*ReadSubmission* norm). This entity would be obliged to detect when the conference details have changed (*DetectChanges* norm) and to provide the service *Execute review* (*ProvideReview* norm).

2.4 PHASE 4: Activity Description

During this phase, each identified task, service, and protocol is described by means of instances of the *activity model view*.



Fig. 22 Phase 4: activity tasks

Table 12 Phase 4: activity tasks

ID.task	Task	Description	Roles involved
4.1	Describe ontology	System domain concepts are analyzed. These concepts will be used to define the inputs, outputs and attributes of tasks, protocols and services.	System analyst and domain expert
4.2	Describe services	Define service profile attributes for each service. One <i>activity view</i> diagram is created for specifying each service implementation. If there are services that should be published to other members of the system or to external stakeholders, the <i>organizational view diagram</i> of the system should be refined by adding a <i>BulletinBoard</i> . This abstraction is an artifact where authorized entities can publish and search services.	System analyst
4.3	Describe tasks and protocols	Create one instance of the <i>activity view diagram</i> for each task and protocol to specify them. In addition to the protocols associated to objectives and roles, the contracts of the system should be completed by adding specific negotiation, execution and conflict resolution protocols.	System analyst

2.4.1 Process Roles

The domain expert should provide the domain ontology and should give support to the system analyst in the definition of the protocols, tasks, and services.

2.4.2 Activity Details

The flow of tasks inside this activity is reported in Fig. 22, and the tasks are detailed in Table 12. Figure 23 describes the relation between these work products and the metamodel elements in terms of which elements are *defined* (D), *refined* (F), *quoted* (Q), *related* (R), or *relationship-quoted* (RQ).

2.4.3 Work Products

Figure 24 shows graphically the products used and produced by each task. One *activity view diagram* is created for each task, protocol, or service identified in the previous phases of the methodology. Phase 2 shows the tasks, services, and protocols that each role should implement, and phase 3 identifies the negotiation, execution, and conflict resolution protocols for the contract templates.

An example is presented in Fig. 25. It shows the description of the *reviewer play role negotiation protocol*. First, the chair sends to the user that tries to play the role reviewer the details about the conference (deadlines, topics of interest, ...). The user analyzes this information, and if necessary, proposes a change in the review

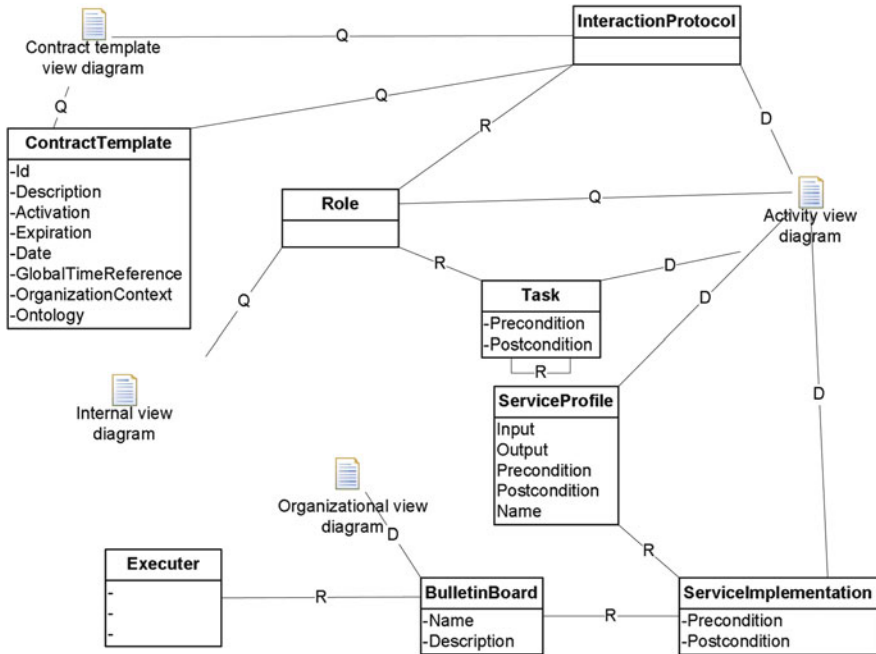


Fig. 23 Phase 4: relations between work products and metamodel elements

deadlines. This change can be accepted or rejected by the chair. If the chair rejects the change, he can finish the interaction or modify his proposal and send it again to the user. Once they have agreed on the conference details, the chair sends the user the specification of the contract, that is, the rights and duties that the user will acquire if he becomes a reviewer. This contract cannot be negotiated, so the user can reject it and finish the interaction or accept it and begin playing the role reviewer within this conference.

2.5 PHASE 5: Agents Description

During this phase, each identified agent is described by means of an instance of the *internal view* metamodel.

2.5.1 Process Roles

The tasks of this phase are executed by the collaboration between the system analyst and the domain expert. The domain expert should provide the information related to agent development requirements. The system analyst should formalize these requirements using the ROMAS diagrams.

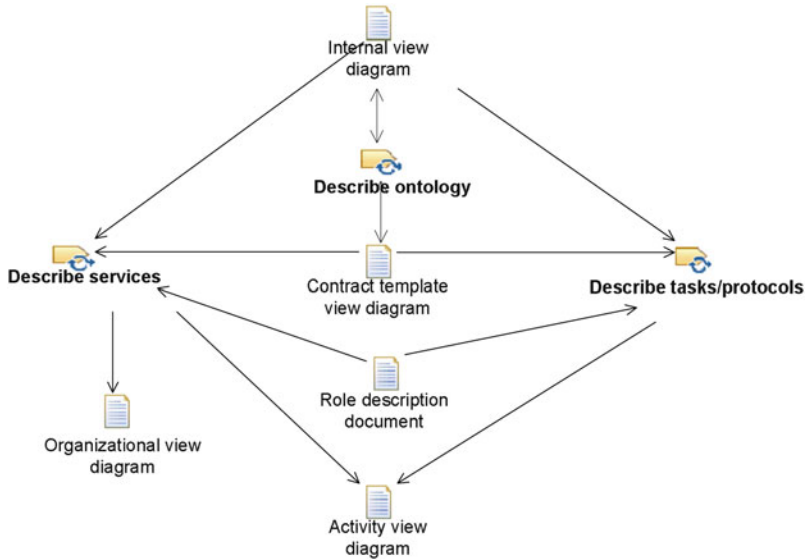


Fig. 24 Phase 4: resources and products used

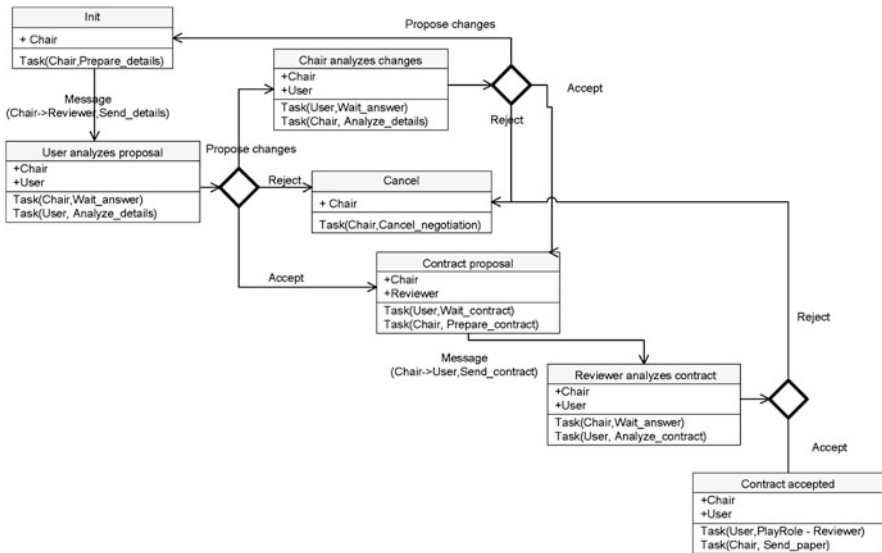


Fig. 25 Phase 4: Case study—reviewer play role negotiation protocol

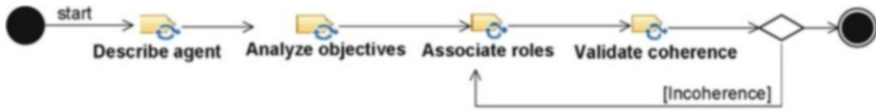


Fig. 26 Phase 5: activity tasks

Table 13 Phase 5: activity tasks

ID.task	Task	Description	Roles involved
4.3	Describe agent	Following the guideline <i>agent description document</i> , the development requirements of each agent are analyzed.	System analyst and domain expert
4.1	Analyze objectives	Following the guideline <i>objectives description document</i> detailed in Phase 1, the agent’s objectives are analyzed and decomposed in operational objectives.	System analyst and domain expert
4.2	Associate with system roles	Identify which roles the agent must play in order to achieve its objectives. This analysis is performed by matching the agent objectives with the roles functionality. Therefore, the <i>objective description document</i> of the agent is compared with the analysis of the roles presented in the <i>roles description documents</i> .	System analyst
4.4	Validate coherence	Validar que puede cumplir sus objetivos y que puede cumplir las tareas de los contratos y que las normas del sistema no van contra ninguna de sus normas personales	System analyst

2.5.2 Activity Details

The flow of tasks inside this activity is reported in Fig. 26, and the tasks are detailed in Table 13. Figure 27 describes the relation between these work products and the metamodel elements in terms of which elements are *defined* (D), *refined* (F), *quoted* (Q), *related* (R), or *relationship-quoted* (RQ).

2.5.3 Work Products

Figure 28 shows graphically the products used and produced by each task. First, an *agent description document* is created for each agent. Table 14 shows the related guideline and an example from the CMS case study. After that, each identified objective is analyzed following the guideline *objective description document* described in Phase 1. The analysis of the objectives in our running example shows that the main objective of the Ph.D. student agent, *Improve CV*, is decomposed into: *Submit thesis draft*, *Increase number of publications*, and *Collaborate in conferences*. The first objective is not related to any objective of the system, so it cannot be achieved inside the conference management system. The second objective, *Increase number of publications*, could be achieved if the agent joined conferences as an author. The authors’ play role contract templates establish that any agent that wants to join a conference as an author should submit an abstract of the paper. Since Bob has unpublished papers that he could submit to a conference, he can play the role *author*.

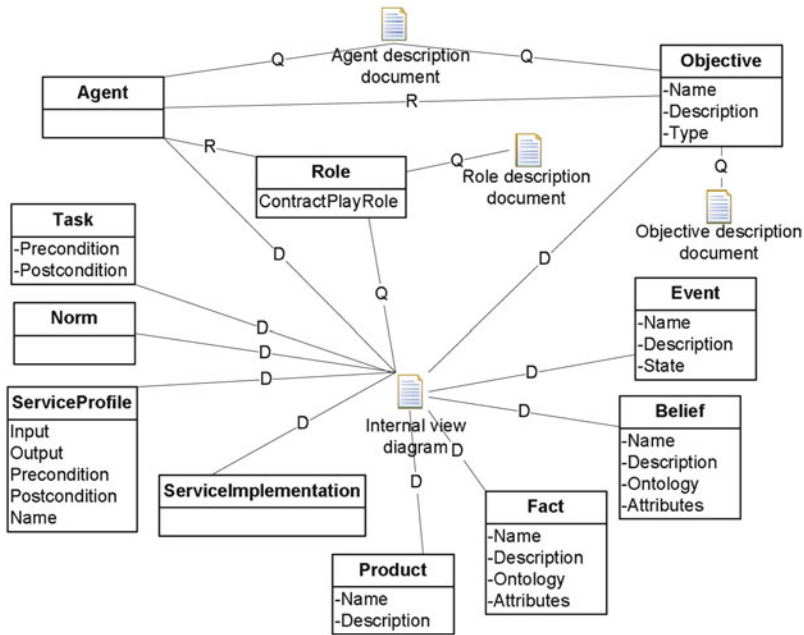


Fig. 27 Phase 5: relations between work products and metamodel elements

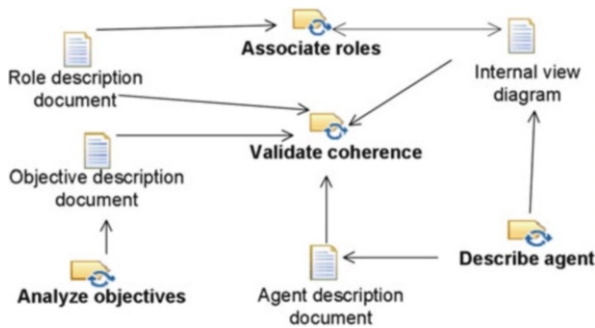


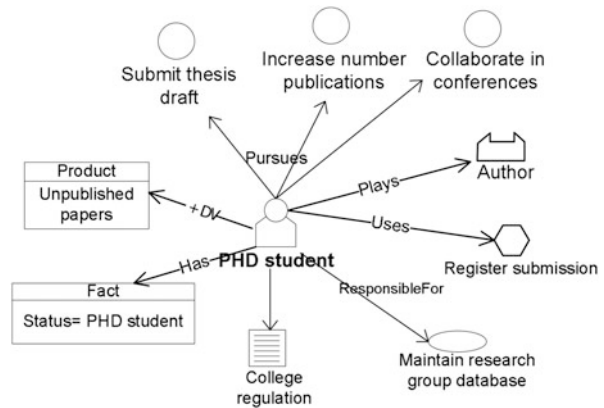
Fig. 28 Phase 5: resources and products used

The third objective, *Collaborate in conferences*, could be achieved by being the PC member of a conference. However, after the validation step, it is shown that Bob cannot play the role PC member because any agent that wants to play this role must be a doctor, and the agent is a Ph.D. student. One *internal view diagram* is created for each to specify the features of each agent. As an example, Fig. 29 shows the internal view diagram of the Ph.D. student agent.

Table 14 Phase 5: agent description document

Property	Description	Example
Identifier	General name of the agent. It is recommended to select a short name or an abbreviation.	Ph.D. student
Description	Informal description of the agent. There is no length limitation on this text.	It is a Ph.D. student who wants to participate in the system in order to improve its CV.
Objectives	Informal description of the agent's purposes of the agent.	- Improve its CV
Resources: Available for the agent	A list of the resources (products, services and applications) that the agent has or provides.	- Unpublished papers
Resources: Required by the agent	A list of the resources (products, services and applications) that the agent requires to develop its functionality. This text should specify which type of access the role needs (reading, executing, writing, partial or full access)	
Events	A list of the events that this agent handles.	
Other memberships	A text explaining if this agent is interacting with other active systems or organizations.	This agent plays the role Ph.D. student inside its college.
Restrictions	A list of the restrictions that are inherent to the agent.	This agent must follow the regulation of its college and that he is responsible of the maintenance of the research group database.

Fig. 29 Phase 5: Case study—Ph.D. student agent description



3 Work Product Dependencies

Figure 30 describes the dependencies among the different work products. For instance, the analysis of the *system description document* is necessary to define the *objective description document* and the *use case diagrams*.

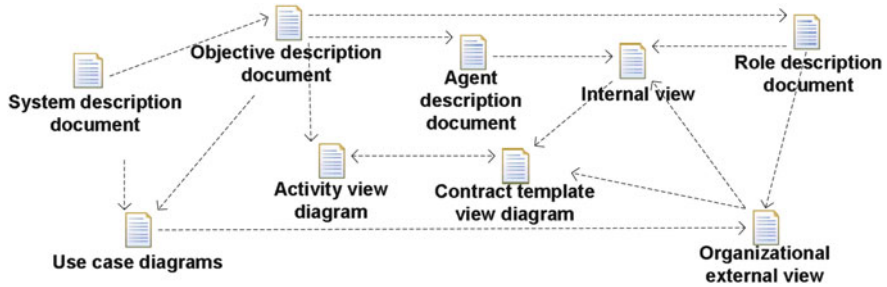


Fig. 30 Work product dependencies

Acknowledgements This work is partially supported by the TIN2008-04446, TIN2009-13839-C03-01, PROMETEO 2008/051 projects, CONSOLIDER INGENIO 2010 under grant CSD2007-00022 and FPU grant AP2007-01276 awarded to Emilia Garcia.

References

1. Argente, E., Botti, V., Julian, V.: GORMAS: an organizational-oriented methodological guideline for open MAS. In: Agent-Oriented Software Engineering, pp. 85–96. Springer, Berlin (2009)
2. Cossentino, M.: Design process documentation template. Technical Report, FIPA (2010)
3. Fenech, S., Pace, G.J., Schneider, G.: Automatic conflict detection on contracts. In: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC '09), pp. 200–214 (2009)
4. Garcia, E., Giret, A., Botti, V.: A model-driven CASE tool for developing and verifying regulated open MAS. *Sci. Comput. Program.* (2011). doi:10.1016/j.scico.2011.10.009
5. Holzmann, G.: Spin Model Checker, the: Primer and Reference Manual. Addison-Wesley Professional, Reading (2003)
6. Tolvanen, J.-P., Marttiin, P., Smolander, K.: An integrated model for information systems modeling. In: Proceedings of the 26th Annual Hawaii International Conference on Systems Science, pp. 470–476. IEEE Computer Society Press, USA (1993)

INGENIAS with the Unified Development Process

Alma Gómez-Rodríguez, Rubén Fuentes-Fernández,
Juan C. González-Moreno, and Francisco J. Rodríguez-Martínez

Abstract

This chapter introduces the definition of a process for the INGENIAS methodology. It is an adaptation of the unified development process (UDP). The process adopts the organization in phases of the UDP and proposes activities and work products equivalent to the ones of UDP and suitable to develop multi-agent-systems (MAS). As INGENIAS follows a model-driven approach, these activities are largely focused on obtaining a specification of the MAS, conforming to the INGENIAS metamodel. The case study of the conference management system illustrates the application of this process to get a description of the environment and users of the system and the workflows where it will act.

1 Introduction

The INGENIAS methodology [14] covers the full development cycle of multiagent systems (MAS). It includes support for requirements elicitation (basic), analysis, design, coding, and testing. It is intended for general use, that is, with no restrictions on application domains. The methodology is supported by the INGENIAS development kit (IDK), which contains a graphical editor for MAS specifications. Besides, the INGENIAS agent framework (IAF) [6] integrated in the IDK is a set of libraries that facilitate the implementation of INGENIAS agents and is built on top of the Java Agent DEvelopment Framework (JADE) [1].

A. Gómez-Rodríguez (✉) • J.C. González-Moreno • F.J. Rodríguez-Martínez
Universidad de Vigo, Campus As Lagoas s/n, 32004 Ourense, Spain
e-mail: alma@uvigo.es; jcgmoreno@uvigo.es; franjrm@uvigo.es

R. Fuentes-Fernández
Universidad Complutense de Madrid, Avda. Complutense s/n, 28040 Madrid, Spain
e-mail: ruben@fdi.ucm.es

Two alternative development processes have been proposed for the INGENIAS methodology: a heavy one based on the unified development process (UDP) [2] and an agile one adapting Scrum [17]. Brief descriptions of both of them can be found in the papers [14] for the UDP and [4] for Scrum. This chapter discusses with detail the first process, and chapter *INGENIAS-Scrum* addresses the second one.

The UDP [2], and its extended version the rational unified process (RUP) [16], distribute the tasks of analysis and design in four consecutive phases: *inception*, *elaboration*, *construction*, and *transition*. Each phase may have several *iterations*, where *iteration* means a complete development cycle. An iteration performs several *workflows* (i.e., disciplines), e.g., *requirements*, *analysis and design*, and *implementation*. Performing the sequence of iterations of all the phases leads to the final system.

The development process of the INGENIAS methodology following the UDP [2] is often represented by its authors in a tabular form (see Table 1). The table shows that INGENIAS considers only three phases: *inception*, *elaboration*, and *construction*, with two different types of workflow, *analysis* and *design*. When compared with the UDP, INGENIAS pays less attention to the *transition* phase and the traditional manual coding and testing of the system. It also splits in two the original UDP workflow of *analysis and design*, as it considers different activities for them. These differences are largely due to the model-driven approach that INGENIAS adopts [14].

The model-driven approach adopted in INGENIAS [14] is adapted from model-driven development (MDD) [3] and promotes focusing the development on the specification of systems and semi-automatically generating from them the other artifacts, such as documentation, code, tests, and configuration scripts. INGENIAS supports this kind of development with tools like the IDK [7] and the IAF [6], which automatically generate running code from the system specifications. This approach is based on the definition of a metamodel [5] that describes the elements that constitute an MAS from several viewpoints (i.e., *models*). These viewpoints are

- The definition, control, and management of each agent's mental state
- The agent interactions
- The MAS organization
- The environment
- The tasks and goals assigned to each agent

The process guides the specification of these viewpoints to a suitable degree according to the development goals. These goals can vary from eliciting requirements using the INGENIAS modeling language to the generation of a running MAS. Interested readers can find detailed references about the methodology from these authors in the following:

- Pavón, J., Gómez-Sanz, J.J., Fuentes-Fernández, R.: The INGENIAS methodology and tools. In: Henderson-Sellers, B., Giorgini, P. (eds) *Agent-Oriented Methodologies*, Chapter IX, pp. 36–276. Idea Group Publishing (2005).
- Gómez-Sanz, J.: *Modelado de sistemas multi-agente* (in Spanish). Ph.D. thesis, Facultad de Informática, Universidad Complutense de Madrid (2002).

Table 1 Main goals in the lifecycle for the INGENIAS methodology

Workflows	Phases		
	Inception	Elaboration	Construction
Analysis	<p>To generate use cases and identify actions of these use cases with the corresponding <i>interaction model</i>.</p> <p>To outline the system architecture with the <i>organization model</i>.</p> <p>To generate the <i>environment model</i> that reflects requirement elicitation.</p>	<p>To refine use cases.</p> <p>To generate the <i>agent model</i> that details the elements of the system architecture.</p> <p>To continue with the <i>organization model</i>, identifying workflows and tasks.</p> <p>To obtain the <i>tasks and goals model</i> that highlights control constraints (main goals and goal decomposition).</p> <p>To refine the <i>environment model</i> introducing new elements.</p>	<p>To study the remaining use cases.</p>
Design	<p>To generate a prototype using tools for rapid application development such as ZEUS [9].</p>	<p>To focus the <i>organization model</i> on workflows.</p> <p>To refine the <i>tasks and goals model</i>, reflecting the dependencies and needs identified in workflows and the relationships with system goals.</p> <p>To show how tasks are executed using the <i>interaction model</i>.</p> <p>To generate an <i>agent model</i> that shows the required mental state patterns.</p>	<p>To generate new diagrams or refining existing ones in the <i>agent model</i>.</p> <p>To study social relationships in order to refine the organization.</p>

- Grupo de Investigación en Agentes Software: Ingeniería y Aplicaciones. INGENIAS Section. <http://grasia.fdi.ucm.es/main/?q=es/node/61> (2010).
- Pavón, J., Gómez-Sanz, J.: Agent Oriented Software Engineering with INGENIAS. Lecture Notes in Computer Science, vol. 2691. Springer (2003).

1.1 Global Process Overview

The process of the INGENIAS methodology is designed following principles present in the UDP [2]. The UDP takes the system architecture as the guideline for development, and INGENIAS focuses the MAS definition and construction on



Fig. 1 Phases of the lifecycle for the INGENIAS methodology

its organization model. INGENIAS also organizes the development using three of the four UDP phases shown in Fig. 1. Each of these phases includes two workflows, *analysis* and *design*, focused respectively on the early definition of the system and on a closer description to the target platform. Being a MDD methodology [3], models are the key element in these definitions. The software engineering process proposes activities where these models are their inputs and products. Next sections discuss all these elements in detail.

1.2 Metamodel

INGENIAS is based on the concept of metamodel [14]. A metamodel defines the primitives and syntactic properties of a model. The INGENIAS metamodel is organized into five metamodels that correspond to the five different views of the system. Figure 2 shows the relationships between these metamodels. Each metamodel, in turn, specifies several kinds of diagram that can be used to describe its perspective of the system. Thus, the specification of a system in INGENIAS includes five models (one for each perspective), which in turn can each include different diagrams. The original work [5] explains in depth each of these metamodels.

Each of the considered metamodels supports several levels of abstraction, from the very abstract one that can be used in requirements elicitation to the low-level one required for coding. This is due to two main reasons. First, these metamodels are intended to be a precise definition of the specification language and its particular syntax and semantics. Second, they introduce all the modeling elements of the INGENIAS methodology required by MDD tools (in particular by the IDK [7]) to guide code generation, both of MAS and support tools.

An example of these metamodels is the *agent metamodel* shown in Fig. 3 (taken from [5] and translated to English). This metamodel shows several entities and their relationships. An *agent* is identified as an *autonomous entity*, with particular *goals* and a unique *identity*. Three more elements define the behavior of an *agent*: the *roles* it must play, the *tasks* it can perform, and its *mental state*. The relationships among them show how an *agent* can pursue its *goals* and how it achieves those *goals* executing particular *tasks*. It is important to note that this metamodel has some entities in common with other metamodels, such as *agents* and *roles*.

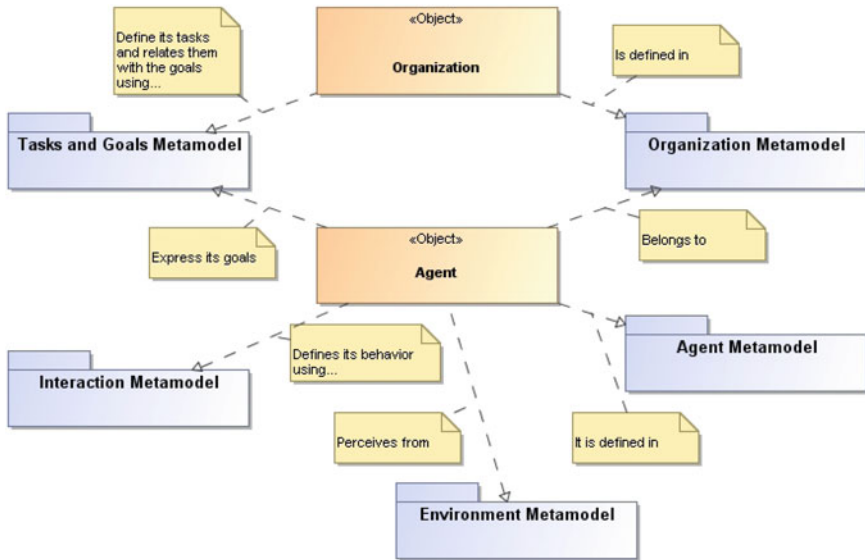


Fig. 2 Metamodel of the INGENIAS methodology

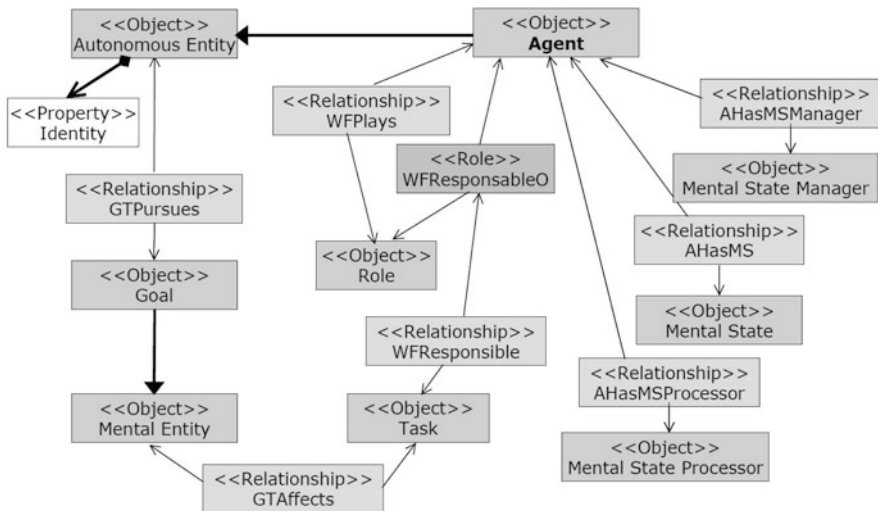


Fig. 3 Agent Metamodel of the INGENIAS methodology

1.3 Definition of MAS Metamodel Elements

Table 2 summarizes the main elements of the complete INGENIAS metamodel. For further details, the original documentation of the methodology must be considered: [5] for the metamodels and [13, 14] for their relationships with the MDD approach. Most of this information is also available at the INGENIAS website [8].

Table 2 Definition of MAS metamodel elements

Concept	Definition	Cross refs.
Agent	It is an autonomous entity with identity and goals that performs activities to achieve those goals.	Autonomous Entity
Application	An application is a wrapper to computational entities that are not agents. These entities are systems that have an interface and a concrete behavior.	
Autonomous Entity	It is the root concept that represents an entity with identity that pursues goals.	Goal
Goal	According to the BDI (belief–desire–intention) model for agents [15], a goal is a desired state that an agent wants to reach. In the planning stage, a goal is represented by a world state. Here a goal is an entity by itself, though it can be related to a representation of the world state using satisfactory relationships with tasks. These relationships contain references to descriptions of mental states of agents, so they refer to the image of the world that agents have.	Agent
Interaction	An interaction represents an exchange between two or more agents or roles. There can be only one initiator and at least one collaborator. An interaction also details the goals pursued. These goals should be related to the goals of the participants.	Agent, Role and Goal
Mental State	A mental state represents the information an agent has in a certain moment. It is an aggregate of mental entities.	Agent
Organization	It is a set of agents, roles, and resources that get together to achieve one or several goals. Inside an organization, there are no other organizations, just groups. You can think of an organization as a company that is internally organized into departments, which may be restructured without affecting the external image of the company.	Agent
Resource	A resource is an element available in the environment for use but lacking a defined interface. INGENIAS adopts the notion of resource in TÆMS [10], though it makes no distinction between consumable and nonconsumable resources.	
Role	A role is a self-contained grouping of functionalities. When an agent plays a role, it is able to execute all the tasks associated to the role and participate in the same interactions.	Agent
Task	Tasks encapsulate actions or nondistributable algorithms. They can use applications and resources. Tasks generate changes in the mental state of the agent that executes them. These changes can include (a) modifying, creating, or destroying mental entities and (b) changing the perceived world by acting over applications, which in turn act over the world producing the events that agents perceive. Tasks can be assigned to roles, but at the end, they belong to agents that provide their actual implementation.	Agent and Role
Workflow	A workflow is an abstraction of a process that has been automatized using activities and identifying responsibility relationships.	

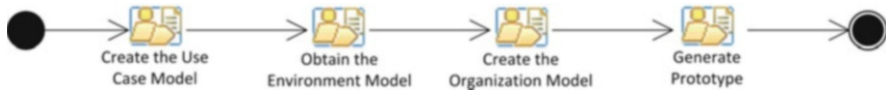


Fig. 4 Activities and workflows of the inception phase

2 Phases of the Process

2.1 Process Documentation: Phase

INGENIAS considers that the development starts from a document describing the problem. This can be considered as the initial input of the process. From this document, the *inception* phase introduces several activities, which Fig. 4 shows. As stated previously, each of the INGENIAS phases may have several iterations, so the figure just describes a single iteration.

Regarding the *analysis* workflow, the activities to make at this level are

- Create use case model
- Obtain environment model
- Create the organization model to initiate the architecture

Regarding Design phase only the construction of a rapid prototype must be addressed. Figures 5 and 6 show the details of these activities, the tasks associated to them, and the related work products. The Figure also identifies the roles responsible for each task and the kind of responsibility they assume.

2.1.1 Process Roles

The INGENIAS methodology makes no explicit reference to the roles implied in the development. Nevertheless, and considering the activities to be done and their level of abstraction, two roles are described as implied in the process: the *system analyst* and the *designer*.

The *system analyst* is responsible or performs most of the activities in this phase. In particular, the analyst's responsibilities are

- Identify use cases and construct and refine use case diagrams. From the initial description of the problem to be solved, the analyst must obtain the use cases that will guide the creation of the interaction model.
- Define the environment model, showing the interaction of the system with its environment. This implies: to identify applications, i.e., all the software and hardware that interact with the system and are not designed as agents; to associate operations to particular applications; and to define the agent perception on applications.
- Obtain the architectural view of the system using the organization model. This means generating a structural definition of the system by identifying groups in the organization, their members, and goals.

The other role present in this phase, the *designer*, is responsible for generating the prototypes to validate the system analyst's specifications. This development is

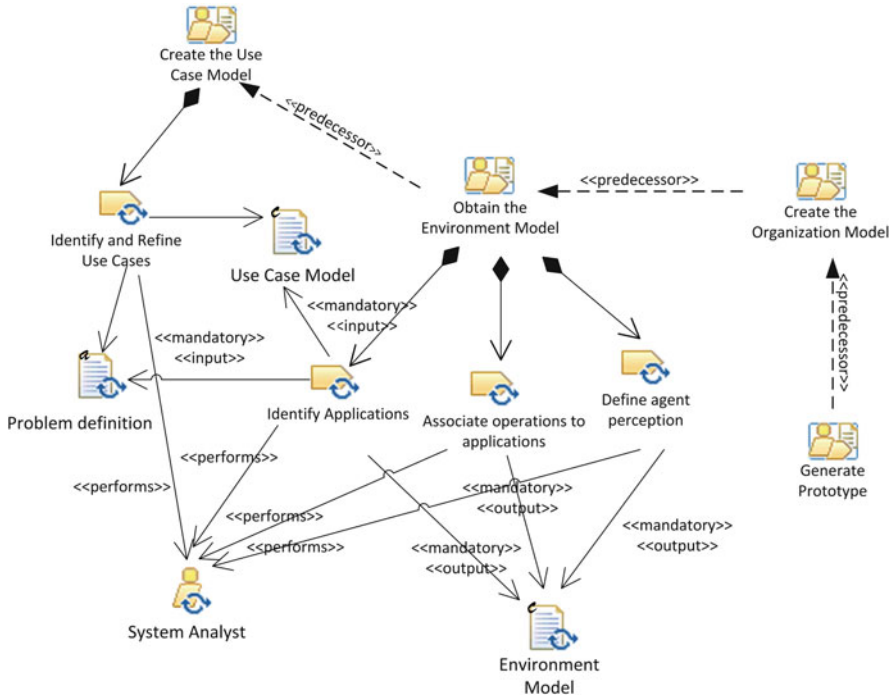


Fig. 5 Detailed tasks of inception activities (part A)

done using rapid application development tools such as ZEUS [9] or the IDK with the IAF [6].

2.1.2 Activity Details

This section details the activities previously outlined for the inception phase through their tasks.

Create the Use Case Model. The generation and refining of use cases has been identified as a unique task. Its goal is to identify the intended functionality of the system. Knowing this functionality allows identifying interactions and establishing their initiators, collaborators, and nature. The nature of the interaction influences the type of control applied to the involved agents and can be planning, cooperation, contract-net, or competition.

Obtain the Environment Model. The environment model shows the elements that constitute the environment of the system and, in consequence, that agents have to perceive. This model defines elements of three main types: *agents*, *resources*, and *applications*. The tasks that must be accomplished to obtain the environment model of the system to develop are outlined in Fig. 7 and further explained in Table 3.

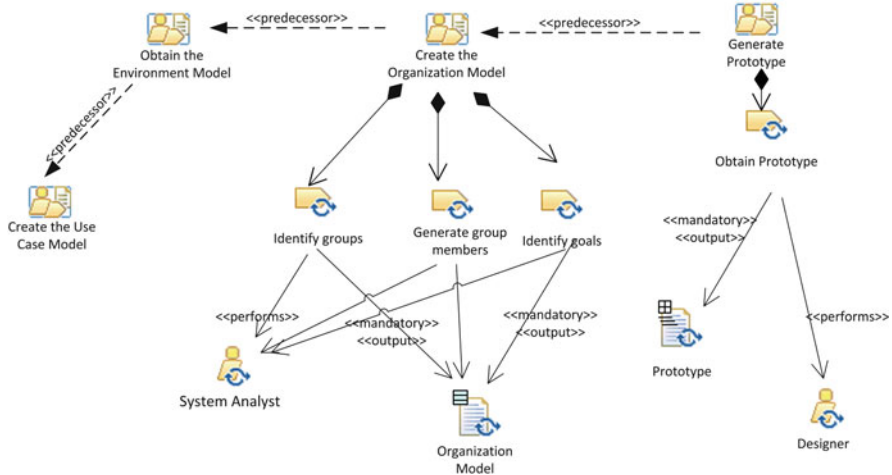


Fig. 6 Detailed tasks of inception activities (part B)

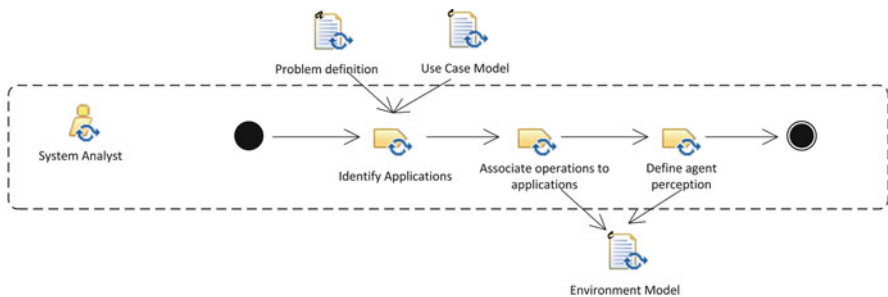


Fig. 7 Obtain the environment model in the inception phase of INGENIAS

Create the Organization Model. One of the key activities in the inception phase is to start the definition of the system architecture. This is done by specifying the organization model, which reflects mainly the system structure and workflows. Figure 8 shows the main tasks related with this activity. These tasks try to obtain an organizational view of the system, attending to its structural, functional, and social aspects. Table 4 provides the detailed definition of these tasks.

Generate Prototype. The construction of a prototype is a unique task. As said previously, the prototype is generated using a rapid application development tool.

2.1.3 Work Products

The inception phase produces as a result four basic work products: a use case model, an environment model, an organization model, and a prototype of the system to be built. Figure 9 shows the relationships among these models and the metamodel elements. For instance, the organization model defines the *organization* and *agent*

Table 3 Tasks of activity *Obtain the Environment Model* of the inception phase of INGENIAS

Activity	Task	Description	Involved roles
Obtain the Environment Model	Identify applications	All the software and hardware that interact with the system and that are not designed as an agent are considered <i>environment applications</i> .	System Analyst
	Associate operations to applications	<i>Operations</i> are associated to <i>environment applications</i> defined by requirements. These operations have a signature, preconditions, and postconditions. The identification of operations is a conventional engineering task.	System Analyst
	Define agent perception	This task defines agent perception on environment applications. At this moment of the process, it is enough to link <i>agents</i> and <i>environment applications</i> .	System Analyst

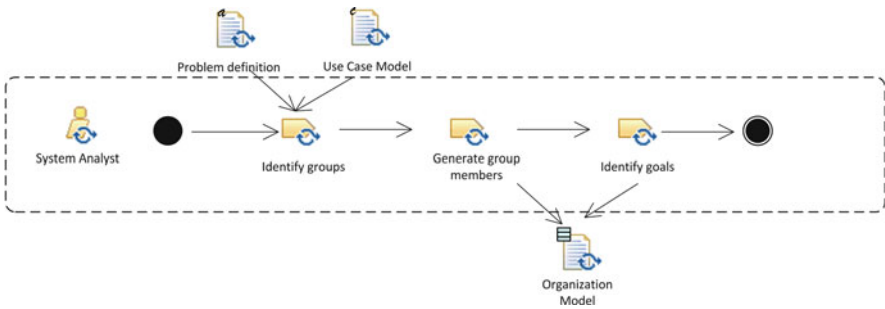


Fig. 8 *Create the Organization Model* in the inception phase of INGENIAS

metamodel elements and uses the *roles* and *goals* previously defined in the use case model. In this particular case, the *organization* concept also includes the *groups* within the organization (see the *organization* definition in Table 2).

Work Product Kinds

Table 5 introduces all the work products of the inception phase with their description and kind.

Use Case Model

The use case model gathers the use case diagrams, describing the functionalities that must be provided by the system. These diagrams follow the standard unified modeling language (UML) [12] with some INGENIAS extensions.

Figure 21 shows the initial use case diagram for the case study of the conference management system (CMS).

Table 4 Tasks of activity *Create the Organization Model* of the inception phase of INGENIAS

Activity	Task	Description	Involved roles
Create the Organization Model	Identify groups	Identify the <i>groups</i> in the system as participants in each particular workflow.	System Analyst
	Generate group members	The members of a <i>group</i> can be <i>agents</i> , <i>roles</i> , <i>resources</i> , and <i>applications</i> . These are assigned to groups, creating the corresponding relationships. If needed, the groups can be decomposed in order to reduce their complexity.	System Analyst
	Identify goals	The organization has a set of goals that must justify the collaboration between agents. The <i>goals</i> identified in this task (or their decomposing goals) will be assigned later to individual <i>agents</i> or <i>roles</i> in the <i>tasks and goals model</i> .	System Analyst

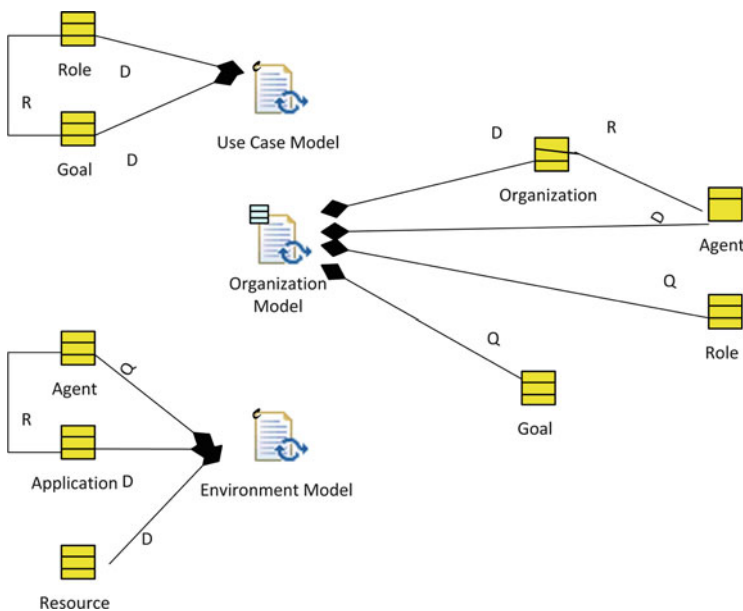


Fig. 9 Structure of the work products of the inception phase in INGENIAS

Environment Model

The environment model shows the applications outside the system, which condition the perception of the agents. In order to simplify such an environment, the model distinguishes several observable entities: the resources the system will consume, the applications it will use, and the agents that will interact with the previous elements.

Figure 22 shows a diagram of the environment model in the case study.

Table 5 Work product kinds of the inception phase of INGENIAS

Name	Description	Work Product Kind
Use Case Model	A description of the functionalities that must be provided by the system, as well as the actors implied. It may include free text descriptions.	Composite
Environment Model	The model provides a description of the applications outside the system and determines the perception of agents.	Composite
Organization Model	The model describes the organization, introducing the groups the system is formed of, and the pursued goals.	Structural

Organization Model

The organization model defines the structure where agents, resources, tasks, and goals relate. This structure includes among other elements: organizations, groups, and workflows.

In Fig. 23, an example of this kind of work product is shown. The diagram shows the groups in the CMS, the roles belonging to them, and the goals these roles pursue.

2.2 Process Documentation: Elaboration Phase

The second phase of the methodology is *Elaboration*. It constitutes the central part of the process, as it includes the most time-consuming activities. As in the previous phase of inception, the elaboration phase comprehends several activities related to the *analysis* workflow and others that can be identified with the traditional concept of design.

Figure 10 shows the activities of this phase. They are as follows:

- Refine the use case model
- Refine the organization model
- Create the agent model
- Create the tasks and goals model
- Refine the environment model
- Show task execution using the interaction model

Figures 11 and 12 show the activities of the elaboration phase. For the sake of simplicity, only the most significant relationships are shown. The diagram also includes the tasks that make up each activity, as well as the work products they use and/or produce. A more detailed description of all these elements will be introduced in the next sections.

2.2.1 Process Roles

As stated before, INGENIAS makes no explicit reference in its definition to the roles implied in the development. Nevertheless, and attending to the different focuses that



Fig. 10 Activities and workflows of the elaboration phase

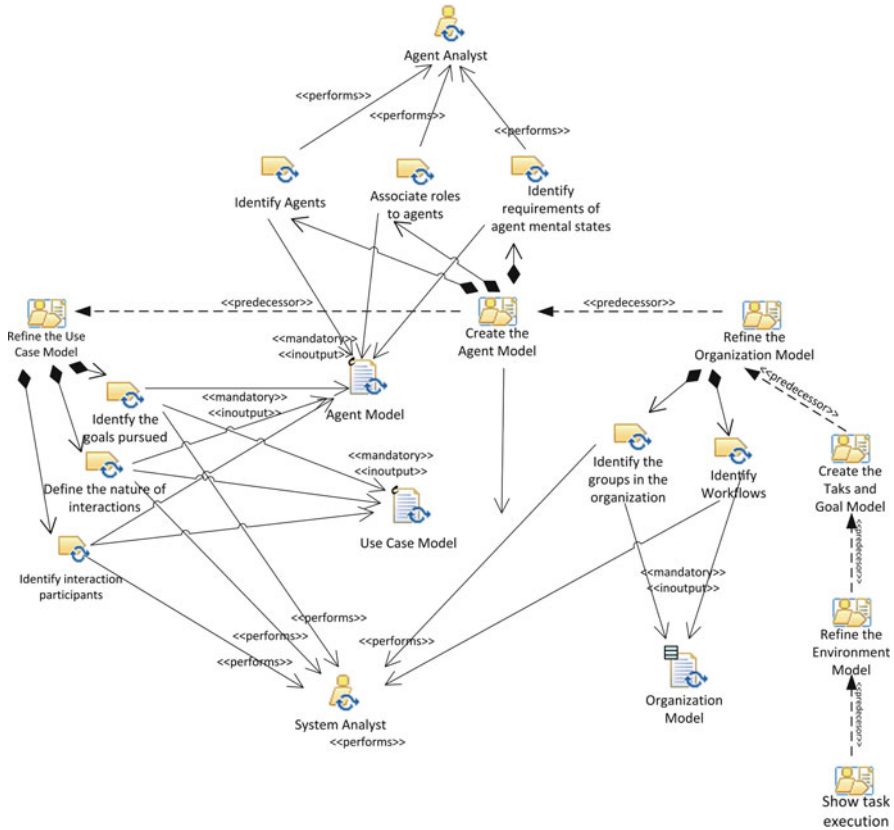


Fig. 11 Detailed tasks of elaboration activities (part A)

the activities suggest, we have identified three different participant roles within this phase: the *system analyst*, the *agent analyst*, and the *designer*.

The *system analyst* has identified the use cases in the previous phase, so that, in this phase also it is the role responsible for refining that model. In addition, this analyst continues refining the organization and environment models.

Another role implied in this phase is the *agent analyst*. This is a specialist analyst that provides a detailed definition of the system in terms of its *agents*. Therefore, the responsibilities of this analyst include the activities to create the agent and the tasks and goals models.

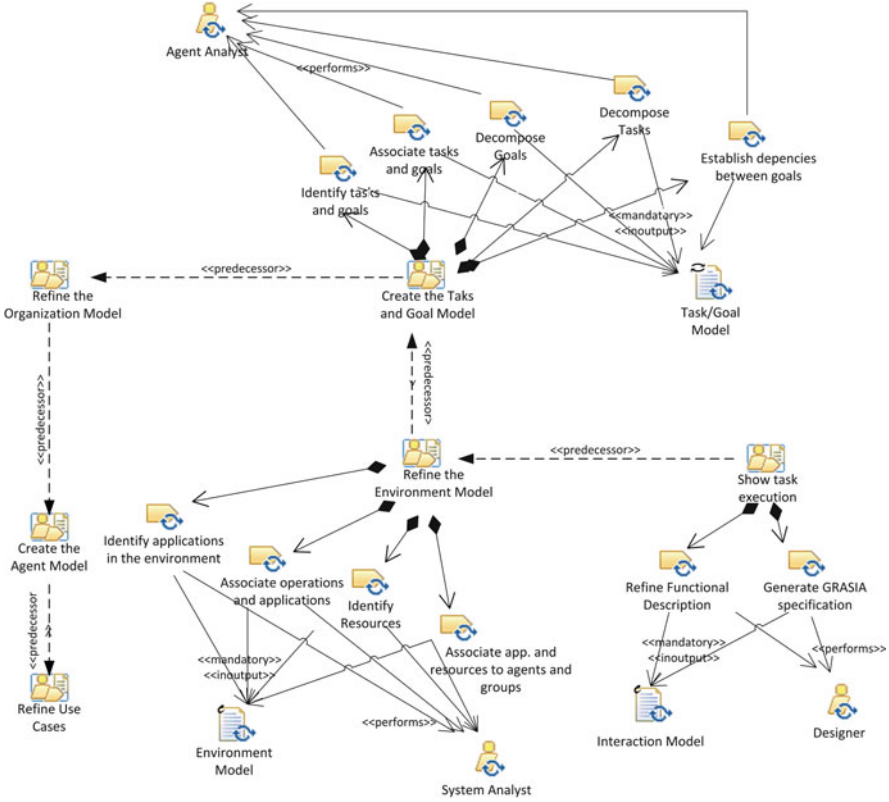


Fig. 12 Detailed tasks of elaboration activities (part B)

The third role is the *designer*. In this phase, the only responsibility of this role is to describe tasks execution using the interaction model.

2.2.2 Activity Details

This section details the activities previously outlined for the elaboration phase through their tasks. These details are provided through their decomposition in tasks.

Refine the Use Case Model. This activity identifies several components and features of the already identified use cases. It establishes their participants and their roles in the interaction, their nature (i.e., type according to agent literature), and the goals these interactions pursue. Figure 13 introduces a detailed view of the activity, including its tasks, work products, and involved roles. The details of these tasks appear in Table 6.

Refine the Organization Model. The initial organization model is improved during this activity with two dimensions of decomposition. A structural definition of

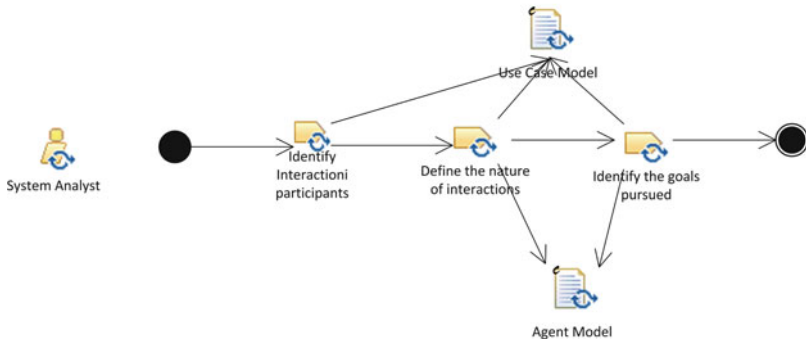


Fig. 13 Refine the Use Case Model in the elaboration phase of INGENIAS

Table 6 Tasks of activity *Refine the Use Case Model* of the elaboration phase of INGENIAS

Activity	Task	Description	Involved roles
Refine the Use Case Model	Identify interaction participants	Use cases can involve several participants that must be identified. In addition, the role played by each participant in the interaction must also be established, distinguishing between <i>initiators</i> and <i>collaborators</i> .	System Analyst
	Define the nature of interactions	The methodology distinguishes several kinds of interaction: planning, cooperation, negotiation and competition. These are extracted from research in the agent’s field. These kinds of interaction have an influence on the type of processing of the mental state for each agent. For instance, a planning interaction probably requires planning capabilities from the involved agents.	System Analyst
	Identify the goals pursued	The participants in a use case interact in order to achieve a goal or a set of goals. This task identifies these <i>goals</i> .	System Analyst

the system is provided through the identification of its groups and their members. A functional definition is established through the identification of workflows and their relationships. Table 7 details these tasks.

Create the Agent Model. The agent model tries to provide a view of the system that takes into account the individual agents that compose it. In order to obtain such model, this activity identifies agents and their features, e.g., social and negotiation skills or learning capabilities. Then it links the roles identified in previous activities with the agents that will play them. Having a clear initial picture of the kind of responsibilities of each agent, this activity also identifies requirements for their mental state manager and mental processor. Table 8 provides more details on these tasks.

Table 7 Tasks of activity *Refine the Organization Model* of the construction phase of INGENIAS

Activity	Task	Description	Involved roles
Refine the Organization Model	Identify the groups in the organization	This task identifies the <i>groups</i> that form a part of the <i>organization</i> and generates their members. When groups are too complex, this task can decompose them into several groups.	System Analyst
	Identify workflows	This step identifies <i>workflows</i> to give a functional definition of the system. After their identification, workflow decomposition can be applied to reduce complexity.	System Analyst

Table 8 Tasks of activity *Create the Agent Model* of the elaboration phase of INGENIAS

Activity	Task	Description	Involved roles
Create the Agent Model	Identify agents	INGENIAS identifies as <i>agents</i> those entities of the system that are modeled following the Rationality Principle [11]. This task also establishes what are the qualities each agent should exhibit, e.g., intelligence degree, learning skills, or social reasoning.	Agent Analyst
	Associate roles to agents	The <i>roles</i> previously identified must be associated to the <i>agents</i> that will play them. This assignment helps to understand how the goals of a particular role are accomplished.	Agent Analyst
	Identify requirements of agent mental states	This task introduces a higher level of detail in agent definitions. It considers the requirements on the mental entities an agent will need to manipulate in order to establish its mental state manager and mental processor.	Agent Analyst

Create the Tasks and Goals Model. This activity decomposes the goals and tasks identified in previous diagrams. It includes several interleaved tasks. Analysts need to further decompose goals and tasks in hierarchies and, at the same time, identify which goals and tasks are linked by the different INGENIAS relationships, such as satisfaction or creation. Goals are initially identified with requirements. As their decomposition becomes more detailed, desires of agents also appear as goals (see the BDI model in [15] regarding the definition of desires). According to the INGENIAS methodology, each agent task should have at the end of the specification a goal associated, as only when that goal is not satisfied the task is candidate for execution. The reciprocal for goals is not always true, as a goal can be satisfied because the related task generates evidences of its satisfaction or because the goals of its decomposition have been satisfied. Table 9 shows the tasks that must be accomplished to obtain the tasks and goals model in this activity.

Table 9 Tasks of activity *Create the Tasks and Goals Model* of the elaboration phase of INGENIAS

Activity	Task	Description	Involved roles
Create the Tasks and Goals Model	Identify tasks and goals	The <i>goals</i> can be identified with requirements, but also with the desires associated to agents as the goal decomposition proceeds. As for goals, two approaches are possible: to relate the <i>tasks</i> with the functions required in the system or to obtain them from association to objectives.	Agent Analyst
	Associate tasks and goals	The association between tasks and goals determines how a goal is satisfied. It must be noted that it is not mandatory that each and every goal is associated with a task. A <i>goal</i> can be satisfied through the execution of a <i>task</i> or the satisfaction of its <i>subgoals</i> . The associations between tasks and goals do not only indicate satisfaction, as other relationships are possible. Examples of them are <i>create</i> , <i>destroy</i> , and <i>affects</i> .	Agent Analyst
	Decompose goals	If there are complex goals, they can be decomposed in hierarchies of goals using a decomposition relationship without specific meaning.	Agent Analyst
	Decompose tasks	If there are complex tasks, they can be decomposed. There is only one decomposition relationship without specific meaning for this. Complex decomposition relationships between tasks can be represented using their inputs and outputs and conditions involving them. This decomposition task is closely related to the previous one for goals. Generally, the creation of new tasks should produce as a result the creation of new goals.	Agent Analyst
	Establish dependencies between goals	The achievement of a goal can depend on the achievement of other goal or goals. In this case, the goal is decomposed into other goals. When these dependencies are evident, they can be directly introduced in the diagrams. INGENIAS includes AND, OR and nonspecified decompositions for goals.	Agent Analyst

Refine the Environment Model. The initial version of the environment model is revised and refined within this activity. The tasks in this activity identify and define applications in the environment and resources available to the system. Applications and resources represent entities that are related or part of the system but are not agents. Applications provide some functionalities accessible as methods and trigger

Table 10 Tasks of activity *Refine the Environment Model* of the elaboration phase of INGENIAS

Activity	Task	Description	Involved roles
Refine the Environment Model	Identify applications in the environment	Attending to the methodology definitions, all the software and hardware that interact with the system and cannot be designed as an agent will be considered an application or a resource. <i>Environment applications</i> have a functional interface described in terms of methods and events.	Agent Analyst
	Associate operations to applications	The applications identified in the previous task must be associated with the definition of the operations they provide. The different operations are characterized by their signatures, preconditions, and postconditions. Applications can also trigger events that agents perceive. This perception must also be explicitly represented.	System Analyst
	Identify resources	The environment also contains <i>resources</i> . These are elements available in limited numbers and can be consumable or not consumable.	System Analyst
	Associate applications and resources to agents and groups	The agents and groups of the system are associated to the applications and resources they interact with. In fact, <i>resources</i> are associated with the <i>role</i> or <i>agent</i> that is responsible for their management and <i>environment application</i> with the roles or agents that can access them.	Agent Analyst

events to inform agents. Resources represent entities available in limited numbers. These elements are linked to groups and agents to represent access constraints. The details of these tasks are provided in Table 10.

Show Task Execution Using the Interaction Model. This activity is related mainly with the *design* workflow, because it brings models closer to implementation issues. It adds to the functional decomposition details, depending on the target platform. For instance, a platform like JADE [1], with a strong focus on interactions, requires more details on communications between agents than a basic Java implementation. Regarding interactions, this activity also provides their *GRASIA-specification* diagrams. These diagrams include information on participants, related goals and tasks, and interaction units (i.e., pieces of information) exchanged between participants. Table 11 shows the details of these tasks.

2.2.3 Work Products

The elaboration phase produces as its final result the models that define the whole system, which are used as a previous step to automated code generation. These

Table 11 Tasks of activity *Show task execution using the interaction model* of the construction phase of INGENIAS

Activity	Task	Description	Involved roles
Show task execution using the Interaction Model	Refine functional description	The functional description has to be extended with information required to guide the transition toward the target platform.	Designer
	Generate GRASIA specification	The GRASIA specification diagram describes for an interaction its interaction units, their sorting, the association between transmitter and receiver, the involved tasks, and the goals participants pursue with the interaction.	Designer

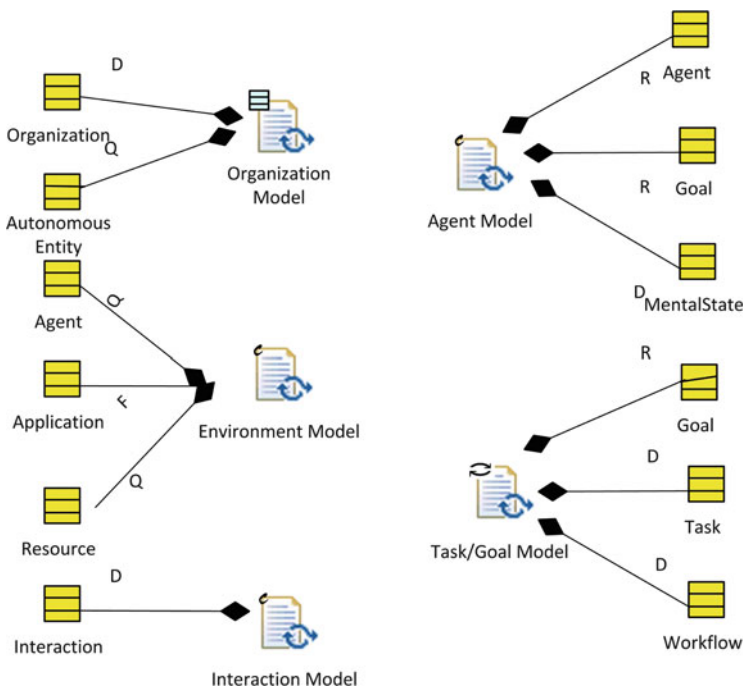


Fig. 14 Structure of the work products of the elaboration phase of INGENIAS

models are the organization model, the agent model, the tasks and goals model, the environment model, and the interaction model, which includes the GRASIA specification diagrams. Figure 14 shows the relationships among these models and the related metamodel elements. For instance, the agent model defines the *agent* and *mental state* concepts and uses, mainly, *autonomous entity*, *role*, and *goal* concepts.

Table 12 Work product kinds of the elaboration phase of INGENIAS

Name	Description	Work Product Kind
Use Case Model	See Sect. 2.1.3	
Environment Model	See Sect. 2.1.3	
Organization Model	See Sect. 2.1.3	
Agent Model	It describes the responsibilities and behavior of each particular agent.	Composite (Structured + Behavioral)
Tasks and Goals Model	This model introduces the inputs and outputs of tasks and the goals their execution satisfies.	Behavioral
Interaction Model	This model shows how agents exchange messages and share knowledge.	Composite (Structured + Behavioral)

Work Product Kinds

Table 12 introduces all the work products of the elaboration phase with their description and kind.

Agent Model

The agent model describes the agent responsibilities, that is, the tasks to be accomplished and the goals to be achieved. The agent actual behavior will be determined by its mental state (i.e., known information), the management of this state, and the decision mechanism.

Tasks and Goals Model

This model introduces the behavioral part of the system definition. Agent behavior will be oriented to achieve its goals by accomplishing several tasks.

There are several examples of this kind of diagram in the case study. Each diagram represents a different way in which the model can be used. For instance, Fig. 25 shows a decomposition of goals, while Fig. 26 shows the consequences of task execution.

Interaction Model

This model defines the interactions between agents or between agents and human beings. At a high level, only information about participants and their role in the communication is required. When needed, additional details can be included, such as interaction protocols, information exchanged, and tasks to process that information. In this last case, structured GRASIA specifications can be used.

Figure 27 shows an example of the GRASIA specification diagram for an interaction. It illustrates the e-mail communications in the CMS case study between members of the organizing committee (OC) and the candidates for the program committee (PC).

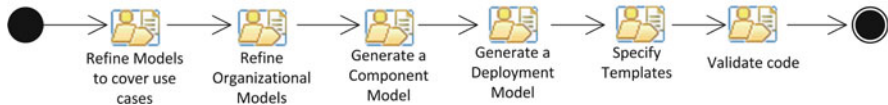


Fig. 15 Activities and workflows of the construction phase

2.3 Process Documentation: Construction Phase

The *construction* phase introduces activities intended to refine the work products of the previous phases in order to generate the final code. Figure 15 shows a general description of it.

The *analysis* workflow accomplishes the following activities in this phase:

- Refine existing models to cover use cases
- Validate code

For the *design* workflow, the following activities must be considered:

- Refine organizational models with social relationships, which includes refining the organization, agent and environment models
- Generate a component model
- Generate a deployment model
- Specify code templates to apply
- Validate code

Although the previous list does not include explicitly model validation, this is an activity that actually appears in all the INGENIAS phases. In the construction phase, the models previously created by a role are changed by other, so suitable validation activities must be performed to cross-check the different artifacts, including diagrams, code, and documentation.

Figures 16 and 17 show an overview of these activities, their tasks, and the related work products. It also includes the roles responsible for the different aspects of the phase.

2.3.1 Process Roles

As for the other phases, the INGENIAS methodology makes no explicit reference to the roles implied in the construction phase. Considering the activities involved in it, this process identifies two roles in this phase: the *designer* and the *programmer*.

The *designer* is responsible for most of the activities in this phase, as INGENIAS adopts an MDD approach. The designer's responsibilities include refining existing models to identify complex algorithmic behaviors. The deployment model describes the nodes where the code is deployed. The designer also generates tests to validate the code using an MDD approach.

The *programmer* translates the specifications of the *designer* to actual code. The code structure is described with the component model. Getting the code can include programming or getting external code to describe complex behaviors, programming new code templates to implement transformations, and programming new scripts when required for deployment.

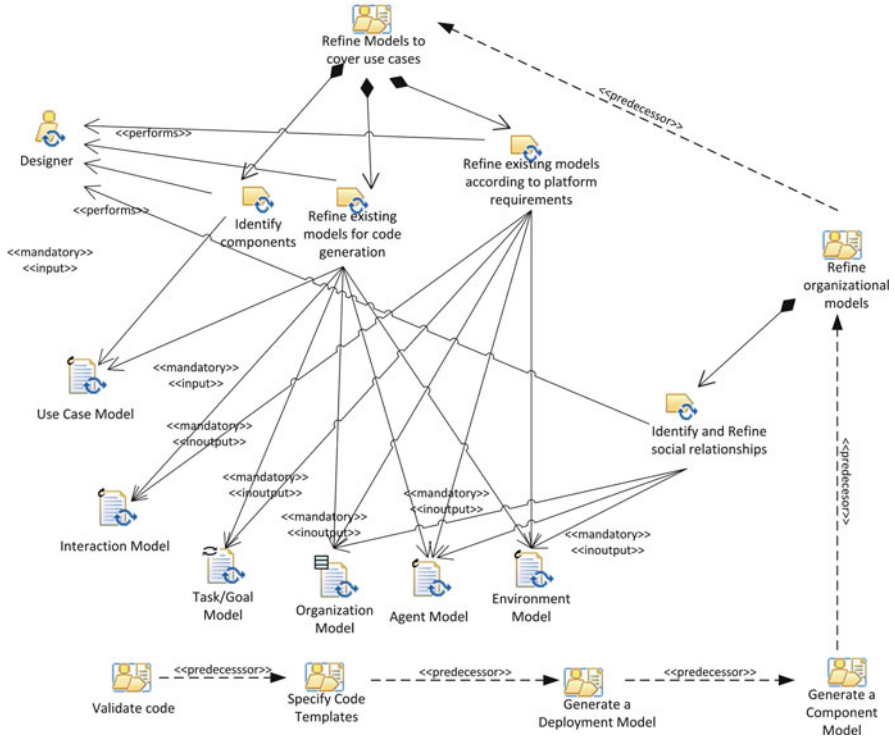


Fig. 16 Detailed tasks of construction activities (part A)

2.3.2 Activity Details

This section details the activities previously outlined for the Construction phase through their tasks.

Refine Existing Models to Cover Use Cases. The refinement of models to cover use cases includes two main tasks in the construction phase. First, the identification of those components whose detailed behavior cannot be specified with the modeling language. This behavior will be later specified using programming code. Second, including the model-specific information used by MDD tools for code generation. These tasks are explained in Table 13.

Refine Organizational Models with Social Relationships. The generation of code implies a review of the relationships between components identified in previous phases. These relationships influence, among others, visibility and access to code.

Generate a Component Model. The component model focuses on the lowest-level issues of a system. This activity specifies the organization of the code, for instance,

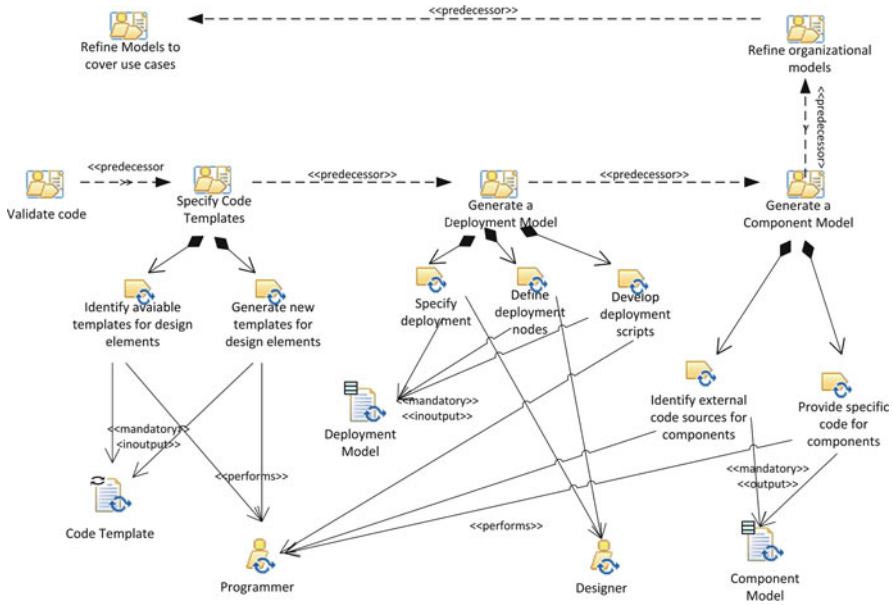


Fig. 17 Detailed tasks of construction activities (part B). Tasks in the lower row use as input the work products in the upper row (showed in part A), though this is omitted in the diagram

in packages and components, it identifies components with elements whose code cannot be automatically generated, and links manually generated code to those elements in the specifications that needed it. This activity includes several tasks that are shown in Fig. 18 and further explained in Table 14.

Generate a Deployment Model. The specification of a system includes the features of the computational devices where it runs. It also indicates the nodes where the different components will be deployed and the characteristics of this deployment. Table 15 describes the tasks of this activity.

Specify Code Templates to Apply. The INGENIAS MDD relies on code templates to generate code for the target platforms from specifications. These templates are coded for the first project in a given target platform, and reused, checked, and refined in later developments for the same target platform. Table 16 describes the related tasks.

Validate Code. The code, whether automatically generated or manually developed, must be checked against the models and for the absence of bugs. In an MDD approach, this implies the generation of the running and test code. The component and deployment models include information on the test code through the *testing package* and *test* elements, respectively. *Tests* specify the code to be run for a given

Table 13 Tasks of activity *Refine existing models to cover use cases* of the construction phase of INGENIAS

Activity	Task	Description	Involved roles
Refine existing models to cover use cases	Identify components with algorithmic behavior	All the <i>tasks</i> and <i>applications</i> whose behavior cannot be expressed using the modeling language are included. For these, a specification of the algorithm must be provided. INGENIAS does not consider any specific language for this purpose but recommends some level of formalism, for instance, with pseudocode, logics or some programming language.	Designer
	Refine existing models with information required for code generation with respect to tools	The tools used for code generation have specific requirements regarding this task. For instance, the IDK [7] requires a specification of the management of mental entities by agents. This task adds this information to available models.	Designer
	Refine existing models according to the requirements of target platforms	Part of the information of models can depend on the target platform. For instance, there must be a specification of the target programming language to be used. Note that these requirements are not only about new information, but also the level of detail of the information already available. For instance, a target platform with a focus on agent interactions would probably require a higher level of detail in the interaction model than other platforms.	Designer

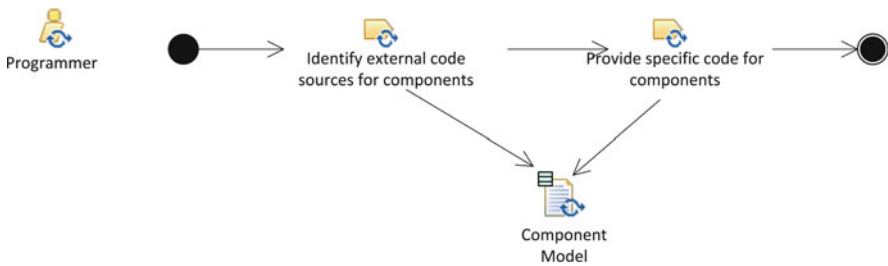


Fig. 18 *Generate a component model* in the construction phase of INGENIAS

test, and *testing packages* the configuration of the system (e.g., agents, nodes, and initial parameters) and the *tests* to run. The development of these elements is similar to that of the rest of the components and packages in the already mentioned models.

2.3.3 Work Products

The construction phase produces as a result five basic work products: a component model, a deployment model, code templates, external and generated code, and platform-oriented versions of models from previous phases. The relationships among the models and the metamodel elements are shown in Fig. 19.

Table 14 Tasks of activity *Generate a Component Model* of the construction phase of INGENIAS

Activity	Task	Description	Involved roles
Generate a Component Model	Identify external code sources for components	The algorithm specifications can be solved linking them to externally available implementations. This is done in the component model using <i>INGENIASComponents</i> . The MDD tools insert this external code in the generated code following the applied code templates for the target platform.	Programmer
	Provide specific code for components	Some algorithms are manually coded and inserted in the specifications as linked to particular elements, e.g., to a <i>Task</i> or <i>Application</i> method. This code is added in the <i>INGENIASCodeComponents</i> of the component model. The MDD tools use this code in a similar way to the external code in task <i>Identify external code sources for components</i> .	Programmer

Table 15 Tasks of activity *Generate a Deployment Model* of the construction phase of INGENIAS

Activity	Task	Description	Involved roles
Generate a Deployment Model	Define deployment nodes	Identify the features of the computational resources for the deployment. This information appears in deployment diagrams as the description of <i>TestingPackage</i> , <i>SimulationPackage</i> , or <i>Deployment Package</i> elements. INGENIAS only supports the textual description of further information on these resources.	Designer
	Specify deployment	The deployment includes the description in Deployment diagrams of deployment units that specify the type, number, and initialization of components.	Designer
	Develop deployment scripts	The deployment of the system may need the setup of a given execution environment in the target nodes. For this purpose, specific configuration scripts can be developed. These appear in deployment diagrams as <i>INGENIASComponents</i> or <i>Applications</i> .	Programmer

The component model defines the *Component* metamodel element to group code components and describes its code with *INGENIASCodeComponents*. These elements can appear linked to *application*, *goal*, *task*, and *test* elements to indicate the organization of their code. The deployment model defines different types of *deployment units*, *packages*, *INGENIASComponents*, and *applications*. The model does not include any relationship between these elements. Deployment units mainly

Table 16 Tasks of activity *Specify Code Templates to apply* of the construction phase of INGENIAS

Activity	Task	Description	Involved roles
Specify Code Templates to apply	Identify available templates for design elements	Code templates define the transformations for code generation. These templates are codes for a given target platform annotated with tags that correspond to modeling primitives. The code generation process injects information from the specifications in the templates following the tags. The instantiated templates are the actual code. These templates can be largely reused among projects with the same target platform. This task examines and identifies available templates for reuse in the current project.	Programmer
	Generate new templates for design elements	When there are no available templates for a given transformation, new ones must be coded. The programmer starts from the specification of the designer and completes the mapping from modeling structures to constructions of the target platform. Then, the programmer manually codes a complete implementation of the new mapping. When it works, the programmer replaces those identifiers in the code that correspond to elements from the specifications with the corresponding tags from the template markup language. The result of this replacement is the template.	Programmer

specify types of agents to create during the system startup. Packages also include code and parameters for the initialization. The other elements are introduced to support linking components and packages to nodes (e.g., for compiled files or scripts) or limit the visibility of some of their components (e.g., methods of *applications*).

Work Product Kinds

Table 17 introduces all the work products of the construction phase with their description and kind.

Code Template

A code template is a fragment of code with marks that corresponds to modeling elements. They are used for code generation. Their actual language depends on the target platform.

The case study does not include examples of templates. Nevertheless, the interested reader can find them in the IDK distribution [7], for instance, for JADE [1].

Fig. 19 Structure of the work products of the construction phase in INGENIAS

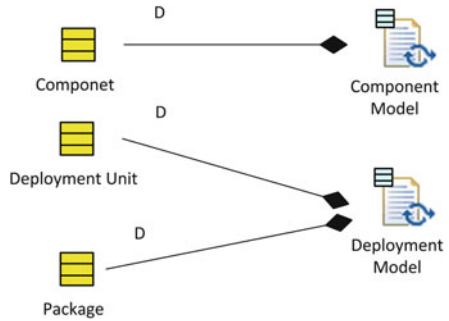


Table 17 Work product kinds of the construction phase of INGENIAS

Name	Description	Work Product Kind
Use Case Model	See Sect. 2.1.3	
Environment Model	See Sect. 2.1.3	
Organization Model	See Sect. 2.1.3	
Agent Model	See Sect. 2.2.3	
Tasks and Goals Model	See Sect. 2.2.3	
Interaction Model	See Sect. 2.2.3	
Code Template	Fragment of marked code for a target platform. It corresponds to a given modeling structure that can appear in specifications.	Behavioral
Component Model	This model introduces organization of the code, whether automatically generated or manually developed, that implements the system specifications.	Structural
Deployment Model	This model introduces the computational devices related to the system and their features. It also maps the code components and the nodes where they run.	Structural

Component Model

The component model includes different primitives to organize the system code. It also supports the linking of modeling elements to the actual code used to implement parts of them.

A model introduces special icons to reflect each of the previous observable entities, which can be identified in Fig. 22. Figure 28a shows a component diagram in the case study. It indicates dependencies between the code of tasks and a code component.

Deployment Model

The deployment model describes the computational devices where the system runs and what components run in each. Examples of these diagrams can be found in the IDK distribution [7] and the INGENIAS website [8].

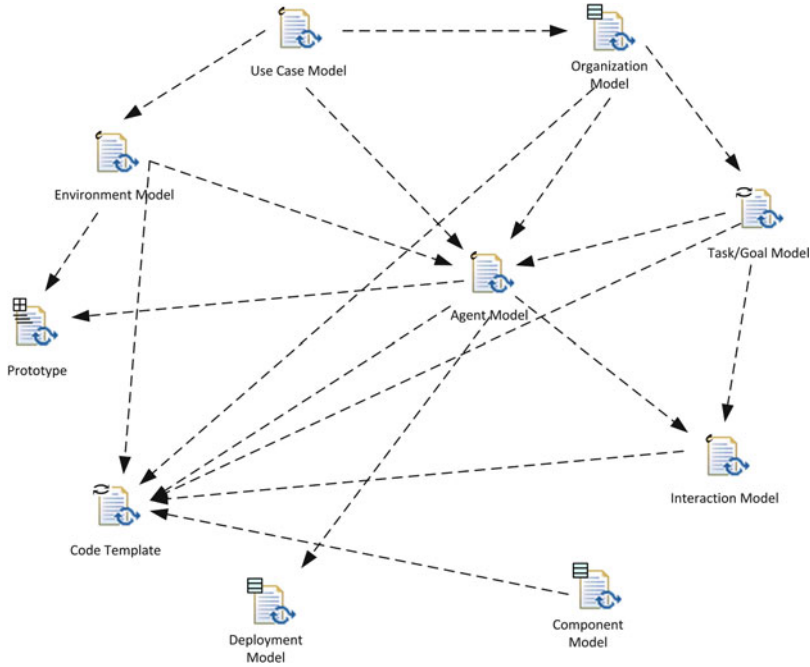


Fig. 20 Dependencies among INGENIAS work products

3 Work Product Dependencies

Figure 20 shows an overview of the INGENIAS work products, as well as their dependencies. As shown in the figure, the agent model depends on the organization and environment models, and the interaction model has dependencies with the agent and tasks and goals models among others.

4 Case Study: Conference Management System

The development of an INGENIAS project following the adaptation of the UDP [2] and RUP [16] proposed in this chapter is organized around three phases: *inception*, *elaboration*, and *construction*. As INGENIAS follows an MDD approach [3], their activities mainly focus on the development and refinement of models. Only in *construction*, engineers deal with artifacts such as code templates, source files, and scripts, but in a limited way. Usually, all these phases include work in all the process disciplines, e.g., requirements, analysis, design, and implementation, although with different devoted effort to each. The presentation of this case study on a conference management system (CMS) follows this phase organization, and for each phase the previous list of disciplines.

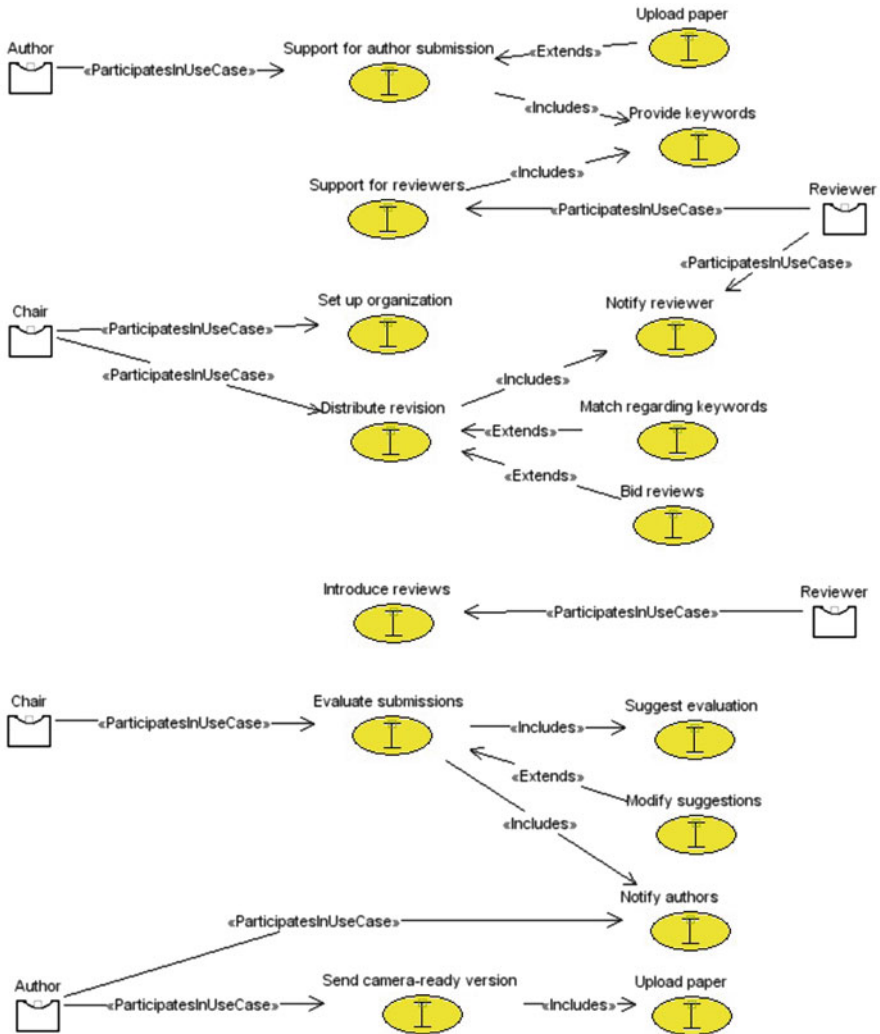


Fig. 21 Use cases for the CMS

Inception includes the activities for requirements elicitation that are the starting point of this development. This process is case-driven, so it starts with the *Create the Use Case Model* activity. The development of this model identifies the main groups of functionality as perceived by the user. Figure 21 shows the related use cases according to the description of the problem.

The use cases provide the initial information to work on the other models. Due to space reasons, this case study only considers the system interacting with external entities. The development process follows the *Obtain the Environment Model* activity, which includes several tasks. *Identify applications* identifies *environment*



Fig. 22 Initial environment diagram of the CMS

applications that interact with the system but are not agents. Figure 22 shows the result of this task. There is a *mail manager* to send notifications; a *database* to store information such as submissions, author profiles, and the program committee, some *keyword utilities* to manipulate lists of keyword and decide on their similarity; and a *submission manager* to upload and download files.

Two additional tasks are used to refine the environment model, *Associate operations to applications* and *Define agent perception*. The first one adds to the model (not shown in the figure) the methods of the applications. For instance, the *mail manager* has methods to send notifications to authors such as “paper successfully uploaded” or “submission review.” The second task indicates the kind of information that agents perceive from the applications. Here, only the *mail manager* can generate some events, for instance, when it receives a mail for the organization. The other applications are just invoked via methods.

The third activity in inception is *Create the Organization Model*, whose main work product is an instance of the organization model. In this case, we apply this activity to capture information about the organization of the conference. The activity includes three tasks. *Identify groups* looks for groups of agent or role that share features such as common goals or access to resources. *Generate group members* establishes the agents, roles, resources, and applications that belong to each group. Finally, *Identify goals* determines the group goals that justify the collaboration between its members.

There are several groups in the CMS case study (see Fig. 23). The *OC* (organizing committee) includes the *chairs*, the *PC* (program committee) for the *reviewers*, and the *contributors* with the *authors*. The *OC* has as the main goal *Run a successful conference*, the *PC* *Identified quality papers* and *Evaluated papers*, and *Contributors Submitted quality papers* and *Got papers admitted*. There are also differences in the access to applications. While all the groups have access to the *submission manager*, only the *OC* can use the *mail manager*. The *database* and *keyword utilities* will be used by internal components of the system.

The elaboration phase starts with a *Refine the Use Case Model* activity, which includes several tasks. *Identify interaction participants* considers the use cases where several actors participate. These use cases are candidates to be specified by means of interaction diagrams. In this case, the use case diagram (see Fig. 21) only includes the *Notify authors* use case with several participants. The *Chair* is its initiator and the *Author* the collaborator. In fact, all the interactions in the CMS happen through applications such as the *mail manager* and the *database* (see Fig. 22). Therefore, there is no direct communication between participants. The other two tasks of the activity Refine the Use Case Model, i.e., *Define the nature of*

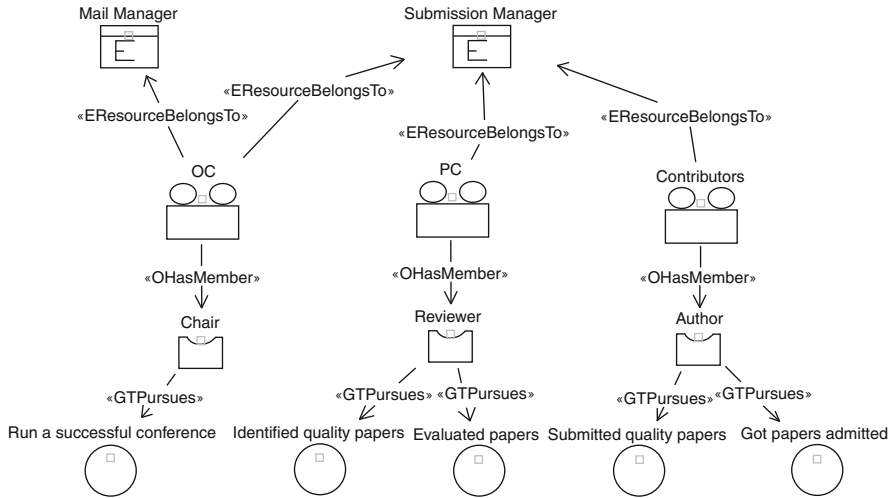


Fig. 23 Organization diagram with the main user groups of the CMS and their access to applications

interaction and *Identify the goals pursued*, identify for the use case of the example that the interaction is of kind *cooperation* and its goal is *Provided review results*.

The next activity of elaboration is *Refine the Organization Model*, with two tasks: *Identify the groups in the organization* and *Identify workflows*. The first task does not introduce new elements in the organization model in this case (see Fig. 23). The second task identifies several workflows that Fig. 24 shows in an organization diagram. These workflows are decomposed into two main groups, those related to the *Set up conference* workflow and those with *Perform evaluation*. The first one includes the workflows to organize the different committees, i.e., *Organize OC* and *Organize PC*, and the workflow *Get submissions* with the activities intended to promote paper submissions, such as mailing the call for papers, setting up a website, or directly contacting well-known researchers in the area. The second one considers the workflows that organize the review process, with the *Assign reviews* and *Process reviews* workflows.

The next activity is *Create the Agent Model*. Its first tasks are *Identify agents* and *Associate roles to agents*. We skip these tasks because this case study focuses on eliciting information about the general behavior of the CMS, and not on how specific tasks are done. Only when addressing this last issue, agents are mandatory. Therefore, roles are enough to model the current information as mentioned before in this section. The other task, *Identify requirements of agent mental states*, is related to the identification of suitable processor and managers for the mental state of agents. In this case, we adopt the standard INGENIAS model implemented in the IAF [6].

The workflows and goals previously identified are the basis to start the functional decomposition in tasks. This is achieved in the *Create the Tasks and Goals Model* activity. As for other components of the process, there is no clear ordering of the

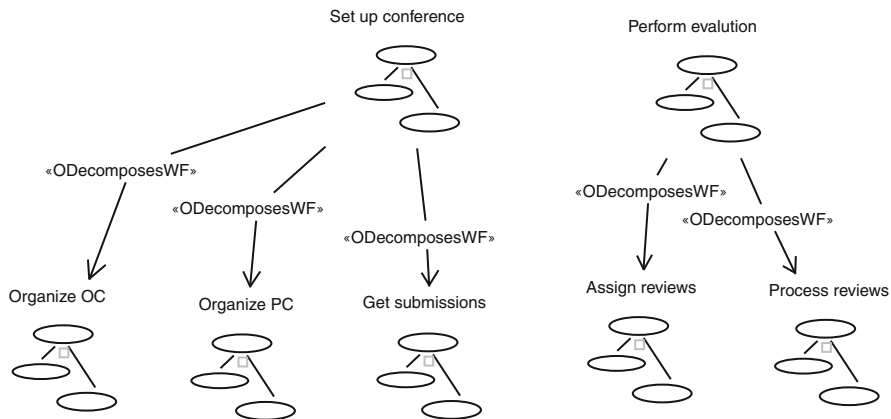


Fig. 24 Main workflows of the CMS

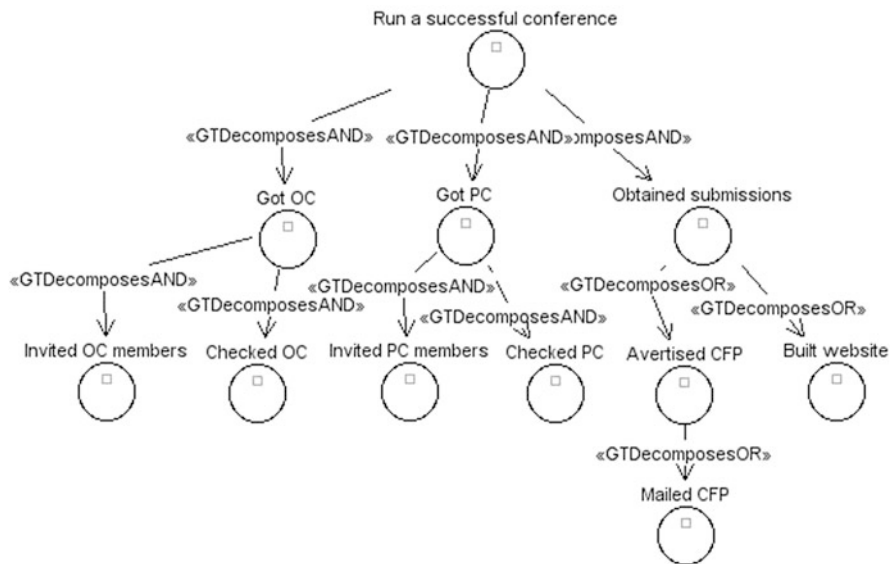


Fig. 25 Initial decomposition of the *Run a successful conference* goal

tasks of this activity. For instance, a task decomposition leads to new goals that in turn require new tasks. Here, we perform together the tasks *Identify tasks and goals* and *Decompose goals* for the high-level goal *Run a successful conference*. The result appears in Fig. 25. Note that the decomposition is aligned with the workflows in Fig. 24.

Goals can be achieved through the execution of INGENIAS tasks or thanks to external events. Here, goals are the result of the execution of tasks by roles. The identification of these INGENIAS tasks and their assignment to roles are

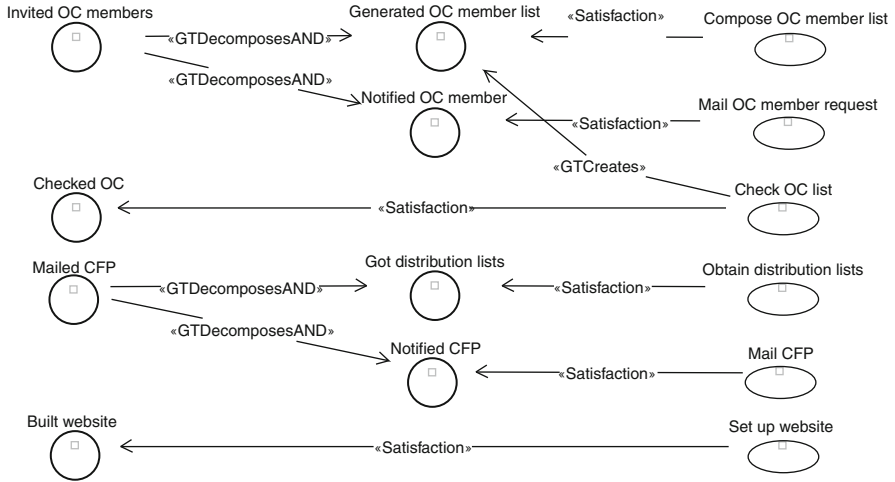


Fig. 26 Initial assignment of tasks related to the *Run a successful conference* goal

made in the *Associate tasks and goals* and *Decompose tasks* tasks. The result for the goals in Fig. 25 appears in Fig. 26. As mentioned before, the process tasks to identify and specify this information are interleaved. The new diagram introduces additional decompositions for goals *Invited OC members* and *Mailed CFP*. For most assignments of tasks to goals in the diagram, the link is a *GTSatisfies* (i.e., *Satisfaction*) relationship. However, the *Check OC list* task is also able to create (i.e., *GTCreates* relationship) the *Generate OC member list* goal when the number of members in the OC is insufficient for the conference. The treatment of PC-related goals is similar to that for OC-related goals.

The next activity is *Refine the Environment Model*. Its activities identify resources, applications, their methods, and assign them to agents, groups, and roles. In this case study, these tasks do not introduce further information with respect to Fig. 22.

The last activity of the elaboration phase is *Show tasks execution using the Interaction Model*, with two tasks: *Refine functional description* and *Generate GRASIA specification*. Regarding the first task, we do not introduce further refinements at this step. For the second task, we consider the specification of the interaction to invite OC members, which fulfills the *Notified OC members* goal and the *Mail OC member request* task starts (see Fig. 26). Figure 27 shows the resulting interaction diagram. The diagram introduces several new elements: the *Candidate PC member* role, tasks to process the request and answer to the invitation, and several interaction units that represent the flow of information between roles.

The last phase considered in this INGENIAS process is construction. In this case study, we skip its first two activities *Refine existing models to cover use cases* and *Refine organizational models with social relationships*, as we do not introduce new information for them and similar tasks have been already applied in previous steps.

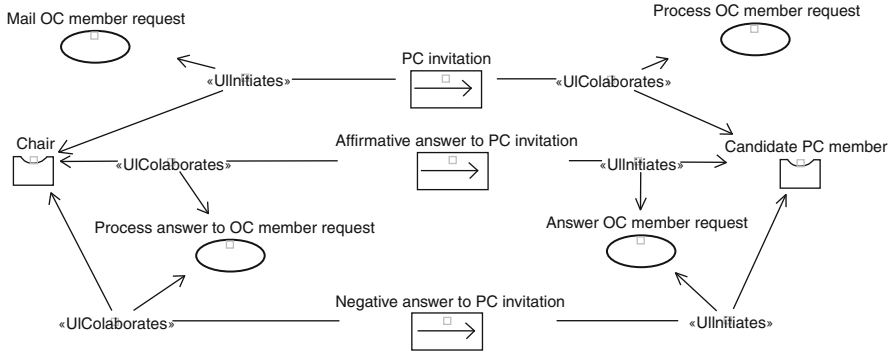


Fig. 27 Interaction to invite OC members

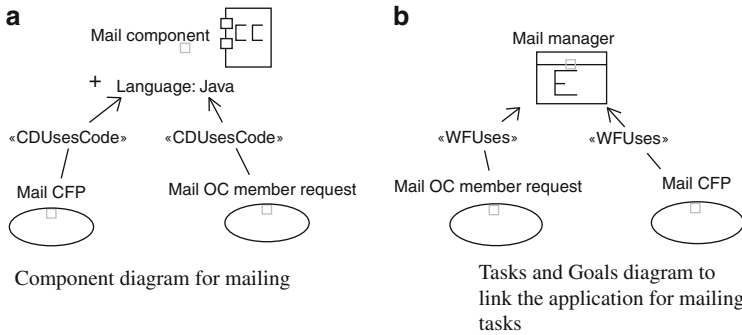


Fig. 28 Diagrams for the mailing tasks, their code, and external applications. **a** Component diagram for mailing. **b** Tasks and Goals diagram to link the application for mailing tasks

The third activity of Construction is *Generate a Component Model*. As this case study focuses on requirements elicitation, most of CMS tasks only have a textual description. Nevertheless, code can be considered to access the external applications of Fig. 22. Two diagrams are introduced for this specification: a component diagram (see Fig. 28a) and a new tasks and goals diagram (see Fig. 28b). The first one indicates the INGENIASCodeComponent that contains the code that the mailing tasks use to manage the *mail manager* external application. The second one links the mailing tasks to the application that they use to send and receive mails.

The remaining activities of the construction are *Generate a Deployment Model*, *Specify Code Templates to apply*, and *Validate code*. These activities are beyond the scope of this case study, which is focused on requirements. It deserves to be mentioned that the facilities included in the IDK [7], and in particular the IAF [6], would allow generating testing code for these specifications. This prototype could be used for a preliminary validation with users.

References

1. Bellifemine, F., Poggi, A., Rimassa, G.: JADE: a FIPA2000 compliant agent development environment. In: Fifth International Conference on Autonomous Agents (AGENTS 2001), pp.16–217. ACM, New York (2001)
2. Booch, G., Jacobson, I., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
3. France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: 2007 Future of Software Engineering (FOSE 2007), pp. 37–54. IEEE Computer Society, Washington, DC (2007)
4. García-Magariño, I., Gómez-Rodríguez, A., Gómez-Sanz, J., González-Moreno, J.C.: INGENIAS-SCRUM development process for multi-agent development. *Adv. Soft Comput.* **50**, 108–117 (2009)
5. Gómez-Sanz, J.: Modelado de sistemas multi-agente. Ph.D. thesis, Facultad de Informática, Universidad Complutense de Madrid, Madrid (2002)
6. Gómez-Sanz, J.: INGENIAS Agent Framework. Development Guide V. 1.0. Tech. Rep., Universidad Complutense de Madrid, Madrid (2008)
7. Gómez-Sanz, J.J., Pavón, J., Fuentes-Fernández, R., García-Magariño, I., Rodríguez-Fernández, C.: INGENIAS Development Kit, V. 2.8. Tech. Rep., Universidad Complutense de Madrid, Madrid (2008)
8. Grupo de Investigación en Agentes Software: Ingeniería y Aplicaciones. INGENIAS Section. <http://grasia.fdi.ucm.es/main/?q=es/node/61> (2010)
9. Intelligent System Lab at British Telecom: Zeus Agent Toolkit, V. 2.0. Tech. Rep., Intelligent System Lab, British Telecom (2006)
10. Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Prasad, M., Raja, A., Vincent, R., Xuan, P., Zhang, X.Q.: Evolution of the GPGP/TÆMS Domain-Independent Coordination Framework. *Auton. Agents Multi-Agent Syst.* **9**(1), 87–143 (2004)
11. Newell, A.: The knowledge level. *Artif. Intell.* **18**(1), 87–127 (1982)
12. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.3. Tech. Rep., OMG (2010)
13. Pavón, J., Gómez-Sanz, J.: Agent Oriented Software Engineering with INGENIAS. *Lect. Notes Comput. Sci.* **2691**, 394–403 (2003)
14. Pavón, J., Gómez-Sanz, J.J., Fuentes-Fernández, R.: The INGENIAS methodology and tools. In: Henderson-Sellers, B., Giorgini, P. (eds) *Agent-Oriented Methodologies*, Chapter IX, pp. 236–276. Idea Group Publishing, Hershey (2005)
15. Rao, A.S., Georgeff, M.P.: BDI agents: from theory to practice. In: First International Conference on Multi-Agent Systems (ICMAS 1995), pp. 312–319. AAAI, Menlo Park (1995)
16. Rational Software: Rational Unified Process: White Paper (1998)
17. Schwaber, K., Beedle, M.: *Agile Software Development with Scrum*. Prentice Hall, Englewood Cliffs (2001)

The SODA Methodology: Meta-model and Process Documentation

Ambra Molesini and Andrea Omicini

Abstract

The SODA methodology deals with MAS analysis and design, and focuses on critical issues such as agent coordination and MAS-environment interaction. After its first formulation, in order to further meet the needs of complex MAS engineering, SODA was extended to embody both the *layering principle* and the Agents & Artifacts (A&A) meta-model. As a result, both the SODA meta-model and the SODA process were re-defined, also to include two new phases—Requirement Analysis and Architectural Design. This chapter is then devoted to the documentation of the complete SODA process according to the FIPA standard.

1 Introduction

SODA (Societies in Open and Distributed Agent spaces) [13] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts (A&A) meta-model [10], and introduces a *layering principle* as an effective tool for scaling with system complexity, applied throughout the analysis and design process [2, 3, 9]. Since its first version [9], SODA is not concerned with *intra-agent* issues: designing a multi-agent system (MAS) with SODA amounts at defining agents in terms of their required observable behaviour as well as their role in the MAS. Then, whichever methodology one may choose to define the structure

A. Molesini (✉)

DISI, Alma Mater Studiorum – Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy

e-mail: ambra.molesini@unibo.it

A. Omicini

DISI, Alma Mater Studiorum – Università di Bologna, Via Sacchi 3, 47521 Cesena, Italy

e-mail: andrea.omicini@unibo.it

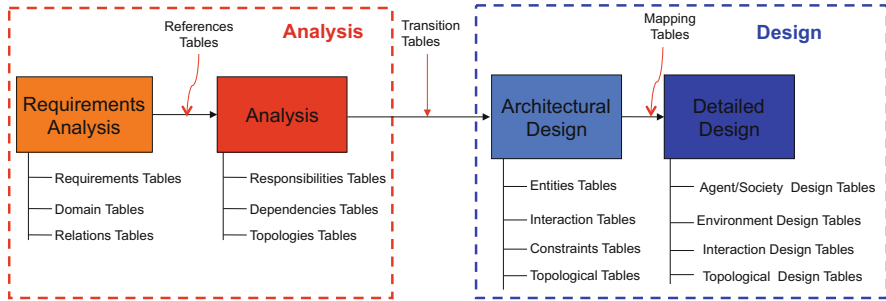


Fig. 1 An overview of the SODA process

and inner functionality of individual agents, it could be used in conjunction with SODA. Instead, SODA focus on *inter-agent* issues, like the engineering of agent societies and MAS environment.

When designing a new system in SODA, three things are to be understood: which activities have to be performed, which functions are available and required, and how activities and functions relate to each other. Accordingly, SODA abstractions are logically divided into three categories: (1) the abstractions for modelling/designing the system's active part (task, role, agent, etc.); (2) those for the reactive part (function, resource, artifact, etc.); and (3) those for interaction and organisational rules (relation, dependency, interaction, rule, etc.).

The SODA *process* is organised in two phases, each structured in two sub-phases: the *Analysis phase*, including the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. Each sub-phase models the system through a subset of the SODA abstractions: in particular, each subset always includes at least one abstraction for each of the above categories—that is, at least one abstraction for the system's active part, one for the reactive part, one for interaction and organisational rules.

Figure 1 overviews the methodology by describing each step in terms of a set of relational tables. In the remainder of this chapter, the SODA process is described first as a whole process then through its four steps, following the FIPA standard [1].

Useful references about the SODA methodology and process are the following:

- A. Omicini. *SODA: Societies and Infrastructures in the Analysis and Design of Agent-based Systems* [9].
- A. Molesini, A. Omicini, A. Ricci, E. Denti. *Zooming Multi-Agent Systems* [3].
- A. Molesini, A. Omicini, E. Denti, A. Ricci. *SODA: A Roadmap to Artefacts* [2].
- A. Molesini, E. Denti, A. Omicini. *Agent-based Conference Management: A Case Study in SODA* [6].
- A. Molesini, E. Nardini, E. Denti, A. Omicini. *Advancing Object-Oriented Standards Toward Agent-Oriented Methodologies: SPEM 2.0 on SODA* [4].
- A. Molesini, E. Nardini, E. Denti, A. Omicini. *Situated Process Engineering for Integrating Processes from Methodologies to Infrastructures* [5].

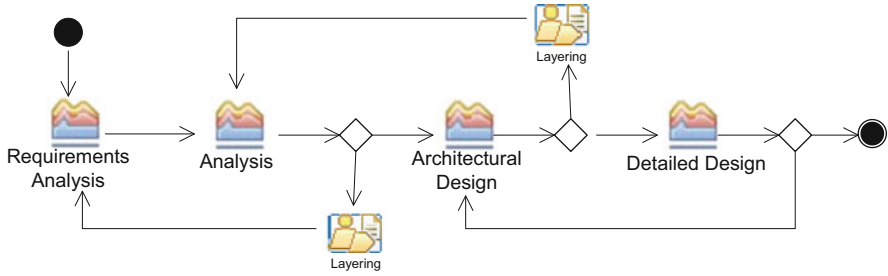


Fig. 2 Phases of the SODA process

- A. Molesini, A. Omicini. *Documenting SODA: An Evaluation of the Process Documentation Template* [7].

1.1 Process Life Cycle

SODA includes two phases, each structured in two sub-phases: the *Analysis* phase, which includes the *Requirements Analysis* and the *Analysis* steps, and the *Design* phase, which includes the *Architectural Design* and the *Detailed Design* steps. SODA phases and steps are arranged according to an iterative process model (see Fig. 2):

- Requirements Analysis* covers all the phases related to actor identification, requirements elicitation and analysis, and analysis of the existing environment.
- Analysis* investigates all the aspects related to the problem domain trying to understand the tasks satisfying the requirements, their connected functions, the environment topology and all the dependencies among these entities.
- Architectural Design* defines a set of admissible architectures for the final system.
- Detailed Design* determines the best system architecture and designs the environment and the system interactions.

Each step in SODA produces several sets of relational tables, each describing a specific MAS Meta-model Element (MMMElement) and its relationships with other MMMElements. The details of each step will be discussed in the following section.

1.2 Meta-model

The meta-model adopted by SODA is represented in Fig. 3, where SODA abstract entities are depicted along with their mutual relations, and distributed according to the four SODA steps: Requirements Analysis, Analysis, Architectural Design and Detailed Design.

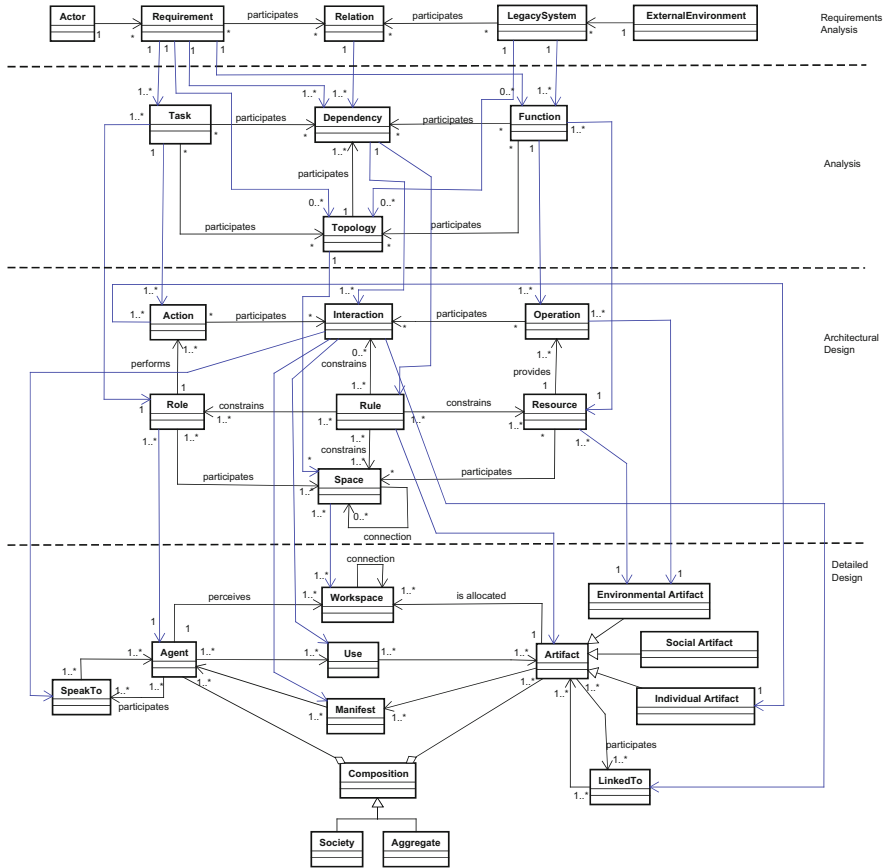


Fig. 3 The SODA meta-model

1.2.1 Requirements Analysis

Several abstract entities are introduced for requirement modelling: in particular, *Requirement* and *Actor* are used for modelling the customers' requirements and the requirement sources, respectively, while the notion of *External Environment* is used as a container of the *Legacy Systems* that represent the legacy resources of the environment. The relationships between requirements and legacy systems are then modelled in terms of *Relation* entities.

1.2.2 Analysis

The Analysis expresses the abstract requirement representation in terms of more concrete entities such as *Tasks* and *Functions*. Tasks are activities requiring one or more competences, while functions are reactive activities aimed at supporting tasks. The relations highlighted in the previous step are now the starting point for the definition of *Dependencies*—interactions, constraints, etc.—among the

abstract entities. The structure of the environment is also modelled in terms of *Topologies*, that is, topological constraints over the environment. Topologies are often derived from functions, but can also constrain/affect task achievement.

1.2.3 Architectural Design

The main goal of this stage is to assign responsibilities to achieve tasks to *Roles*, and responsibilities to provide functions to *Resources*. To this end, roles should be able to perform *Actions*, and resources should be able to execute *Operations* providing one or more functions. The dependencies identified in the previous phase become here *Interactions* and *Rules*. *Interactions* represent the acts of the interaction among roles, among resources and between roles and resources; rules, instead, enable and bound the entities' behaviour. Finally, the topology constraints lead to the definition of *Spaces*, that is, conceptual places structuring the environment.

1.2.4 Detailed Design

The active and passive parts are expressed in the Detailed Design in terms of individual entities (*Agents* and *Artifacts*) as well as of composite entities, such *Societies* and *Aggregates*. Agents are intended here as autonomous entities able to play several roles, whereas the resources identified in the previous step are now mapped onto suitable artifacts.

Artifacts have “types” according to the following taxonomy [11]:

- An *individual artifact* handles the interaction of a single agent within a MAS and essentially works as a mediator between the agent and the MAS itself. Since they can be used to shape admissible interactions of individual agents in MAS, individual artifacts play an essential role in engineering both organisational and security concerns in MAS.
- An *environmental artifact* brings an external resource within a MAS by mediating agent actions and perceptions over resources. As such, environmental artifacts play an essential role in enabling, disciplining and governing the interaction between agents and MAS environment.
- A *social artifact* rules social interactions within a MAS by mediating interactions between individual, environmental and possibly other social artifacts. Social artifacts in SODA play the role of the coordination artifacts that embody the rules around which societies of agents can be built.

Interactions between agents and artifacts in SODA take the form of *Use* (agent to artifact), *Manifest* (artifact to agent), *SpeakTo* (agent to agent) and *LinkedTo* (artifact to artifact).

In SODA, a group of individual entities can be abstracted away as a single composite entity. In particular, a group of interacting agents and artifacts can be seen as a SODA *Society* when its overall behaviour is essentially an autonomous, proactive one; it can be seen as a SODA *Aggregate*, instead, when its overall behaviour is essentially a functional, reactive one. Finally, SODA *Workspaces* take the form of an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces, and agents can dynamically enter (join) or exit workspaces (Table 1).

Table 1 The SODA entities definitions

Concepts	Definition	Step
Actor	<i>System's stakeholder</i>	Requirements Analysis
Requirement	<i>Service that the stakeholder requires from a system and the constraints under which it operates and is developed</i>	Requirements Analysis
Legacy System	<i>Legacy resources</i>	Requirements Analysis
External Environment	<i>Legacy environment in which the new system will execute</i>	Requirements Analysis
Relation	<i>A tie among the entities of the Requirements Analysis</i>	Requirements Analysis
Task	<i>An activity aimed at the satisfaction of a specific requirement</i>	Analysis
Function	<i>Function or service aimed at supporting task accomplishment</i>	Analysis
Topology	<i>Topological constraints over the environment. Often derived from legacy systems and requirements, however also functions and tasks could induce some topological constraints</i>	Analysis
Dependency	<i>Any kind of dependency relationships among abstract entities, as a conceptual premise to any sort of interaction</i>	Analysis
Role	<i>An entity responsible to accomplish some tasks</i>	Architectural Design
Action	<i>An activity that changes the environment in order to meet roles design objectives</i>	Architectural Design
Resource	<i>Entity that provides functions</i>	Architectural Design
Operation	<i>A resource access point in order to achieve a function</i>	Architectural Design
Space	<i>Conceptual places structuring the environment</i>	Architectural Design
Rule	<i>Any prescription over roles, resources, interactions, and spaces</i>	Architectural Design
Interaction	<i>Any interaction among roles and resources</i>	Architectural Design
Agent	<i>Pro-active components of the systems, encapsulating the autonomous execution of some kind of activities inside an environment</i>	Detailed Design
Artifact	<i>Passive components of the systems such as resources and media that are intentionally constructed, shared, manipulated and used by agents to support their activities, either cooperatively or competitively</i>	Detailed Design
Individual Artifact	<i>Mediator between an individual agent and the MAS</i>	Detailed Design

(continued)

Table 1 (continued)

Concepts	Definition	Step
Social Artifact	<i>Mediator of social interactions within a MAS</i>	Detailed Design
Environmental Artifact	<i>Mediator of the interaction between MAS and the external environment</i>	Detailed Design
Composition	<i>A collection of agents and artifacts working together as an ensemble</i>	Detailed Design
Society	<i>A composition whose overall behaviour is essentially an autonomous, proactive one</i>	Detailed Design
Aggregate	<i>A composition whose overall behaviour is essentially a functional, reactive one</i>	Detailed Design
Workspace	<i>Conceptual containers of agents and artifacts, providing a notion of locality for MAS</i>	Detailed Design
Use	<i>The act of interaction between agent and artifact: agent uses artifact</i>	Detailed Design
Manifest	<i>The act of interaction between artifact and agent: artifact manifests itself to agent</i>	Detailed Design
SpeakTo	<i>The act of interaction among agents: agent speaks to another agent</i>	Detailed Design
LinkedTo	<i>The act of interaction among artifact: artifact is linked to another artifact</i>	Detailed Design

1.3 Guidelines and Techniques

SODA exploits a technique called *Layering* that can be applied to the overall process before the Detailed Design step. In SODA, during the Analysis phase and the Architectural Design step, the system is described in principle by all the layers defined, and could then be modelled by a number of different—although related—*design views*. This of course does not hold for the Detailed Design step since the developer should be provided with a single system representation among all the potentially admissible ones based on the Architectural Design layers.

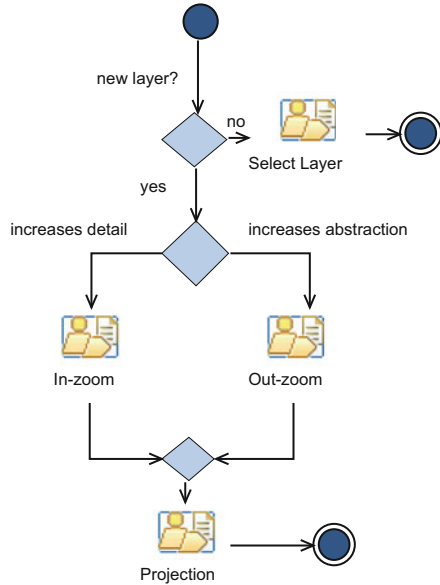
Accordingly, the next section presents the SODA Layering technique.

1.3.1 Layering

Complexity is inherent in real-life systems. While *modelling* complex systems and understanding their behaviour and dynamics is the most relevant concern in many areas, such as economics, biology or social sciences, also the complexity of *construction* becomes an interesting challenge in artificial systems like software ones. An integral part of a system development methodology must therefore be devoted to controlling and managing complexity.

To this end, SODA introduces *Layering*, a conceptual tool to deal with the complexity of system representation. Using Layering, a system in SODA can be represented as composed by different *layers of abstraction*, with a *layering operation* to conceptually move between them.

Layering can be represented as a *capability pattern* [8]—that is, a reusable portion of the process, as shown in Fig. 4, where the layering process is detailed. In particular, the layering process has two activities: (1) the selection of a specific layer

Fig. 4 The layering process

for refining/completing the abstractions models in the methodology process (Select Layer activity), and (2) the creation of a new layer in the system by *in-zooming*—that is, increasing the system detail—or *out-zooming*—that is, increasing the system abstraction—activities. In the last case, the layering process ends with the projection activity where the abstractions are projected “as they are” from one layer to another so as to maintain the consistency in each layer.

In general, when working with SODA, the reference layer, called *core layer*, is labelled with C , and is by definition *complete*—that is, it contains all the entities required to fully describe a given abstract layer. Any other layer—labelled with either $C + i$, for more detailed layers, where i is the number of in-zoom steps from the C layer, or $C - i$, for more abstract layers, where i is the number of out-zoom steps from the C layer—contains just the entities (in/out-) zoomed from another layer, along with the entities projected “as they are” from other layers. So, in general, the other (non-core) layers are not required to be complete—though of course they might be so, as in the case of layer $C + 1$ in Fig. 5. The projected entities are identified by means the prefix “+” if they are projected from a more abstract layer (see entity E2 in Fig. 5), with “-” otherwise—see entity E1 in Fig. 5.

Figure 6 depicts a more detailed view of the Layering capability pattern showing the flow of activities, the process roles involved and the input and work products of each activity.

1.3.2 Process Roles

One role is involved in the Layering pattern: the Layering Expert. Layering Expert is responsible for

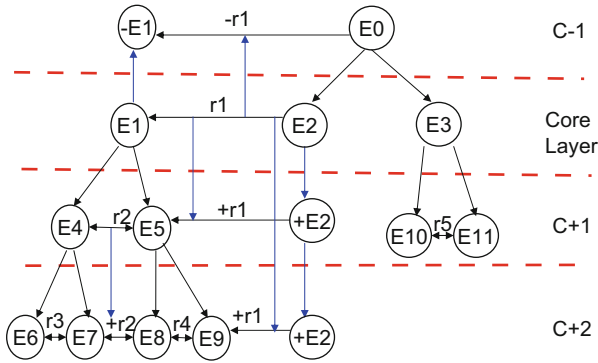


Fig. 5 An example of layering

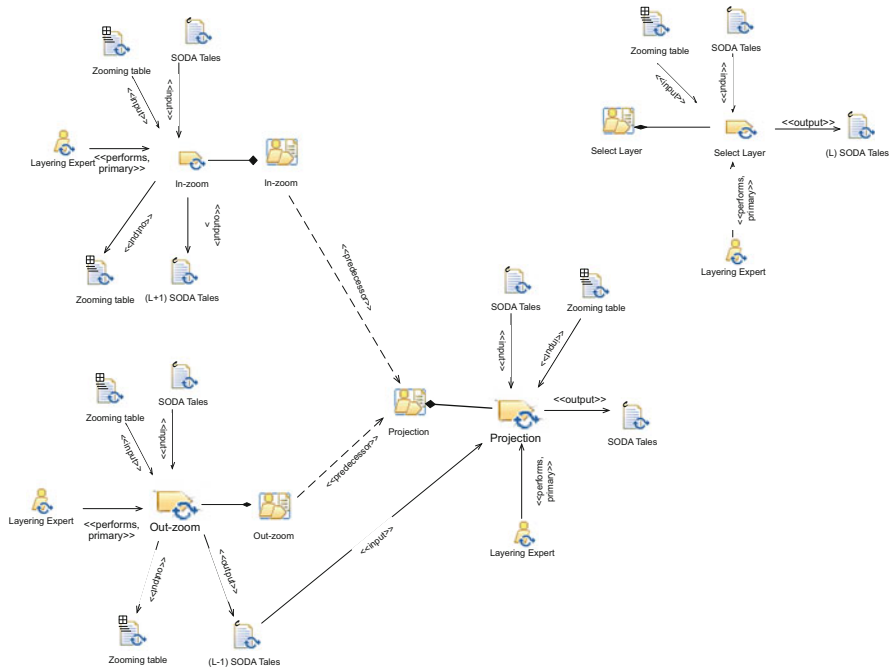


Fig. 6 The layering flow of activities, roles and work products

- Selecting the specific abstraction layer
- Either in-zooming or out-zooming the system by creating the specific Zooming table or modifying an existing Zooming table
- Projecting the necessary entities in the new created layer
- Partially filling all the newly created SODA tables

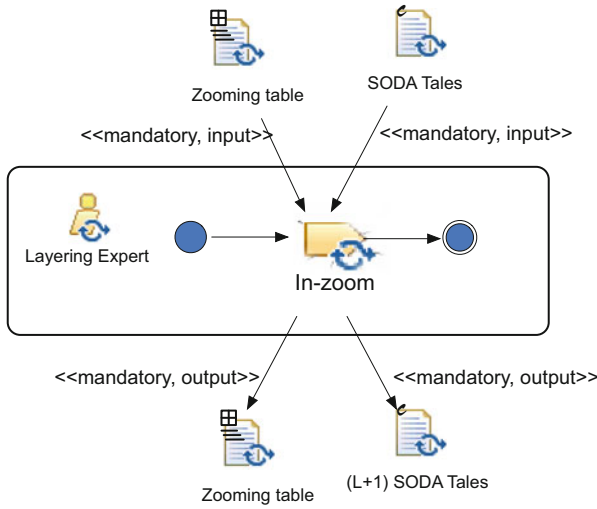


Fig. 7 The task in the in-zoom activity

1.3.3 Activity Details

In-Zoom Activity

The flow of tasks inside *in-zoom* activity is reported in Fig. 7; the tasks are detailed in the following table.

Activity	Task	Task description	Role involved
In-zoom	In-zoom	Allowing the creation of a new, more detailed layer modifying the zooming table, and introducing the work products for the new layer	Layering Expert (<i>perform</i>)

Out-Zoom Activity

The flow of tasks inside *out-zoom* activity is reported in Fig. 8; the tasks are detailed in the following table.

Activity	Task	Task description	Role involved
Out-zoom	Out-zoom	Allowing the creation of a new, more abstract layer modifying the Zooming table, and introducing the work products for the new layer	Layering Expert (<i>perform</i>)

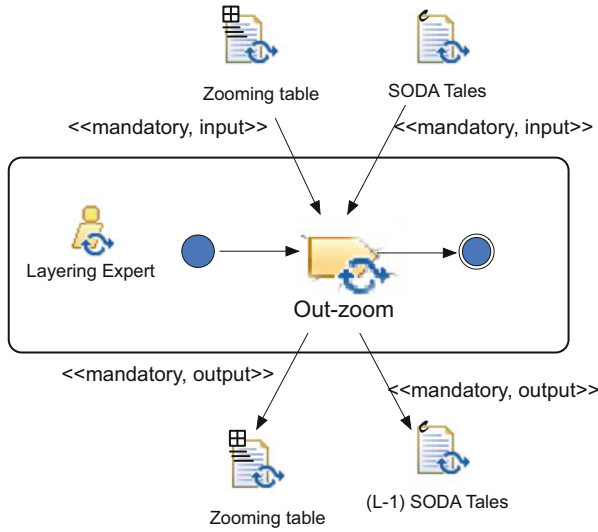


Fig. 8 The task in the out-zoom activity

Select Layer Activity

The flow of tasks inside *Select Layer* activity is reported in Fig. 9; the tasks are detailed in the following table.

Activity	Task	Task description	Role involved
Select Layer	Select Layer	Allowing the selection of a specific layer in order to either redefine or complete it	Layering Expert (perform)

Projection Activity

The flow of tasks *Projection activity* this activity is reported in Fig. 10; the tasks are detailed in the following table.

Activity	Task	Task description	Role involved
Projection	Projection	Allowing the projection of a non-zoomed entity from a layer to another in order to preserve the layer consistency	Layering Expert (perform)

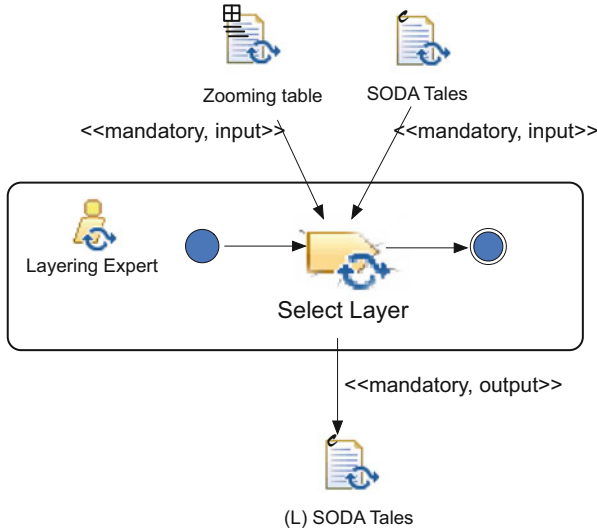


Fig. 9 The task in the select layer activity

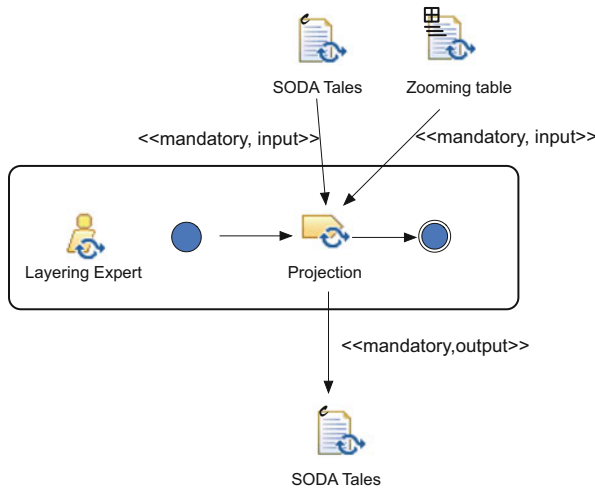


Fig. 10 The task in the projection activity

1.3.4 Work Products

Layering generates one work product: the Zooming table. Its relationships with the MMMElements are described in Fig. 11.

This diagram represents the Layering in terms of the Work Product and its relationships with the SODA meta-model (Sect. 1.2) elements. Each MMMElement is represented using an UML class icon (yellow) and, in the documents, such

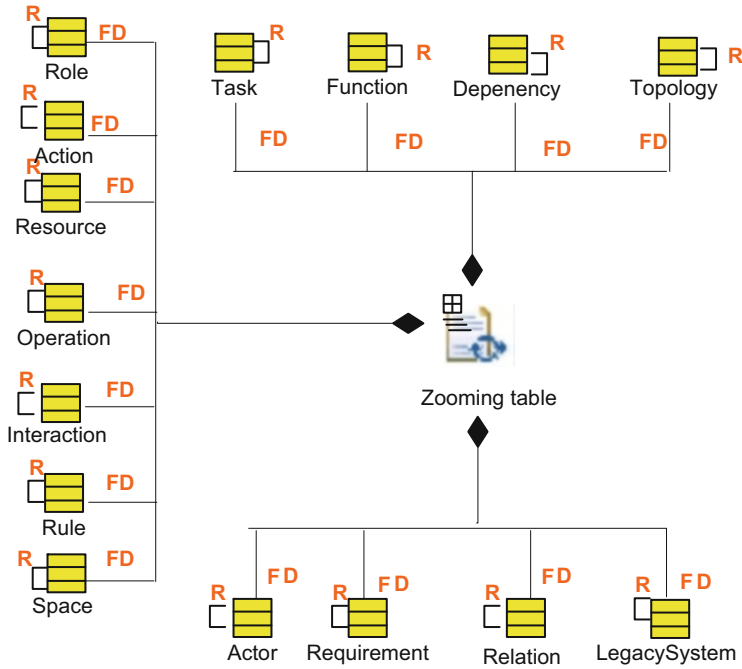


Fig. 11 The layering work products

elements can be Defined, reFined, Quoted, Related or Relationship Quoted as defined in [12] and briefly reported in the following:

- *Defined* (D label)—this means that the element is introduced for the first time in the design in this artifact (the MMMElement is instantiated in this artifact)
- *reFined* (F label)—this means that the MMMElement is refined in the work product (for instance by means of attribute definition)
- *Related* (R label)—this means that an already defined element is related to another, or, from a different point of view, that one of the MAS meta-model relationships is instantiated in the document
- *Quoted* (Q label)—this means that the element was already defined, and it is reported in this artifact only to complete its structure, but no work has to be done on it
- *Relationship Quoted* (RQ label)—this means that the relationship is reported in the work product, but it was defined in another part of the process

Kinds of Work Products

Layering is represented by means of a Zooming table ((C)Z_t)—see Fig. 12. The Zooming table formalises the in-zoom of a layer into the more detailed layer; of course, the same table can be used to represent the dual out-zoom process. One column of the table contains the name of the abstraction at layer C, while the

Fig. 12 $(L)Z_t$

Layer L	Layer $L + 1$
<i>out-zoomed entity</i>	<i>in-zoomed entities</i>

Fig. 13 $(C)Z_t$

Layer C	Layer $C + 1$
E1	E4, E5, r2, +E2, +r1
E3	E10, E11, r5

Fig. 14 $(C - 1)Z_t$

Layer $C - 1$	Layer C
-E1, -r1, E0	E2, E3

Fig. 15 $(C + 1)Z_t$

Layer $C + 1$	Layer $C + 2$
E4	E6, E7, r3, +r2
E5	E8, E9, +E2, r4, +r1

other column reports the name of the corresponding zoomed abstractions at the subsequent layer $C + 1$ (in-zooming) or $C - 1$ (out-zooming).

In general when in-zooming an entity from layer C to layer $C + 1$, we obtain a new group of entities, but also a set of relationships among these new entities that allow the entities' coordination as shown in Fig. 5.

Examples of Work Products

Figures 13, 14, and 15 report the Zooming tables modelling the example proposed in Fig. 5. In particular, Fig. 13 shows the relationships between layer C —the core layer—and layer $C + 1$, where entity E1 is in-zoomed into E4 and E5, and E3 is in-zoomed into E10 and E11. The E2 entity and r1 relationship are projected from C to $C + 1$: this is reported in the in-zoom table of E1, since the relation between E1 and E2 in layer C has to be maintained also in layer $C + 1$ in order to maintain consistency. In addition, two new relationships are necessary after the in-zoom operation: r2 comes from the in-zooming of E1 in order to coordinate the E4 and E5; in a similar way r5 comes from the in-zooming of E3. Figure 14 reports the relation between layer $C - 1$ and layer C . Here there is an out-zoom operation, E2 and E3 are collapsed in E0, while E1 and r1 are projected for consistency reason. Finally, Fig. 15 depicts the relation between layer $C + 1$ and layer $C + 2$ where E4 is in-zoomed in E6, E7 and r3; E5 is in-zoomed in E8, E9 and r4; and E2, r1 and r2 are projected.

2 Phases of the SODA Process

2.1 The Requirements Analysis

The goals of Requirements Analysis are (1) characterising both the customers' requirements and the legacy systems with which the system should interact, as well as (2) highlighting the relationships among requirements and legacy systems. Requirements can be categorised in [14]:

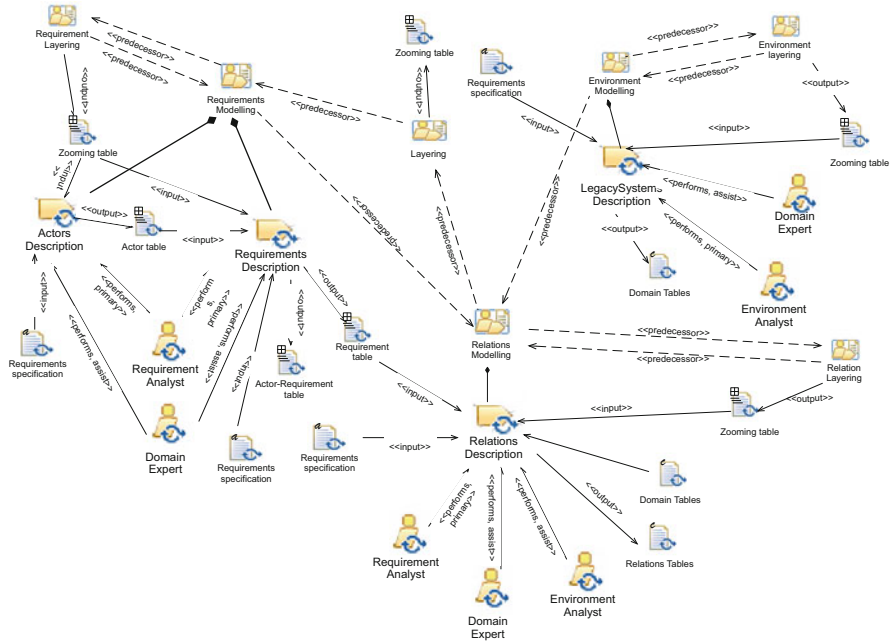


Fig. 16 The requirements analysis flow of activities, roles and work products

- *Functional Requirements*—statements about which functionalities the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- *Non-Functional Requirements*—constraints on the services and functions offered by the system such as timing constraints, constraints on the development process, standards, security, privacy, etc. Non-functional requirements could be more critical than functional requirements. If these are not met, the system is useless.
- *Domain Requirement*—requirements that come from the application domain of the system, and that reflect features of that domain. Domain requirements could be new functional requirements, constraints on existing requirements or define specific computations. If domain requirements are not satisfied, the system may be unworkable.

In this step, we take into account several abstract entities to model the system’s requirements: actors, requirements, external environment, legacy systems and relations. The Requirements Analysis involves three different process roles, and eight work products, as described in Fig. 16. Figure 17 presents the Requirements Analysis process composed by three main activities—Requirements Modelling, Environment Modelling, and Relations Modelling—and several different layering activities—see Sect. 1.3.1.

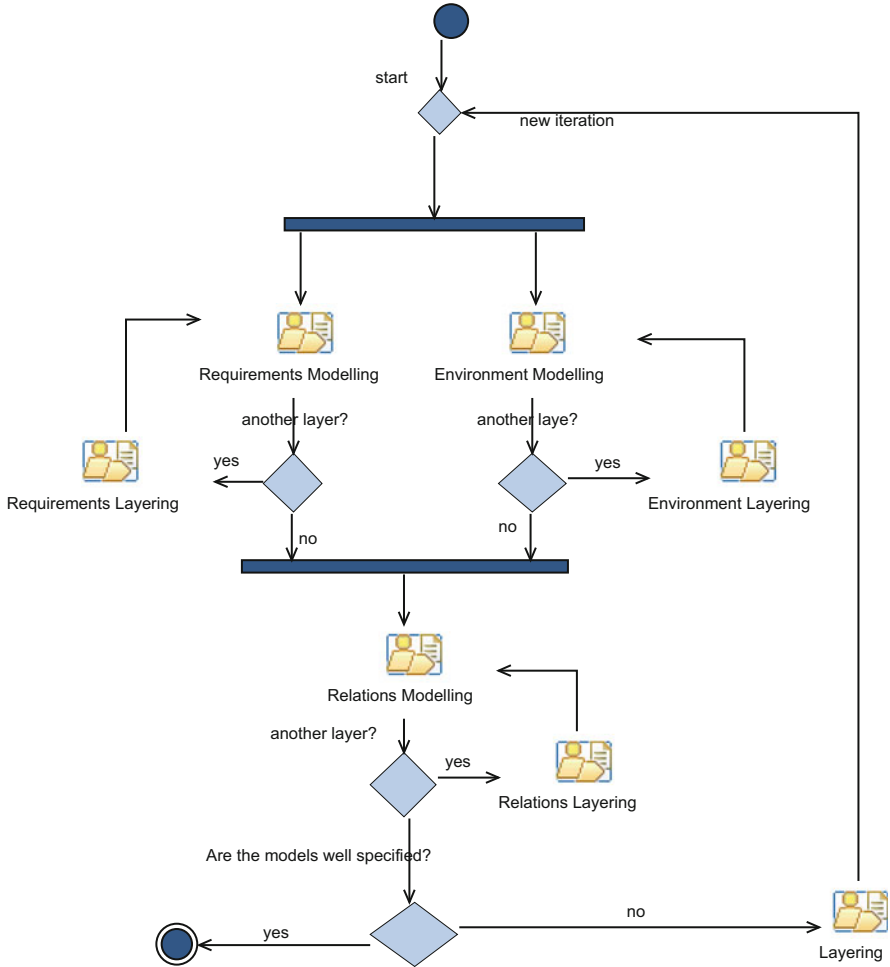


Fig. 17 The requirements analysis process

2.1.1 Process Roles

Three roles are involved in the Requirements Analysis: the Requirement Analyst, the Environment Analyst and the Domain Analyst.

Requirement Analyst

The Requirement Analyst is responsible for

- The identification of the main actors and system stakeholders
- The identification of the system functional and non-functional requirements
- The analysis of the system’s requirements

- The identification of the any kinds of relationship among requirements, and between requirements and legacy systems

Environment Analyst

The Environment Analyst is responsible for

- The identification of the legacy systems already present in the environment
- The analysis of the legacy systems

In addition, the Environment Analyst should help the Requirement Analyst in identification of the any kinds of relationship between requirements and legacy systems.

Domain Expert

The Domain Expert supports the Requirement Analyst and the Environment Analyst during the description of the application domain.

2.1.2 Activity Details

For the details about the different Layering activities please refer to Sect. 1.3.1.

Requirements Modelling Activity

The Requirements Modelling activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Requirements Modelling	Actors Description	<i>Identification of the actors and their description</i>	Requirement Analyst (<i>perform</i>) Domain Expert (<i>assist</i>)
Requirements Modelling	Requirements Description	<i>Identification of the requirements and their description</i>	Requirement Analyst (<i>perform</i>) Domain Expert (<i>assist</i>)

The flow of tasks inside the Requirements Modelling activity is reported in Fig. 18.

Environment Modelling Activity

The Environment Modelling activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Environment Modelling	Legacy Systems Description	<i>Identification of the legacy systems and their description</i>	Environment Analyst (<i>perform</i>) Domain Expert (<i>assist</i>)

The flow of tasks inside the Environment Modelling activity is reported in Fig. 19.

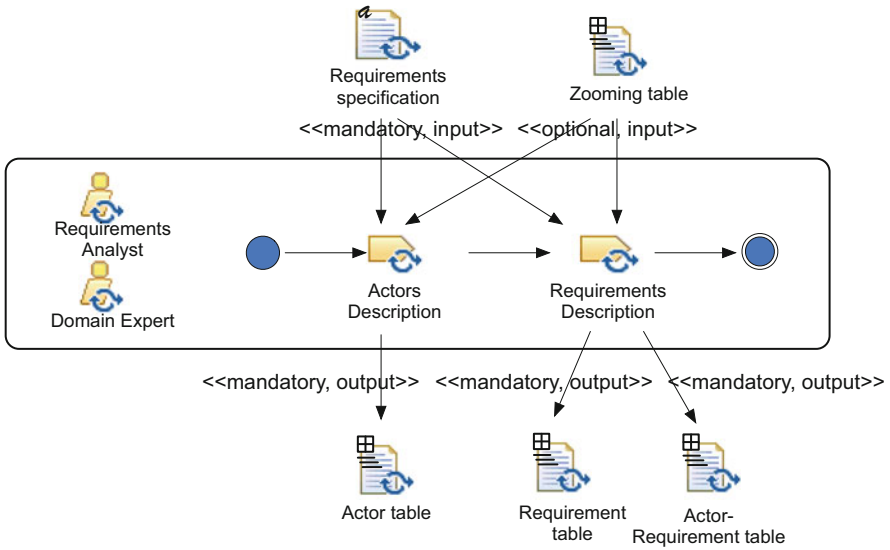


Fig. 18 The flow of tasks in the requirements modelling activity

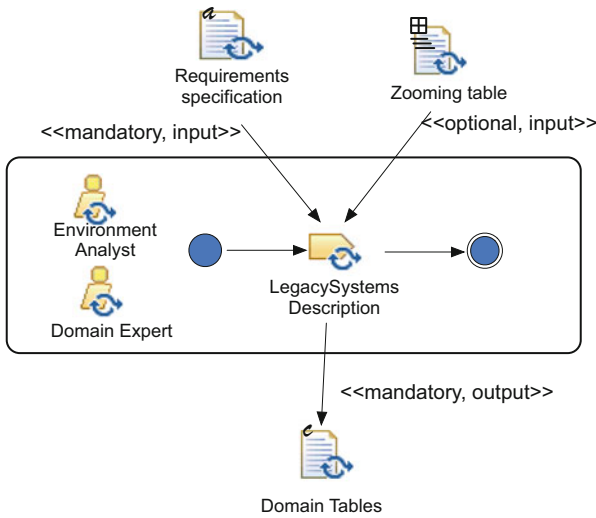


Fig. 19 The flow of tasks in the environment modelling activity

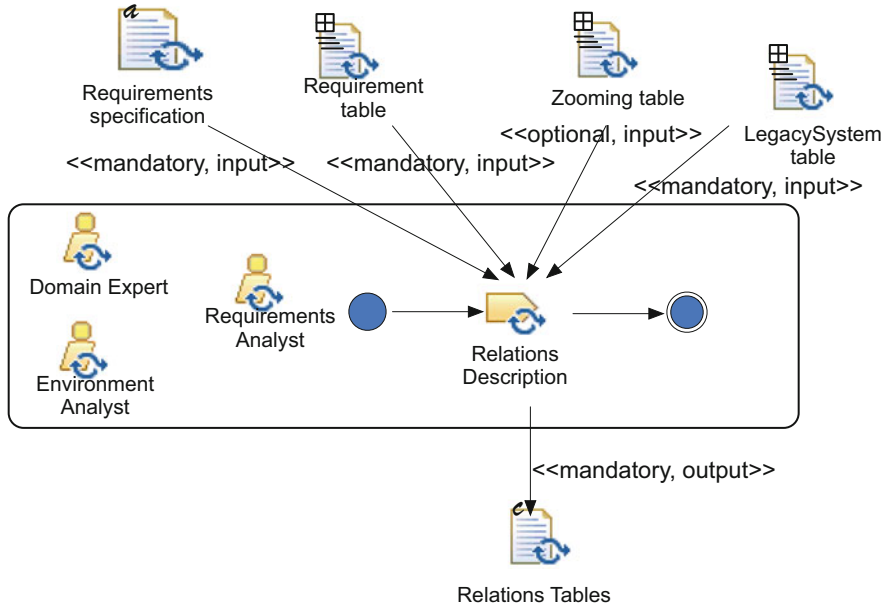


Fig. 20 The flow of tasks in the relations modelling activity

Relations Modelling Activity

The Relations Modelling activity is composed by the following tasks:

Activity	Task	Task Description	Role Involved
Relations Modelling	Relations Description	Identification of the relations and their description	Environment Analyst (perform) Domain Expert (assist) Environment Analyst (assist)

The flow of tasks inside the Relations Modelling activity is reported in Fig. 20.

2.1.3 Work Products

The Requirements Analysis step consists of three sets of tables: Requirements Tables, Domain Tables and Relations Tables. Figure 21 reports the relationships among the work products of this step and the MMMElements of the Requirements Analysis. In Fig. 21, the relationships among the Zooming table and the MMMElements of the Requirements Analysis are also reported—see Sect. 1.3.1 for details.

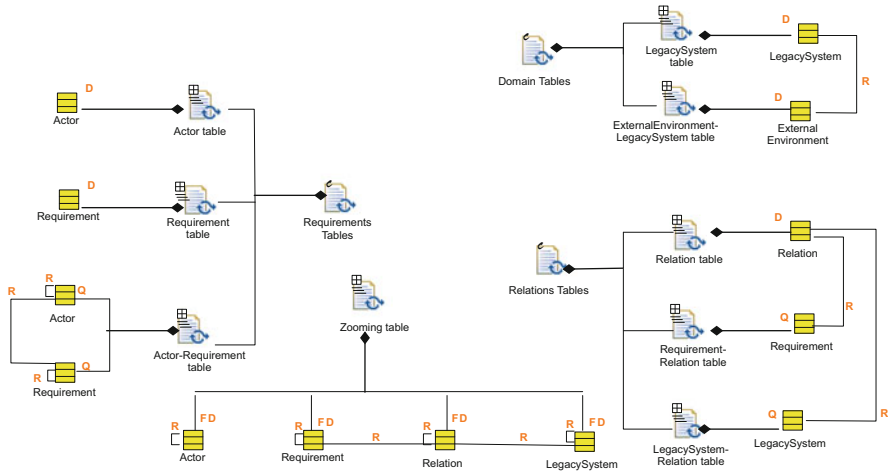


Fig. 21 The requirement analysis work products

Kinds of Work Products

Table 2 describes all the work products of the Requirements Analysis. In particular, the first entry (Requirements Specification) represents the input of the all SODA process, the second set is the outcome of the Requirement Modelling activity, the third set is the outcome of the Environment Modelling activity and the last set is the outcome of the Relation Modelling activity.

Requirements Tables

Figure 22 provides an example of the Requirements Tables for the conference management case study [6].

Domain Tables

In the conference management system case study, there is only one Legacy System, called “WebServer”, which represents the container for the web application of the conference: the reason to include it in the description is that the conference management system will obviously interact with it, and such an interaction should be captured and constrained. Figure 23 presents Legacy System table where the WebServer system is described.

In our system, there is just one relation, called “Web”, which involves all the abstract entities since all requirements need to access the web server to retrieve or store information.

Relations Tables

An example of the Relations Tables in illustrated in Fig. 24.

Table 2 Requirements Analysis work products kinds

Name	Description	Work product kind
Requirements Specification	<i>A description of the problem to be solved</i>	Free Text
<i>Requirements Tables</i>	<i>A composition of others tables that defines the abstract entities tied to the concept of "requirement"</i>	Composite (structured)
Actor table ((L)Ac _i)	<i>Description of each single actor</i>	Structured
Requirement table ((L)Re _i)	<i>Description of each single requirement</i>	Structured
Actor-Requirement table ((L)AR _i)	<i>Specification of the collection of the requirements associated to each actor</i>	Structured
<i>Domain Tables</i>	<i>A composition of others tables that defines the abstract entities tied to the concept of "external environment"</i>	Composite (structured)
Legacy System table ((L)LS _i)	<i>Description of each single legacy system</i>	Structured
External Environment – Legacy System table ((L)EELS _i)	<i>Specification of the legacy systems associated to the external environment</i>	Structured
<i>Relations Tables</i>	<i>A composition of others tables that links the abstract entities with each other</i>	Composite (structured)
Relation table ((L)Rel _i)	<i>Description of all the relationships among abstract entities</i>	Structured
Requirement-Relation table ((L)RR _i)	<i>Specification of the relations where each requirement is involved</i>	Structured
Legacy System – Relation table ((L)LSR _i)	<i>Specification of the relations where each legacy system is involved</i>	Structured

Requirement	Description
ManageStartUp	<i>creating call for papers and defining the rules of the organisation</i>
ManageSubmission	<i>managing users registration, papers submission and keywords insertion</i>
ManagePartitioning	<i>partitioning papers basing on the conference structure</i>
ManageReviewers	<i>managing reviewers registrations and insertion of the keywords representing their expertise area</i>
ManageAssignment	<i>managing the assignment process according to the organisation rules</i>
ManageReview	<i>managing the review process and sending reviews to authors</i>

Fig. 22 Requirement table (C)Re_i

Legacy System	Description
WebServer	<i>the container for the web application of the conference</i>

Fig. 23 Legacy System table (C)LS_i

Relation	Description
Web	<i>access to the web in order to retrieve or storage some information</i>

Fig. 24 Relation table $((C)Rel_t$

Layer C	Layer $C + 1$
ManagePartitioning	UpdateStartUp, ManageSubCommittee, ManageClassification, PartitionPapers, UpSubCooRel, SubCommPartRel, ClassPartRel, Vice-Chair

Fig. 25 Zooming table $((C)Z_t$: paper partitioning in-zoom

Requirement	Description
UpdateStartUp	<i>it could be necessary to update the structure and the rules of the organisation in order to manage a large number of paper submitted</i>
ManageSubCommittee	<i>if necessary, sub-committes will be created and the Vice-Chairs elected</i>
ManageClassification	<i>classification of the papers according to keywords suggested by authors</i>
PartitionPapers	<i>partitioning of papers in order to match authors keywords and reviewers keywords, and according to the organisation's rules</i>

Fig. 26 Requirement table $(C + 1)Re_t$

Requirements Tables at Layer $C + 1$

In Figs. 25 and 26, we report some examples of the SODA tables modelling the conference management systems at layer $C + 1$.

2.2 The Analysis

In the Analysis step, SODA takes into account four abstract entities to analyse the system: tasks, functions, dependencies and topologies. Figure 27 presents the Analysis process, while Fig. 28 presents the flow of activities, the roles involved and the work products.

2.2.1 Process Roles

One role is involved in the Analysis step: the System Analyst.

System Analyst

The System Analyst is responsible for

- Mapping the MMMElements of the Requirements Analysis to the MMMElements of the Analysis
- Identifying new tasks coming from system analysis and description of the all tasks (new tasks and tasks coming from the mapping)
- Identifying new functions coming from system analysis and description of the all functions (new tasks and tasks coming from the mapping)

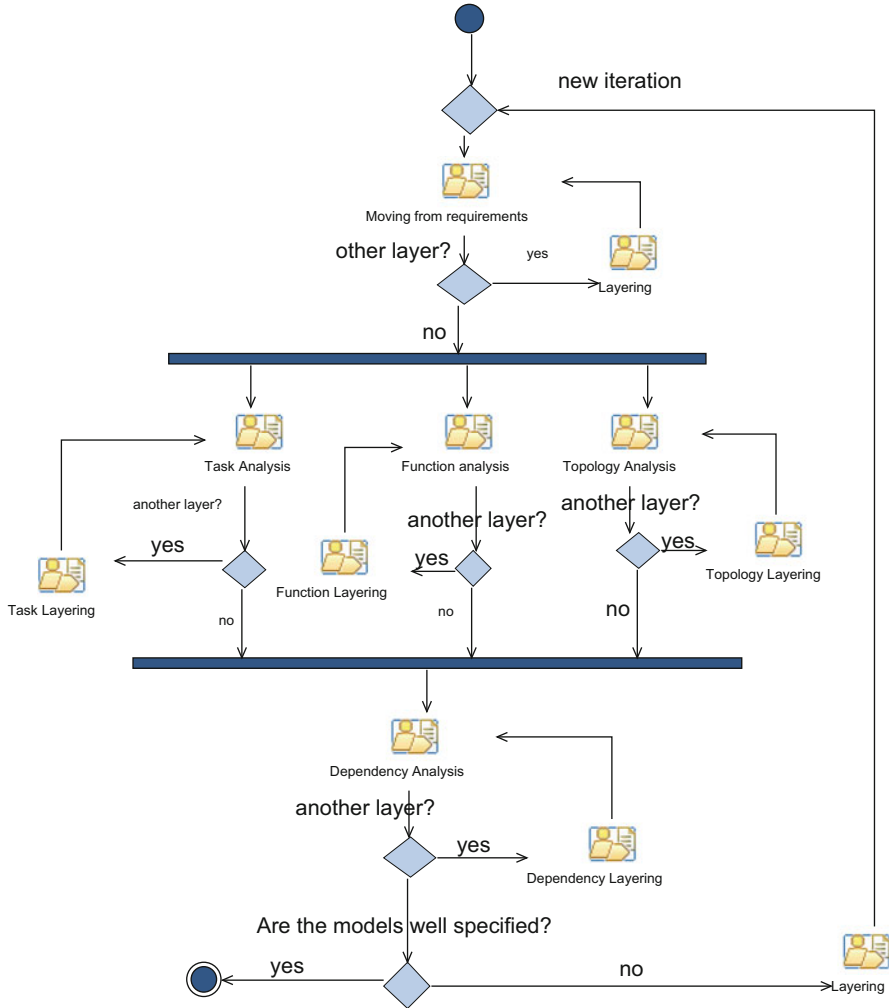


Fig. 27 The analysis process

- Identifying new dependencies coming from system analysis and description of the all dependencies (new dependencies and dependencies coming from the mapping)
- Identifying new topologies coming from system analysis and description of the all topologies (new topologies and topologies coming from the mapping)

2.2.2 Activity Details

For the details about the different Layering activities, please refer to Sect. 1.3.1.

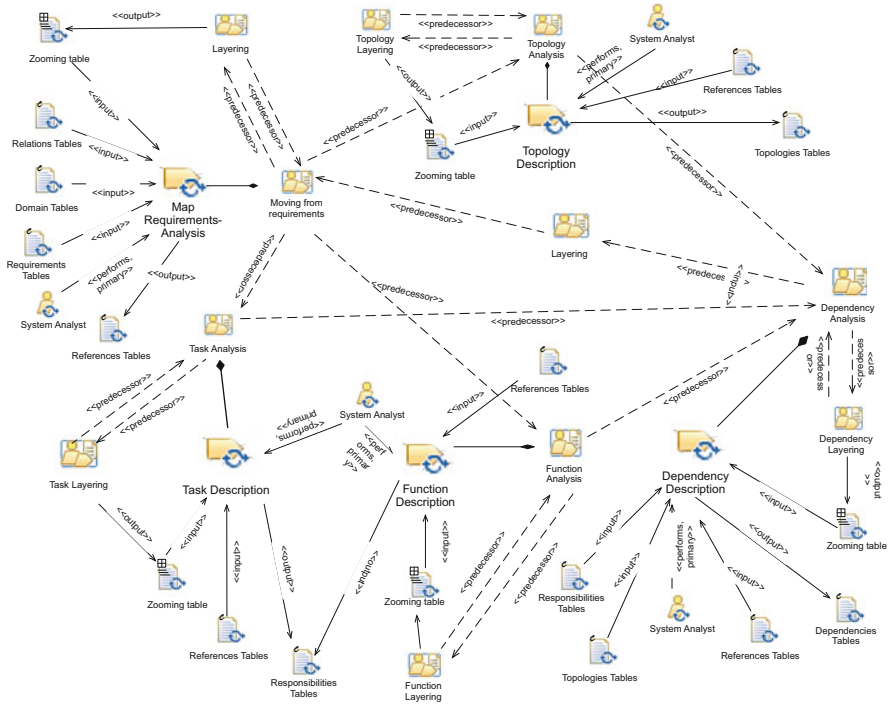


Fig. 28 The analysis flow of activities, roles and work products

Moving from Requirements Activity

The Moving from Requirements activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Moving from Requirements	Map Requirements-Analysis	Mapping of the MMMElements defined in Requirements Analysis to the Analysis MMMElements	System Analyst (perform)

The flow of tasks inside the Moving from Requirements activity is reported in Fig. 29.

Task Analysis Activity

The Task Analysis activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Task Analysis	Task Description	Identification of the tasks and their description	System Analyst (perform)

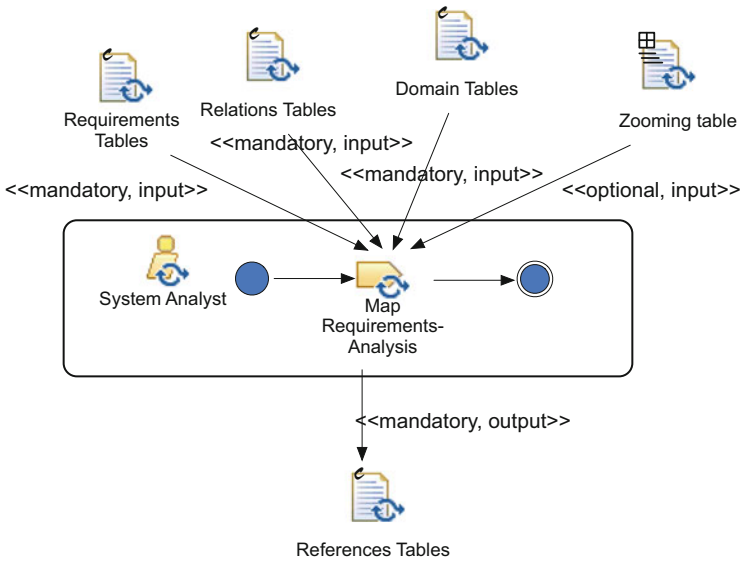


Fig. 29 The flow of tasks in the moving from requirements activity

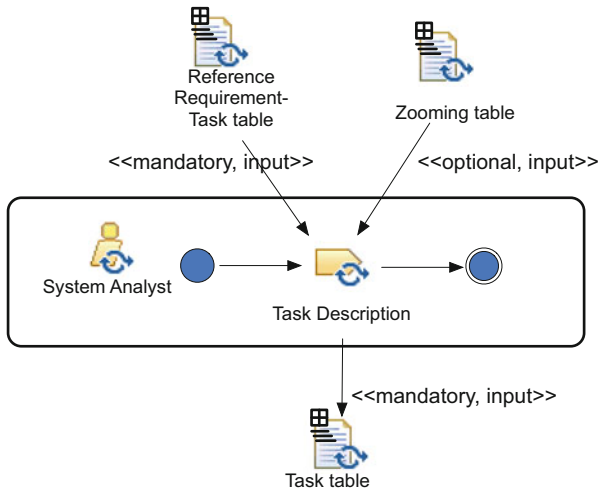


Fig. 30 The flow of tasks in the task analysis activity

The flow of tasks inside the Task Analysis activity is reported in Fig. 30.

Function Analysis Activity

The flow of tasks inside this activity is reported in Fig. 31; the tasks are detailed in the following table.

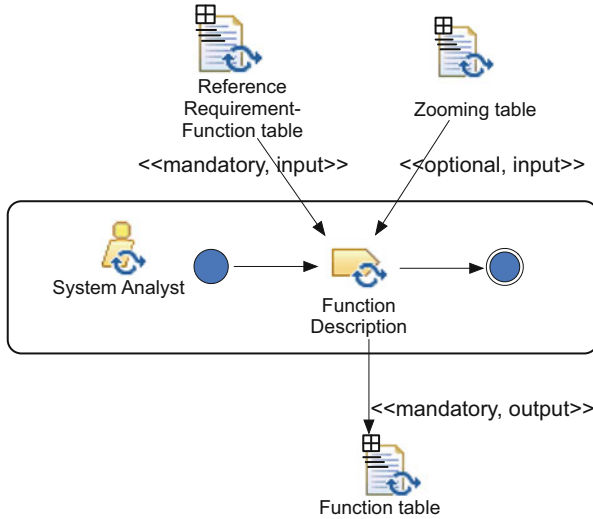


Fig. 31 The flow of tasks in the function analysis activity

Activity	Task	Task description	Role involved
Function Analysis	Function Description	Identification of the functions and their description	System Analyst (perform)

Dependency Analysis Activity

The flow of tasks inside this activity is reported in Fig. 32, and the tasks are detailed in the following table.

Activity	Task	Task description	Role involved
Dependency Analysis	Dependency Description	Identification of the system dependencies and their description. Identification of relations with tasks, functions and topology	System Analyst (perform)

Topology Analysis Activity

The flow of tasks inside this activity is reported in Fig. 33; the tasks are detailed in the following table.

Activity	Task	Task description	Role involved
Topology Analysis	Topology Description	Identification of the topological constraints and their description. Identification of relations with tasks and functions	System Analyst (perform)

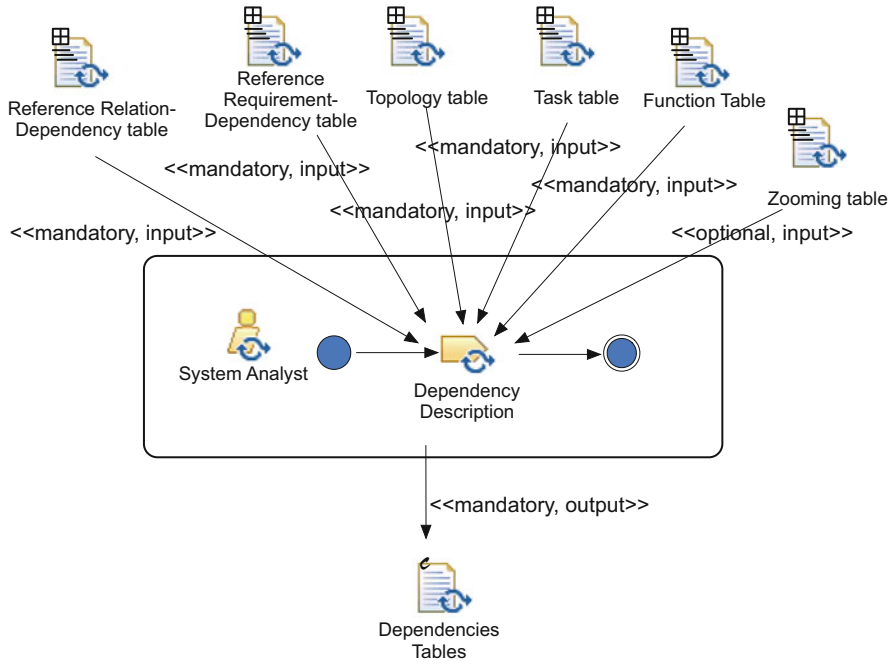


Fig. 32 The flow of tasks in the dependency analysis activity

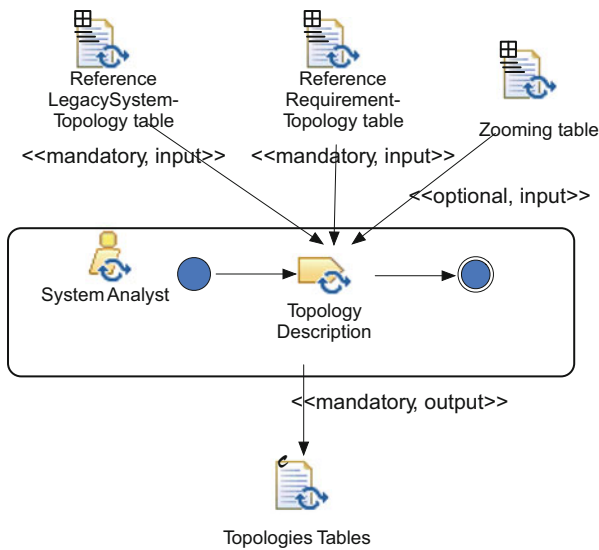


Fig. 33 The flow of tasks in the topology analysis activity

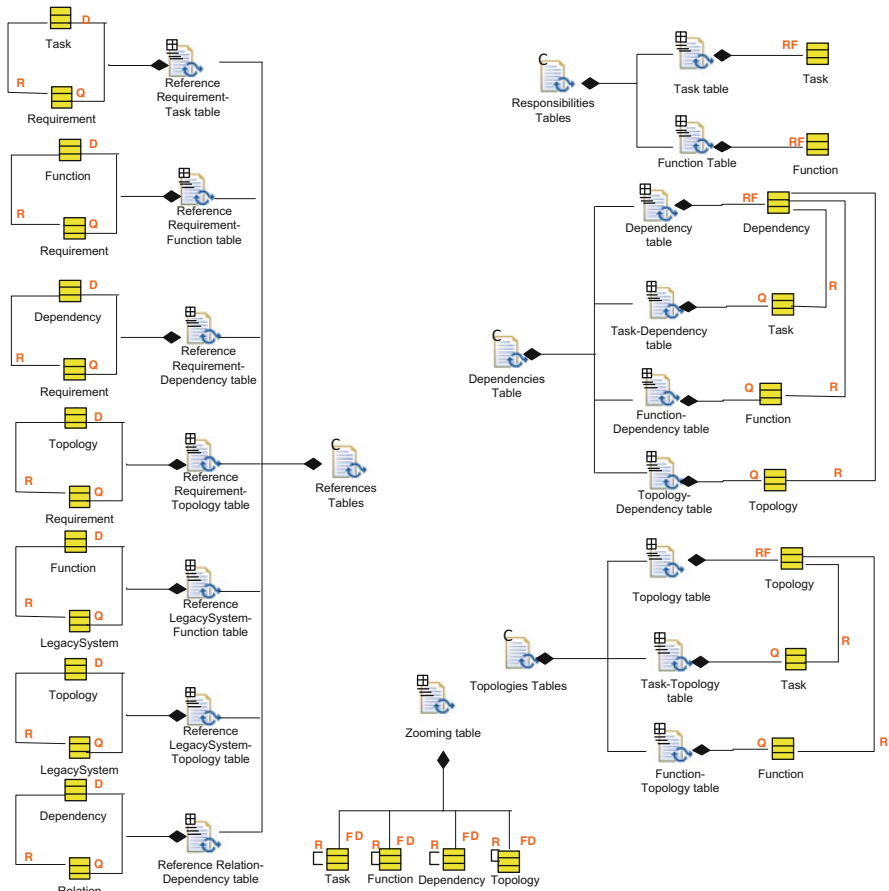


Fig. 34 The analysis work products

2.2.3 Work Products

The Analysis step exploits four sets of tables: Reference Tables, Responsibilities Tables, Dependencies Tables and Topologies Tables. Figure 34 reports the relationships among the work products of this step and the MMMElements of the Analysis. In Fig. 34 are also reported the relationships among the Zooming table and the MMMElements of the Analysis—see Sect. 1.3.1 for details.

Kinds of Work Products

Table 3 describes all the work products of the Analysis. In particular, the first set of work products is the outcome of the Moving from Requirements activity, the second set is the outcome of the Task Analysis and Function Analysis activities, the third is

Table 3 Requirements Analysis work products kinds

Name	Description	Work product kind
<i>References Tables</i>	<i>A composition of others tables that allow to move from Requirements Analysis to Analysis</i>	Composite (structured)
Reference Requirement-Task table ((L)RRT _i)	<i>Specification of the mapping between each requirement and the generated tasks</i>	Structured
Reference Requirement-Function table ((L)RRF _i)	<i>Specification of the mapping between each requirement and the generated functions</i>	Structured
Reference Requirement-Topology table ((L)RRT _o _i)	<i>Specification of the mapping between each requirement and the generated topologies</i>	Structured
Reference Requirement-Dependency table ((L)RReqD _i)	<i>Specification of the mapping between each requirement and the generated dependencies</i>	Structured
Reference Legacy System-Function table ((L)RLSF _i)	<i>Specification of the mapping between each Legacy System and the corresponding functions</i>	Structured
Reference Legacy System-Topology table ((L)RLST _i)	<i>Specification of the mapping between Legacy Systems and topologies</i>	Structured
Reference Relation-Dependency table ((L)RRelD _i)	<i>Specification of the mapping between relations and dependencies</i>	Structured
<i>Responsibilities Tables</i>	<i>A composition of others tables that defines the abstract entities tied to the concept of “responsibilities centre”</i>	Composite (structured)
Task table ((L)T _i)	<i>Description of the all tasks</i>	Structured
Function table ((L)F _i)	<i>Description of the all functions</i>	Structured
<i>Topologies Tables</i>	<i>A composition of others tables that express the topological constraints over the environment</i>	Composite (structured)
Topology table ((L)Top _i)	<i>Description of the topological constraints</i>	Structured
Task-Topology table ((L)TTop _i)	<i>Specification of the list of the topological constraints where each task is involved</i>	Structured
Function-Topology table ((L)FTop _i)	<i>Specification of the list of the topological constraints where each function is involved</i>	Structured
<i>Dependencies Tables</i>	<i>A composition of others tables that relates functions and tasks with each other</i>	Composite (structured)
Dependency table ((L)D _i)	<i>Description of the all dependencies among abstract entities</i>	Structured
Task-Dependency table ((L)TD _i)	<i>Specification of the set of dependencies where each task is involved</i>	Structured
Function-Dependency table ((L)FD _i)	<i>Specification of the list of dependencies where each function is involved</i>	Structured
Topology-Dependency table ((L)TopD _i)	<i>Specification of the list of dependencies where each topology is involved</i>	Structured

Requirement	Task
ManageStartUp	<i>start up</i>
ManageSubmission	<i>paper submission user registration</i>
ManageReviewers	<i>reviewer registration</i>
ManagePartitioning	<i>paper partitioning</i>
ManageAssignment	<i>assignment papers</i>
ManageReview	<i>review process</i>

Fig. 35 Reference Requirement-Task table (C)RRT_i

Task	Description
start up	<i>insertion of the setup information</i>
submission	<i>the paper has to be submitted and the keywords have to be indicated</i>
user registration	<i>user inserts his data</i>
reviewer registration	<i>reviewer inserts his data and the keywords representing his expertise areas</i>
paper partitioning	<i>partitioning of the set of papers according to the conference rules</i>
assignment papers	<i>assignment papers to reviewers</i>
review process	<i>creation and submission of the reviews</i>

Fig. 36 Task table (C)T_i

Topology	Description
place	<i>it is the locus where functions are allocated</i>

Fig. 37 Topology table (C)Top_i

the outcome of the Topology Analysis activity and the last set is the outcome of the Dependency Analysis activity.

References Tables

Figure 35 represents an example of the References Tables for the conference management case study.

Responsibilities Tables

Figure 36 represents an example of the Responsibilities Tables for the conference management case study. Figure 37 represents an example of the Topologies Tables for the conference management case study.

Dependencies Tables

Figure 38 represents an example of the Dependencies Tables for the conference management case study.

Responsibilities Tables at Layer C + 1

Figures 39 and 40 report some examples of the SODA tables modelling the conference management systems at layer C + 1.

Dependency	Description
RegSubDep	<i>paper submission to be done after author registration</i>
RegAssDep	<i>the paper assignment has to be done after reviewers registration</i>
PartAssDep	<i>the paper assignment has to be done after the conclusion of the paper partitioning process</i>
AssRevDep	<i>the paper revision has to be started only after the conclusion of the paper assignment process</i>
WebAccessDep	<i>access to website for retrieving or storing information</i>
StartUpInfDep	<i>access of all the information about start up process</i>
UserInfDep	<i>access to all the users' information</i>
ReviewerInfDep	<i>access to all the reviewers' information</i>
PaperInfDep	<i>access to all the paper information</i>
PartInfDep	<i>access to all the information about partitioning process</i>
SubInfDep	<i>access to all the information about submission process</i>
AssInfDep	<i>access to all the information about assignment process; a reviewer cannot be the author of the papers assigned to him</i>
ReviewInfDep	<i>access to all the information about review process</i>

Fig. 38 Dependency table (C)D_i

Layer C	Layer C + 1
paper partitioning	modifying startup, create sub-committees, Vice-Chair elections, paper classification, partition papers, NewOrganisationDep, ElectionDep

Fig. 39 Zooming table (C)Z_i

Task	Description
modifying startup	<i>update the structure and the rules of the organisation</i>
create sub-committees	<i>creation of sub-committees</i>
Vice-Chair elections	<i>for each sub-committee elect the Vice-Chair</i>
papers classification	<i>classification of papers according to the keywords</i>
partition papers	<i>partitioning papers according to their classification</i>

Fig. 40 Task table (C + I)T_i

2.3 The Architectural Design

In this step, we take into account several abstract entities in order to design the system’s general architecture: role, resource, action, operation, interaction, environment and place. Figure 41 presents the Architectural Design process, while Fig. 42 presents the flow of activities, the involved roles and the work products.

2.3.1 Process Roles

One role is involved in the Architectural Design: the Architectural Designer.

Architectural Designer

The Architectural Designer is responsible for

- Mapping the MMMElements of the Analysis to the MMMElements of the Architectural Design

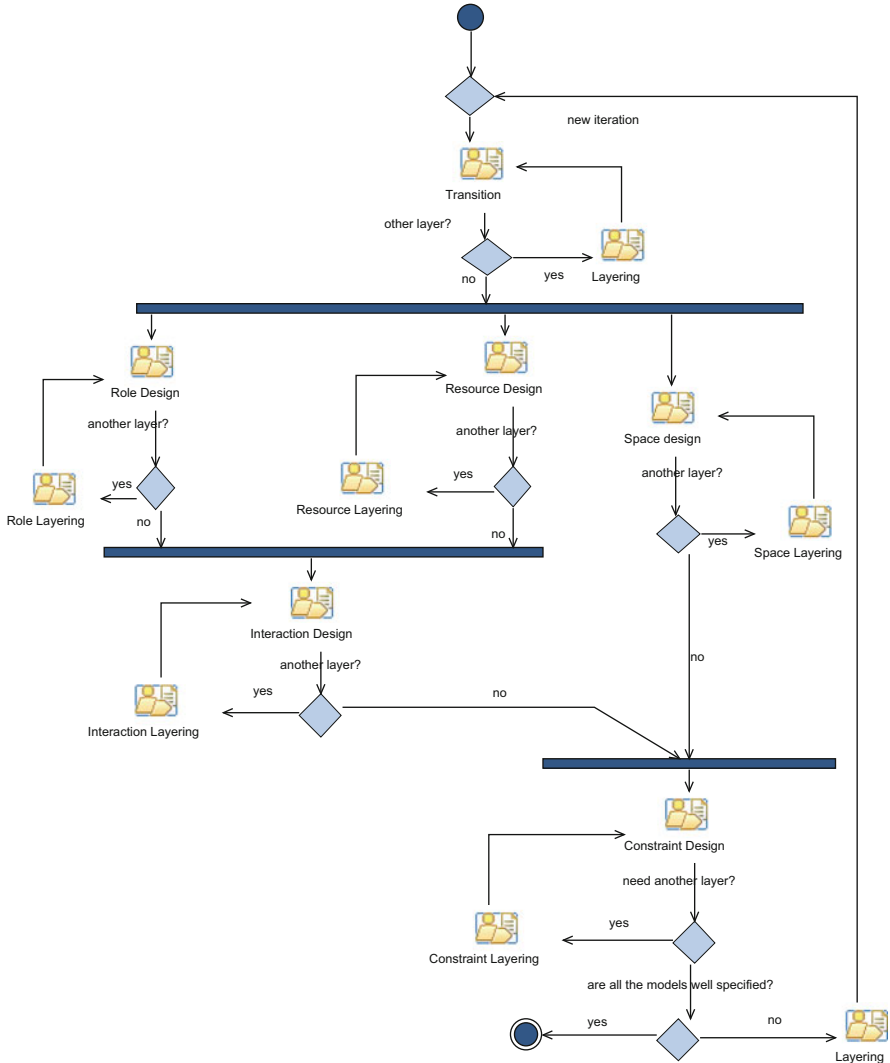


Fig. 41 The architectural design process

- Assigning tasks to roles
- Assigning functions to resources
- Identifying new actions coming from system design and describing all the actions (new actions and actions coming from the mapping)
- Identifying operations coming from system design and describing all the operations (new operations and operations coming from the mapping)
- Identifying new interactions coming from system design and describing all the interactions (new interactions and interactions coming from the mapping)

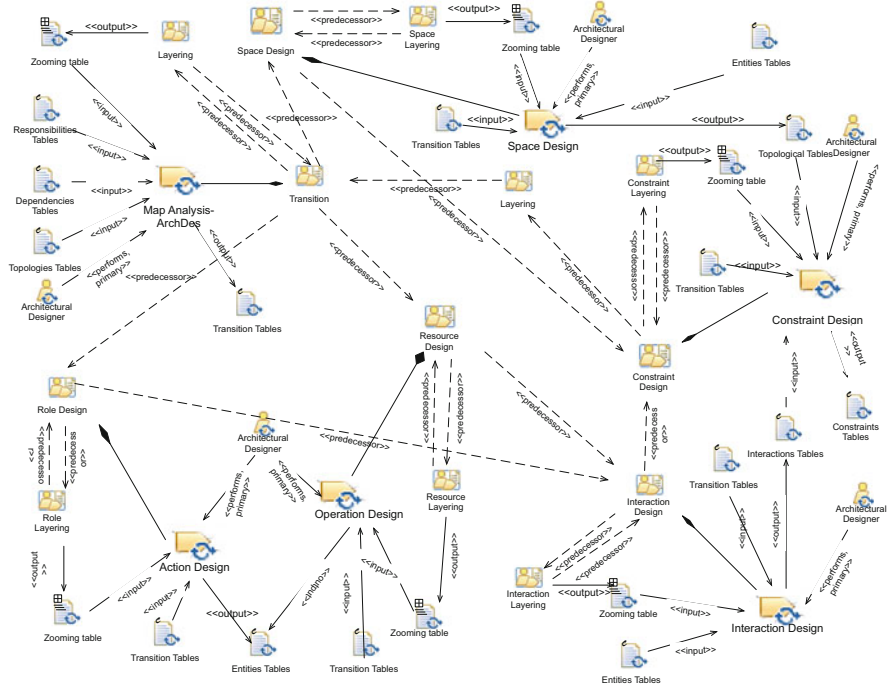


Fig. 42 The architectural design flow of activities, roles, and work products

- Identifying new rules coming from system design and describing all the rules (new rules and rules coming from the mapping)
- Identifying new spaces coming from system design and describing all the spaces (new spaces and spaces coming from the mapping)

2.3.2 Activity Details

For the details about the different Layering activities, please refer to Sect. 1.3.1.

Transition Activity

The Transition activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Transition	Map Analysis-ArchDes	Mapping of the MMMElements defined in Analysis to the Architectural Design MMMElements so as to generate the initial version of the Architectural Design models	Architectural Designer (perform)

The flow of tasks inside the Transition activity is reported in Fig. 43.

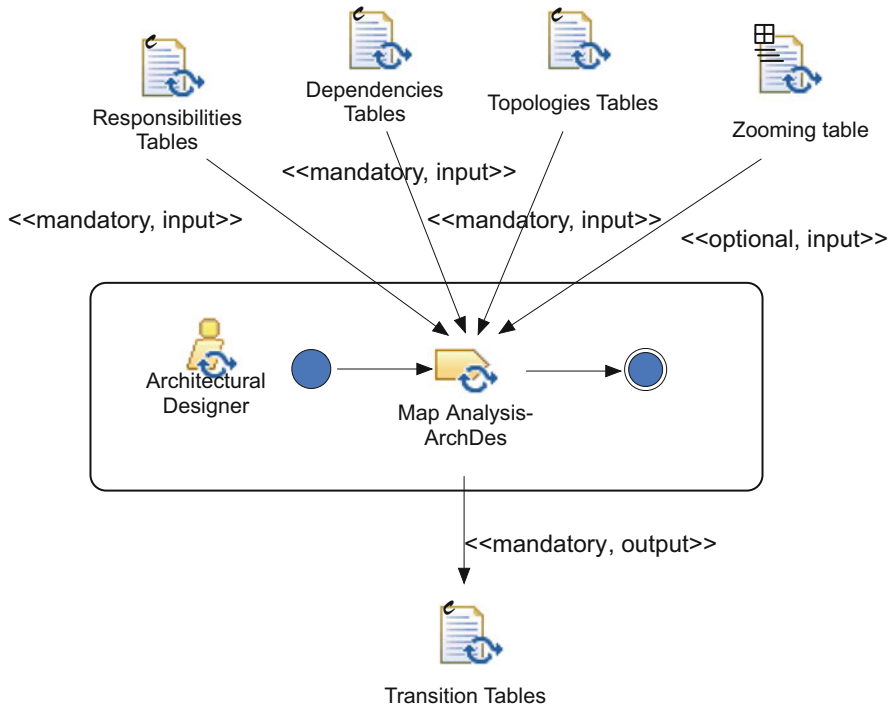


Fig. 43 The flow of tasks in the transition activity

Role Design Activity

The Role Design activity is composed by the following tasks:

Activity	Task	Task Description	Role Involved
Role Design	Action Design	Assignment of tasks to roles and identification of the actions necessary in order to achieve each specific task	Architectural Designer (perform)

The flow of tasks inside the Role Design activity is reported in Fig. 44.

Resource Design activity

The Resource Design activity is composed by the following tasks:

Activity	Task	Task Description	Role Involved
Operation Design	Resource Design	Assignment of functions to resources and identification of the operations necessary for providing each specific function	Architectural Designer (perform)

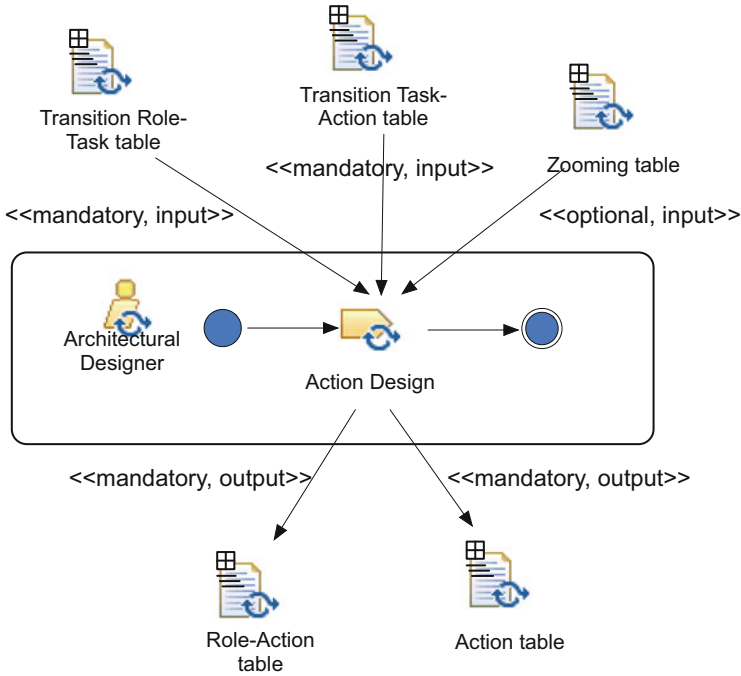


Fig. 44 The flow of tasks in the role design activity

The flow of tasks inside the Resource Design activity is reported in Fig. 45.

Constraint Design Activity

The Constraint Design activity is composed by the following tasks:

Activity	Task	Task Description	Role Involved
Constraint Design	Constraint Design	Identification of the rules that enable and bound the entities' behaviour starting from the dependencies analysed in the previous step	Architectural Designer (perform)

The flow of tasks inside the Constraint Design activity is reported in Fig. 46.

Interaction Design Activity

The Interaction Design activity is composed by the following tasks:

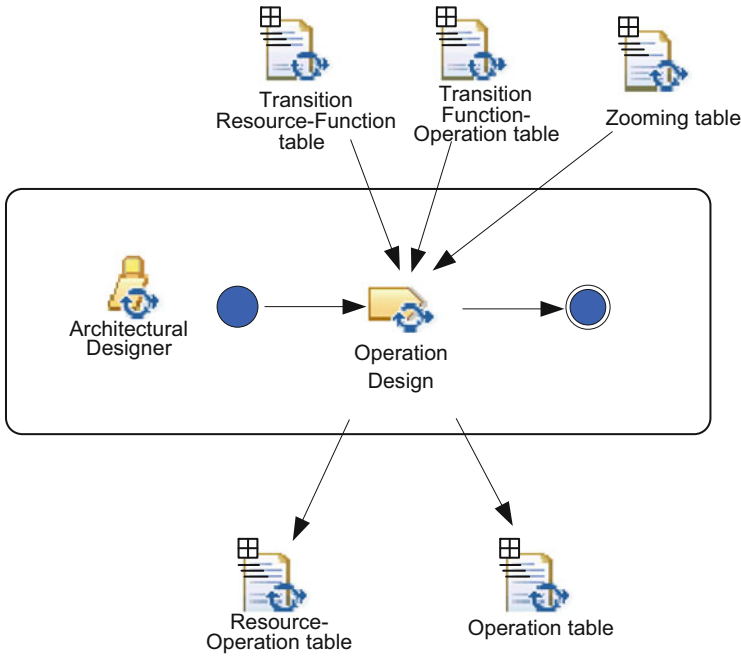


Fig. 45 The flow of tasks in the resource design activity

Activity	Task	Task Description	Role Involved
Interaction Design	Interaction Design	<i>Identification of the interactions that represent the acts of the interaction among roles, among resources and between roles and resources starting from the dependencies analysed in the previous step</i>	Architectural Designer (perform)

The flow of tasks inside the Interaction Design activity is reported in Fig. 47.

Space Design Activity

The Space Design activity is composed by the following tasks:

Activity	Task	Task Description	Role Involved
Space Design	Space Design	<i>Identification of the spaces starting from the topology constraints analysed in the previous step</i>	Architectural Designer (perform)

The flow of tasks inside the Space Design activity is reported in Fig. 48.

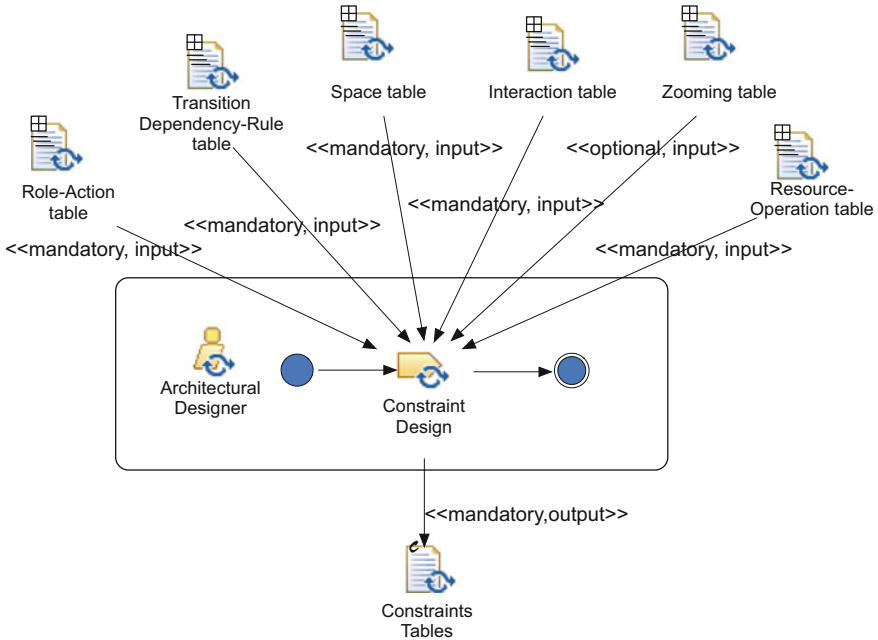


Fig. 46 The flow of tasks in the constraint design activity

2.3.3 Work Products

The Architectural Design step consists of five sets of tables: Transition Tables, Entities Tables, Interactions Tables, Constraints Tables and Topological Tables. Figure 49 reports the relationships among the work products of this step and the MMMElements of the Architectural Design step. In Fig. 49 are also reported the relationships among the Zooming table and the MMMElements of the Architectural Design—see Sect. 1.3.1 for details.

Kinds of Work Products

Table 4 describes all the work products of the Architectural Design. In particular, the first set of work products is the outcome of the Transition activity, the second is the outcome of the Role Design and Resource Design activities, the third is the outcome of the Interaction Design activity, the fourth is the outcome of the Constraint Design activity and the last is the outcome of the Space Design activity.

Transition Tables

Figure 50 presents an example of the Transition Tables for the conference management case study.

Entities Tables

Figure 51 presents an example of the Entities Tables for the conference management case study.

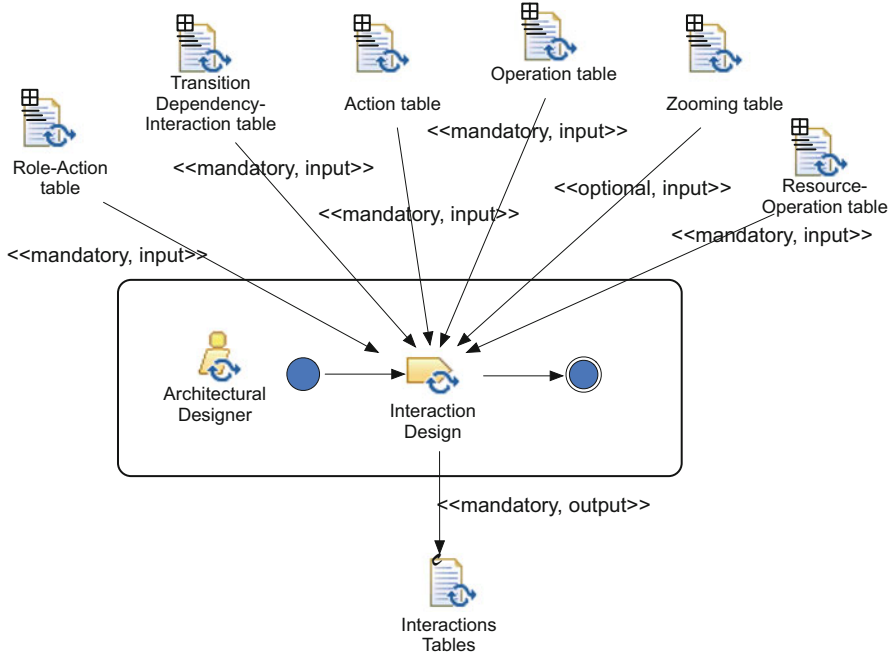


Fig. 47 The flow of tasks in the interaction design activity

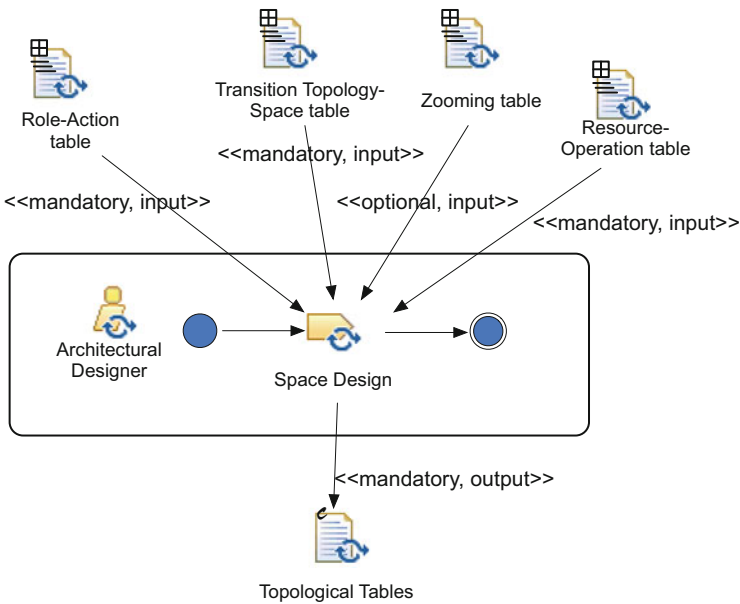


Fig. 48 The flow of tasks in the space design activity

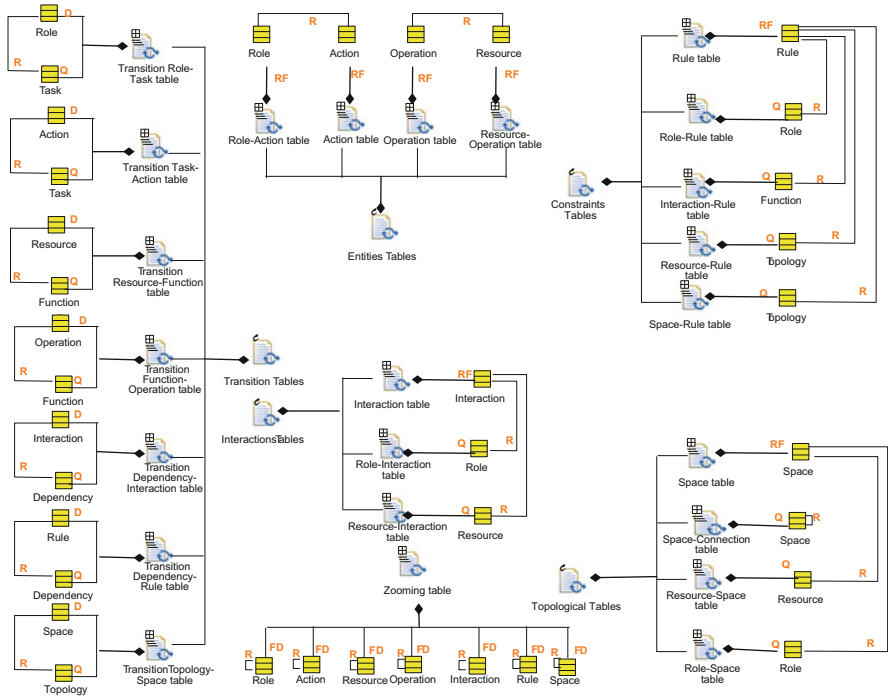


Fig. 49 The architectural design work products

Interactions Tables

Figure 52 presents an example of the Interactions Tables for the conference management case study.

Constraints Tables

Figure 53 presents an example of the Constraints Tables for the conference management case study.

Topological Tables

Figure 54 presents an example of the Topological Tables for the conference management case study.

2.4 The Detailed Design

The goal of Detailed Design is to choose the most adequate representation level for each architectural entity, thus leading to depict one (detailed) design from the many potential alternatives outlined in the Architectural Design step. Figure 55

Table 4 Architectural Design work products kinds

Name	Description	Work product kind
<i>Transition Tables</i>	<i>A composition of others tables that links the Analysis step with the Architectural Design step</i>	Composite (structured)
Transition Role-Task table ((L)TRT _i)	<i>Specification of the mapping between each role and the tasks assigned to it</i>	Structured
Transition Task-Action table ((L)TTA _i)	<i>Specification of the mapping between each task and the generated actions</i>	Structured
Transition Resource-Function table ((L)TRF _i)	<i>Specification of the mapping between each functions and the functions assigned to it</i>	Structured
Transition Function-Operation table ((L)TFO _i)	<i>Specification of the mapping between each functions and the generated operations</i>	Structured
Transition Dependency-Interaction table ((L)TDI _i)	<i>Specification of the mapping between each dependency and the generated interactions</i>	Structured
Transition Dependency-Rule table ((L)TDRu _i)	<i>Specification of the mapping between each dependency and the generated rules</i>	Structured
Transition Topology-Space table ((L)TTopS _i)	<i>Specification of the mapping between each topology and the generated spaces</i>	Structured
<i>Entities Tables</i>	<i>A composition of others tables that describes both the active entities (able to perform some actions in the system) and the passive entities which provide services</i>	Composite (structured)
Action table ((L)A _i)	<i>Description of the actions executable by some roles</i>	Structured
Operation table ((L)O _i)	<i>Description of the operations provided by resources</i>	Structured
Role-Action table ((L)RA _i)	<i>Specification of the actions that each role can do</i>	Structured
Resource-Operation table ((L)RO _i)	<i>Specification of the operations that each resource can provide</i>	Structured
<i>Interactions Tables</i>	<i>A composition of others tables that describe the interaction between roles and resources</i>	Composite (structured)
Interaction table ((L)I _i)	<i>Description of the single interactions</i>	Structured
Action-Interaction table ((L)AcI _i)	<i>Specification of the interactions where each action is involved</i>	Structured
Operation-Interaction table ((L)OpI _i)	<i>Specification of the interactions where each operation is involved</i>	Structured
<i>Constraints Tables</i>	<i>A composition of others tables that describes the constraints over the entities behaviours</i>	Composite (structured)
Rule table ((L)Ru _i)	<i>Description of the rules</i>	Structured
Rule-Interaction table ((L)IRu _i)	<i>Specification of the constraints over the interactions</i>	Structured

(continued)

Table 4 (continued)

Name	Description	Work product kind
Resource-Rule table ((L)ReI _t)	<i>Specification of the rules where each resource is involved</i>	Structured
Role-Rule table ((L)RoRu _t)	<i>Specification of the rules where each role is involved</i>	Structured
Space-Rule table ((L)SRu _t)	<i>Specification of the rules where each space is involved</i>	Structured
<i>Topological Tables</i>	<i>A composition of others tables that describes the logical structure of the environment</i>	Composite (structured)
Space table ((L)S _t)	<i>Description of the spaces</i>	Structured
Space-Connection table ((L)SC _t)	<i>Specification of the connections among the spaces of a given layer (the hierarchical relations between spaces are expressed via the Zooming Table)</i>	Structured
Resource-Space table ((L)ReS _t)	<i>Specification of the all spaces where resources is involved</i>	Structured
Role-Space table ((L)RoS _t)	<i>Specification of the all spaces where role is involved</i>	Structured

Role	Task
Conference Secretary	start up
Chair	paper partitioning, assignment papers
Author	submission, user registration
Reviewer	reviewer registration, review process
PC-member	reviewer registration, review process

Fig. 50 Transition role-task table (C)TRT_t

Action	Description
login	<i>user authentication</i>
send paper	<i>user compiles form and sends his paper</i>
publish deadline	<i>user generates/modifies deadline</i>
partition	<i>user splits papers according to keywords</i>
assignment	<i>user assigns papers</i>
read paper	<i>user reads papers</i>
download paper	<i>user downloads paper from the web</i>
write review	<i>user writes the review</i>

Fig. 51 Action table (C)A_t

presents the Detailed Design process, while Fig. 56 presents the flow of activities, the involved roles and the work products.

2.4.1 Process Roles

One role is involved in the Detailed Design: the Detailed Designer.

Interaction	Description
UserInfInteraction	<i>accessing user information</i>
ReviewerInfInteraction	<i>accessing reviewer information</i>
PaperInfInteraction	<i>accessing paper information</i>
PartInfInteraction	<i>accessing partitioning information</i>
SubInfInteraction	<i>accessing submission information</i>
AssInfInteraction	<i>accessing assignment information</i>
ReviewInfInteraction	<i>accessing review information</i>
WebAccessInteraction	<i>accessing website</i>

Fig. 52 Interaction table $(C)I_t$

Rule	Description
RegSubRule	<i>the submission has to be done after the author registration</i>
RegAssRule	<i>the assignment has to be done after reviewer registration</i>
PartAssRule	<i>the assignment has to be done after partitioning</i>
AssRevRule	<i>write review after the assignment</i>
UserInfRule	<i>user can access & modify only his own information</i>
ReviewerInfRule	<i>reviewer can access & modify only his own information</i>
AuthorInfRule	<i>author can access & modify only public information of owned paper(s)</i>
MatchRule	<i>papers can be partitioned according to their keywords</i>
SubInfRule	<i>send paper only before deadline submission</i>
AutRevRule	<i>PC-Member/Reviewer cannot review his own papers</i>
ReviewRule	<i>PC-Member/Reviewer cannot access private information about owned papers</i>
WebAccessRule	<i>access to the system must be authorised</i>

Fig. 53 Rule table $(C)Rt_t$

Space	Description
S-place	<i>the space where resources have to be allocated</i>

Fig. 54 Space table $(C)S_t$

Detailed Designer

The Detailed Designer is responsible for

- Mapping the MMMElements of the Architectural Design to the MMMElements of the Detailed Design
- Identifying the most suitable system architecture among all the possibilities provided in the Architectural Design step
- Assigning roles to agents
- Assigning actions to individual artifacts
- Assigning roles to societies
- Assigning resources to environmental artifacts
- Assigning resources to aggregate
- Assigning operations to environmental artifacts
- Assigning rules to artifacts
- Designing artifacts usage interfaces

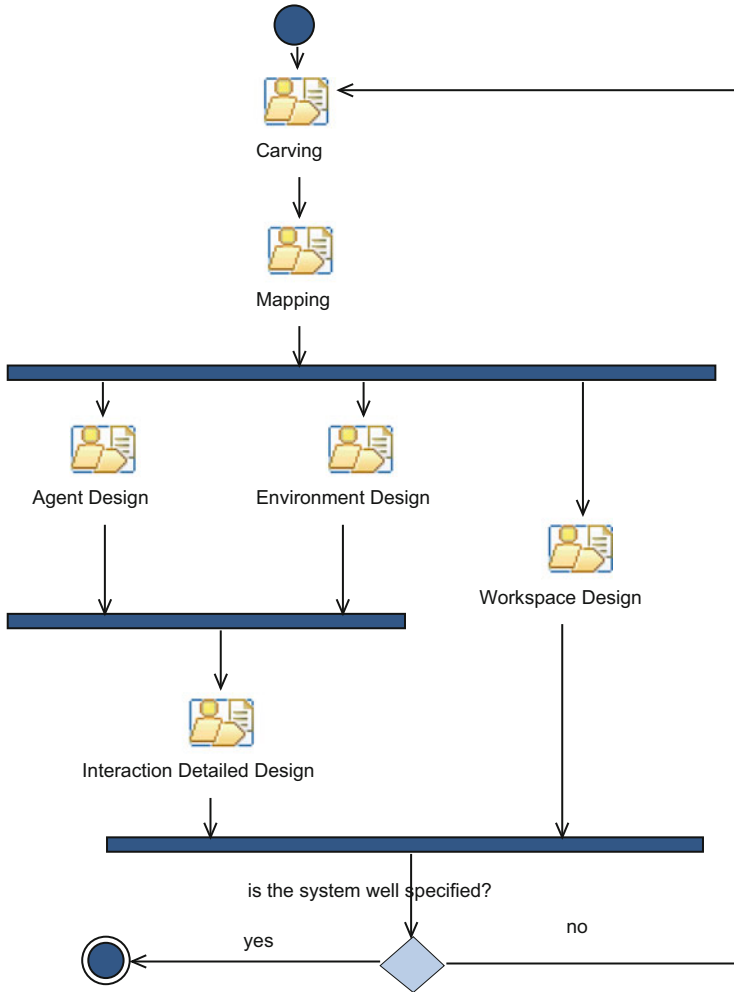


Fig. 55 The detailed design process

- Assigning interactions to uses and designing the specific protocols
- Assigning interactions to manifests and designing the specific protocols
- Assigning interactions to speakTo and designing the specific protocols
- Assigning interactions to linkedTo and designing the specific protocols
- Assigning spaces to workspaces and designing them

2.4.2 Activity Details

Carving Activity

The Carving activity is composed of the following tasks:

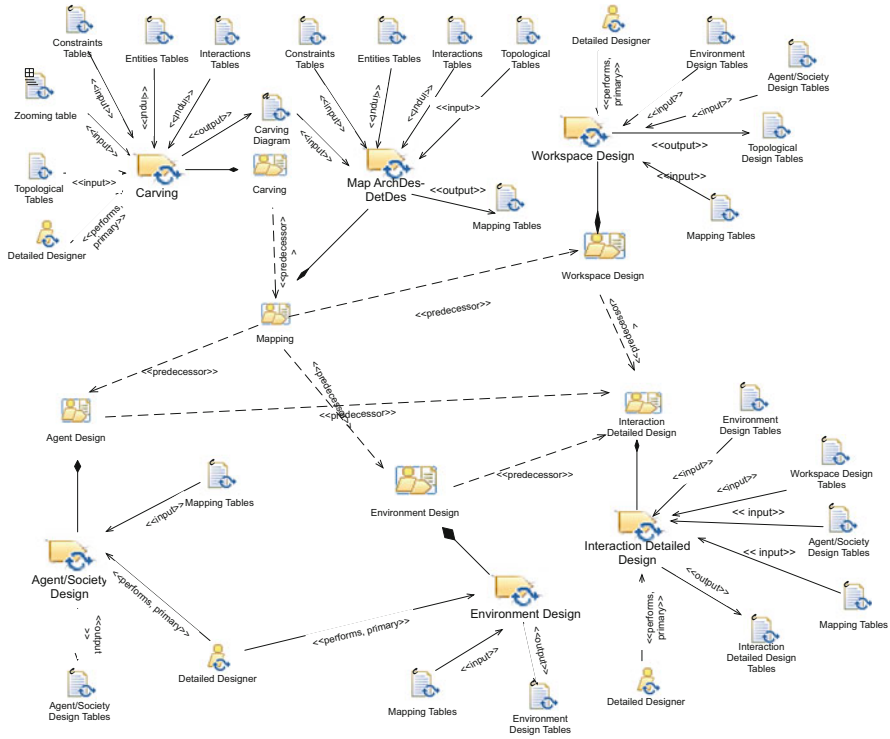


Fig. 56 The detailed design flow of activities, roles, and work products

Activity	Task	Task description	Role involved
Carving	Carving	For each entity the appropriate layer of representation is chosen	Detailed Designer (perform)

The flow of tasks inside the Carving activity is reported in Fig. 57.

Mapping Activity

The Mapping activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Mapping	Map ArchDes-DetDes	Mapping of the MMMElements defined in Architectural Design to Detailed Design MMMElements so as to generate the initial version of the Detailed Design models	Detailed Designer (perform)

The flow of tasks inside the Mapping activity is reported in Fig. 58.

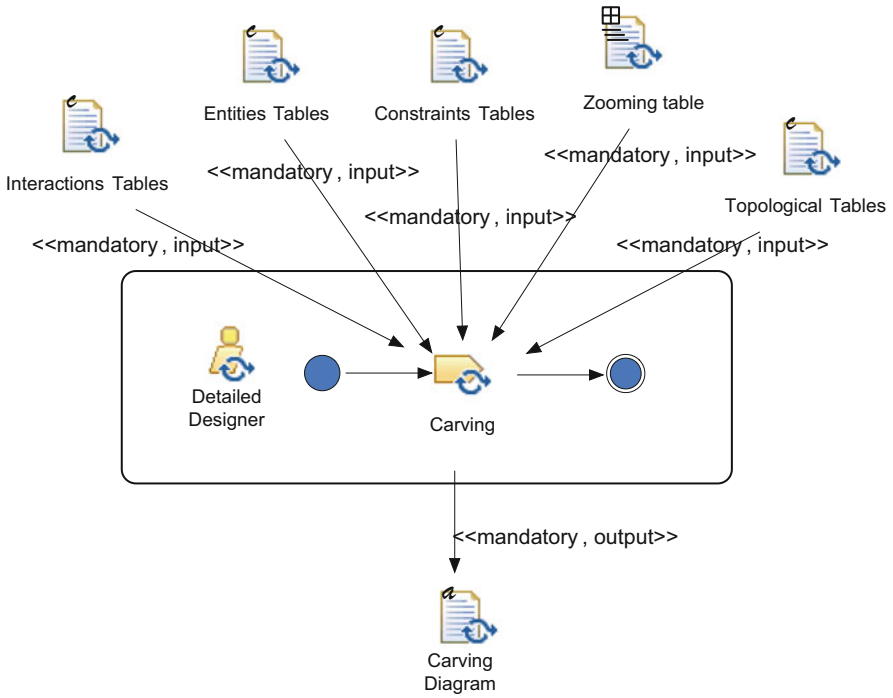


Fig. 57 The flow of tasks in the carving activity

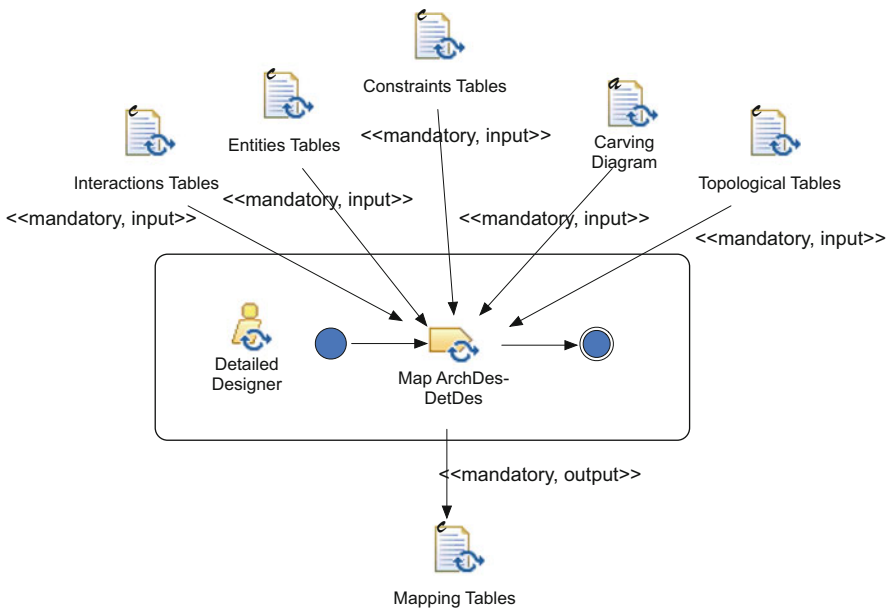


Fig. 58 The flow of tasks in the mapping activity

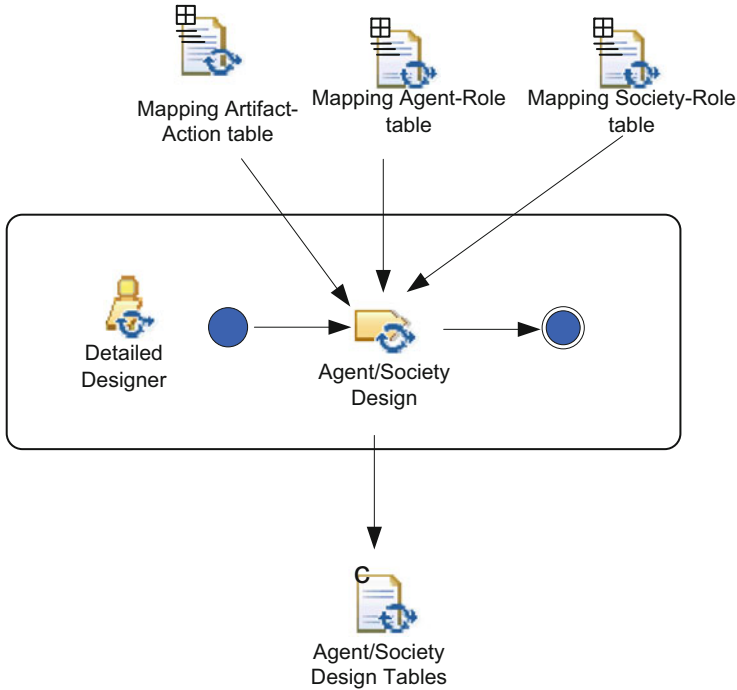


Fig. 59 The flow of tasks in the agent design activity

Agent Design activity

The Agent Design activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Agent Design	Agent/Society Design	<i>Design of Agents and Societies</i>	Detailed Designer (<i>perform</i>)

The flow of tasks inside the Agent Design activity is reported in Fig. 59.

Environment Design Activity

The Environment Design activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Environment Design	Environment Design	<i>Design of Artifacts and Aggregates</i>	Detailed Designer (<i>perform</i>)

The flow of tasks inside the Environment Design activity is reported in Fig. 60.

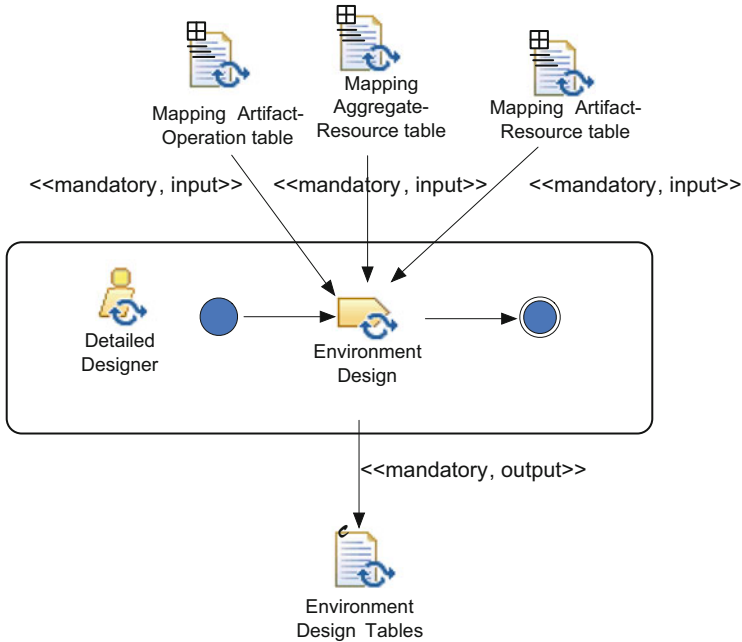


Fig. 60 The flow of tasks in the environment design activity

Interaction Detailed Design Activity

The Interaction Detailed Design activity is composed of the following tasks:

Activity	Task	Task description	Role involved
Interaction Detailed Design	Interaction Detailed Design	<i>Design of the interaction protocols for Uses, Manifests, SpeakTo and LinkedTo identified in the carving</i>	Detailed Designer (perform)

The flow of tasks inside the Interaction Detailed Design activity is reported in Fig. 61.

Workspace Design Activity

The Workspace Design activity is composed by the following tasks:

Activity	Task	Task description	Role involved
Workspace Design	Workspace Design	<i>Design of workspaces starting from spaces identified in the carving</i>	Detailed Designer (perform)

The flow of tasks inside the Workspace Design activity is reported in Fig. 62.

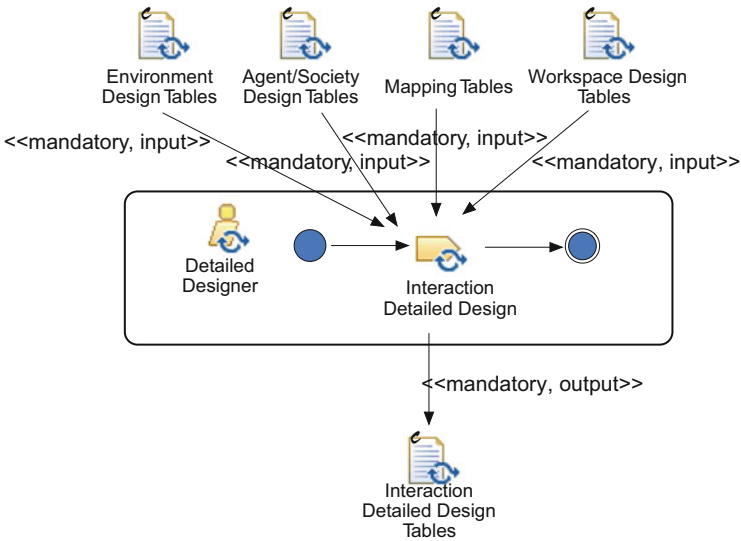


Fig. 61 The flow of task in the interaction detailed design activity

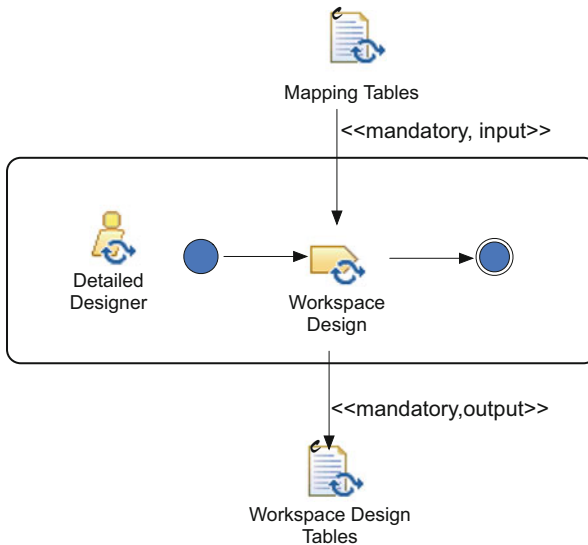


Fig. 62 The flow of task in the workspace design activity

2.4.3 Work Products

The Detailed Design step exploits several sets of tables: namely, Mapping Tables, Agent/Society Design Tables, Environment Design Tables, Interaction Detailed Design Tables and Workspace Design Tables. Figure 63 reports the relationships among the work products and the MMMElements of the Detailed Design step.

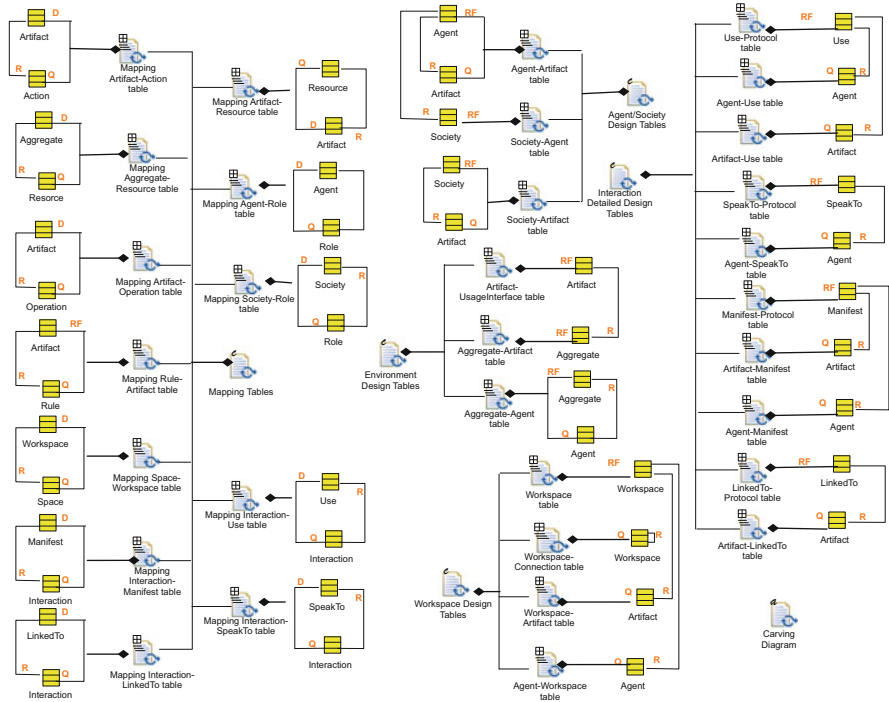


Fig. 63 The detailed design work products

Kinds of Work Products

Table 5 describes all the work products of the Detailed Design. In particular, the first entry is the outcome of the Carving Activity, the second set of work products is the outcome of the Mapping activity, the third set is the outcome of the Agent Design activity, the fourth set is the outcome of the Environment Design activity, the fifth set is the outcome of the Interaction Detailed Design activity and the last set is the outcome of the Workspace Design activity.

Carving Diagram

An example of the Carving Diagram for the conference management system is reported in Fig. 64.

Mapping Tables

Figures 65, 66, and 67 present some examples of the Mapping Tables for the conference management case study.

Agent/Society Design Tables

Figure 68 presents an example of the Agent/Society Design Tables for the conference management case study.

Table 5 Detailed Design work products kinds

Name	Description	Work product kind
Carving Diagram	<i>Diagram that shows the chosen system architecture</i>	Structured
Mapping Tables	<i>A composition of others tables that links the Architectural Design step with the Detailed Design step</i>	Composite (structured)
Mapping Agent-Role table (MAR_t)	<i>Specification of the mapping between roles and agents</i>	Structured
Mapping Society-Role table (MSR_t)	<i>Specification of the mapping between role and society</i>	Structured
Mapping Artifact-Action table ($MAAc_t$)	<i>Specification of the mapping between actions and individual artifacts</i>	Structured
Mapping Artifact-Resource table ($MARr_t$)	<i>Specification of the mapping between resources and artifacts</i>	Structured
Mapping Aggregate-Resource table ($MAGgR_t$)	<i>Specification of the mapping between resources and aggregate</i>	Structured
Mapping Artifact-Operation table ($MAROp_t$)	<i>Specification of the mapping between operations and environmental artifacts</i>	Structured
Mapping Artifact-Rule table ($MARRu_t$)	<i>Specification of the mapping between rules and the artifacts that implement and enforce them</i>	Structured
Mapping Artifact-Operation table (MSW_t)	<i>Specification of the mapping between spaces and workspaces</i>	Structured
Mapping Interaction-Use table (MIU_t)	<i>Specification of the mapping between interactions and uses</i>	Structured
Mapping Interaction-Manifest table (MIM_t)	<i>Specification of the mapping between interactions and manifests</i>	Structured
Mapping Interaction-SpeakTo table ($MISp_t$)	<i>Specification of the mapping between interactions and speakTos</i>	Structured
Mapping Interaction-LinkedTo table (MIL_t)	<i>Specification of the mapping between interactions and linkedTos</i>	Structured
<i>Agent/Society Design Tables</i>	<i>A composition of others tables that depicts agents, individual artifacts, and the societies derived from the carving operation</i>	Composite (structured)
Agent-Artifact table (AA_t)	<i>Specification of the individual artifacts related to each agent</i>	Structured
Society-Agent table (SA_t)	<i>Specification of the list of agents belonging to a specific society</i>	Structured

(continued)

Table 5 (continued)

Name	Description	Work product kind
Society-Artifact table (SAR_i)	<i>Specification of the list of artifacts belonging to a specific society</i>	Structured
<i>Environment Design Tables</i>	<i>A composition of others tables that depicts artifacts, aggregates, agents derived from the carving operation</i>	Composite (structured)
Artifact-UsageInterface table (AUI_i)	<i>Specification of the operations provided by each artifact</i>	Structured
Aggregate-Artifact table ($AggArt_i$)	<i>Specification of the list of artifacts belonging to a specific aggregate</i>	Structured
Aggregate-Agent table ($AggAge_i$)	<i>Specification of the list of agents belonging to a specific aggregate</i>	Structured
<i>Interaction Detailed Design Tables</i>	<i>A composition of others tables that concerns the design of interactions among entities</i>	Composite (structured)
Use-Protocol table (UP_i)	<i>Description of the protocols for each "use"</i>	Structured
Agent-Use table ($AgeU_i$)	<i>Specification of the "use" where each agent is involved</i>	Structured
Artifact-Use table ($ArtU_i$)	<i>Specification of the "use" where each artifact is involved</i>	Structured
SpeakTo-Protocol table (SP_i)	<i>Description of the protocols for each "speakTo"</i>	Structured
Agent-SpeakTo table ($AgeSp_i$)	<i>Specification of the "speakTo" where each agent is involved</i>	Structured
Manifest-Protocol table (MP_i)	<i>Description of the protocols for each "manifest"</i>	Structured
Agent-Manifest table ($AgeM_i$)	<i>Specification of the "manifest" where each agent is involved</i>	Structured
Artifact-Manifest table ($ArtM_i$)	<i>Specification of the "manifest" where each artifact is involved</i>	Structured
LinkedTo-Protocol table (LP_i)	<i>Description of the protocols for each "linkedTo"</i>	Structured
Artifact-LinkedTo table ($ArtL_i$)	<i>Specification of the "linkedTo" where each artifact is involved</i>	Structured
<i>Workspace Design Tables</i>	<i>A composition of others tables that describes the structure of the environment</i>	Composite (structured)
Workspace table ($(L)W_i$)	<i>Description of the workspaces</i>	Structured
Workspace-Connection table ($(L)WC_i$)	<i>Specification of the connections among the workspaces</i>	Structured
Workspace-Artifact table ($(L)WArt_i$)	<i>Specification of the allocation of artifacts in the workspaces</i>	Structured
Workspace-Agent table ($(L)WA_i$)	<i>Specification of the list of the workspaces that each agent can perceive</i>	Structured

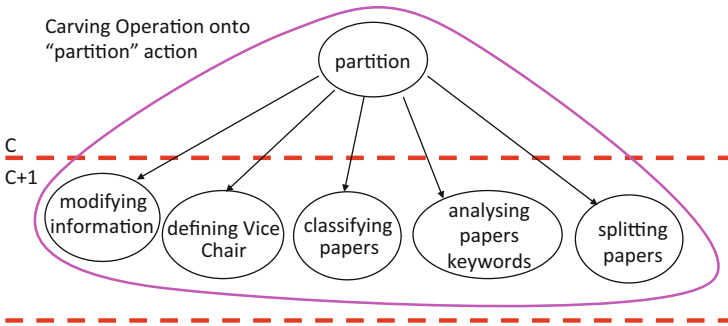


Fig. 64 Carving operation in the conference management system

Agent	Role
Conference Secretary Agent	Conference Secretary
Chair Agent	Chair
Author Agent	Author
Reviewer Agent	Reviewer
PC-Member Agent	PC-member

Fig. 65 Mapping agent-role table MAR_t

Artifact	Resource
Paper Artifact	Paper DB
People Artifact	People DB
Process Artifact	Process DB
Web Artifact	WebService

Fig. 66 Mapping artifact-resource table MAR_r

Artifact	Rule
User Artifact	UserInfRule, ReviewerInfRule
Paper Artifact	AuthorInfRule, MatchRule,
Process Artifact	RegSubRule, RegAssRule, PartAssRule, AssRevRule, SubInfRule
Review Artifact	AutRevRule, ReviewRule
Web Artifact	WebAccessRule

Fig. 67 Mapping artifact-rule table MAR_{ru}

Agent	Artifact
Conference Secretary Agent	Conference Secretary
Chair Agent	Chair Artifact
Author Agent	Author Artifact
Reviewer Agent	Reviewer Artifact
PC-Member Agent	PC-Member Artifact

Fig. 68 Agent-artifact table AA_t

Artifact	Usage Interface
Chair Artifact	read start up information, modify start up information, get info, login, partition, assignment
Author Artifact	login, registration, submit paper
Reviewer Artifact	login, registration, read paper, write review, download paper
PC-Member Artifact	login, read paper, write review, download paper
User Artifact	store user, get user, modify user
Paper Artifact	store paper, get paper, store classification, store partitioning, get partitioning, get assignment, store assignment, store review, check authors, check reviewer, check user, get review
Process Artifact	start conference process, get process, store process, next stage, deadline extension, update rule, read rule
Web Artifact	login
Review Artifact	check access to review information

Fig. 69 Artifact-usageInterface table *AUI_i*

Use	Protocol
ReadUserInfo	get user (id) information
ReadReviewInfo	check user ack get review (paperID) review
PaperInf- Interaction	check user ack get paper (paperID) paper
PartInf- Interaction	check user ack get partitioning partitioning info
SubInf- Interaction	get info info
AssInf- Interaction	check user ack get assignment assignment info
ReviewInf- Interaction	check access to review information ack get review (paperID) review
WebAccess- Interaction	login

Fig. 70 Use-protocol table *UP_i*

Environment Design Tables

Figure 69 presents an example of the Environment Design Tables for the conference management case study.

Interaction Detailed Design Tables

Figure 70 presents an example of the Interaction Detailed Design Tables for the conference management case study.

Workspace	Artifact
Wplace	Chair Artifact, Author Artifact, Reviewer Artifact, PC-Member Artifact, People Artifact, Process Artifact, Web Artifact, Review Artifact, Paper Artifact

Fig. 71 Workspace-artifact table WA_i

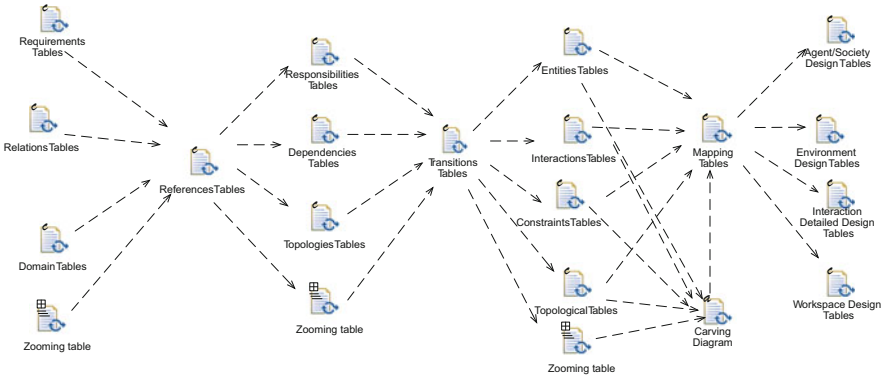


Fig. 72 The work products dependencies

Workspace Design Tables

Figure 71 presents an example of the Workspace Design Tables for the conference management case study.

3 Work Products Dependencies

Figure 72 describes the dependencies among the different SODA composite work products.

Acknowledgements This work was supported by the EU-FP7-FET Proactive project SAPERE—Self-aware Pervasive Service Ecosystems under contract no. 256873.

References

1. DPDF Working Group: FIPA design process documentation template. <http://fipa.org/specs/fipa00097/> (2011)
2. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: a roadmap to artefacts. In: Dikenelli, O., Gleizes, M.P., Ricci, A. (eds.) Engineering Societies in the Agents World VI. Lecture Notes in Artificial Intelligence, vol. 3963, pp. 49–62. Springer, Berlin (2006). doi:10.1007/11759683_4. 6th International Workshop (ESAW 2005), Kuşadası, Aydın, 26–28 Oct 2005. Revised, Selected & Invited Papers
3. Molesini, A., Omicini, A., Ricci, A., Denti, E.: Zooming multi-agent systems. In: Müller, J.P., Zambonelli, F. (eds.) Agent-Oriented Software Engineering VI. Lecture Notes in Computer

- Science, vol. 3950, pp. 81–93. Springer, Berlin (2006). doi:10.1007/11752660_7. 6th International Workshop (AOSE 2005), Utrecht, 25–26 Jul 2005. Revised and Invited Papers
4. Molesini, A., Nardini, E., Denti, E., Omicini, A.: Advancing object-oriented standards toward agent-oriented methodologies: SPEM 2.0 on SODA. In: Baldoni, M., Cossentino, M., De Paoli, F., Seidita, V. (eds.) 9th Workshop “From Objects to Agents” (WOA 2008) – Evolution of Agent Development: Methodologies, Tools, Platforms and Languages, pp. 108–114. Seneca Edizioni, Palermo (2008). <http://www.pa.icar.cnr.it/woa08/materiali/Proceedings.pdf>
 5. Molesini, A., Nardini, E., Denti, E., Omicini, A.: Situated process engineering for integrating processes from methodologies to infrastructures. In: Shin, S.Y., Ossowski, S., Menezes, R., Viroli, M. (eds.) 24th Annual ACM Symposium on Applied Computing (SAC 2009), vol. II, pp. 699–706. ACM, Honolulu (2009). doi:10.1145/1529282.1529429
 6. Molesini, A., Denti, E., Omicini, A.: Agent-based conference management: a case study in SODA. *Int. J. Agent Oriented Softw. Eng.* **4**(1), 1–31 (2010). doi:10.1504/IJAOSE.2010.029808
 7. Molesini, A., Omicini, A.: Documenting SODA: an evaluation of the process documentation template. In: Omicini, A., Viroli, M. (eds.) WOA 2010 – Dagli oggetti agli agenti. Modelli e tecnologie per sistemi complessi: context-dependent, knowledge-intensive, nature-inspired e self-*, CEUR Workshop Proceedings, vol. 621, pp. 95–101. Sun SITE Central Europe, RWTH Aachen University, Rimini (2010). <http://CEUR-WS.org/Vol-621/paper14.pdf>
 8. Object Management Group: Software & systems process engineering meta-model specification 2.0. <http://www.omg.org/spec/SPEM/2.0/PDF> (2008)
 9. Omicini, A.: SODA: societies and infrastructures in the analysis and design of agent-based systems. In: Ciancarini, P., Wooldridge, M.J. (eds.) Agent-Oriented Software Engineering. Lecture Notes in Computer Science, vol. 1957, pp. 185–193. Springer, Berlin (2001). doi:10.1007/3-540-44564-1_12. 1st International Workshop (AOSE 2000), Limerick, 10 June 2000. Revised Papers
 10. Omicini, A.: Formal ReSpecT in the A&A perspective. *Electron. Notes Theor. Comput. Sci.* **175**(2), 97–117 (2007). doi:10.1016/j.entcs.2007.03.006. 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA’06), CONCUR’06, Bonn, 31 Aug 2006. Post-proceedings
 11. Omicini, A., Ricci, A., Viroli, M.: *Agens Faber*: toward a theory of artefacts for MAS. *Electron. Notes Theor. Comput. Sci.* **150**(3), 21–36 (2006). doi:10.1016/j.entcs.2006.03.003. 1st International Workshop “Coordination and Organization” (CoOrg 2005), COORDINATION 2005, Namur, 22 April 2005. Proceedings
 12. Seidita, V., Cossentino, M., Gaglio, S.: Using and extending the SPEM specifications to represent agent oriented methodologies. In: Luck, M., Gómez-Sanz, J.J. (eds.) Agent-Oriented Software Engineering IX. Lecture Notes in Computer Science, vol. 5386, pp. 46–59. Springer, Berlin (2009). doi:10.1007/978-3-642-01338-6. 9th International Workshop (AOSE 2008), Estoril, 12–13 May 2008, Revised Selected Papers
 13. SODA: Home page. <http://soda.apice.unibo.it> (2009)
 14. Sommerville, I.: *Software Engineering*, 8th edn. Addison-Wesley, Reading (2007)

The Tropos Software Engineering Methodology

Mirko Morandini, Fabiano Dalpiaz, Cu Duy Nguyen,
and Alberto Siena

Abstract

The agent-oriented software engineering methodology *Tropos* offers a structured development process for the development of socio-technical systems. Such systems explicitly recognise the interplay between social actors (humans and organisations) and technical systems (software). *Tropos* adopts the state-of-the-art *i** requirements modelling language throughout the development cycle, giving special attention to the early phases of domain and requirements analysis. The system is modelled in terms of the goals of the involved actors and their social interdependencies, allowing for a seamless transition from the requirements to the design and potentially to an agent-oriented implementation. *Tropos* prescribes a limited set of domain-independent models, activities and artefacts, which can be complement with domain- and application-specific ones.

1 Introduction

Tropos is a comprehensive, agent-oriented methodology for developing socio-technical systems. Such systems explicitly recognise the existence of and interplay between technical systems (software) and social actors (humans and organisations).

M. Morandini (✉) • C.D. Nguyen
Fondazione Bruno Kessler, via Sommarive 18, 38123 Trento, Italy
e-mail: morandini@fbk.eu; cunduy@fbk.eu

F. Dalpiaz
Department of Computer Science, University of Toronto, 40 St. George Street, Toronto, ON,
Canada M5S 2E4
e-mail: dalpiaz@cs.toronto.edu

A. Siena
DISI, Università di Trento, via Sommarive 5, 38123 Trento, Italy
e-mail: siena@disi.unitn.it

Tropos adopts a requirement-driven approach to software development, recognising a pivotal role to the modelling of domain stakeholders (social actors) and to the analysis of their goals and interdependencies, before designing a technical system-to-be that supports the social actors. System design results in the specification of a multi-agent system. The methodology was first introduced in [4] and has been extended in different ways in the last decade. For instance, its modelling language has been adapted to support the analysis of crucial issues in distributed systems, such as trust, security and risks [1, 10].

Tropos encompasses all the software development phases, from *Early Requirements* to *Implementation and Testing*. *Tropos* introduces an *Early Requirements Analysis* phase in software development in which the analysts consider the stakeholders, their strategic goals and the organisational aspects of the system *as it is*, before the software system under design comes into play.

Throughout the design phases, the aim of *Tropos* is to understand and analyse the goals of the stakeholders and to operationalise them, obtaining the requirements for the software system to be built. *Tropos* was proposed to ease traceability issues in the software development life cycle by relying upon consistent concepts and artefacts throughout the development phases. As a result, problems that arise during software construction can be traced back to their original requirements, and an advanced analysis can be conducted to check whether a specific implementation satisfies a stakeholder's goals. Besides this, *Tropos* also provides guidelines for implementation and testing. Designed artefacts are used to generate agent-oriented prototypes and for the derivation of test cases, whose execution result can tell the fulfilment of stakeholders' goals (see [18]).

Several support tools for *Tropos* are available: *TAOM4e* covers the entire development cycle [13,21], *t2x* supports generating agent-oriented implementations [14], the T-Tool [8] provides model checking for *Tropos* specifications, the GR-Tool supports formal reasoning on goal models [9] and multi-agent planning enables the selection among alternative networks of delegations [5].

The *Tropos* methodology was applied to various research and industrial case studies. Research contributions, including several Ph.D. theses from various universities and research centers including, among others, FBK Trento and the Universities of Trento, Toronto, Louvain, Haifa and Recife, consolidated and extended the *Tropos* methodology in several directions, covering topics such as security, norms, risks, agent testing, adaptive systems, socio-technical systems, traceability, services, formal analysis and model interchange. Furthermore, various empirical studies conducted on *Tropos*, its extensions and concurrent methodologies, demonstrated its applicability in different domains [6, 11, 16].

This chapter provides guidance on how to use *Tropos* to develop a multi-agent system (MAS), performing analysis and design activities, generating code and performing testing on it, with the support of a set of tools. Moreover, it enables comparison with other tool-supported AOSE methodologies through a description of the main steps of these activities and of excerpts of the resulting artefacts, with reference to a common case study, namely, the Conference Management System (CMS) case study. Following the IEEE FIPA standard XC00097A for agent-oriented

Tropos engineering process

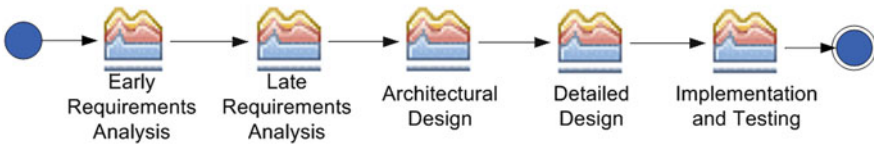


Fig. 1 The phases of the Tropos software engineering process

software engineering, first the methodology life cycle and meta-model are presented. Then, each phase of the methodology is presented by describing participating roles, included activities and resulting work products. Finally, the dependencies between the work products are outlined.

2 The Tropos Process Life Cycle

The software development process in *Tropos* consists of five subsequent phases, shown in Fig. 1:

- *Early Requirements Analysis* is concerned with understanding and modelling the existing organisational setting where the system-to-be will be introduced (the so-called “as-is” setting). The organisation is represented in terms of goal-oriented actors that socially depend one on another to fulfil their respective goals.
- *Late Requirements Analysis* starts from the output of the Early Requirements phase and introduces the system-to-be in the organisational setting.
- *Architectural Design* defines the system-to-be’s overall architecture in terms of interacting subsystems (agents). These agents are those to be implemented.
- *Detailed Design* further refines the system specification. It defines the functionalities to be implemented in each agent as well as the interaction protocols.
- *Implementation and Testing* are concerned with the actual development of the system agents and verifying whether they operate and interact as specified, respectively.





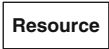
Even though not expressly shown in the figure, backwards iterations are possible and often needed, since the analysis carried out in a phase can provide feedback for the refinement of a previous phase. For example, while modelling the system-to-be in the late requirements phase, additional actors in the organisational setting may be identified. Their analysis is carried out through another iteration of the early requirements phase.

The *Tropos* software engineering process is model-based. Diagrammatic requirements and design models are used and refined throughout the process. These models are created using a variant of the conceptual modelling language *i** [24]. Modelling activities are central to the first four phases in the software development process, from early requirements analysis to detailed design. The basic concepts of the goal-oriented modelling language (see Sect. 3) are those of actor, goal, plan and social dependency. Goal modelling is complemented by UML activity and sequence diagrams, in the Detailed Design.

Goal modelling can be performed using *i** modelling tools such as *TAOM4e* [13], which implements the *Tropos* metamodel presented in Sect. 3, provides explicit support for the different modelling phases and is able to derive skeletons of a BDI-based agent implementation from goal models.

3 The Tropos Metamodel and Language

The following table describes the key concepts in the *Tropos* language and their graphical notation. The definition of the concepts is summarised from [4, 20].

Actor	It models an entity that has strategic goals and intentionality within the system or the organisational setting. It represents a physical, social or software agent as well as a role or position.	
Goal	It represents actors' strategic interests. We distinguish <i>hard goals</i> (often simply called <i>goals</i>) from <i>softgoals</i> . The latter have no clear-cut definition and/or criteria for deciding whether they are satisfied or not, and are typically used to describe preferences and quality-of-service demands.	 
Plan	It represents, at an abstract level, a way of doing something. The execution of plan can be a means for satisfying a goal or for <i>satisficing</i> (i.e. sufficiently satisfying) a softgoal.	
Resource	It represents a physical or an informational entity.	
Dependency	It is specified between two actors to indicate that one actor depends, for some reason, on the other in order to attain some goal, execute some plan, or deliver a resource.	
Capability	It represents both the <i>ability</i> of an actor to perform some action and the <i>opportunity</i> of doing this. These concepts are represented by a plan, together with the link to the goal which is the means to execute the plan, and eventual positive or negative contribution links to softgoals, which describe the opportunity of executing this plan in favour of alternative ones.	

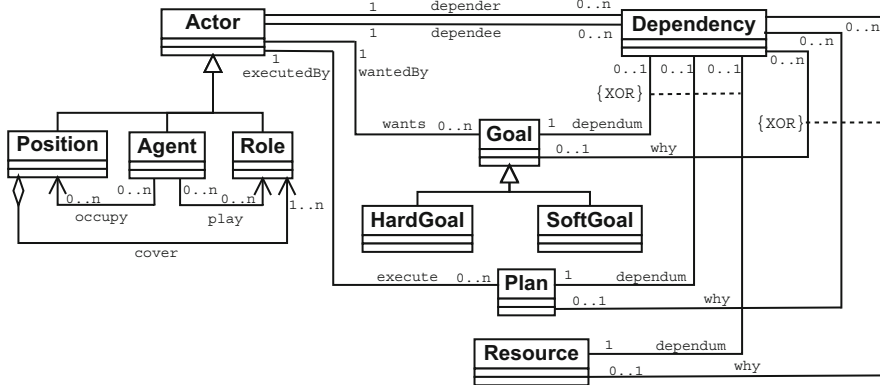


Fig. 2 The UML class diagram specifying the strategic part of the *Tropos* metamodel

The *Tropos* metamodel has been specified in [23]. Here we recall the concepts and relationships of the language defined in the metamodel.

Concerning the concepts related to the *Tropos* actor diagram (as shown in Fig. 2), a “position” can *cover* 1 . . . n roles, whereas an “agent” can *play* 0 . . . n “roles” and can *occupy* 0 . . . n “positions”. An “actor” can have 0 . . . n “goals”, which can be both hard and softgoals and are wanted by 1 actor.

An actor *dependency* is a quaternary relationship relating two actors, the depender and the dependee, by a dependum, which can be a goal, a plan or a resource and describes the nature of the relationships between the two actors. The *why* relationship between the dependum and another concept (goal, plan or resource) in the scope of the depender actor allows to specify the reason for the dependency.

Figure 3 shows the concepts of the *Tropos* goal diagram. The distinctive concept of goal is represented by the class *Goal*. Goals can be analysed, from the point of view of an actor, by *Boolean decomposition*, *Contribution analysis* and *Means-end analysis*. *Decomposition* is a ternary relationship which defines a generic boolean decomposition of a root goal into subgoals, which can be an *AND*- or an *OR-decomposition* specified via the attribute *Type* in the class *Boolean Decomposition* specialisation of the class *Decomposition*. *Contribution analysis* is a ternary relationship between an *actor*, whose point of view is represented, and two goals. Contribution analysis strives to identify goals that can contribute positively or negatively towards the fulfilment of other goals (see association relationship labelled *contribute* in Fig. 3). A contribution can be annotated with a qualitative metric, as proposed in [7], denoted by +, ++, -, --. In particular, if the goal g_1 contributes positively to the goal g_2 , with metric ++ then if g_1 is satisfied, so is g_2 . Analogously, if the plan p contributes positively to the goal g , with metric ++, this says that p fulfils g . A + label for a goal or plan contribution represents a partial, positive contribution to the goal being analysed. With labels --, and - we have the dual situation representing a sufficient or partial negative contribution towards the fulfilment of a goal. The *Means-end relationship* is also a ternary relationship

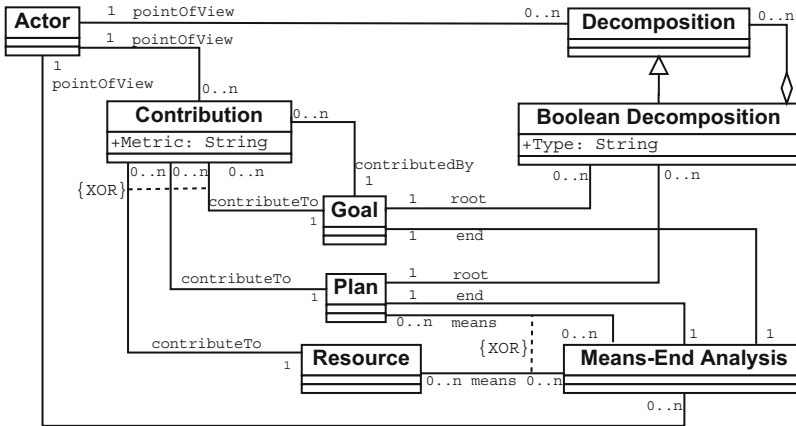


Fig. 3 The UML class diagram specifying the concepts related to the goal diagram in the *Tropos* metamodel

defined among an *Actor*, whose point of view is represented in the means-end analysis, a goal (the end), and a *Plan*, *Resource* or *Goal*, representing the means which will be able to satisfy that goal, i.e. the *operationalisation* of the goal.

Plan analysis in *Tropos* is specified in Fig. 3. *Means-end analysis* and *AND/OR decomposition*, defined above for goals, can also be applied to plans. In particular, AND/OR decomposition allows for modelling the detailed plan structure.

4 Early Requirements Phase

The Early Requirements (ER) phase (Figs. 4 and 5) concerns the understanding of the organisational context within which the system-to-be will eventually function. During early requirements analysis, the requirements engineer identifies the domain stakeholders and acquires through them domain knowledge about the organisational setting. The organisational setting is further detailed by eliciting and elaborating the needs, preferences and responsibilities of the stakeholders. This domain knowledge is captured in ER models describing social actors, who have goals and depend on each other for goals to be fulfilled, plans to be performed and resources to be furnished.

4.1 Process Roles

Two roles are involved in this phase: Requirements Analysts and Domain Stakeholders.

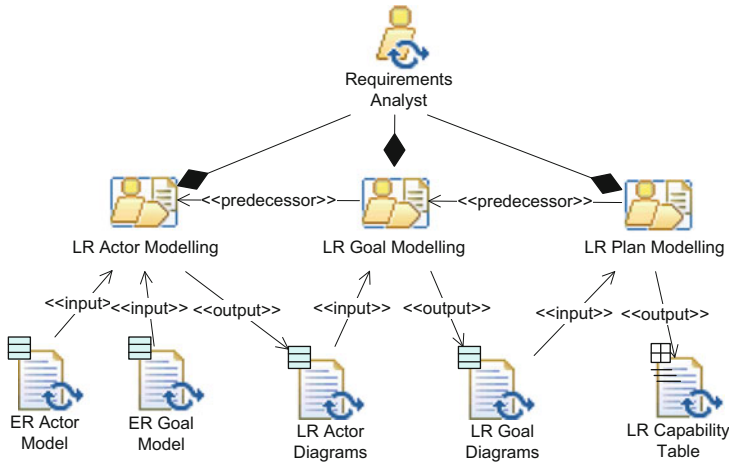


Fig. 4 The structure of the Early Requirements phase described in terms of activities, involved roles and work products

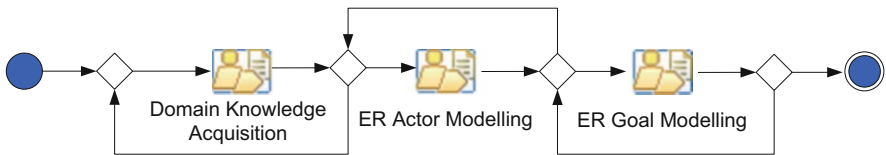


Fig. 5 Flows of activities in the Early Requirements phase

4.1.1 Requirements Analyst

Requirements Analysts gather knowledge about the domain and transform them into models of early requirements. They are in charge of

- Analysing domain documents
- Setting up interviews and focus groups with Domain Stakeholders
- Preparing scenarios
- Gathering information elicited during interviews and focus groups
- Producing ER Actor models
- Producing ER Goal models

4.1.2 Domain Stakeholder

Domain stakeholders own the knowledge about the domain. They are in charge of

- Providing domain knowledge to Requirements Analysts
- Participating to interviews and focus groups
- Informally validating the requirements models

4.2 Activity Details

4.2.1 Domain Knowledge Acquisition

Domain Knowledge acquisition consists in gathering knowledge from domain stakeholders, by analysing documents, acquiring information by setting up stakeholder interviews or focus groups, modelling stakeholder knowledge and producing scenarios. The output of this activity consists of a domain documentation that will be the basis for the next two activities in this phase.

In the CMS scenario, in this phase the domain is investigated to capture its fundamental organisational settings, such as domain actors and their responsibilities. A guide to start building the early requirements model is given by the following analysis questions: Who are the stakeholders in the domain? What are their goals and how are they related to each other? What are there strategic dependencies between actors for goal achievement? The answers to these questions contain information about stakeholders such as the program committee, the program chair, the paper submission and peer reviewing mechanism, and so on. They allow, for example, to insert the publisher as a major stakeholder in the domain, and to demote the vice chair as not relevant with respect to the CMS mechanisms.

4.2.2 ER Actor Modelling

During early requirements actor modelling, the stakeholders, their desires, needs and preferences are modelled in *Tropos* in terms of actors, goals and actor dependencies. The stakeholders' goals are then identified and, for every goal, the analyst decides on the basis of the domain documentation, if the goal is achievable by the actor itself or if the actor has to delegate it to another actor, revealing a dependency relationship between the two actors. This activity produces as output an ER actor model.

The CMS domain is modelled in terms of its main stakeholders (actors), as shown in Fig. 6: the papers' authors, modelled as actor Author; the conference's program committee and its chair—the PC PC Chair actors respectively—papers reviewers, modelled as the actor Reviewer and the proceedings publisher, actor Publisher. Dependency relationships between actors are identified, such as in the case of the dependency between Author and PC for the achievement of the goal Publish proceedings. An analogous analysis can be carried out for the domain softgoals and resources, according to the *Tropos* modelling process.

4.2.3 ER Goal Modelling

The early requirements goal modelling activity is intended to model the goals of each given actor, producing a goal model that represent the actor's rationale. This activity includes a modelling algorithm comprised by three steps:

- Refine. Goals are refined from a higher abstraction level (typically, the top-level strategic goals coming form the actor model) into more fine-grained goals. The refinement can include an AND (i.e. decomposition) and OR (i.e. alternative) relations among the refining goals.

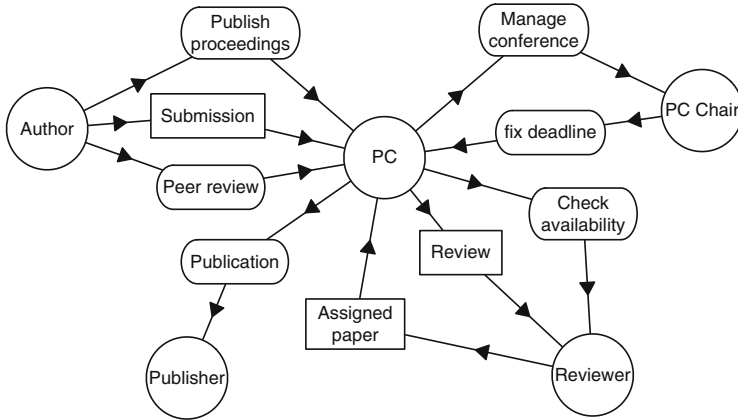


Fig. 6 The early requirements actor diagram for the Conference Management System domain

- Delegate. Goals that are desired by an actor but are able to be satisfied by another actor are delegated from the former to the latter.
- Contribute. Contribution relations are established between goals whenever the achievement of a certain goal helps in the achievement of another goal or softgoal.

Additional stakeholder needs and preferences can be modelled in this activity by means of adding new goals and softgoals, and dependency links representing social relationships. Early requirements analysis consists of several refinements of the models identified so far, and, for example, further dependencies can be added through a means-ends analysis of each goal.

In Fig. 7, an early requirements goal diagram is shown. This diagram represents a (partial) view on the model. Only two actors of the model, PC and PC Chair, are represented with two goal dependencies, Manage conference and Fix deadlines. The goal Manage conference is analysed from the point of view of its responsible actor, the PC Chair, through an AND- decomposition into several goals: Get papers, Select papers, Print proceedings, Nominate PC and Decide deadlines. Moreover, softgoals can be specified inside the actor goal diagram, with their contribution relationships to/from other goals (see, e.g., the softgoal Conference quality and the positive contribution relationship from the softgoal Better quality papers).

4.3 Work Products

Table 1 shows the work products for the Early Requirements phase. The relationships of these work products with the *Tropos* meta-model elements are described in Fig. 8. Since these relationships are valid for all goal modes (including goal and actor diagrams and the capability table) throughout the *Tropos* process, the figure does not attribute the work products to a specific phase.

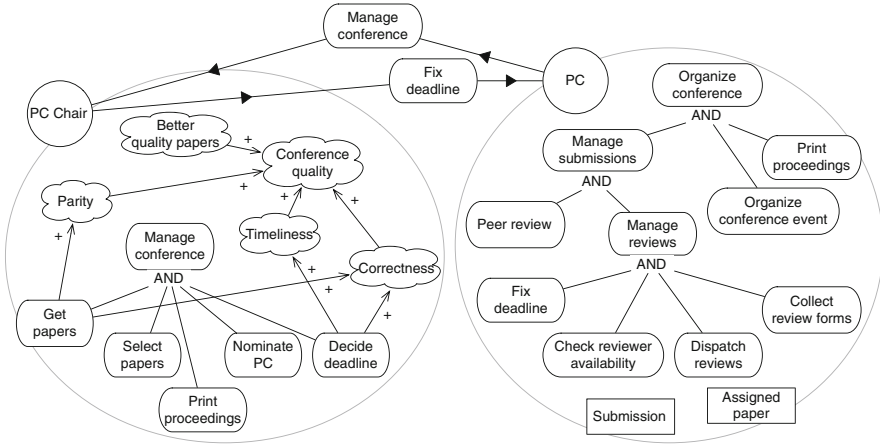


Fig. 7 Early requirements of CMS: goal diagram

Table 1 Work products for the Early Requirements phase

Name	Description	Work product kind
Domain Documentation	It contains the knowledge about the domain as gathered by requirements analysts.	Composite
ER Actor Diagram	It represents the actors identified in the domain and the strategic dependencies among them.	Structural
ER Goal Diagram	For each identified actor, it represents the gathered hard and softgoals that form the rationale of that actor, decomposed into sub-goals and operationalised through plans.	Structural

4.3.1 Work Products Examples

The ER Actor Diagram is illustrated by Fig. 6, which shows the actors in the Early Requirements phase as long as their social dependencies. Figure 7 illustrates an ER Goal Diagram for the system-to-be actor.

5 Late Requirements Phase

The Late Requirements phase is concerned with the definition of functional and non-functional requirements of the system-to-be. This is accomplished by treating the system as another actor (or a small number of actors representing system components) who are dependers or dependees in dependencies that relate them to external actors. The shift from early to late requirements occurs when the system actor is introduced and it participates in delegations from/to other actors. Here, modelling activities consist of introducing the system-to-be as a new actor with

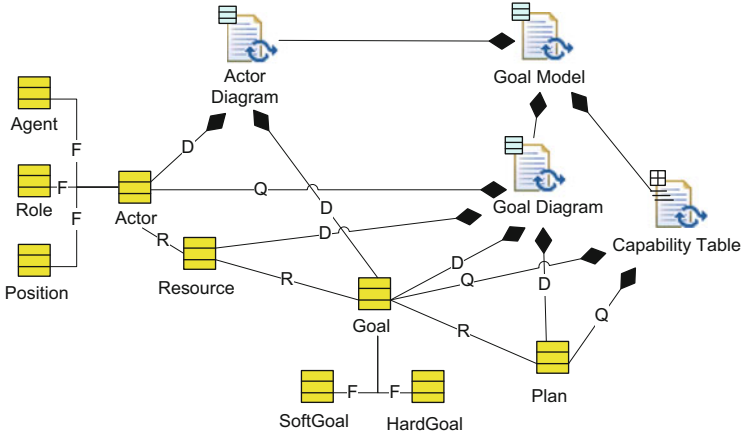


Fig. 8 General structure of a *Goal Model*, composed of actor- and goal diagrams and the capability table, in relation to the meta-model elements, as used in the Early and Late Requirements analysis phases and in the architectural and Detailed Design phases (*D* define, *R* relate, *F* refine, *Q* query)

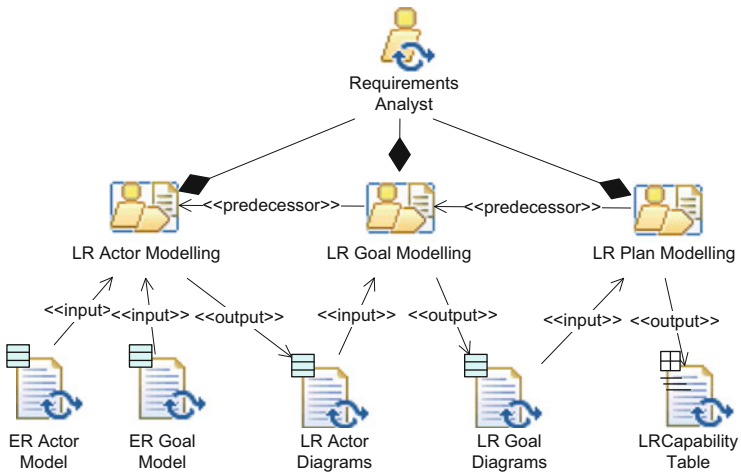


Fig. 9 The structure of the late requirements phase described in terms of activities, involved roles and work products

specific dependencies from/to stakeholder actors. These dependencies lead to the identification of system goals, which will be further analysed in terms of sub-goals, of plans providing means for their achievement, of positive and negative contributions to stakeholders’ preferences (typically modelled as softgoals).

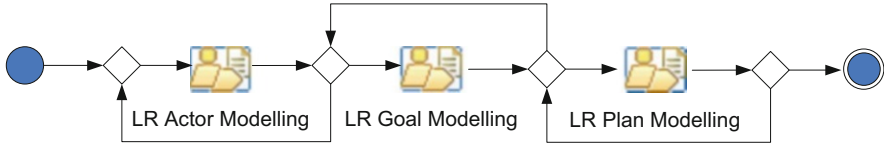


Fig. 10 The flow of activities in the Late Requirements phase

5.1 Process Roles

The Late Requirements phase is accomplished by Requirements Analysts.

5.1.1 Requirements Analyst

During this phase (Figs. 9 and 10), Requirements Analysts are in charge of

- Introducing the system-to-be as a new actor
- Delegating stakeholder goals to the system to be through dependency links
- Refining the top goals assigned to the system to be to leaf-level goals
- Identifying the capabilities of the system-to-be and model them as plans

5.2 Activity Details

5.2.1 LR Actor Modelling

During late requirements actor modelling, the system-to-be is introduced into the models as a new actor. Stakeholder needs and preferences are assigned to the system-to-be actor by establishing goal dependency links from stakeholder actors to the system actor. Top-level goals received by stakeholders form the functionalities which the system is responsible for.

A partial view of the LR actor model for the CMS domain is shown in Fig. 11 where the CMS actor is represented.

5.2.2 LR Goal Modelling

During late requirements goal modelling, the top-level goals of the system-to-be are refined into more fine-grained goals, resulting in system-specific goal models. This activity includes the same goal modelling algorithm used in ER goal modelling. The system's goals are AND or OR decomposed into more fine-grained goals. These goals are analysed *from the system actor perspective*.

In Fig. 12, the relative goal diagram is shown for the CMS domain. The goals Coordinate conference and Manage proceedings are decomposed in new sub-goals. Moreover, operative plans are specified and associated to the system goals as means to achieve them (means-ends relationships), such as in the case of the goal Manage decision.

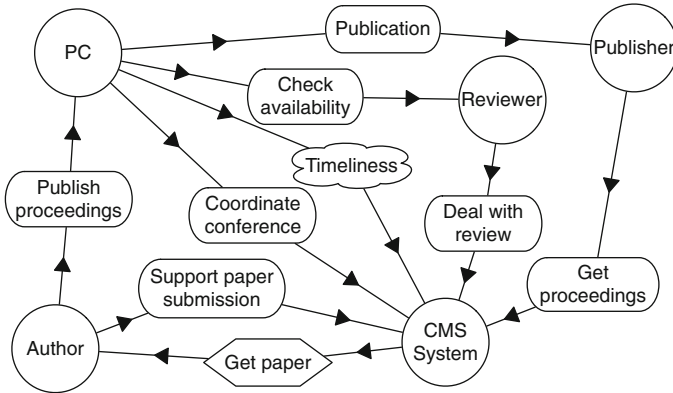


Fig. 11 Late requirements: actor diagram

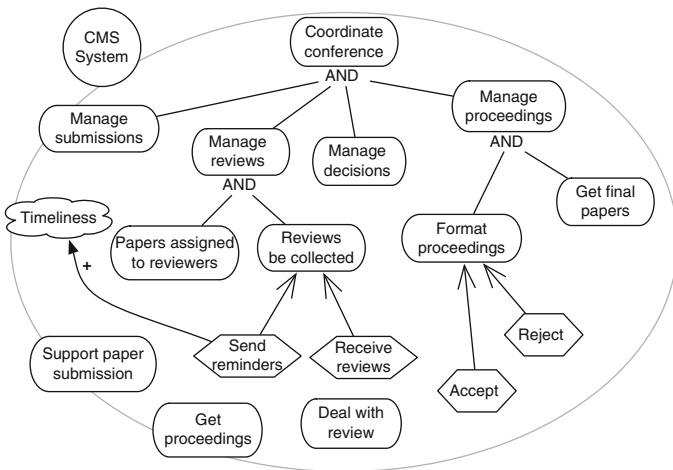


Fig. 12 Late requirements: goal diagram

5.2.3 LR Plan Modelling

During plan modelling, the goals of each actor are operationalised into concrete plans. If an actor owns the capability to fulfil a leaf-level goal by performing a specific activity, this is modelled in the goal model by a plan bound through a means-end relationship to the goal. Otherwise, the goal may require to be delegated to some other actor. The activity produces as output a modified version of the goal model and a capability table, which consists in a view on the goal model, highlighting the capabilities identified for each actor. In Fig. 12, the two plans **Accept** and **Reject** operationalise the goal **Manage decision**.

Table 2 Work products for the Late Requirements phase

Name	Description	Work product kind
LR Actor Diagram	It represents the system-to-be as a new actor of the model, and the functionalities assigned to it as functional dependencies from domain actor to the system actor.	Structural
LR Goal Diagram	It represents the operationalisation of the system-to-be strategic top goals in terms of tactical goals, including alternative ways to achieve them.	Structural
LR Capability Model	It represents the operationalisation of the system-to-be functional goals in terms of plans and contribution to softgoals.	Structured

Table 3 Example of an LR Capability Table

Goal	Capabilities
Format proceedings	Accept; reject
Reviews be collected	Send reminders, receive reviews
...	...

5.3 Work Products

Table 2 shows the work products for the Late Requirements phase. For the content of these work products, refer to the general Fig. 8.

5.3.1 Work Products Examples

The LR Actor Diagram is illustrated in Fig. 11, showing the actors involved in this phase as long as their social dependencies. Figure 12 illustrates an LR Goal Diagram for the system-to-be actor, while Table 3 displays an LR Capability Table.

6 The Architectural Design Phase

The *Architectural Design* (AD) phases drives the definition of the actual system architecture. It comprises both the overall multi-agent system structure and the detailed design for each single agent of the system. The phase is outlined in Fig. 13 (activity flow) and Fig. 14 (structural view), while details are provided in the following sub-sections.

The produced artefact consists of the system's overall structure, which is represented in terms of its sub-systems (*AD Goal Diagrams*), their inter-dependencies (*AD Actor Diagram*) and their capabilities (*AD Capability Table*). Adopting the multi-agent system paradigm, sub-systems are autonomous agents that communicate through message passing.

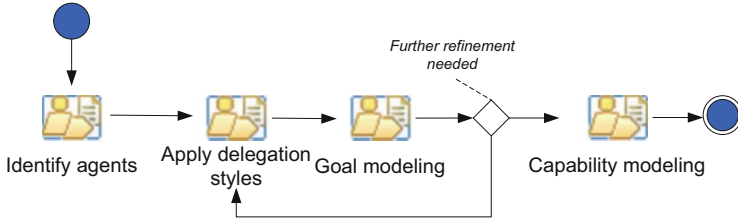


Fig. 13 Flow of activities in the *Tropos* Architectural Design phase

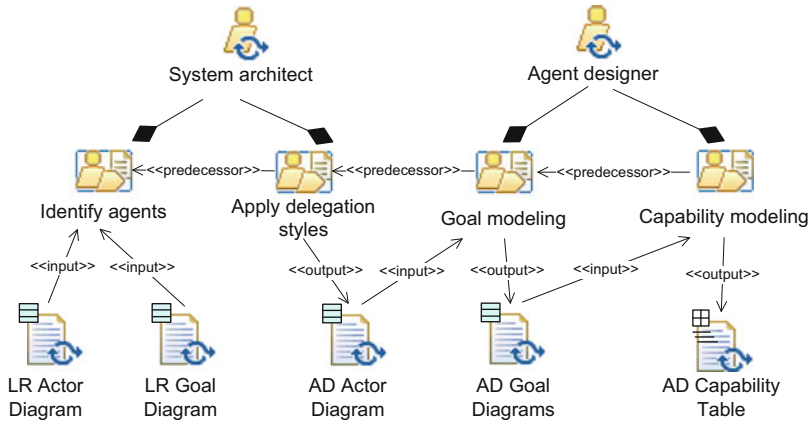


Fig. 14 The structure of the Architectural Design phase described in terms of activities, involved roles and work products

6.1 Process Roles

Two roles are involved: the *System Architect* and the *Agent Designer*.

6.1.1 System Architect

It is responsible for refining the system actor (*LR Goal Diagram*) by introducing system agents, which are delegated responsibility for the fulfilment of the system’s goals. While delegating responsibilities, the architect may apply organisational patterns [12] so as to structure the relationships between and responsibilities of agents.

6.1.2 Agent Designer

Given the *AD Actor Diagram* created by the *System Architect*, it is responsible for detailing the identified agents by applying goal modelling (as in the early and Late Requirements phases). Additionally, the agent designer is responsible for performing capability modelling, which defines how each agent will fulfil the goals it is responsible for.

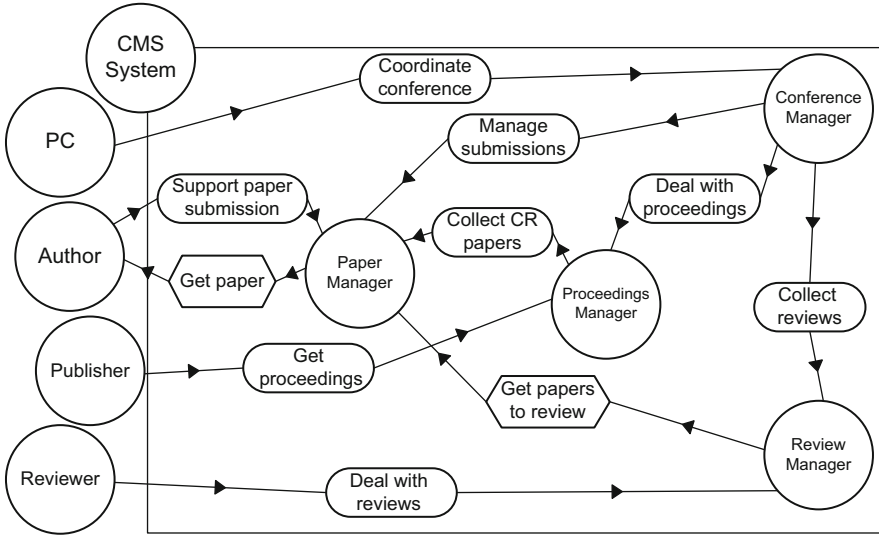


Fig. 15 Architectural design: CMS decomposition into agents

6.2 Activity Details

We detail each of the activities of the phase, showing the main involved artefacts, as well as examples from the CMS case study.

6.2.1 Identify Agents

The aim of this step is to split the complexity of the system, which is described in terms of high-level goals, into smaller components, easier to design, to implement and to manage. These components are autonomous agents.

6.2.2 Apply Delegation Styles

The identified agents take up responsibilities (through delegations) from the system actor. In other words, the goals of the system actor (*LR Goal Diagram*) are delegated to specific agents. During this activity, when applicable, organisational patterns [12] (e.g., structure-in-five or joint-venture) can be applied to decide how to relate the agents. Figure 15 displays the resulting architectural design diagram for the CMS actor. Analysing this actor’s goal model (see Fig. 12), the engineer should be able to extract a proper decomposition into agents.

In our example, we introduce four new agents. The **Conference Manager** manages the top-level goal *Coordinate conference*, delegated to the system by the program committee actor **PC**. The **Paper Manager** gets goal *Support paper submission* delegated from the domain actor **Author**. Further, some internal agents depend on it to manage submissions. To do this, the agent depends on authors to get papers. Similarly, the **Review Manager** and **Proceedings Manager** get goals delegated from the **Reviewer** and the **Publisher**, respectively.

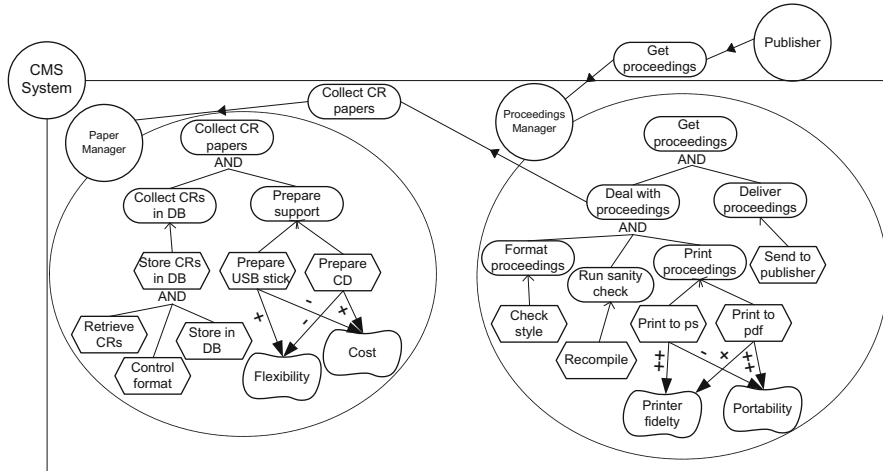


Fig. 16 Architectural design: simplified goal model of two agents of CMS

6.2.3 Goal Modelling

Once the sub-actors have been modelled and have been assigned responsibilities through goal and task delegation, each of them is analysed by building a goal model. For more details about this activity, look at the goal modelling activity in the Early and Late Requirements phases. Figure 16 shows an excerpt of the goal models for two agents, namely Paper Manager and Proceedings Manager. We focus on the analysis of the goal *Get proceedings* delegated from the Publisher agent, and the resulting dependency between the two agents. The delegated goal is AND-decomposed into sub-goals, which are either operationalised through a plan or further decomposed. To be achieved, one of the sub-goals, *Deal with proceedings*, causes the Proceedings Manager to depend on the Paper Manager for the goal *collect finals*.

Notice that there may exist alternative operationalisations of the system agents. For example, in Fig. 16, the Proceedings Manager can print proceedings either via postscript (ps) or pdf. These alternatives differ in terms of their contribution to softgoals. While the postscript option is better in terms of contribution to softgoal printer fidelity, the pdf option is better for what concerns portability. The decision about which option to select is delegated either to the Detailed Design phase or, envisioning an adaptive system, to runtime.

6.2.4 Capability Modelling

This activity details whether and how a specific agent is capable of achieving a goal by executing a specific plan. As suggested in [20], modelling capabilities include assessing both ability and opportunity. Ability means that the agent can carry out a plan without interacting with (delegating to) other agents. A greater opportunity means that the plan contributes better than others to the stakeholders'

Table 4 Work products for the Architectural Design phase

Name	Description	Work product kind
AD Actor Diagram	It shows the dependencies from the system actor to the agents as well as the dependencies between the agents.	Structural
AD Goal Diagrams	They represent the goal diagram for each of the identified and analysed agents.	Structural
Capability Table	It lists all agent capabilities along with their contributions to softgoals	Structured

Table 5 Example of an AD Capability Table

Goal	Ability	Contribution
Prepare support	Prepare USB stick	Flexibility (+); Cost (-)
Prepare support	Prepare CD	Flexibility (-); Cost (+)

preferences and QoS needs (i.e. to modelled softgoals). Plans can also be detailed by decomposing them in AND and OR to more concrete sub-plans. See, for instance, the AND decomposition of the plan store finals in DB into the sub-plans retrieve finals, control format, store in DB, in Fig. 16.

6.3 Work Products

Table 4 lists the work product types for the Architectural Design phase. The relationships of these work products, representing a *Tropos* goal model, with the meta-model elements, are described in Fig. 8.

6.3.1 Work Products Examples

The AD Actor Diagram is illustrated in Fig. 16, which shows the delegations between the system actor and other agents. Figure 15 compactly shows two AD goal diagrams for agents “Paper Manager” and “Proceedings Manager”. Table 5 illustrates an AD Capability Table.

7 Detailed Design

The Detailed Design (DD) phase is concerned with the specification of the capabilities of the software agents in the system and of the interactions taking place, focusing on dynamic and input–output aspects, leading to a detailed definition of how each agent needs to behave in order to execute a plan or satisfy a goal. This includes the detailed specification of the single functionalities composing the plans associated to each agent’s goals, the definition of interaction protocols and of the dynamics of the interactions occurring between agents and with systems and humans in the environment.

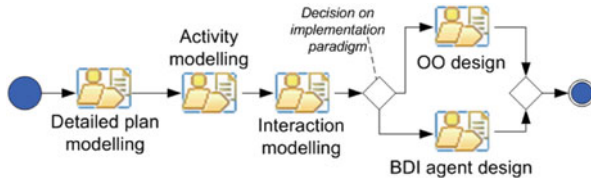


Fig. 17 Flow of activities in the *Tropos* detailed design phase. Iterations are not shown in the diagram, but possible starting at each activity

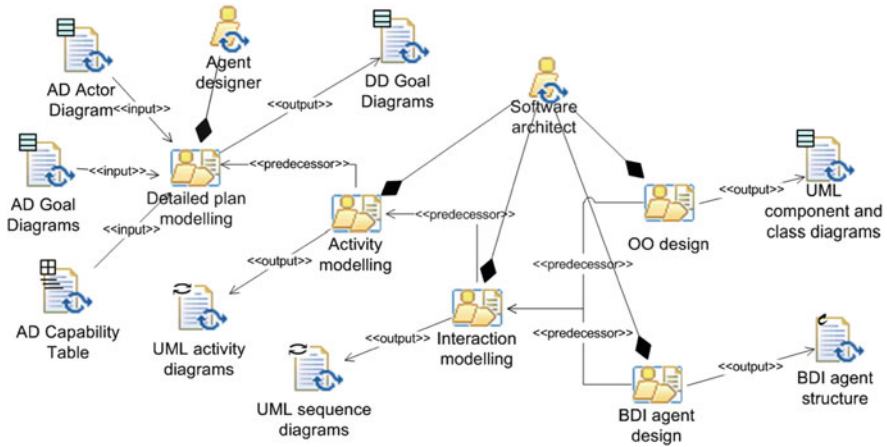


Fig. 18 The structure of the detailed design phase described in terms of activities, involved roles and work products. Please note that the internal use of work products in output of an activity as inputs of the subsequent activities is not shown explicitly, to keep the diagram concise

In this phase, also a decision on the implementation paradigm has to be made, either for going to a traditional object-oriented, or to an agent-based, goal-driven implementation. The activities in the phase are outlined in Fig. 17 while, Fig. 18 shows the internal structure of the phase.

7.1 Process Roles

Two roles are involved, the *Agent Designer* and the *Software Architect*.

7.1.1 Agent Designer

The Agent Designer, involved also in the AD phase, is responsible for detailing the goal models obtained from the previous step, decomposing the plans into AND/OR hierarchies, till arriving to the concrete functionalities that compose each plan.

7.1.2 Software Architect

It is responsible for the detailed design of the system as a whole and of the single functionalities that have to be implemented. He has to take a decision for the programming paradigm to follow and the implementation platform to use, and to detail the models according to its decisions.

7.2 Activity Details

7.2.1 Detailed Capability Modelling

The agent designer details the capabilities identified in the AD phase through plan modelling by AND/OR decomposition to sub-plans, until reaching a fine-grained level defining the single activities that need to be available in the system.

7.2.2 Activity Modelling

At this step, the software architect needs to take a first decision on the alternative capabilities and sub-plans which should be further specified and finally implemented, depending on the desired adaptability of the system and the affordable implementation effort. This selection should be made considering positive and negative contributions to softgoals, which represent preferences and quality-of-service demands, or by employing a requirements prioritisation technique. If the aim is to create an adaptive system, a higher number of alternatives should be detailed and eventually implemented. To model the execution workflow for the activities that a capability is composed of, UML activity diagrams are adopted. UML activity diagrams can be directly derived from *Tropos* plan decompositions by model transformation, and further elaborated by detailing the dynamic aspects of a capability (see an example in Fig. 19), including sequential and parallel workflows and alternative choices.

7.2.3 Interaction Modelling

The software architect analyses the interactions that take place, between agents in the system which need to be detailed for each activity and between the agents in the system and actors in its environment (including software agents and human actors). UML sequence diagrams (Fig. 19) are used for specifying interaction protocols and details for each agent interaction and for single activities (i.e. leaf-level plans).

7.2.4 Platform-Dependent Design

At this stage, a decision on the implementation paradigm has to be made either for going to a traditional object-oriented or to an agent-based, goal-driven implementation. Following the central idea of *Tropos* to keep the notions of actor/agent and goal throughout all phases to avoid conceptual gaps, an agent-oriented implementation is recommended. Agents are autonomous entities with independent threads of control that can interact with each other to reach their goals. Various frameworks are available, supporting these implementation concepts. An object-oriented implementation

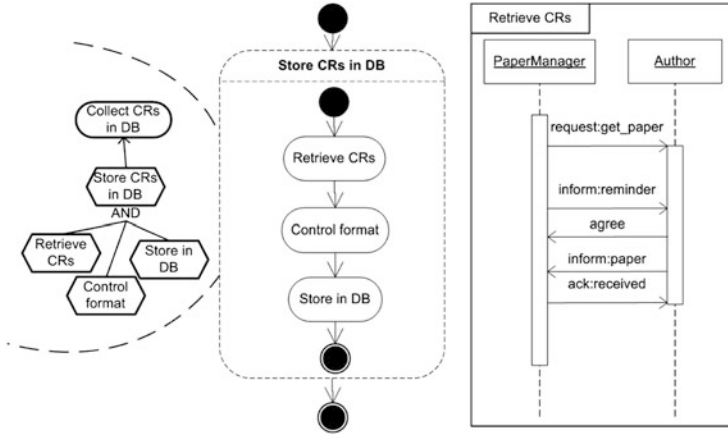


Fig. 19 Activity and interaction modelling: goal diagram with a plan decomposition for a part of the CMS Paper Manager Agent, detailed in UML activity and sequence diagrams

could be of benefit for performance-critical systems and for a better integration with existing software. In the following, we briefly touch on an object-oriented design, while the specific implementation phase proposed for *Tropos* relies on an agent-oriented design.

Object-Oriented Design At this point, a traditional OO approach can be followed for the detailed design by using UML component and class diagrams for the description of the agents in the system and their relationships.

BDI Agents Design Aiming at an agent-oriented implementation, after having modelled the single activities and interactions to be carried out by an agent (i.e. the *capability level*), in this activity the behavioural aspects, called the *knowledge level* [20], are detailed. This is achieved by mapping the high-level goal model of an agent, including dependencies and contribution relationships, into a generic BDI (belief–desire–intention) agent structure, according to a mapping such as the one defined in [19]. This structure can be enriched defining goal types (maintenance, achievement), goal satisfaction and goal failure criteria [15]. Selecting the proposed agent development platform Jadex, the mapping to an Agent Definition File specification is tool-supported [14].

7.3 Work Products

The Detailed Design phase generates four work products, which in part depend on the chosen implementation architecture, out of the ones listed in Table 6.

Table 6 Work products created during the Detailed Design phase

Name	Description	Work product kind
DD Goal Diagram	It details the AD goal diagram of each agent, with the plans decomposed to sub-plans which represent the single functionalities that should be available.	Structural
UML Activity Diagram	It details a plan to the single activities it is composed of, and captures their temporal order.	Behavioural
UML Sequence Diagram	It explicitly shows the interactions between two software or human entities, possibly basing on interaction protocols, and defines the effects of an interaction.	Behavioural
UML Component and Class Diagrams	They illustrate the whole system by the use of UML diagrams, suitable if the target language is object-oriented.	Structural
BDI Agent Structure	It has a structure basing on the concepts of goal, plan and belief [3]. Adopting Jadex [22] as implementation platform, this structure is represented in an Agent Definition File.	Composite

7.3.1 Work Products Examples

Figure 19 illustrates the main work products of the Detailed Design phase. Left: an excerpt of a DD goal diagram with plan decomposition; middle: UML activity diagram for plan Store CRs in DB; right: UML sequence diagram detailing the interactions that need to take place in one of the leaf-level plans. Portions of a BDI agent structure are shown, implemented as Jadex Agent Definition File, on the right side of Fig. 22.

8 Implementation and Testing

The last phase in the *Tropos* development process includes coding and testing activities, leading to the deployment of the final software.

Tropos does not impose the use of a specific implementation platform. However, it recommends the adoption of an agent-oriented language with the notions of agent, goal and plan. In this way, conceptual gaps are reduced and the traceability of artefacts and decisions through the phases is simplified. Specifically, we describe an implementation on the *Jadex* [22] agent platform. Nevertheless, the activities described are general and apply in principle also to a development with other agent languages, such as Jack, Jason or 2APL. Also, non-BDI agent languages such as JADE can be used for the full implementation, mapping goals to the artefacts available in the language. The activities in this phase, outlined in Fig. 20, are typically executed in an iterative way and, to some degree, in parallel, to take

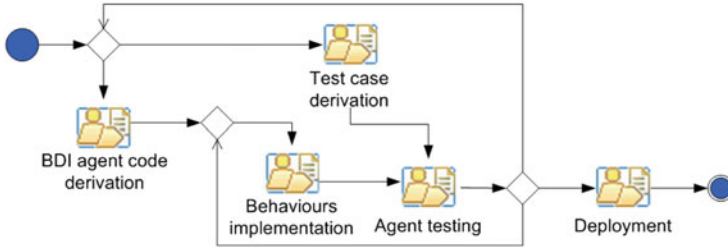


Fig. 20 Flow of activities for the implementation and testing phase

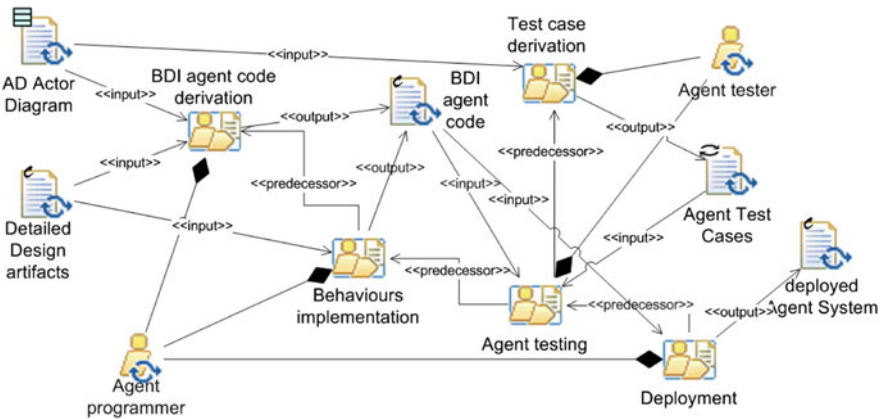


Fig. 21 SPEM model showing the structure of the implementation and testing phase with in input and output work products. The phase takes in input the AD actor diagram and the outputs of the DD phase

account of the revisions needed to reach a final software product. Figure 21 shows the in- and output work products and the involved actors.

8.1 Process Roles

Two roles are involved: the *Agent Programmer* and the *Agent Tester*.

8.1.1 Agent Programmer

The Agent Programmer is responsible for carrying out the implementation of the agent system, starting from goal models and UML diagrams. Typically, he also carries out the deployment of the software system.

8.1.2 Agent Tester

On the basis of goal models and sequence diagrams, the agent tester derives test cases, which are then executed on the software agents.

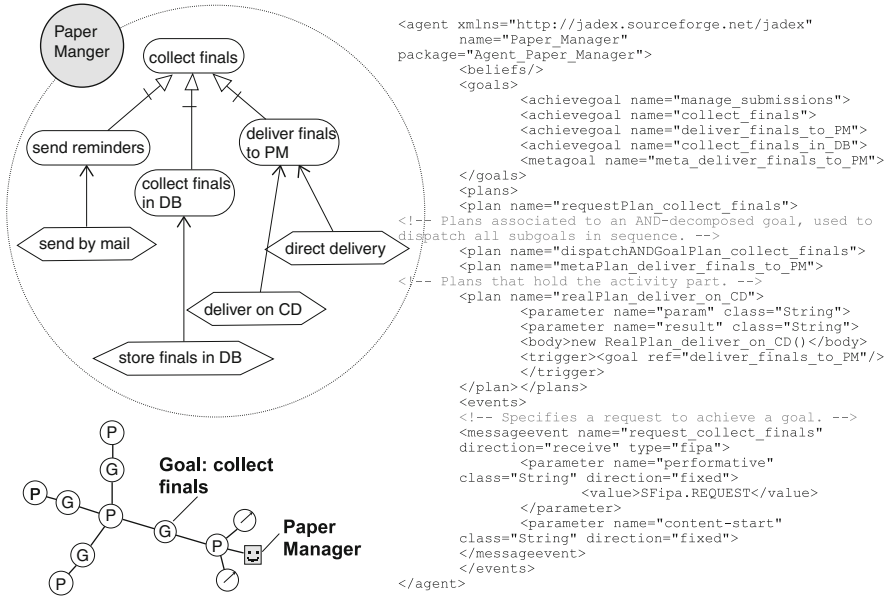


Fig. 22 Left: Simplified goal diagram for PaperManger modelled using the TAOM4e tool. Right: Part of the Jadex XML code generated with the t2x tool. Bottom: Example Jadex run-time agent instance with activated goals and plans, visualised with the Introspector tool provided by the Jadex platform

8.2 Activity Details

We detail each of the activities of the phase and explain the artefacts involved.

8.2.1 BDI Agent Code Derivation

The goal and UML models created in the DD phase and the AD actor diagram are the basis for the implementation of software agents. Selecting Jadex as a target platform and using the t2x tool, Jadex code can be generated basing on the BDI agent structures previously defined. The t2x tool analyses a GM exploring goal decomposition trees. The goal hierarchy is mapped to Jadex goals along with Java files containing the decomposition logic, while plans are implemented in Java files and connected to the relative goals by a triggering mechanism. The generated code implements the agent’s reasoning mechanisms needed to select correct plans at run-time to achieve desired goals defined in the agent’s AD goal model. It has to be customised adding the temporal and operational aspects defined in the detailed design UML models, and implementing the interaction protocols. FIPA standard agent interaction protocols such as Request and Contract Net are predefined. As an example, Fig. 22 shows part of the generated Jadex code in XML format of the agent Paper Manager from the CMS example. This fragment of code corresponds

to the *Tropos* goal model on the top-left side of the figure, and its reasoning trace at runtime is presented on the bottom-left corner of the figure.

8.2.2 Behaviours Implementation

With the DD activity and sequence diagrams in input, in this phase the *behaviours* of the agents are implemented. The behaviours realise the plans defined in a goal model to operationalise the goals. Behaviours can be implemented with OO concepts (e.g. in JAVA) or using specific concepts present in languages such as *JADE* [2].

8.2.3 Test Case Derivation

A systematic way of deriving test cases from goal-oriented specifications and techniques to automate test case generation and their execution has been introduced in [18]. Such approach considers different testing levels, from unit testing to acceptance testing, and different aspects of testing a multi-agent system that adopts *Tropos* design. Test cases are derived from the agent specifications (specifically, the AD actor diagram) with the aim to test and validate the interactions between agents (represented as dependencies in *Tropos*) and the achievement of goals, adopting domain-specific metrics. Test cases derivation is supported by a semi-automatic tool [17], providing a GUI-based editor to detail test scenarios and inputs.

In the case of the CMS, the *eCAT* testing tool [17] takes the architectural diagram in Fig. 15 as an input and generates a set of test suites for each agent.

8.2.4 Agent Testing

The agent testing activity consists of an automated testing of the agents in a virtual environment, observing the interactions with the environment and with the peer agents, while varying environment and inputs for each test case. This activity can be automated and parallelised by applying the testing framework proposed in [18].

8.3 Deployment

The BDI agent code, including the implemented behaviours, can be executed on the *Jadex* platform. Interactions and the goal achievement process can be visualised via tools provided by the platform. Regarding the present case study, code was generated for the two system agents *ProceedingsManager* and *PaperManager*. As an example, Fig. 22 shows an excerpt *Jadex* code in XML format for the agent *Paper Manager*. This fragment of code corresponds to the *Tropos* goal model on the top-left side of the figure, and its reasoning trace at runtime is presented on the bottom-left corner of the figure.

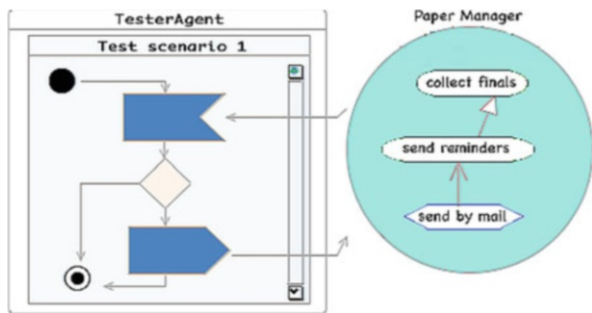
8.4 Work Products

Focusing on an agent-oriented implementation, the implementation and testing phase generates four main work products, listed in Table 7.

Table 7 Work products created during the Implementation and Testing phase

Name	Description	Work product kind
BDI agent code	It represents the executable agent code that exhibits the behaviour defined in the previous phases.	Composite
Agent test cases	They define a list of inputs and corresponding outputs (data, interactions, exhibited behaviours), to test the correctness of the single agent implementations and of the whole system. Defined in XML format, they can be used as input for automated testing agents.	Behavioural
Deployed agent system	It represents the running agent system.	Composite

Fig. 23 An example of a test case for the Paper Manager, specified using the eCAT editor



8.4.1 Work Products Examples

The BDI agent code depends on the selected implementation platform. In Jadex, it corresponds to an Agent Definition File, such as on the right side of Fig. 22, for each agent, in combination with Java code. The behaviour of the deployed agents can be visualised at runtime on the agent platform, such as in the lower left part of Fig. 22. Figure 23 depicts an example of a test case that checks the goal send reminders of the Paper Manager. The test case is specified using the eCAT editor; following the test scenario of the test case, the Agent Tester waits for a message from the agent under test (Paper Manager), checks for the content of the message and sends back a notification.

9 Work Product Dependencies

The work product dependency diagram in Fig. 24 describes the dependencies among the different work products created in the five development phases. Focusing on an agent-oriented implementation, an eventual object-oriented implementation is not considered.

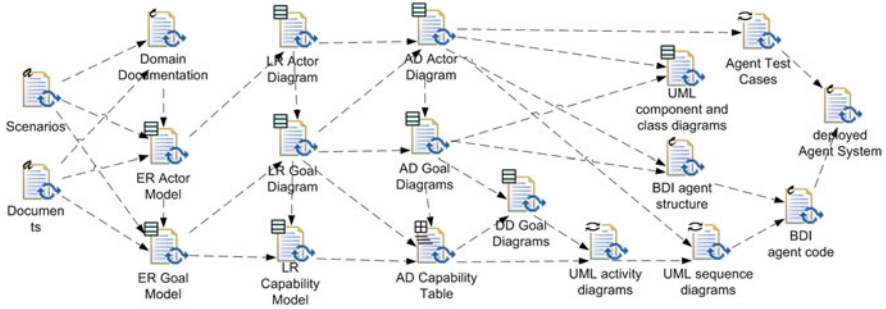


Fig. 24 Work product dependency diagram for the *Tropos* methodology

Acknowledgements The authors thank Angelo Susi (Fondazione Bruno Kessler) Anna Perini (Fondazione Bruno Kessler), and Paolo Giorgini (University of Trento) for their support.

References

1. Asnar, Y., Giorgini, P., Mylopoulos, J.: Goal-driven risk assessment in requirements engineering. *Requir. Eng.* **16**(2), 101–116 (2011)
2. Bellifemine, F., Poggi, A., Rimassa, G.: JADE: a FIPA compliant agent framework. In: *Practical Applications of Intelligent Agents and Multi-agents*, pp. 97–108, April 1999
3. Braubach, L., Pokahr, A., Moldt, D., Lamersdorf, W.: Goal representation for BDI agent systems. In: *Programming Multi-agent Systems*, pp. 44–65 (2004)
4. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: an agent-oriented software development methodology. *Auton. Agent Multi Agent Syst.* **8**(3), 203–236 (2004)
5. Bryl, V., Giorgini, P., Mylopoulos, J.: Designing cooperative IS: exploring and evaluating alternatives. In: *OTM Conferences (1)*, pp. 533–550 (2006)
6. Bryl, V., Dalpiaz, F., Ferrario, R., Mattioli, A., Villafiorita, A.: Evaluating procedural alternatives: a case study in e-voting. *Electron. Gov.* **6**(2), 213–231 (2009)
7. Chung, L.K., Nixon, B., Yu, E., Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer, Dordrecht (2000)
8. Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P.: Model checking early requirements specifications in Tropos. In: *IEEE International Symposium on Requirements Engineering*, pp. 174–181. IEEE Computer Society, Toronto (Aug 2001)
9. Giorgini, P., Mylopoulos, J., Sebastiani, R.: Goal-oriented requirements analysis and reasoning in the tropos methodology. *Eng. Appl. Artif. Intell.* **18**(2), 159–171 (2005)
10. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling security requirements through ownership, permission and delegation. In: *Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05)* (2005)
11. Hadar, I., Kuflik, T., Perini, A., Reinhartz-Berger, I., Ricca, F., Susi, A.: An empirical study of requirements model understanding: *Use Case vs. tropos* models. In: *Symposium on Applied Computing*, pp. 2324–2329 (2010)
12. Kolp, M., Giorgini, P., Mylopoulos, J.: A goal-based organizational perspective on multi-agents architectures. In: *Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)* (2001)
13. Morandini, M., Nguyen, D.C., Perini, A., Siena, A., Susi, A.: Tool-supported development with tropos: the conference management system case study. In Luck, M., Padgham, L. (eds.)

- Agent Oriented Software Engineering VIII. Lecture Notes in Computer Science, vol. 4951, pp. 182–196. Springer, Berlin (2008). 8th International Workshop, AOSE 2007, Honolulu, May 2007
14. Morandini, M., Penserini, L., Perini, A.: Automated mapping from goal models to self-adaptive systems. In: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), Tool Demo, pp. 485–486 (Sept 2008)
 15. Morandini, M., Penserini, L., Perini, A.: Operational semantics of goal models in adaptive agents. In: 8th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS'09). IFAAMAS, Richland (May 2009)
 16. Morandini, M., Perini, A., Marchetto, A.: Empirical evaluation of tropos4as modelling. In: iStar, pp. 14–19 (2011)
 17. Nguyen, C.D., Perini, A., Tonella, P.: eCAT: a tool for automating test cases generation and execution in testing multi-agent systems. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers, pp. 1669–1670. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2008)
 18. Nguyen, C.D., Perini, A., Tonella, P.: Goal-oriented testing for MASs. *Int. J. Agent Oriented Softw. Eng.* **4**(1), 79–109 (2010)
 19. Penserini, L., Perini, A., Susi, A., Morandini, M., Mylopoulos, J.: A design framework for generating BDI-agents from goal models. In: 6th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS'07), Honolulu, pp. 610–612 (2007)
 20. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: High variability design for software agents: extending tropos. *ACM Trans. Auton. Adapt. Syst.* **2**(4), 16:01–16:27 (2007)
 21. Perini, A., Susi, A.: Agent-oriented visual modeling and model validation for engineering distributed systems. *Comput. Syst. Sci. Eng.* **20**(4), 319–329 (2005)
 22. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: a BDI reasoning engine. In: Dix, J., Bordini, R., Dastani, M., El Fallah Seghrouchni, A. (eds.) *Multi-agent Programming*, vol. 9, pp. 149–174. Springer Science+Business Media Inc., New York (2005). Book chapter
 23. Susi, A., Perini, A., Mylopoulos, J., Giorgini, P.: The tropos metamodel and its use. *Informatica (Slovenia)* **29**(4), 401–408 (2005)
 24. Yu, E.: Modelling strategic relationships for process reengineering. Ph.D. thesis, Department of Computer Science, University of Toronto (1995)

The OpenUp Process

Massimo Cossentino, Vincent Hilaire, and Valeria Seidita

Abstract

The Open Unified Process (OpenUp) is an iterative design process that structures the project lifecycle into four phases: Inception, Elaboration, Construction, and Transition. It is part of the Eclipse Process Framework and embraces a pragmatic, agile philosophy that focuses on the collaborative nature of software development. It is a tools-agnostic, low-ceremony process that can be extended to address a broad variety of project types. The project lifecycle provides stakeholders and team members with visibility and decision points throughout the project and makes them able to manage their work through micro-increments.

1 Introduction

OpenUp is a lean Unified Process that applies iterative and incremental approaches within a structured lifecycle. OpenUp embraces a pragmatic, agile philosophy that focuses on the collaborative nature of software development. It is a tools-agnostic,

M. Cossentino (✉)

Istituto di Reti e Calcolo ad Alte Prestazioni – Consiglio Nazionale delle Ricerche, Viale delle Scienze, 90128 Palermo, Italy

e-mail: cossentino@pa.icar.cnr.it

V. Hilaire

IRTES-SET, UTBM, UPR EA 7274, 90010 Belfort Cedex, France

e-mail: vincent.hilaire@utbm.fr

V. Seidita

Dipartimento di Ingegneria Chimica, Gestionale, Informatica, Meccanica, Viale delle Scienze, 90128 Palermo, Italy

e-mail: valeria.seidita@unipa.it

low-ceremony process that can be extended to address a broad variety of project types.

Personal efforts on an **OpenUp** project are organized in micro-increments. These represent short units of work that produce a steady, measurable pace of project progress (typically measured in hours or a few days).

The process applies intensive collaboration as the system is incrementally developed by a committed, self-organized team. These micro-increments provide an extremely short feedback loop that drives adaptive decisions within each iteration.

OpenUp divides the project into iterations: planned, time-boxed intervals typically measured in weeks. Iterations focus the team on delivering incremental value to stakeholders in a predictable manner. The iteration plan defines what should be delivered within the iteration. The result is a demo-able or shippable build. **OpenUp** teams self-organize around how to accomplish iteration objectives and commit to delivering the results. They do that by defining and “pulling” fine-grained tasks from a work items list. **OpenUp** applies an iteration lifecycle that structures how micro-increments are applied to deliver stable, cohesive builds of the system that incrementally progresses toward the iteration objectives.

OpenUp structures the project lifecycle into four phases: Inception, Elaboration, Construction, and Transition. The project lifecycle provides stakeholders and team members with visibility and decision points throughout the project. This enables effective oversight, and allows you to make “go or no-go” decisions at appropriate times. A project plan defines the lifecycle, and the end result is a released application.

It is worth to note that the **OpenUp** description largely uses the concept of work product slot. This is, indeed, one of the peculiarities of this process approach. The definition of work product slot may be found in [2]: “Work product slots are indirections for the inputs of tasks of a Practice that allow practices to be documented independent of any other practice, i.e. independent of the work products produced by other practices. Practice task refer to work product slots as inputs, rather than refer directly to specific work products.” It is a little bit different from the concept of Work Product for which a specific SPEM 2.0 icon exists; we decided to use the same icon for the two concepts maintaining the name within brackets in the case of Work Product Slot as the **OpenUp** documentation does.

The description of the **OpenUp** process reported in this chapter is taken from the **OpenUp** website [1]. The documentation approach adopted in that website is quite different from the IEEE FIPA SC00097B standard adopted in this book but it has been possible to retrieve most of the necessary information. In order not to introduce any personal interpretation of **OpenUp**, the authors of this chapter preferred not to report the information that may not be explicitly found in the website. This brought to the omission of a few details but it is coherent with the spirit of this book where the proposed chapters have been written by people who are the primary authors of the described process or they are at least deeply involved in it. In this case, this situation was not verified and therefore a specific care has been needed (Fig. 1).

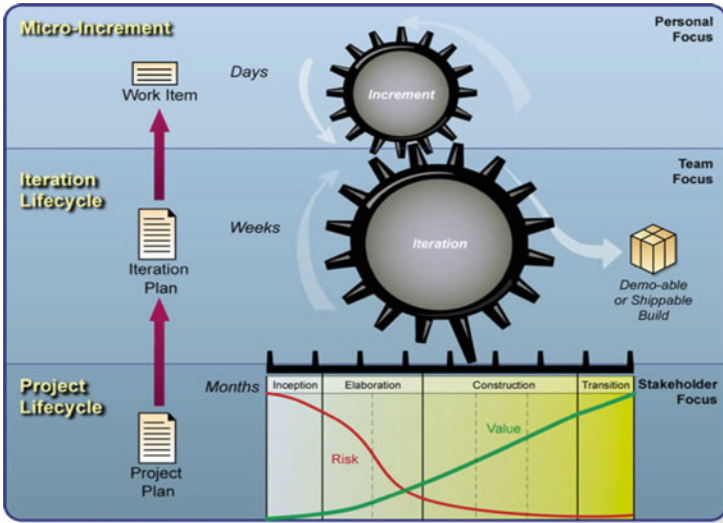


Fig. 1 An overview of the OpenUp process

1.1 The OpenUp Process Lifecycle

OpenUp is an iterative process with iterations distributed throughout four phases: Inception, Elaboration, Construction, and Transition.

Each phase may have as many iterations as needed (depending on the degree of novelty of the business domain, the technology being used, architectural complexity, and project size, to name a few factors).

To offer a quick start for teams to plan their iterations, OpenUp provides work breakdown structure (WBS) templates for each iteration, and a WBS template for an end-to-end process.

Iterations may have variable lengths, depending on project characteristics. One-month iterations are typically recommended because this timeframe provides:

- A reasonable amount of time for projects to deliver meaningful increments in functionality.
- Early and frequent customer feedback.
- Timely management of risks and issues during the course of the project.

In the following sections, all the aspects of OpenUp are described by using SPEM 2.0 [6] and the extensions proposed by Seidita et al. in [7]. Figure 2 shows the SPEM 2.0 icons the reader can find in the following figures (Fig. 3).

1.2 The OpenUp Process System Metamodel

The OpenUp website (see [1]) does not provide an explicit representation of the underlying system metamodel. For this reason, it has been preferred to limit

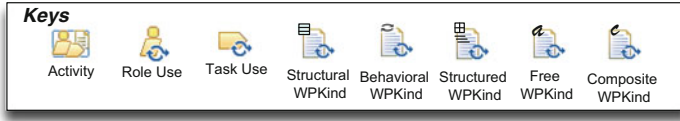


Fig. 2 The SPEM 2.0 Icons



Fig. 3 The OpenUp phases

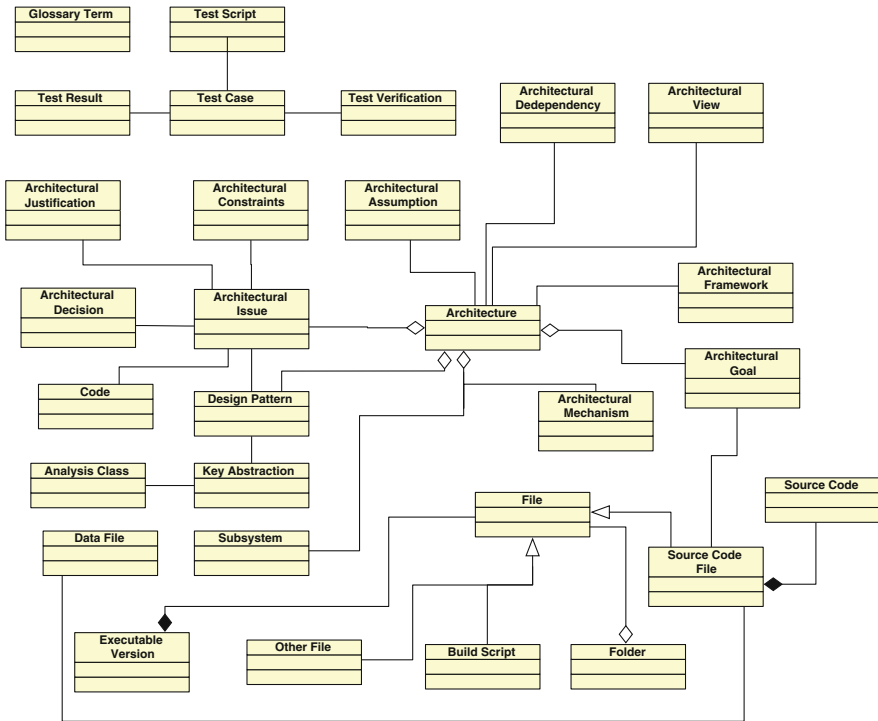


Fig. 4 The OpenUp System Metamodel—the first portion

the information reported in this section to what could be extracted from the website without any margin for misinterpretation. The elements of the metamodel have been deduced from the analysis of available work product descriptions. The following pictures (Figs. 4, 5, 6, and 7) only reports system metamodel elements whose relationships with others may be unambiguously deduced from work product descriptions; for readability reasons we split the whole figure in four ones.

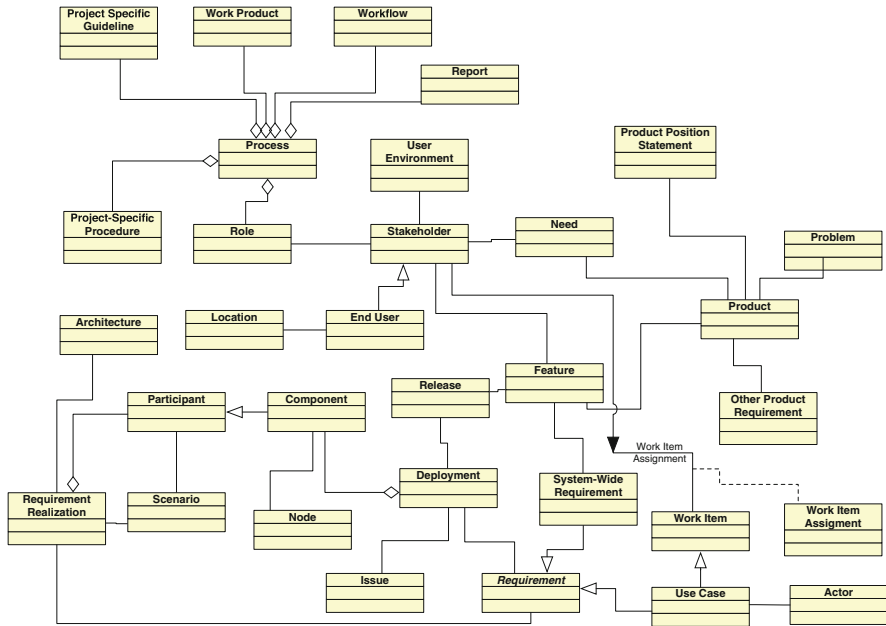


Fig. 5 The OpenUp System Metamodel—the second portion

1.2.1 Definition of the System Metamodel Elements

In the following table, the “Domain” column prescribed by the FIPA SC00097B specification has not been reported because there is no precise information on the OpenUp website about the allocation of concepts to domains like problem, solution and so on.

Table 1 only reports the elements for which a definition may be found in the OpenUp website. The other elements identified in the work product descriptions are reported below the table as a simple list.

List of system metamodel elements without definition: Analysis Class, Architecture, Architectural Issues, Architectural Mechanism, Build scripts, Communication Procedure, Component, Contingency Measure, Data files, Deployment, Design Pattern, End User, Implementation Element, Issue, Node, Other files, Product, Release, Requirement (abstract), Requirement Realization, Rollback, Source code files, Test Result, Workaround.

2 Phases of the OpenUp Process

2.1 The Inception Phase

The Inception phase is composed by the Inception iteration as described in Fig. 8. The process flow within the iteration is detailed in Fig. 9.

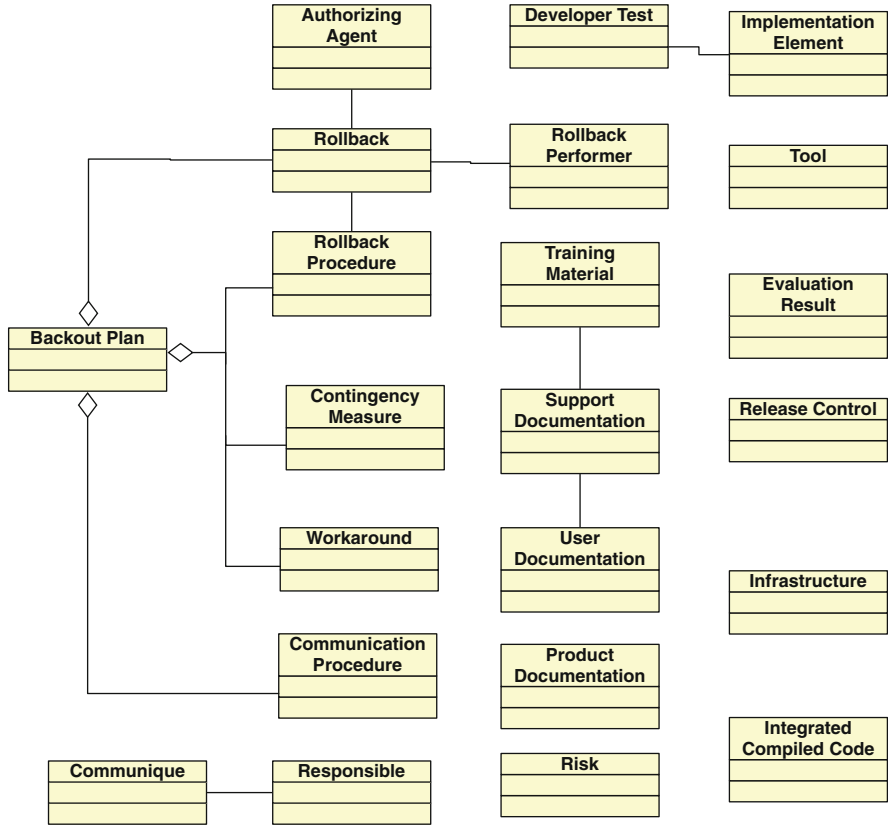


Fig. 6 The OpenUp System Metamodel—the third portion

The workflow inside each activity will be detailed in the following subsections (after the description of process roles). The Inception phase involves 8 different process roles, 24 work products as described in Figs. 10, 11, 12, and 13. The phase is composed of four activities as described in Fig. 10, each of them composed of one or more activities and tasks as described in Figs. 11, 12, 13, 14, and 15.

Details of activities shown in Fig. 10 are reported in Figs. 11, 12, 13, 14, and 15.

2.1.1 Process Roles

Eight roles are involved in the Inception phase, which are described in the following subsections.

Analyst

She/he is responsible for:

1. Detailing system-wide requirements
2. Detailing use-case scenario

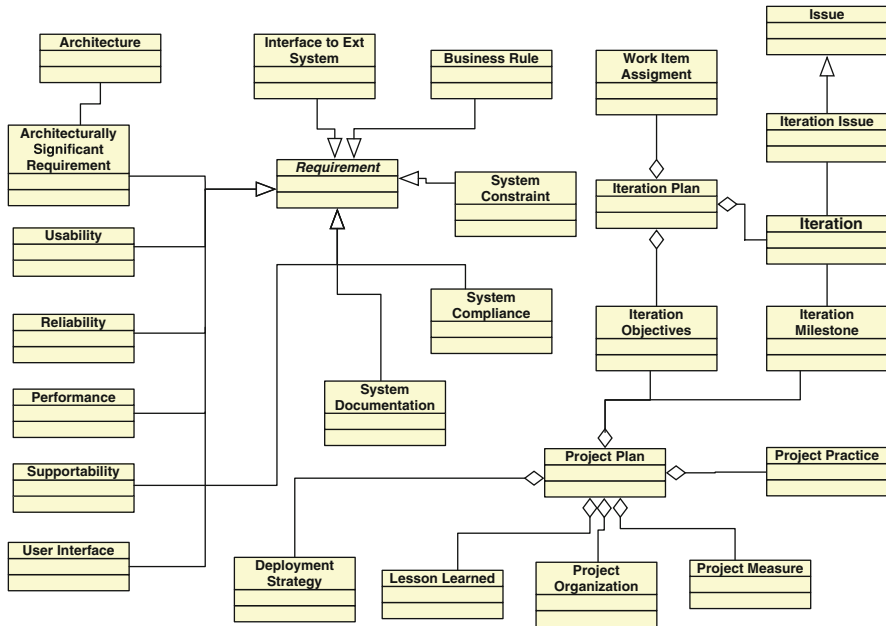


Fig. 7 The OpenUp System Metamodel—the fourth portion

3. Developing technical vision
4. Identifying and outlining requirements
 - She/he assists in:
 1. Assessing results
 2. Creating test cases
 3. Envisioning the architecture
 4. Managing iteration
 5. Planning iteration
 6. Planning project

Architect

- She/he responsible for:
1. Envisioning the architecture.
 - She/he assists in:
 1. Detailing system-wide requirements
 2. Detailing use-case scenario
 3. Developing technical vision
 4. Identifying and outlining requirements
 5. Managing iteration
 6. Planning iteration
 7. Plan project

Table 1 Definition of the system metamodel elements

Concept	Definition
Actor	To fully understand the system's purpose, you must know who the system is for, that is: Who will use the system? The answer to this question is: the Actors. An Actor is a role that a person or external system plays when interacting with the system. Instances of an Actor can be an individual or an external system; however, each Actor provides a unique and important perspective on the system that is shared by every instance of the Actor.
Communicate	While there is no prescribed format for the release communications artifact, each communicate should indicate the preferred delivery mechanisms (e.g., beeper notification, telephone calls, a posting to an internal release website, live or pre-recorded presentations by senior management, etc.) and generally answer the following questions: <ul style="list-style-type: none"> • Who are the parties (stakeholders) that are interested in knowing that a release to production has taken place? • What specifically (features, functions, components) has been placed into production? • Why is this release valuable to stakeholders and what business purpose does it serve? • Where is the product available (on which platforms, geographical locations, business units, etc.)? • How can the stakeholders access the system and under what circumstances? • When was the product released (or when will it be released if the release date is in the future)?
Design Element	The elements that will make up the implemented system. They contribute to define the abstractions of particular portions of the implementation. Design elements may be used to describe multiple static and dynamic views of the system for examination.
Developer Test	It covers all of the steps to validate a specific aspect of an implementation element. A developer test specifies test entries, execution conditions, and expected results. These details are identified to evaluate a particular aspect of a scenario
Envisioned Core Requirement	Define the quality ranges for performance, robustness, fault tolerance, usability, and similar characteristics that are not captured in the Feature Set.
Evaluation Results	Results of the iteration assessment that may be useful for improving the next one
Executable Version	The working version of the system or part of the system is the result of putting the implementation through a build process (typically an automated build script) that creates an executable version, or one that runs. This executable version will typically have a number of supporting files that are also considered part of this artifact.
Feature	The view of the stakeholders of the technical solution to be developed is specified in terms of her/his key needs and features. It also includes envisioned core requirements.
File	Files compose the executable version of the system.
Folder	Folders contain File
Glossary Term	Terms that are being used on the project so that everyone has a common understanding of them
Integrated Compiled Code	A release consists of integrated, compiled code that runs cleanly, independently, and in its entirety.
Term	Definition
Infrastructure	In reference to a release sprint, infrastructure refers to all the hardware, software, and network facilities necessary to support a deployed release. Infrastructure normally is defined as anything that supports the flow and processing of information in an organization. The infrastructure needed to support a release package normally includes: <ul style="list-style-type: none"> • Software, including: Operating systems and applications for servers and clients, Desktop applications, Middleware, Protocols • Hardware • Networks, including: Routers, Aggregators, Repeaters, Other transmission media devices that control movement of data and signals • Facilities

(continued)

Table 1 (continued)

Term	Definition
Iteration	An iteration is a set period of time within a project in which you produce a stable, executable version of the product, together with any other supporting documentation, install scripts, or similar, necessary to use this release. Also referred to as a cycle or a timebox.
Iteration Issue	An iteration is a set period of time within a project in which you produce a stable, executable version of the product, together with any other supporting documentation, install scripts, or similar, necessary to use this release. Also referred to as a cycle or a timebox.
Iteration Objectives	A few objectives should be written for the iteration, these will help guide the performers throughout the iteration. Also, assess at the end if those objectives have been achieved.
Iteration Plan	It helps the team to monitor the progress of the iteration, and keeps the results of the iteration assessment that may be useful for improving the next one. It include Milestones of the Iteration, task assignment and issues to be solved during the iteration.
Milestone	Milestones of an iteration show start and end dates, intermediate milestones, synchronization points with other teams, demos, and so on.
Need	Capabilities needed by stakeholder
Process	The process that a project is to follow in order to produce the project’s desired results.
Product Documentation	It provides a detailed enough understanding of how the product operates and how it meets stated business goals and needs.
Project Plan	It describes how the project is organized, and identifies what practices will be followed. Additionally, it defines the parameters for tracking project progress, and specifies the high-level objectives of the iterations and their milestones.
Release Control	It identifies the requirements to which a release package must conform to be considered “deployable”.
Responsible	Who will execute the communications when a successful release has been declared (normally the Deployment Engineer), as well as the timing and dependencies of the communiques.
Risk	A risk is whatever may stand in the way to success, and is currently unknown or uncertain. Usually, a risk is qualified by the probability of occurrence and the impact in the project, if it occurs.
Stakeholder	Stakeholders express their needs and requested features.
Support Documentation	Support documentation typically includes: <ul style="list-style-type: none"> • User manuals with work instructions, process descriptions, and procedures • Communications, training, and knowledge transfer deliverables • Support and operations manuals • Service information, including Help Desk scripts
System-Wide Requirement	System-wide requirements are requirements that define necessary system quality attributes such as performance, usability and reliability, as well as global functional requirements that are not captured in behavioral requirements artifacts such as use cases. System-wide requirements are categorized according to the FURPS+ model (Functional, Usability, Reliability, Performance, Supportability + constraints). Constraints include design, implementation, interfaces, physical constraints, and business rules. System-wide requirements and use cases, together, define the requirements of the system. These requirements support the features listed in the vision statement. Each requirement should support at least one feature, and each feature should be supported by at least one requirement.
Task Assignment	The task assignments for an iteration are a subset of all tasks on the Artifact: Work Items List
Test Case	A test case specifies the conditions that must be validated to enable an assessment of aspects of the system under test. A test case is more formal than a test idea; typically, a test case takes the form of a specification. It includes the specification of test inputs, conditions, and expected results for a system
Test Script	Test scripts implement a subset of required tests in an efficient and effective manner.

(continued)

Table 1 (continued)

Term	Definition
Test Verification	It reports that a set of tests was run
Tool	The tools needed for supporting the software development effort.
Training Material	Training materials that can be used to train end users and production support personnel might consist of: Presentation slides, Handouts, Job aids, Tutorials, On-line demos, Video vignettes, Lab exercises, Quizzes, Workshop materials, etc.
Use Case	Use cases are used for the following purposes: <ul style="list-style-type: none"> • To reach a common understanding of system behavior • To design elements that support the required behavior • To identify test cases • To plan and assess work • To write user documentation
User Documentation	User documentation might include all or parts of user manuals (electronic or paper-based), tutorials, frequently asked questions (FAQs), on-line Help Files, installation instructions, work instructions, operational procedures, etc.
Vision Constraint	Together with the Stakeholder Requests give an overview of the reasoning, background, and context for detailed requirements.
Work Item	Requests for additional capabilities or enhancement for that application. Work items can be very large in scope, especially when capturing requests for enhancements. To allow the application to be developed in micro-increments, work items are analysed and broken down into smaller work items so that they can be assigned to an iteration.

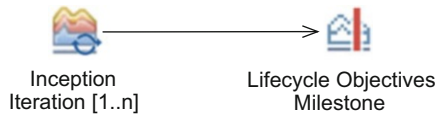


Fig. 8 The Inception iteration inside the Inception phase

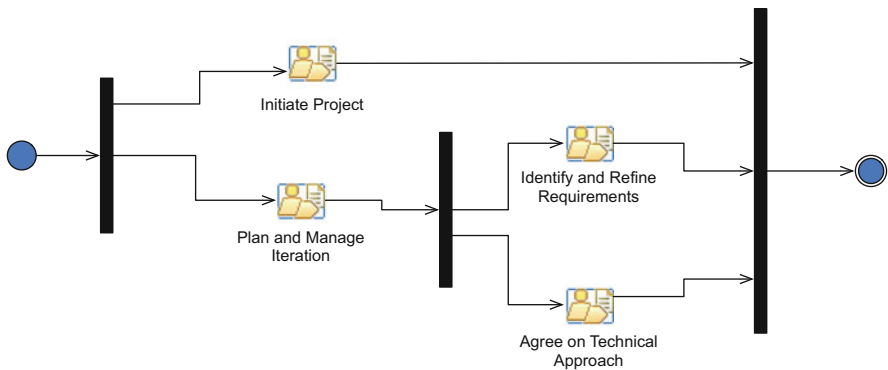


Fig. 9 The Inception phase flow of activities

Developer

She/he assists in

1. Assessing results
2. Creating test cases
3. Detailing use-case scenarios

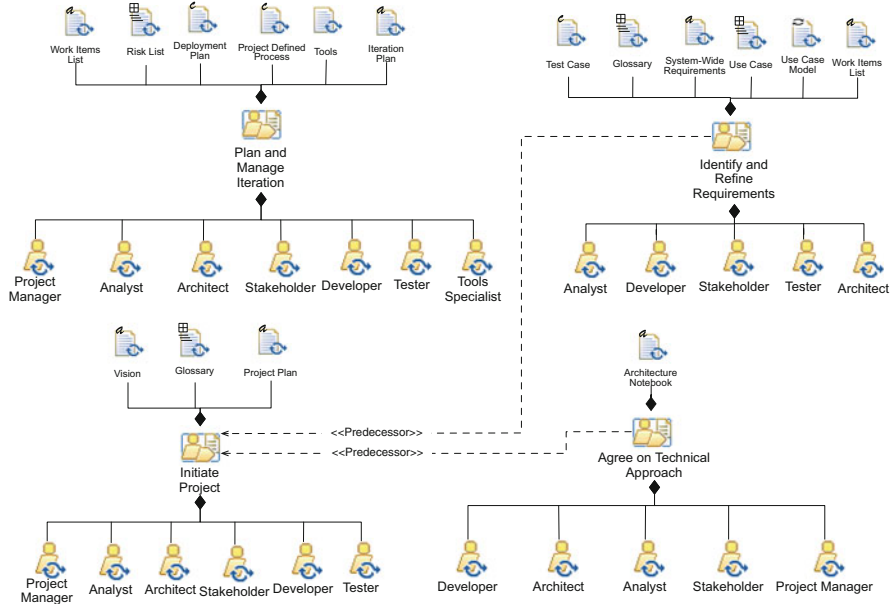


Fig. 10 The Inception phase described in terms of activities, output work products and involved stakeholders

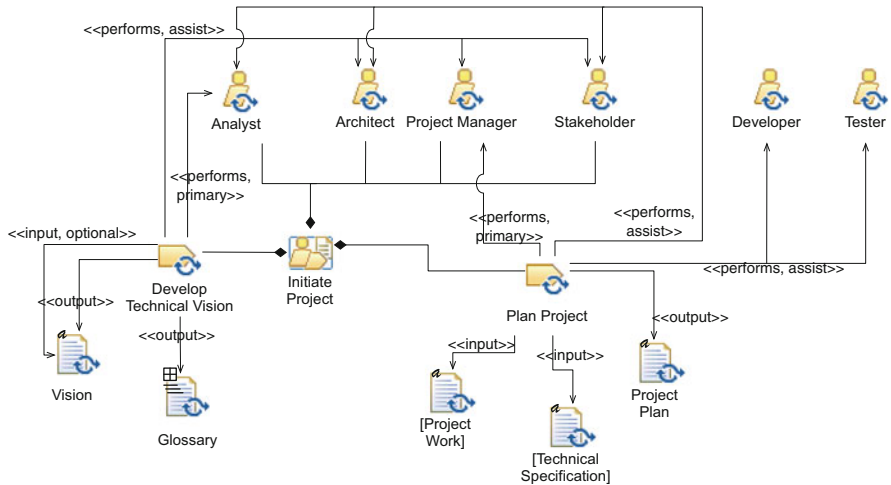


Fig. 11 The Initiate Project activity described in terms of tasks, roles and work products

4. Envisioning the architecture
5. Identifying and outlining requirements
6. Managing iteration
7. Outlining deployment plan

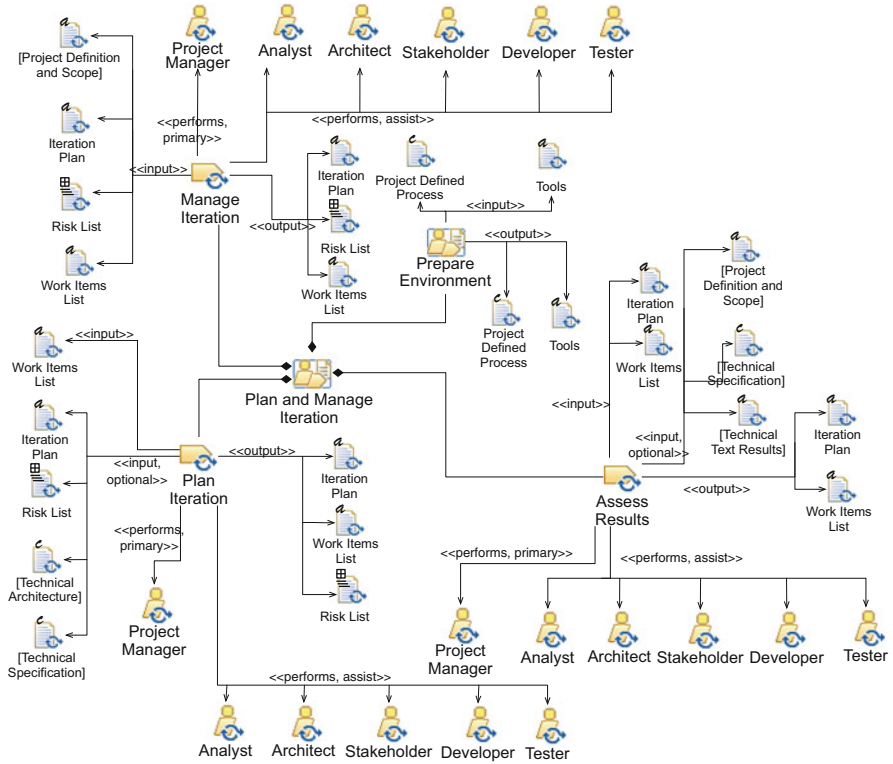


Fig. 12 The Plan and Manage Iteration activity described in terms of activities, tasks, roles and work products (activity Prepare Environment is detailed in Fig. 13)

- 8. Planning iteration
- 9. Planning project

Process Engineer

She/he is responsible for

- 1. Deploying the process
- 2. Tailoring the process

Project Manager

She/he is responsible for

- 1. Assessing results
- 2. Managing iteration
- 3. Planning iteration
- 4. Planning project

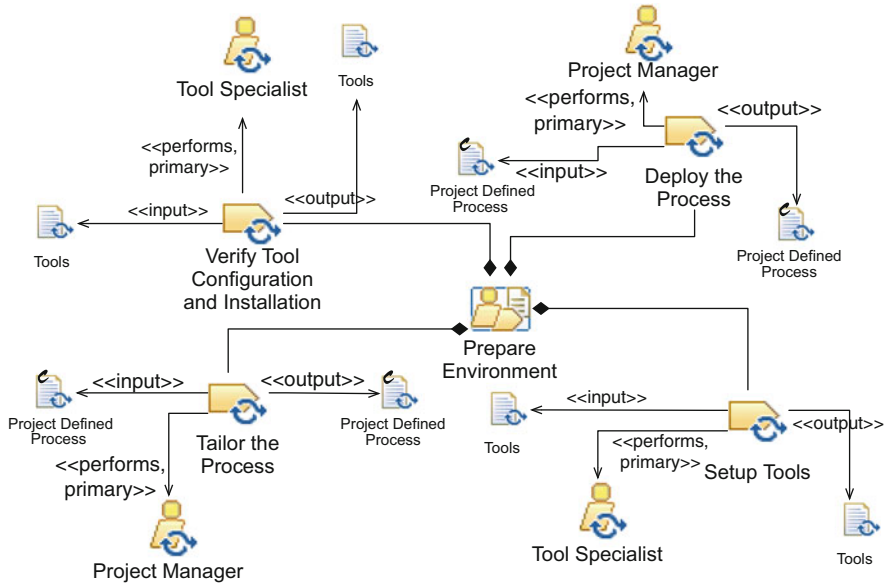


Fig. 13 The Prepare Environment activity of the Plan and Manage Iteration activity described in terms of tasks, roles, and work products

She/he assists in

1. Developing technical vision
2. Envisioning the architecture

Stakeholder

She/he assists in

1. Assessing results
2. Creating test cases
3. Detailing system-wide requirements
4. Detailing use-case scenarios
5. Developing technical vision
6. Envisioning the architecture
7. Identifying and outlining requirements
8. Managing iteration
9. Planning iteration
10. Planning project

Tester

She/he is responsible for

1. Creating test cases

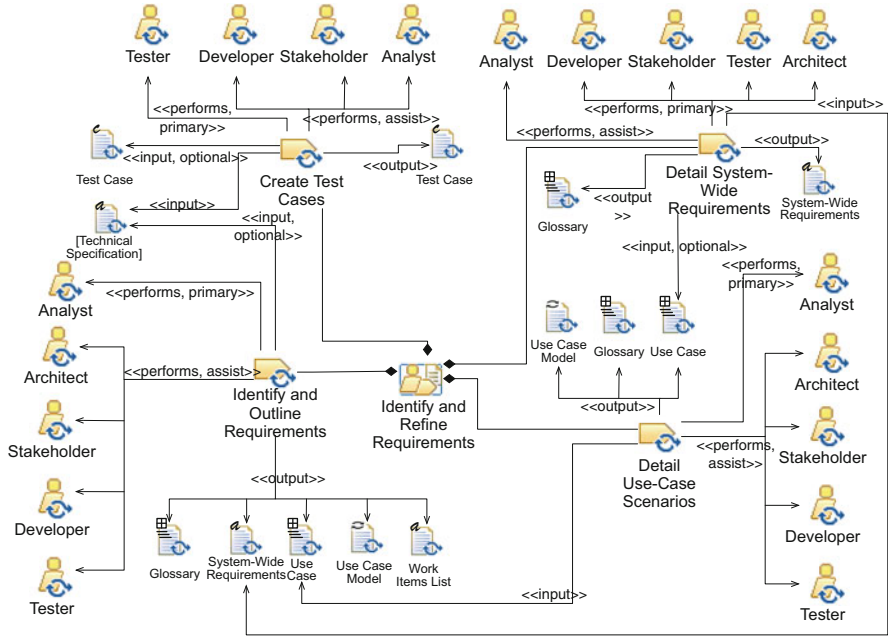


Fig. 14 The Identify and Refine Requirements activity described in terms of activities, tasks, roles, and work products

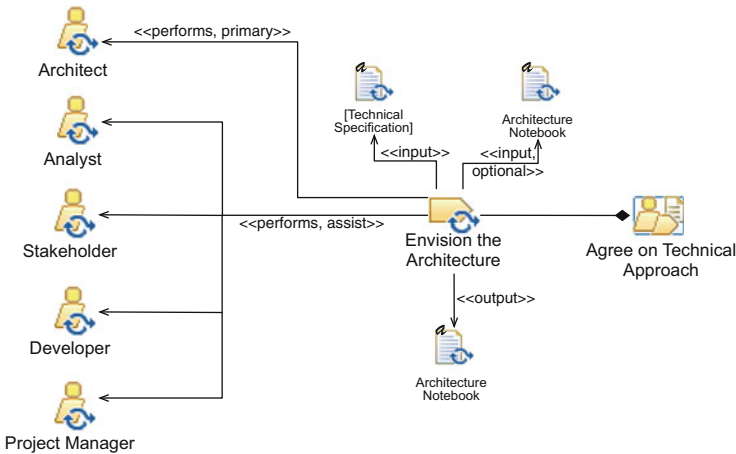


Fig. 15 The Agree on Technical Approach activity described in terms of activities, tasks, roles, and work products

She/he assists in

1. Assessing results
2. Detailing system-wide requirements
3. Detailing use-case scenarios

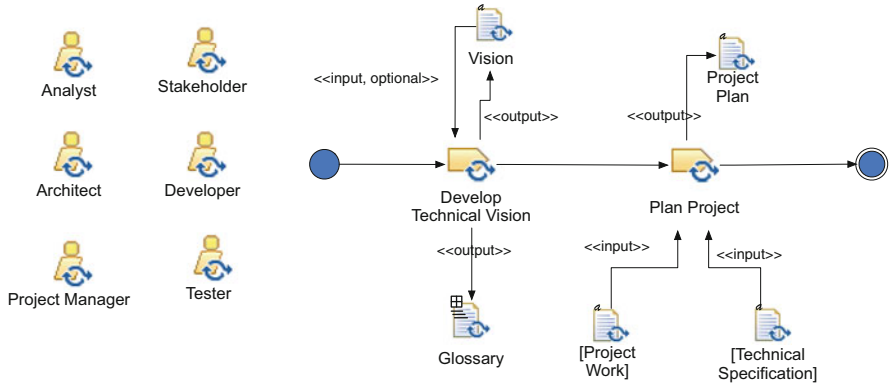


Fig. 16 The flow of tasks of the Initiate Project activity

Table 2 Initiate project—the task description

Activity	Task	Task description	Roles involved
Initiate Project	Initiate Project	The solution is proposed for a problem that everybody agrees on. Stakeholders collaborate with the development team to express and document their problems, needs, and potential features for the system to be, so the project team can better understand what has to be done.	Analyst (perform), Architect (assist), Project Manager (assist), Stakeholder (assist).
Initiate Project	Plan Project	Get stakeholder buy-in for starting the project and team commitment to move forward with it. This plan can be updated as the project progresses based on feedback and changes in the environment.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).

- 4. Identifying and outlining requirements
- 5. Managing iteration
- 6. Planning iteration
- 7. Planning project

Tool Specialist

She/he is responsible for

- 1. Setting up tools
- 2. Verifying tool configuration and installation

2.1.2 Activity Details

The Inception phase includes four activities, which are described in the following subsections.

Initiate Project

The flow of tasks inside this activity is reported in Fig. 16, and the tasks are detailed in Table 2.

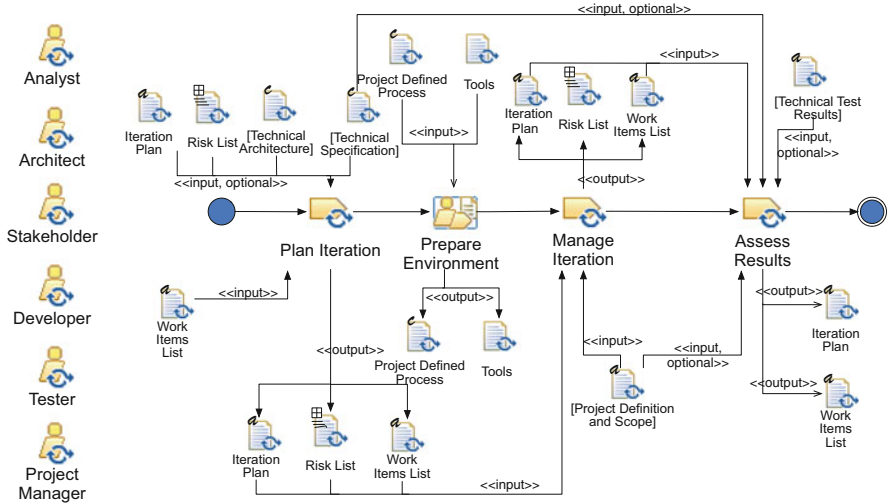


Fig. 17 The flow of tasks of the Plan and Manage Iteration activity

Table 3 Plan and manage iteration—the task description

Activity	Task	Task description	Roles involved
Plan and Manage Iteration	Plan Iteration	The purpose of this task is to identify the next increment of system capability, and create a fine-grained plan for achieving that capability within a single iteration.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).
Plan and Manage Iteration	Manage Iteration	Help the team meet the iteration objectives and keep the project on track. Manage stakeholders' expectations as technical and practical discoveries are made during the project.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).
Plan and Manage Iteration	Assess Results	Demonstrate the value of the solution increment that was built during the iteration and apply the lessons learned to modify the project or improve the process.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).

Plan and Manage Iteration

The flow of tasks inside this activity is reported in Fig. 17, and the tasks are detailed in Table 3.

Prepare Environment

The flow of tasks inside this activity is reported in Fig. 18, and the tasks are detailed in Table 4.

Identify and Refine Requirements

The flow of tasks inside this activity is reported in Fig. 19, and the tasks are detailed in Table 5.

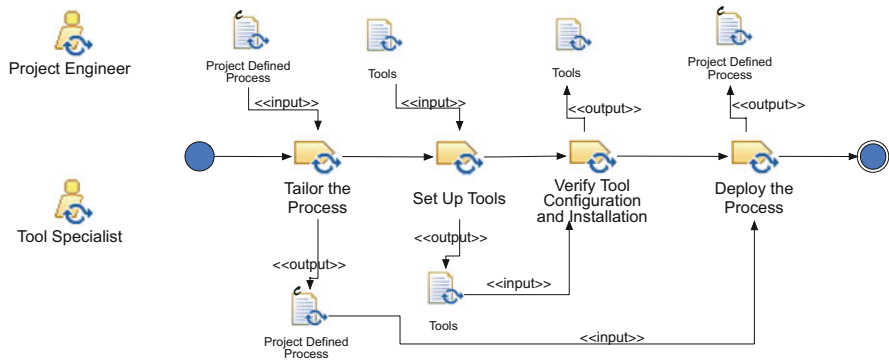


Fig. 18 The flow of tasks of the Prepare Environment activity

Table 4 Prepare environment—the task description

Activity	Task	Task description	Roles involved
Prepare Environment	Tailor the Process	The purpose of this task is to ensure that the project team has a defined process that meets their needs. The purpose of this task is to <ul style="list-style-type: none"> • Install the tools • Customize the tools • Make the tools available to the end users 	Process Engineer (perform)
Prepare Environment	Set Up Tools	The purpose of this task is to <ul style="list-style-type: none"> • Install the tools • Customize the tools • Make the tools available to the end users 	Tool Specialist (perform)
Prepare Environment	Verify Tool Configuration and Installation	The purpose of this task is to verify that the tools can be used to develop the system	Tool Specialist (perform)
Prepare Environment	Deploy the Process	The purpose of this task is to <ul style="list-style-type: none"> • Ensure that the project members are properly introduced to the process • Harvest any feedback on the process and refine the process, as necessary 	Process Engineer (perform)

Agree on Technical Approach

The flow of tasks inside this activity is reported in Fig. 20, and the tasks are detailed in Table 6.

2.1.3 Work Products

The Inception phase generates fourteen work products. Their relationships with the system meta-model elements are described in Fig. 21.

This diagram represents the Iteration phase in terms of output Work Products. Each of these reports one or more elements from the OpenUp system metamodel; each system metamodel element is represented using an UML class icon (yellow filled) and, in the documents, such elements can be Defined, reFined, Quoted, Related or Relationship Quoted.

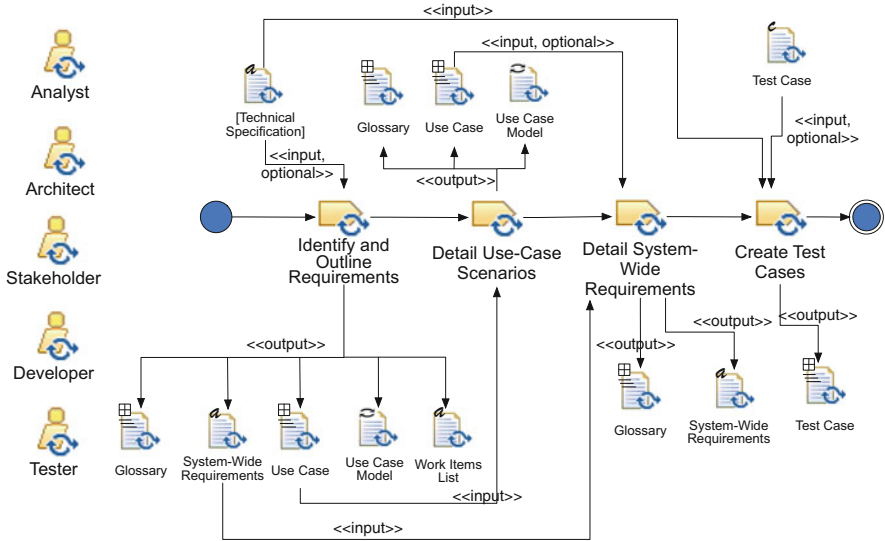


Fig. 19 The flow of tasks of the Identify and Refine Requirements activity

Table 5 Identify and refine requirements—the task description

Activity	Task	Task description	Roles involved
Identify and Refine Requirements	Identify and Outline Requirements	This task describes how to identify and outline the requirements for the system so that the scope of work can be determined.	Analyst (perform), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).
Identify and Refine Requirements	Detail Use-Case Scenarios	This task describes how to detail use-case scenarios for the system.	Analyst (perform), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).
Identify and Refine Requirements	Detail System-Wide Requirements	This task details one or more requirement(s) that do(es) not apply to a specific use case.	Analyst (perform), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).
Identify and Refine Requirements	Create Test Cases	Develop the test cases and test data for the requirements to be tested.	Tester (perform), Analyst (assist), Developer (assist), Stakeholder (assist).

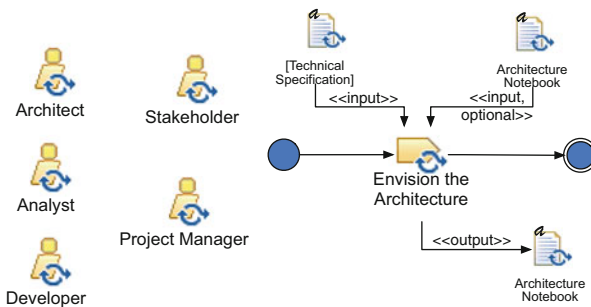


Fig. 20 The flow of tasks of the Agree on Technical Approach activity

Table 6 Agree on technical approach—the task description

Activity	Task	Task description	Roles involved
Agree on Technical Approach	Envision the Architecture	This task is where the ‘vision’ for the architecture is developed through analysis of the architecturally significant requirements and identification of architectural constraints, decisions and objectives.	Architect (perform), Analyst (assist), Developer (assist), Stakeholder (assist), Project Manager (assist).

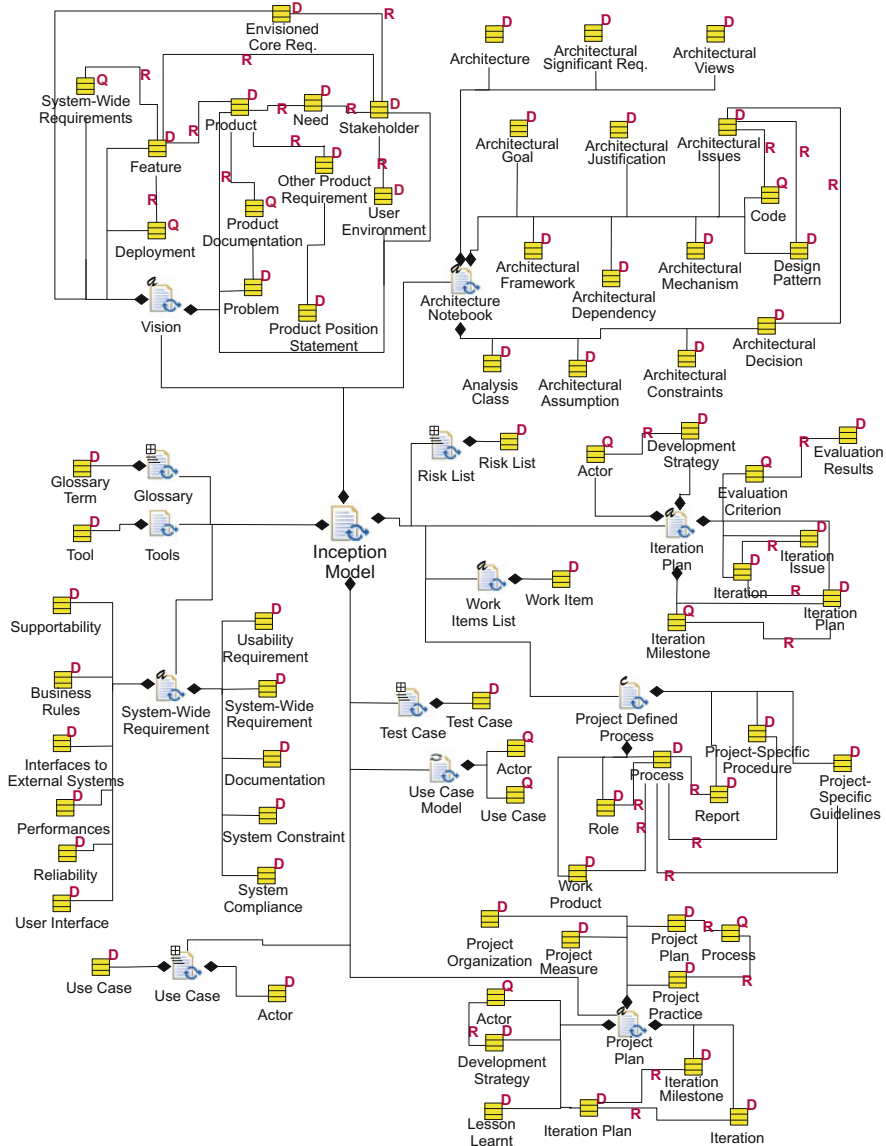


Fig. 21 The Inception phase documents structure

Work Product Kinds

Table 7 describes the work products of the Inception phase according to their kinds.

Work products are detailed in the following sections. No specific examples of notation use are reported since standard UML [3,4] is supposed to be adopted.

Architecture Notebook

The purpose of this artifact is to capture and make architectural decisions and to explain those decisions to developers. This artifact describes the Software Architecture. It provides a place for maintaining the list of architectural issues, along with the associated architectural decisions, designs, patterns, code documented (or pointed to), and so forth—all at appropriate levels to make it easy to understand what architectural decisions have been made and what remain to be made. It is helpful for architects to use this artifact to collaborate with other team members in developing the architecture and to help team members understanding the motivation behind architectural decisions so that those decisions can be robustly implemented. For example, the architect may put constraints on how data is packaged and communicated between different parts of the system. This may appear to be a burden, but the justification in the Architecture Notebook can explain that there is a significant performance bottleneck when communicating with a legacy system. The rest of the system must adapt to this bottleneck by following a specific data packaging scheme. This artifact should also inform the team members how the system is partitioned or organized so that the team can adapt to the needs of the system. It also gives a first glimpse of the system and its technical motivations to whoever must maintain and change the architecture later. At a minimum, this artifact should do these three things:

- List guidelines, decisions, and constraints to be followed
- Justify those guidelines, decisions, and constraints
- Describe the Architectural Mechanisms and where they should be applied

Team members who were not involved in those architectural decisions need to understand the reasoning behind the context of the architecture so that they can address the needs of the system. A small project should not spend a lot of time documenting the architecture, but all critical elements of the system must be communicated to current and future team members. This is all useful content:

- Goals and philosophy of the architecture
- Architectural assumptions and dependencies
- References to architecturally significant requirements
- References to architecturally significant design elements
- Critical system interfaces
- Packaging instructions for subsystems and components
- Layers and critical subsystems
- Key abstractions
- Key scenarios that describe critical behavior of the system

Table 7 Inception phase—work product kinds

Name	Description	Work product kind
Architecture Notebook	This artifact describes the rationale, assumptions, explanation, and implications of the decisions that were made in forming the architecture.	Free Text
Build	An operational version of a system or part of a system that demonstrates a subset of the capabilities to be provided in the final product.	Composite
Design	This artifact describes the realization of required system functionality and serves as an abstraction of the source code.	Composite
Developer Test	The Developer Test validates a specific aspect of an implementation element.	Structured
Glossary	This artifact defines important terms used by the project. The collection of terms clarifies the vocabulary used on the project.	Structured
Implementation	Software code files, data files, and supporting files (such as online help files) that represent the raw parts of a system that can be built.	Composite
Iteration Plan	A fine-grained plan describing the objectives, work assignments, and evaluation criteria for the iteration.	Free Text
Project Defined Process	This work product describes the process that a project is to follow in order to produce the project's desired results.	Composite
Project Definition and Scope	This slot serves as an abstraction of high-level artifacts that define the project and its scope. Typical examples of such artifacts could be a project definition, and a high-level project schedule identifying major milestones and major deliverables. Fulfilling Work Products:	Free Text
	<ul style="list-style-type: none"> • Project Plan 	
Project Plan	This artifact gathers all of the information required to manage the project on a strategic level. Its main part consists of a coarse-grained plan, identifying project iterations and their goals.	
Project Work	This slot serves as an abstraction for any type of work being done on the project. It could be represented as a work items list, an operational schedule, a work breakdown structure, and so on. Fulfilling Work Products:	Free Text
	<ul style="list-style-type: none"> • Iteration Plan • Work Items List 	
Risks List	This artifact is a sorted list of known and open risks to the project, sorted in order of importance and associated with specific mitigation or contingency actions.	Structured
System-Wide Requirements	This artifact captures the quality attributes and constraints that have system-wide scope. It also captures system-wide functional requirements.	Free Text
Technical Architecture	This slot serves as an abstraction of high-level artifacts that represent the documentation of the architecture. Fulfilling Work Products:	Composite
	<ul style="list-style-type: none"> • Architecture Notebook 	
Technical Specification	This slot serves as an abstraction of high-level artifacts that describe requirements, constraints, and goals for the solution. Fulfilling Work Products:	Composite
	<ul style="list-style-type: none"> • Glossary • System-Wide Requirements • Use Case • Use-Case Model • Vision 	
Technical Test Result	This slot serves as an abstraction of high-level artifacts that define the results of testing the hardware and software for the system being developed. Fulfilling Work Products:	Free Text
	<ul style="list-style-type: none"> • Test Log 	

(continued)

Table 7 (continued)

Name	Description	Work product kind
Test Case	This artifact is the specification of a set of test inputs, execution conditions, and expected results that you identify to evaluate a particular aspect of a scenario.	Structured
Test Log	This artifact collects the raw output that is captured during a unique run of one or more tests for a single test cycle run.	Free Text
Test Script	This artifact contains the step-by-step instructions that compose a test, enabling its run. Text scripts can take the form of either documented textual instructions that are manually followed, or computer-readable instructions that enable automated testing.	Structured
Tools	These work products are the tools needed to support the software development effort.	Free Text
Use Case	This artifact captures the system behavior to yield an observable result of value to those who interact with the system.	Structured Behavioral
Use-Case Model	This artifact presents an overview of the intended behavior of the system. It is the basis for agreement between stakeholders and the project team in regards to the intended functionality of the system. It also guides various tasks in the software development lifecycle.	Composite (Structured + Behavioral)
Vision	This artifact provides a high-level basis for more detailed technical requirements. It captures the technical solution by describing the high-level stakeholder requests and constraints that give an overview of the reasoning, background, and context for detailed requirements. The vision serves as input for communicating the fundamental “what and why” for the project and provides a strategy against which all future decisions can be validated. The vision should rally team members behind an idea and give them the context for decision-making in the requirements area. The vision must be visible to everyone on the team.	Free Text
Work Items List	This artifact contains a list of all of the scheduled work to be done within the project, as well as proposed work that may affect the product in this or future projects. Each work item may contain references to information relevant to carry out the work described within the work item.	Free Text

Build

The purpose of this work product is to deliver incremental value to the user and customer, and provide a testable artifact for verification. This working version of the system, or part of the system, is the result of putting the implementation through a build process (typically an automated build script) that creates an executable version. This version will typically have a number of supporting files that are also considered part of this artifact. This work product is almost always a product made up of numerous parts required to make the executable system. Therefore, a Build is more than just executable files; it also includes such things as configuration files, help files, and data repositories that will be put together, resulting in the product that the users will run.

Deployment Plan

The purpose of this work product is to capture, in one document, the unique information that will be consumed by deployment engineers before and during the deployment to production of a particular release package. The deployment

plan should contain the unique instructions for deploying a particular version of a product. By “unique instructions” we mean those things that are not part of a deployment engineer’s normal procedures. Rather, they often are specific procedures and timing constraints that a deployment engineer should be aware of as they are rolling out a particular release. While a draft version of the deployment plan is typically developed by a development team, the deployment engineer is responsible for its contents and existence. A deployment plan normally consists of the following sections:

- The scope of the release and a general overview of the capabilities to be deployed
- The timing and dependencies for deploying components to various nodes
- The risks or issues associated with the release based on a risk assessment
- The customer organization, stakeholders, and end user community that will be impacted by the release
- The person or persons who have the authority to approve the release as “ready for production”
- The development team members responsible for delivering the release package to the Deployment Manager, along with contact information
- The approach for transitioning the release package to the Deployment Engineer, including appropriate communications protocols and escalation procedures
- The success criteria for this deployment; in other words, how will the Deployment Engineer know that the release is successful so it can report success

Design

The purpose of this work product is to describe the elements of the system so they can be examined and understood in ways not possible by reading the source code. This work product describes the elements that will make up the implemented system. It communicates abstractions of particular portions of the implementation. While architecture focuses on interfaces, patterns, and key decisions, the design fleshes out the technical details in readiness for implementation, or as part of implementation. This work product can describe multiple static and dynamic views of the system for examination. Although various views may focus on divergent, seemingly independent issues of how the system will be put together and work, they should fit together without contradiction. It is important that the author of this work product is able to analyse key decisions about the structure and behavior of the system and communicate them to other collaborators. It is also important that these decisions can be communicated at various levels of abstraction and granularity. Some aspects of the design can be represented by source code, possibly with some extra annotations. But more abstract representations of the design will be at a higher-level than source code. The more abstract representation could use various representation options. UML could be used either strictly or informally, it is the preferred notation based on its rich semantics and broad usage in the industry. Other techniques could be used to communicate the design. Or the design could use a mix of techniques as applicable.

This process does not govern whether to record these representations on a white board or to use a formal tool. But any representation, whether characterized as

formal or informal, should unambiguously communicate the technical decisions embodied by the design.

Developer Test

This artifact is used to evaluate whether an implementation element performs as specified. This artifact covers all of the steps to validate a specific aspect of an implementation element. For example, a test could ensure that the parameters of a method properly accept the uppermost and lowermost required values. A developer test specifies test entries, execution conditions, and expected results. These details are identified to evaluate a particular aspect of a scenario. When you collect developer tests for a specific implementation element, you can validate that the element performs as specified. The tests should be self-documenting so that it is clear upon completion of the test whether the implementation element has run correctly. Although there is no predefined template for this work product, and testing tools affect how the work product is handled, you should address the following issues:

- Setup
- Inputs
- Script
- Expected Results
- Evaluation Criteria
- Clean-Up

Suggestions and options for representing this work product: **Suggestion:** Automated code unit The most appropriate technique for running these tests is to use code that tests the implementation element scenarios and that you can run regularly as you update the system during development. When code is the sole form of the tests, ensure that the code is self-documenting. The code should document the specifications of the conditions you are testing and the setup or clean-up that is required for the test to run properly.

Option: Manual instructions: In some cases, you can use manual instructions. For example, when testing a user interface, a developer might follow a script, explaining the implementation element. In this case, it is still valuable to create a test harness that goes straight to the user interface. That way, the developer can follow the script without having to follow a complicated set of instructions to find a particular screen or page.

Option: Embedded code: You can use certain technologies (such as Java™ 5 Test Annotation) to embed tests in the implementation. In these cases, there will be a logical work product, but it will be assimilated into the code that you are testing. When you use this option, ensure that the code is self-documenting.

Glossary

These are the purposes of this artifact:

- To record the terms that are being used on the project so that everyone has a common understanding of them

- To achieve consistency by promoting the use of common terminology across the project
- To make explicit different stakeholders' use of the same terms to mean different things or different terms to mean the same thing
- To provide important terms to the Analysis and Design team

This artifact helps you avoid miscommunication by providing two essential resources:

- A central location to look for terms and abbreviations that are new to development team members
- Definitions of terms that are used in specific ways within the domain

Definitions for the glossary terms come from several sources, such as requirements documents, specifications, and discussions with stakeholders and domain experts.

Implementation

The purpose of this artifact is to represent the physical parts that compose the system to be built and to organize the parts in a way that is understandable and manageable. This artifact is the collection of one or more of these elements:

- Source code files
- Data files
- Build scripts
- Other files that are transformed into the executable system

Implementation files are represented as files in the local file system. File folders (directories) are represented as packages, which group the files into logical units.

Iteration Plan

The main objectives of the iteration plan are to provide the team with the following:

- One central place for information regarding iteration objectives
- A detailed plan with task assignments
- Evaluation results

This artifact also helps the team to monitor the progress of the iteration and keeps the results of the iteration assessment that may be useful for improving the next one. This artifact captures the key milestones of an iteration, showing start and end dates, intermediate milestones, synchronization points with other teams, demos, and so on. This artifact is also used to capture issues that need to be solved during the iteration. A few objectives should be written for the iteration, these will help guide the performers throughout the iteration. Also, assess at the end if those objectives have been achieved. The task assignments for an iteration are a subset of all tasks on the Artifact: Work Items List. Therefore, the iteration plan ideally references those work items. The evaluation criteria and iteration assessment information are captured in this artifact, so that it is possible to communicate results and actions from assessments. Work items assigned to an iteration do not necessarily have the same priority. When selecting work items from the Work Items List, the iteration plan may end up having work items with different priorities (for example, you assign the remaining high priority work items, plus a few mid-priority ones from the Work

Items List). Once work items have been assigned to the iteration, the team ensures that they can complete all work, regardless of original work item priorities. Deciding what to develop first on an iteration will vary across projects and iterations. The level of detail or formality of the plan must be adapted to what you need in order to meet these objectives successfully. The plan could, for example, be captured on the following places:

- A whiteboard listing the objectives, task assignments, and evaluation criteria
- A one-page document listing the objectives and evaluation criteria of the iteration, as well as referencing the Work Items List for assignments for that iteration
- A more complex document, supported by a Gantt or Pert chart in a project planning tool

Project Defined Process

The purpose of the project process is to provide guidance and support for the members of the project. “Information at your finger tips” is a metaphor that aligns well with the purpose of this work product. A project process typically describes or references the following items:

- What organizational processes and policies must be adhered to
- What standard process, if any, is being adopted by the project
- Any tailoring of the standard process, or deviations from policy mandates
- Rationale for tailoring and deviations
- Approvals for deviations
- Which work products are reviewed at which milestones, and their level of completion
- Guidelines and information that the project wants to use in addition to the information contained in the main process
- What reviews will be performed, and their level of formality
- What approvals are required, by whom, and when

Processes can be captured informally in documents, formally captured in a Method Composer configuration, or specified by configuring tools. Typically a project will use a combination of these: start with a Method Composer configuration, create a document to describe variations from this configuration and configure tools to support the process being followed.

Project Plan

The purpose of this artifact is to provide a central document where any project team member can find the information on how the project will be conducted. This artifact describes how the project is organized, and identifies what practices will be followed. Additionally, it defines the parameters for tracking project progress, and specifies the high-level objectives of the iterations and their milestones. The project plan allows stakeholders and other team members to understand the big picture and, roughly, when to expect a certain level of functionality be available. Update the plan as often as necessary, usually at the end of each iteration, in order to reflect changing priorities and needs, as well as record the lessons learned from the project. Create

and update the project plan in planning sessions that involve the whole team and appropriate project stakeholders in order to make sure that everybody agrees with it.

Risk List

The purpose of this work product is to capture the perceived risks to the success of the project. This list identifies, in decreasing order of priority, all the risks associated to a project. It serves as a focal point for project activities, and is the basis around which iterations are organized.

System-Wide Requirements

This artifact is used for the following purposes:

- To describe the quality attributes of the system, and the constraints that the design options must satisfy to deliver the business goals, objectives, or capabilities
- To capture functional requirements that are not expressed as use cases
- To negotiate between, and select from, competing design options
- To assess the sizing, cost, and viability of the proposed system
- To understand the service-level requirements for operational management of the solution

This artifact captures the quality attributes and constraints that have system-wide scope. It also captures system-wide functional requirements. This list should capture the critical and serious risks. If you find this list extending beyond 20 items, carefully consider whether they are really serious risks. Tracking more than 20 risks is an onerous task. A representation option for the risk list is to capture it as a section in the coarse-grained plan for the project. This means the coarse-grained plan has to be constantly revisited as you update risks. The fine-grained plans will contain only the tasks that you will be doing to mitigate risks in the short term.

Test Case

The purpose of this work product is

- To provide a way to capture test inputs, conditions, and expected results for a system
- To systematically identify aspects of the software to test
- To specify whether an expected result has been reached, based on the verification of a system requirement, set of requirements, or scenario

A test case specifies the conditions that must be validated to enable an assessment of aspects of the system under test. A test case is more formal than a test idea; typically, a test case takes the form of a specification. In less formal environments, you can create test cases by identifying a unique ID, name, associated test data, and expected results. Test cases can be derived from many sources, and typically include a subset of the requirements (such as use cases, performance characteristics and reliability concerns) and other types of quality attributes.

Test Log

The purpose of this work product is

- To provide verification that a set of tests was run

- To provide information that relates to the success of those tests

This artifact provides a detailed, typically time-based record that both verifies that a set of tests were run and provides information that relates to the success of those tests. The focus is typically on providing an accurate audit trail, which enables you to undertake a post-run diagnosis of failures. This raw data is subsequently analyzed to determine the results of an aspect of the test effort. Because this is a collection of raw data for subsequent analysis, it can be represented in a number of ways:

- For manual tests, log the actual results on a copy of the manual Test Script
- For automated tests, direct the output to log files that you can trace back to the automated Test Script
- Track raw results data in a test management tool

Test Script

Test scripts implement a subset of required tests in an efficient and effective manner.

Tools

These work products are the tools needed to support the software development effort.

Use Case

Use cases are used for the following purposes:

- To reach a common understanding of system behavior
- To design elements that support the required behavior
- To identify test cases
- To plan and assess work
- To write user documentation

A use case typically includes the following information:

- Name: The name of the use case.
- Brief Description: A brief description of the role and purpose of the use case.
- Flow of Events: A textual description of what the system does in regard to a use-case scenario (not how specific problems are solved by the system). Write the description so that the customer can understand it. The flows can include a basic flow, alternative flows, and subflows.
- Key scenarios: A textual description of the most important or frequently discussed scenarios.
- Special Requirements: A textual description that collects all of the requirements of the use case that are not considered in the use-case model, but that must be taken care of during design or implementation (e.g., non-functional requirements).
- Preconditions: A textual description that defines a constraint on the system when the use case starts.
- Post-conditions: A textual description that defines a constraint on the system when the use case ends.

- **Extension points:** A list of locations within the flow of events of the use case at which additional behavior can be inserted by using the extend-relationship.

You can document the use case as a use-case specification document or you can incorporate the use case in a use-case model. You can also use a requirements management tool to capture use cases and parts of use cases.

Use-Case Model

This artifact presents an overview of the intended behavior of the system. It is the basis for agreement between stakeholders and the project team in regards to the intended functionality of the system. It also guides various tasks in the software development lifecycle. Representation options include reports and diagrams from UML modeling tools, graphical representations created by using drawing tools, and drawings on whiteboards. Most of the information in the use-case model is captured in the use-case specifications.

Vision

This artifact provides a high-level basis for more detailed technical requirements. It captures the technical solution by describing the high-level stakeholder requests and constraints that give an overview of the reasoning, background, and context for detailed requirements. The vision serves as input for communicating the fundamental “what and why” for the project and provides a strategy against which all future decisions can be validated. The vision should rally team members behind an idea and give them the context for decision-making in the requirements area. The vision must be visible to everyone on the team. It is good practice to keep this artifact brief, so you can release it to stakeholders as soon as possible, and to make the artifact easy for stakeholders to read and understand. You can accomplish this by including only the most important features and avoiding details of requirements. Projects that focus on product development might extend the marketing section and include a more detailed product position statement that is based on their needs and research. Typically, the vision is represented in a document. If key stakeholder needs are captured in a requirements management tool, this part of the document can be generated by using reporting capabilities. If the vision serves a set of projects or an entire program, the overall vision might be divided into several vision work products. In this case, the vision of the program brings the visions together by providing program-specific content and referencing the subordinate visions.

Work Items List

The purpose of this artifact is to collect all requests for work that will potentially be taken on within the project, so that work can be prioritized, effort estimated, and progress tracked. This artifact provides a focal point for the entire team:

- It provides one list containing all requests for additional capabilities or enhancement for that application. Note that some of these requests may never be implemented, or be implemented in later projects.
- It provides one list of all the work to be prioritized, estimated, and assigned within the project. The risk list is prioritized separately.

- It provides one place to go to for the development team to understand what micro-increments need to be delivered, get references to material required to carry out the work, and report progress made.

These are the typical work items that go on this list:

- Use cases (and references to use-case specifications)
- System-wide requirements
- Changes and enhancement requests
- Defects
- Development tasks

Work items can be very large in scope, especially when capturing requests for enhancements, such as “Support Financial Planning” for a personal finance application. To allow the application to be developed in micro-increments, work items are analyzed and broken down into smaller work items so that they can be assigned to an iteration, such as a use-case scenario for “Calculate Net Worth”. Further breakdown may be required to identify suitable tasks to be assigned to developers, such as “Develop UI for Calculate Net Worth”. This means that work items often have parent/child relationships, where the lowest level is a specification and tracking device for micro-increments. This artifact should consist of the following information for each work item:

- Name and Description
- Priority
- Size Estimate
- State
- References

Assigned work items should also contain the following:

- Target Iteration or Completion Date
- Assignee
- Estimated Effort Remaining
- Hours Worked

Work Items should contain estimates. The recommended representation for the work items list is to capture it as a separate artifact, represented by a spreadsheet or database table. See Example: Work Items List. Alternatively, the work items list may be captured in tools such as project management, requirements management, or change request. In fact, the work items list may be spread over several tools, as you may choose to keep different types of work items in different repositories to take advantage of features in those tools. For example, you could use a requirements composition or management tool to track information about requirements, and use another tool to capture defects. Work items may start in one representation (such as in a spreadsheet) and move to more sophisticated tools over time, as the number of work items and the metrics you wish to gather grows more sophisticated. As part of the Iteration Plan, the plan typically references work items that are assigned to that iteration. If the team is capturing the iteration plan on a whiteboard, for example, the team may choose to reference high-level work items in the Work Items List that are assigned to the iteration, and maintain low-level child work items used to track day-to-day work only in an iteration plan.

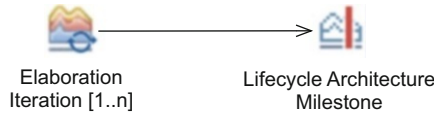


Fig. 22 The elaboration iteration inside the elaboration phase

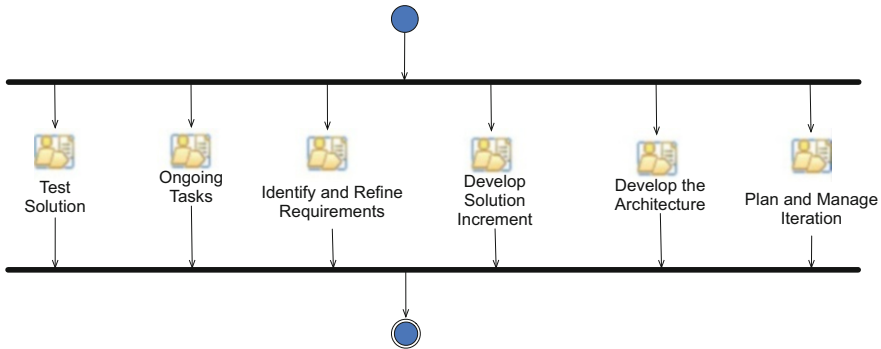


Fig. 23 The Elaboration phase flow of activities

2.2 The Elaboration Phase

The Elaboration starts when the first Milestone, Lifecycle Objectives Milestone, from Inception phase, is available. The Elaboration phase is composed by the Elaboration iteration as described in Fig. 22. The process flow within the iteration is detailed in Fig. 23.

The number and the length of each Elaboration iteration is dependent on, but not limited to, factors such as green-field development compared to maintenance cycle, unprecedented system compared to well-known technology and architecture, and so on. Typically, on the first iteration, it is better to design, implement, and test a small number of critical scenarios to identify what type of architecture and architectural mechanisms you need, so you can mitigate the most crucial risks. You also detail high-risk requirements that have to be addressed early in the project. You test enough to validate that the architectural risks are mitigated. During the subsequent iterations, you fix whatever was not right from the previous iteration. You design, implement, and test the remaining architecturally significant scenarios, ensuring that you check all major areas of the system (architectural coverage), so that potential risks are identified as early as possible [5].

The workflow inside each activity will be detailed in the following subsections (after the description of process roles). The Elaboration phase involves 10 different process roles, 29 work products and six activities (i.e., Plan and Manage Iteration, Identify and Refine Requirements, Develop the Architecture, Develop Solution Increment, Test Solution, Ongoing Tasks), as described in Fig. 24, each activity is

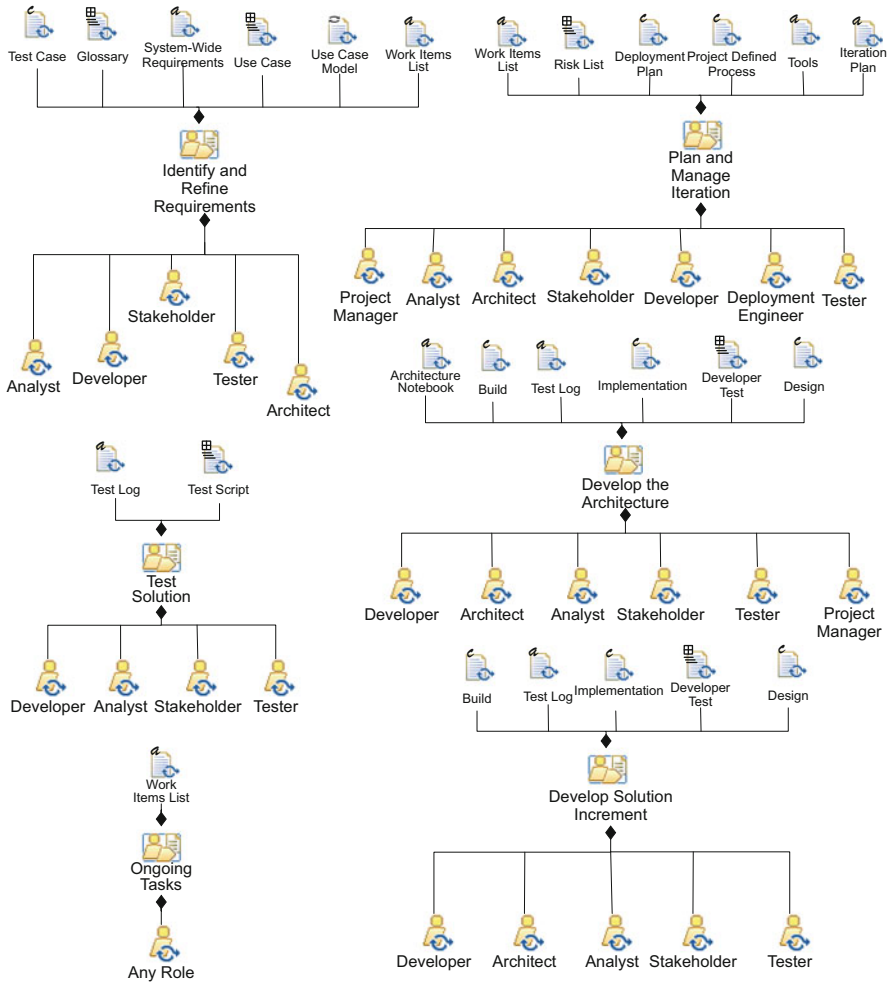


Fig. 24 The elaboration phase described in terms of activities, output work products and involved stakeholders

composed of one or more tasks/activities as described in Figs. 25, 26, 27, 28, and 29.

The description of the Prepare Environment activity in terms of tasks, roles, and work products is reported in Fig. 13. The description of the Identify and Refine Requirements activity in terms of tasks, roles, and work products is reported in Fig. 14.

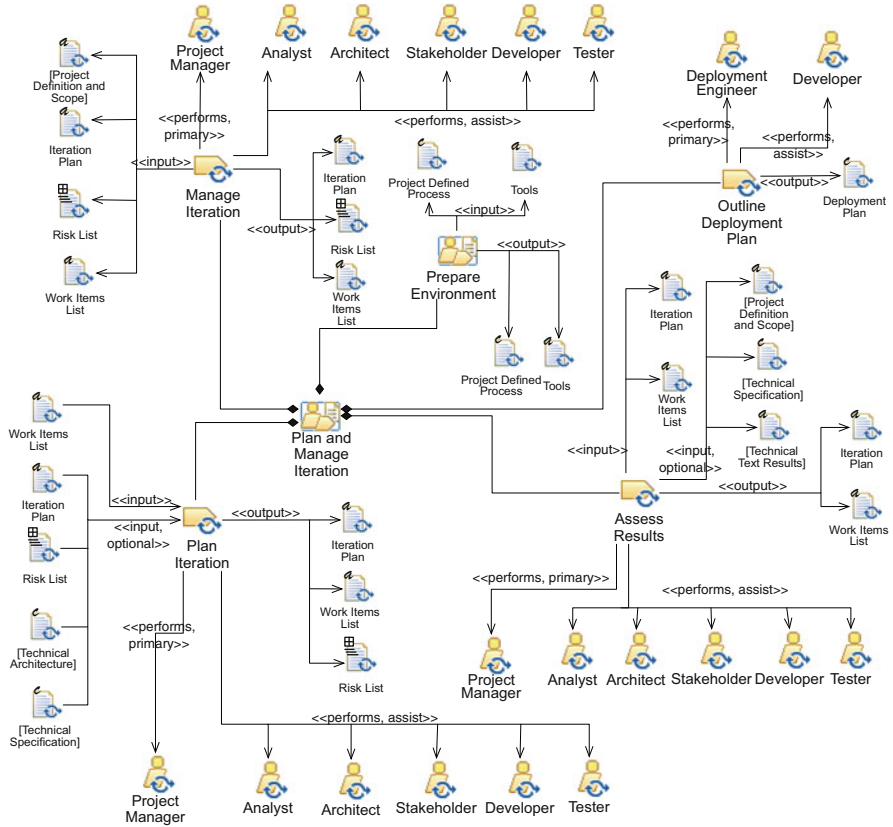


Fig. 25 The Plan and Manage Iteration activity described in terms of tasks, roles, and work products

2.2.1 Process Roles

Ten roles are involved in the Elaboration phase, which are described in the following subsections.

Analyst

She/he is responsible for the following tasks:

1. Detail System-Wide Requirements
2. Detail Use-Case Scenarios
3. Identify and Outline Requirements

She/he assists in the following tasks:

1. Assess Results
2. Create Test Cases
3. Design the Solution
4. Implement Tests

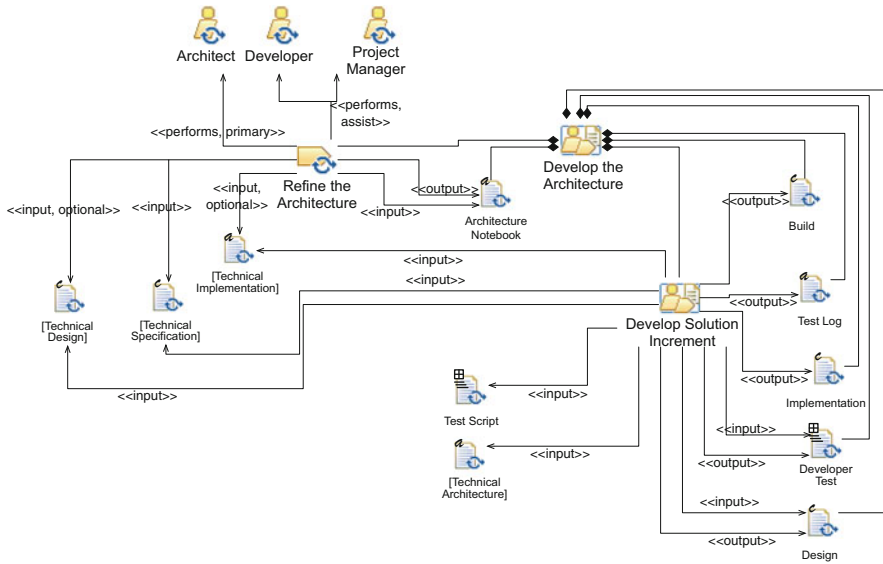


Fig. 26 The Develop the Architecture activity described in terms of tasks, roles, and work products

- 5. Manage Iteration
- 6. Plan Iteration

Any Role

She/he is responsible for the following tasks:

- 1. Request Change

Architect

She/he is responsible for the following task:

- 1. Refine the Architecture
- She/he assists in the following tasks:

- 1. Assess Results
- 2. Design the Solution
- 3. Detail System-Wide Requirements
- 4. Detail Use-Case Scenarios
- 5. Identify and Outline Requirements
- 6. Manage Iteration
- 7. Plan Iteration

Deployment Engineer

She/he is responsible for the following task:

- 1. Outline deployment plan

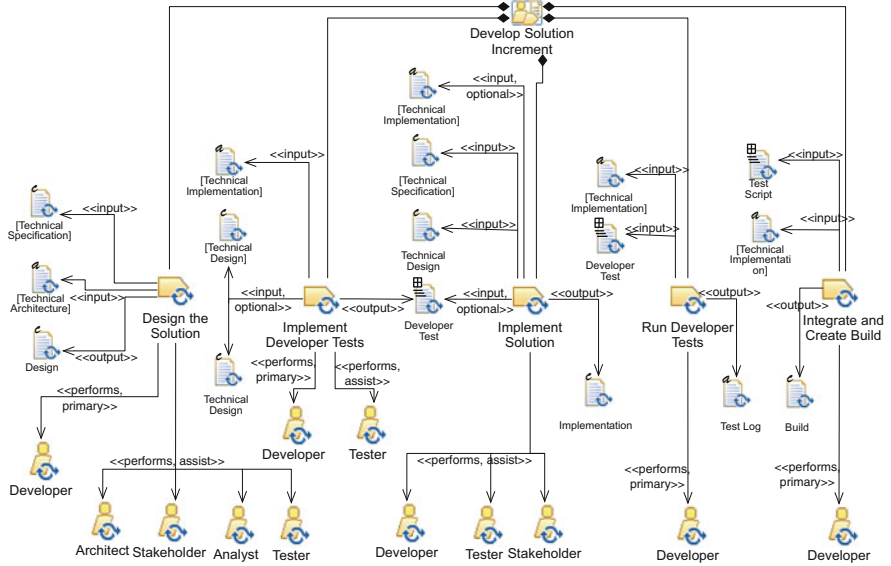


Fig. 27 The Develop Solution Increment activity described in terms of tasks, roles, and work products

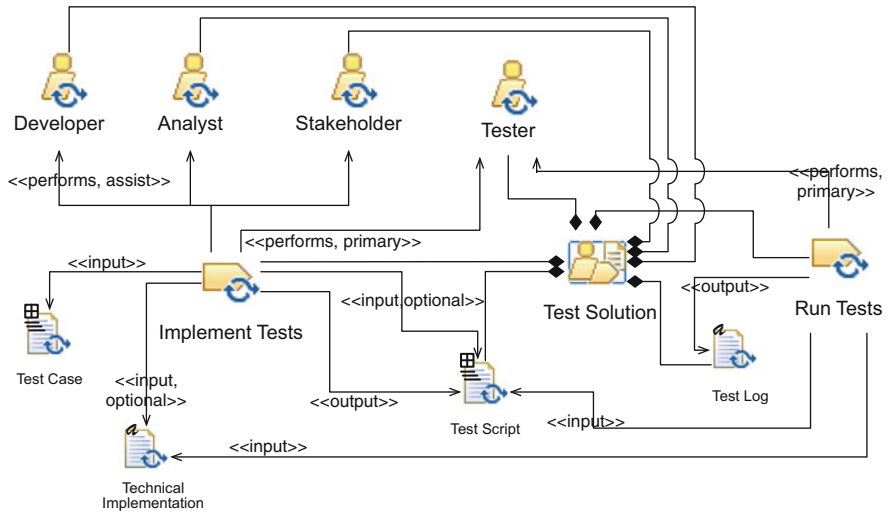


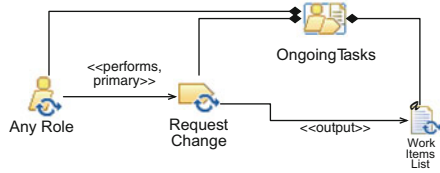
Fig. 28 The Test Solution activity described in terms of tasks, roles, and work products

Developer

She/he is responsible for the following tasks:

1. Design the Solution
2. Implement Developer Tests

Fig. 29 The Ongoing Tasks activity described in terms of tasks, roles, and work products



3. Implement Solution
4. Integrate and Create Build
5. Run Developer Tests

She/he assists in the following tasks:

1. Assess Results
2. Create Test Cases
3. Detail System-Wide Requirements
4. Detail Use-Case Scenarios
5. Identify and Outline Requirements
6. Implement Tests
7. Manage Iteration
8. Outline Deployment Plan
9. Plan Iteration
10. Refine the Architecture

Process Engineer

She/he is responsible for the following tasks:

1. Deploy the process
2. Taylor the process

Project Manager

She/he is responsible for the following task:

1. Assess Results
2. Manage Iteration
3. Plan Iteration

She/he assists in the following task:

1. Refine the Architecture

Stakeholder

She/he assists in the following tasks:

1. Assess Results
2. Create Test Cases
3. Design the Solution
4. Detail System-Wide Requirements
5. Detail Use-Case Scenarios
6. Identify and Outline Requirements
7. Implement Solution

8. Implement Tests
9. Manage Iteration
10. Plan Iteration

Tester

She/he is responsible for the following tasks:

1. Create Test Cases
2. Implement Tests
3. Run Tests

She/he assists in the following tasks:

1. Assess Results
2. Design the Solution
3. Detail System-Wide Requirements
4. Detail Use-Case Scenarios
5. Identify and Outline Requirements
6. Implement Developer Tests
7. Implement Solution
8. Manage Iteration
9. Plan Iteration

Tool Specialist

She/he is responsible for the following tasks:

1. Set Up Tools
2. Verify Tool Configuration and Installation

2.2.2 Activity Details

The Elaboration phase includes six activities, which are described in the following subsections.

Plan and Manage Iteration

The goal of this activity is initiating the iteration, allowing team members to sign up for development tasks, monitoring and communicating project status to external stakeholders, and finally, identify and handling exceptions and problems. The flow of tasks inside this activity is reported in Fig. 30, and the tasks are detailed in Table 8.

Prepare Environment

See section “Prepare Environment” above.

Identify and Refine Requirements

See section “Identify and Refine Requirements” above.

Develop the Architecture

The goal of this activity is to develop the architecturally significant requirements prioritized for this iteration. The flow of tasks inside this activity is reported in Fig. 31, and the tasks are detailed in Table 9.

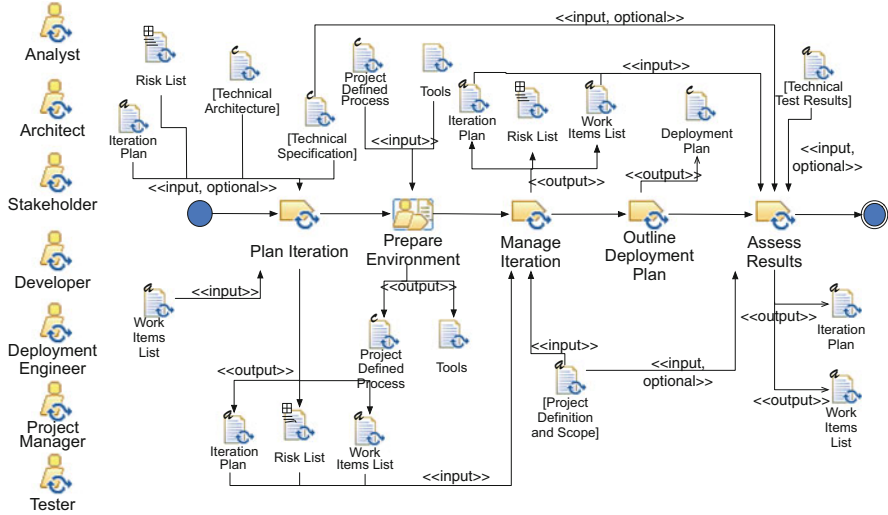


Fig. 30 The flow of tasks of the Plan and Manage Iteration activity

Table 8 Plan and manage iteration—the task description

Activity	Task	Task description	Roles involved
Plan and Manage Iteration	Plan Iteration	The purpose of this task is to identify the next increment of system capability, and create a fine-grained plan for achieving that capability within a single iteration.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).
Plan and Manage Iteration	Manage Iteration	Help the team meet the iteration objectives and keep the project on track. Manage stakeholders’ expectations as technical and practical discoveries are made during the project.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist), (perform), Domain Expert (assist)
Plan and Manage Iteration	Outline Deployment Plan	If a deployment plan for the project already exists, update it to reflect the nature of this release. If this document does not exist, develop a deployment plan to indicate how this release will be rolled out to the production environment.	Deployment Engineer (perform), Developer (assist).
Plan and Manage Iteration	Assess Results	Demonstrate the value of the solution increment that was built during the iteration and apply the lessons learned to modify the project or improve the process.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).

Develop Solution Increment

The goal of this activity is to design, implement, test, and integrate the solution for a requirement within a given context. The flow of tasks inside this activity is reported in Fig. 32, and the tasks are detailed in Table 10.

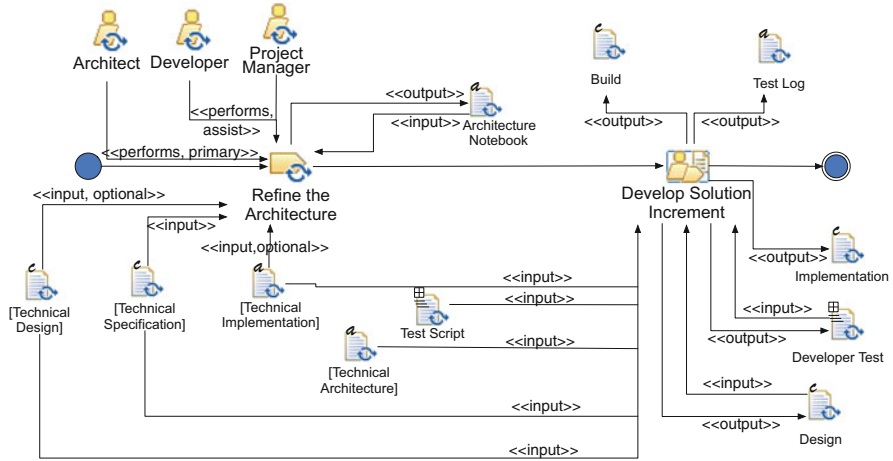


Fig. 31 The flow of tasks of the Develop the Architecture activity

Table 9 Develop the architecture—the task description

Activity	Task	Task description	Roles involved
Develop the Architecture	Develop Solution Increment	<ul style="list-style-type: none"> For developers: To create a solution for the work item for which they are responsible For project managers: To have a goal-based way of tracking project status 	Developer (Primary), architect (additional), Analyst (additional), Stakeholder (additional), Tester (additional)
Develop the Architecture	Refine the Architecture	To make and document the architectural decisions necessary to support development.	Architect (Primary), Developer (additional), Project Manager (additional)

Test Solution

The goal of this activity is to test and evaluate the developed requirements from a system perspective. The flow of tasks inside this activity is reported in Fig. 33, and the tasks are detailed in Table 11.

Ongoing Tasks

The goal of this activity is to perform ongoing tasks that are not necessarily part of the project schedule. The flow of tasks inside this activity is reported in Fig. 34, and the tasks are detailed in Table 12.

2.2.3 Work Products

The Elaboration phase generates thirteen work products. Their relationships with the system metamodel elements are described in Fig. 35.

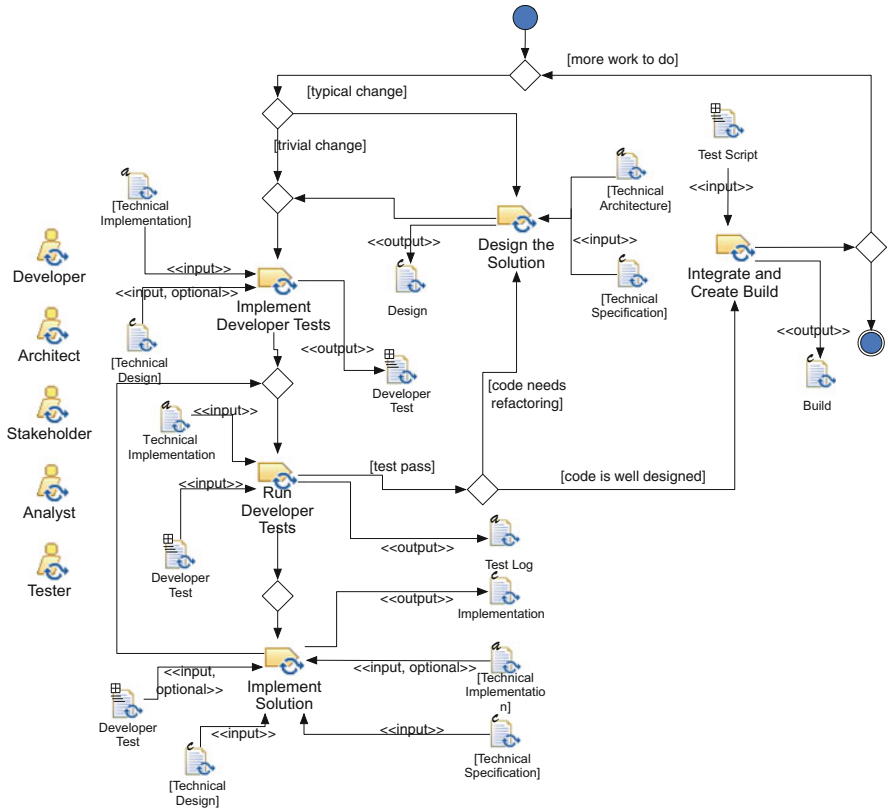


Fig. 32 The flow of tasks of the Develop Solution Increment activity

Work Product Kinds

Table 13 describes the work products of the Elaboration phase according to their kinds.

Architecture Notebook

See section “Architecture Notebook” above.

Build

See section “Build” above.

Deployment Plan

The purpose of this work product is to capture, in one document, the unique information that will be consumed by deployment engineers before and during the deployment to production of a particular release package. The deployment plan should contain the unique instructions for deploying a particular version of a product. By “unique instructions” we mean those things that are not part

Table 10 Develop solution increment—the task description

Activity	Task	Task description	Roles involved
Develop Solution Increment	Design the Solution	Identify the elements and devise the interactions, behavior, relations, and data necessary to realize some functionality. Render the design visually to aid in solving the problem and communicating the solution.	Developer (Primary), Architect (additional), Analyst (additional), Stakeholder (additional), Tester (additional)
Develop Solution Increment	Implement Developer Test	Implement one or more tests that enable the validation of the individual implementation elements through execution.	Developer (Primary), Tester (additional)
Develop Solution Increment	Implement Solution	The purpose of this task is to produce an implementation for part of the solution (such as a class or component), or to fix one or more defects. The result is typically new or modified source code, which is referred to the implementation.	Developer (Primary), Stakeholder (additional), Tester (additional)
Develop Solution Increment	Run Developer Tests	Run tests against the individual implementation elements to verify that their internal structures work as specified.	Developer (Primary)
Develop Solution Increment	Integrate and Create Build	This task describes how to integrate all changes made by developers into the code base and perform the minimal testing to validate the build.	Developer (Primary)

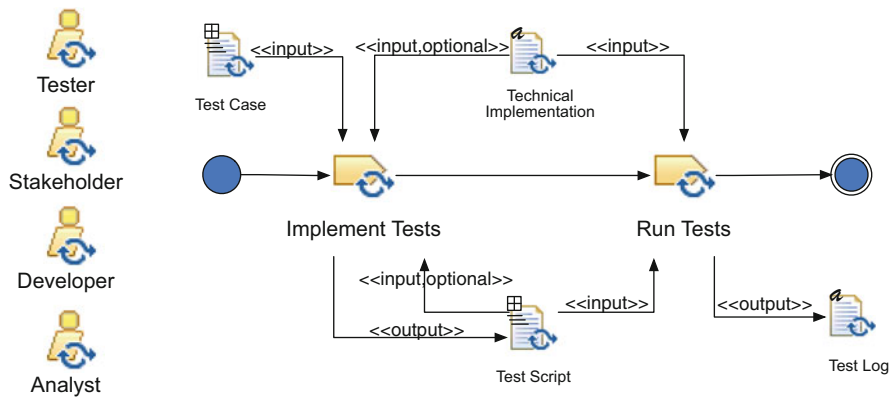


Fig. 33 The flow of tasks of the Test Solution activity

of a deployment engineer’s normal procedures. Rather, they often are specific procedures and timing constraints that a deployment engineer should be aware of as they are rolling out a particular release. While a draft version of the deployment plan is typically developed by a development team, the deployment engineer is responsible for its contents and existence. A deployment plan normally consists of the following sections:

- The scope of the release and a general overview of the capabilities to be deployed
- The timing and dependencies for deploying components to various nodes
- The risks or issues associated with the release based on a risk assessment

Table 11 Test solution—the task description

Activity	Task	Task description	Roles involved
Test Solution	Implement Tests	To implement step-by-step Test Scripts that demonstrate the solution satisfies the requirements.	Tester (Primary), Analyst (Additional), Developer (Additional), Stakeholder (Additional)
Test Solution	Run Tests	Run the appropriate tests scripts, analyze results, articulate issues and communicate test results to the team.	Tester (Primary)

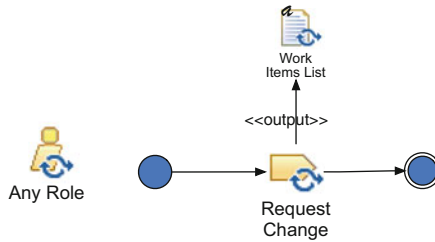


Fig. 34 The flow of tasks of the Ongoing Tasks activity

Table 12 Ongoing tasks—the task description

Activity	Task	Task description	Roles involved
Ongoing Tasks	Request Change	Capture and record change requests.	Any Role (perform)
Roles Identification	Design Scenarios	Each scenario is designed in form of sequence diagram thus depicting the details of agents interactions	System Analyst (perform), Domain Expert (assist)

- The customer organization, stakeholders, and end user community that will be impacted by the release
- The person or persons who have the authority to approve the release as “ready for production”
- The development team members responsible for delivering the release package to the Deployment Manager, along with contact information
- The approach for transitioning the release package to the Deployment Engineer, including appropriate communications protocols and escalation procedures
- The success criteria for this deployment; in other words, how will the Deployment Engineer know that the release is successful so it can report success

Design

See section “Design” above.

Developer Test

See section “Developer Test” above.

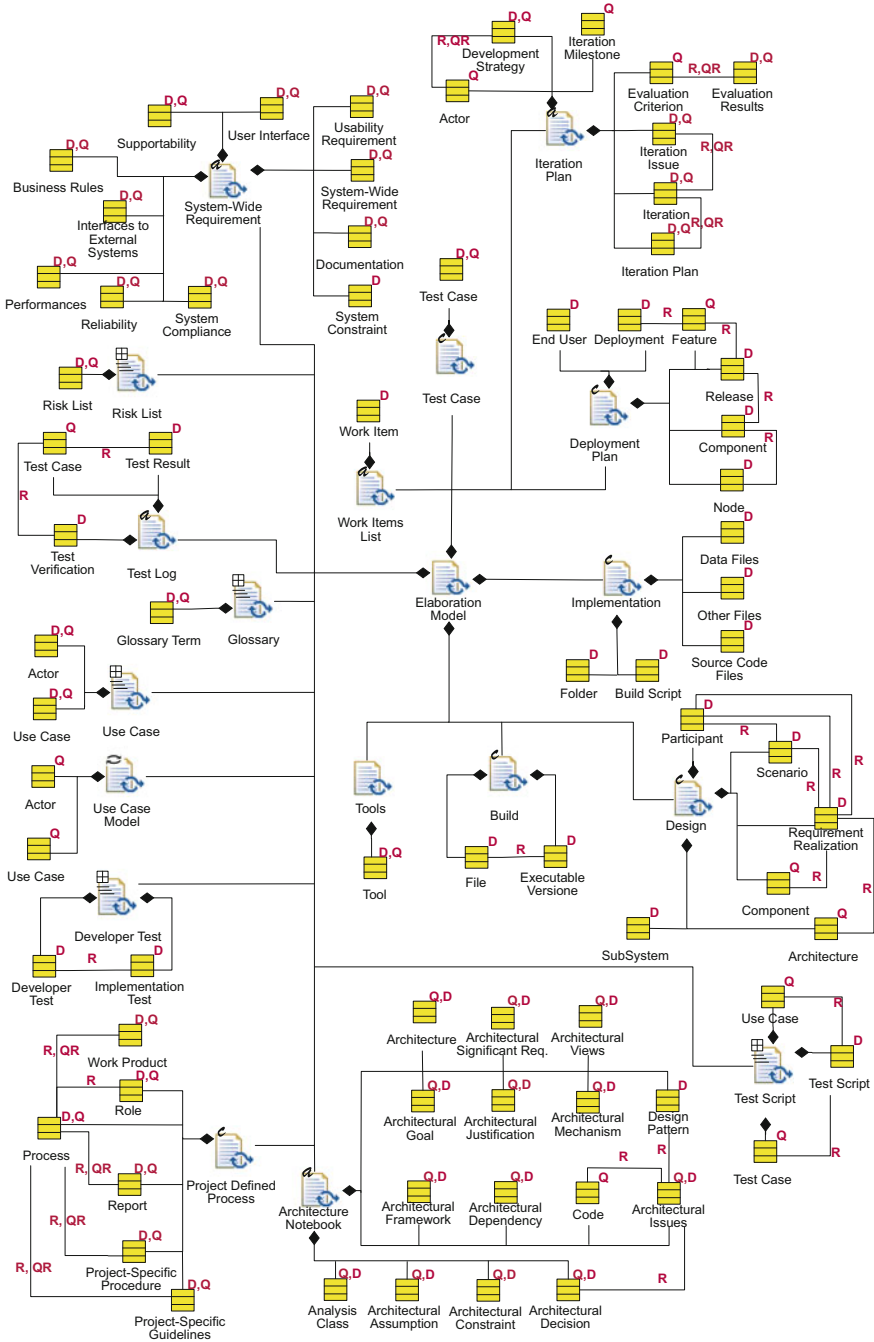


Fig. 35 The Elaboration phase documents structure

Table 13 Elaboration phase—work product kinds

Name	Description	Work product kind
Architecture Notebook	See sections “Architecture Notebook” and “Work Product Kinds”.	
Build	See sections “Build” and “Work Product Kinds”.	
Deployment Plan	A deployment plan is used to document all the information needed by deployment engineers to deploy a release successfully.	Composite
Design	See sections “Design” and “Work Product Kinds”.	
Developer Test	See sections “Developer Test” and “Work Product Kinds”.	
Glossary	See sections “Glossary” and “Work Product Kinds”.	
Implementation	See sections “Implementation” and “Work Product Kinds”.	
Iteration Plan	See sections “Iteration Plan” and “Work Product Kinds”.	
Project Defined Process	See sections “Project Defined Process” and “Work Product Kinds”.	
Project Plan	See sections “Project Plan” and “Work Product Kinds”.	
Risk List	See sections “Risk List” and “Work Product Kinds”.	
System-Wide Requirement	See sections “System-Wide Requirements” and “Work Product Kinds”.	
Technical Architecture	See section “Work Product Kinds”.	
Technical Design	This slot serves as an abstraction of high-level artifacts that describe the realization of required system functionality, and serves as an abstraction of the solution. Fulfilling Work Products: <ul style="list-style-type: none"> • Design 	
Technical Implementation	This slot serves as an abstraction of high-level artifacts that describe the realization of required system functionality, and serves as an abstraction of the solution. Fulfilling Work Products: <ul style="list-style-type: none"> • Build • Implementation 	
Technical Specification	See section “Work Product Kinds”	
Technical Test Results	See section “Work Product Kinds”.	
Test Case	See sections “Test Case” and “Work Product Kinds”.	
Test Log	See sections “Test Log” and “Work Product Kinds”.	
Test Script	See sections “Test Script” and “Work Product Kinds”.	
Tools	See sections “Tools” and “Work Product Kinds”.	
Use Case	See sections “Use Case” and “Work Product Kinds”.	
Use-Case Model	See sections “Use Case Model” and “Work Product Kinds”.	
Vision	See sections “Vision” and “Work Product Kinds”.	
Work Items List	See sections “Work Items List” and “Work Product Kinds”.	

Glossary

See section “Glossary” above.

Implementation

See section “Implementation” above.

Iteration Plan

See section “Iteration Plan” above.

Project Defined Process

See section “Project Defined Process” above.

Release

The purpose of this work product is to

- Bring, at the team level, closure to a sprint/iteration or series of sprint/iterations by delivering working, tested software that can be potentially used by the end user community for whom the system was (or is being) developed.
- Deliver, at the program level, an integrated, value-added product increment to customers that was developed in parallel by multiple, coordinated, and synchronized development team members. A release consists of integrated, compiled code that runs cleanly, independently, and in its entirety. This is an important rule because in order to be released or even “potentially shippable,” a release increment must be able to stand on its own, otherwise it is not shippable. Releases can be created at either the program or team level.

There are three potential uses for a release:

- It is not used outside of the program: It has been produced to minimize risks linked to technology and a program’s capability to integrate components and to produce a Build. This situation usually happens at the beginning of a new product lifecycle.
- It is used by beta customers and the program manager: It allows end users to test it in a Beta test environment, which provides immediate feedback and reduces risks associated with user interface ergonomics. customer feedback will influence the program backlog for later consideration.
- It is deployed or shipped and used by end users: This result provides immediate value to the end users.

In many organizations, a program release typically is timeboxed to 2–3 months of development effort and 2–4 weeks of hardening which results in a scheduled release approximately every 90 days. Releases for individual development teams usually occur more often than those for programs, but there are no hard and fast rules regarding how often releases should be scheduled. The only requirement is that working software should be delivered “frequently” by both development teams and programs. Although the example timeframe described above is arbitrary, empirical evidence suggests it is about right for most companies and fits nicely into quarterly planning cycles. Each release has a set of release objectives and a projected delivery date; it also has a planned number of sprint/iterations.

Release Communications

The purpose of this work product is to inform all the various stakeholders that a release to production has taken place and the implemented features are now generally available. Sometimes, depending on the product user base, separate communiques might need to be prepared for each stakeholder group. In that case, this artifact should specify the different groups to which communications are directed, the method of communication (e.g., email, text or pager message, bulletin, newsletter, phone message, etc.). All communiques should be prepared in advance so that it is a matter of disseminating information when the release to production has been determined to be successful. Also included in this artifact is a listing of

the responsible parties who will execute the communications when a successful release has been declared (normally the Deployment Engineer), as well as the timing and dependencies of the communiques. While there is no prescribed format for the release communications artifact, each communique should indicate the preferred delivery mechanisms (e.g., beeper notification, telephone calls, a posting to an internal release website, live or pre-recorded presentations by senior management, etc.) and generally answer the following questions:

- Who are the parties (stakeholders) that are interested in knowing that a release to production has taken place?
- What specifically (features, functions, components) has been placed into production?
- Why is this release valuable to stakeholders and what business purpose does it serve?
- Where is the product available (on which platforms, geographical locations, business units, etc.)?
- How can the stakeholders access the system and under what circumstances?
- When was the product released (or when will it be released if the release date is in the future)?

Risk List

See section “Risk List” above.

System Wide Requirements

See section “System-Wide Requirements” above.

Test Case

See section “Test Case” above.

Test Log

See section “Test Log” above.

Test Script

See section “Test Script” above.

Tools

See section “Tools” above.

Use Case

See section “Use Case” above.

Use-Case Model

See section “Use Case Model” above.

Vision

See section “Vision” above.

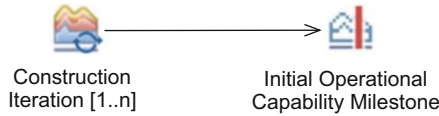


Fig. 36 The construction iteration inside the construction phase

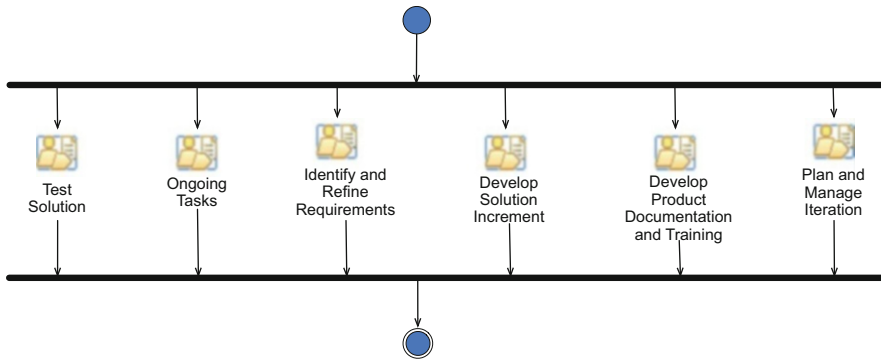


Fig. 37 The construction phase flow of activities

Work Items List

See section “Work Items List” above.

2.3 The Construction Phase

The Construction phase is composed by the Construction iteration as described in Fig. 36. The process flow within the iteration is detailed in Fig. 37.

The workflow inside each activity will be detailed in the following subsections (after the description of process roles). The Construction phase involves eight different process roles, five work products (four UML models and four text documents) and four guidance documents (one for each UML model) as described in Fig. 38. The phase is composed of six activities (i.e., Plan and Manage Iteration, Identify and Refine Requirements, Develop Solution Increment, Test Solution, Ongoing Tasks, Develop Product Documentation and Training), each of them composed of one or more tasks. Details of new activities are reported below (Figs. 39 and 40).

The description of the Identify and Refine Requirements activity in terms of tasks, roles, and work products is reported in Fig. 14.

The description of the Develop Solution Increment activity in terms of tasks, roles, and work products is reported in Fig. 27.

The description of the Test Solution activity in terms of tasks, roles, and work products is reported in Fig. 28. The description of the Ongoing Tasks activity in terms of tasks, roles, and work products is reported in Fig. 29.

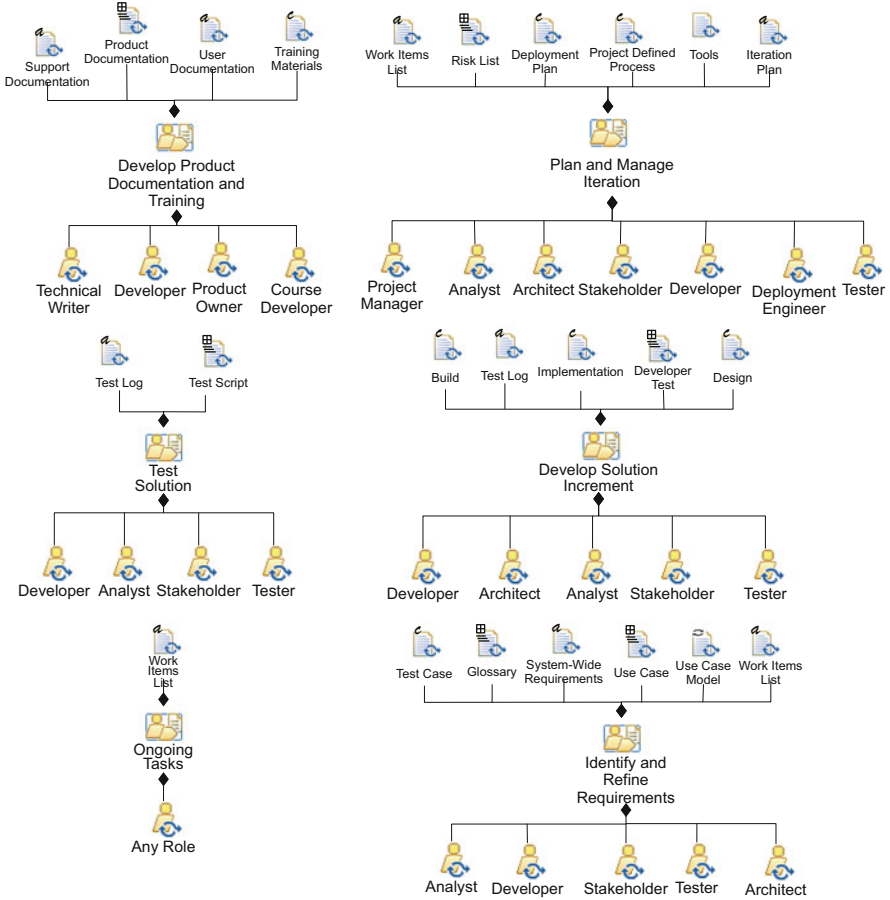


Fig. 38 The Construction phase described in terms of activities, output work products and involved stakeholders

2.3.1 Process Roles

Ten roles are involved in the Construction phase, which are described in the following subsections.

Analyst

She/he is responsible for the following tasks:

1. Detail System-Wide Requirements
2. Detail Use-Case Scenarios
3. Identify and Outline Requirements

She/he assists in the following tasks:

1. Assess Results
2. Create Test Cases

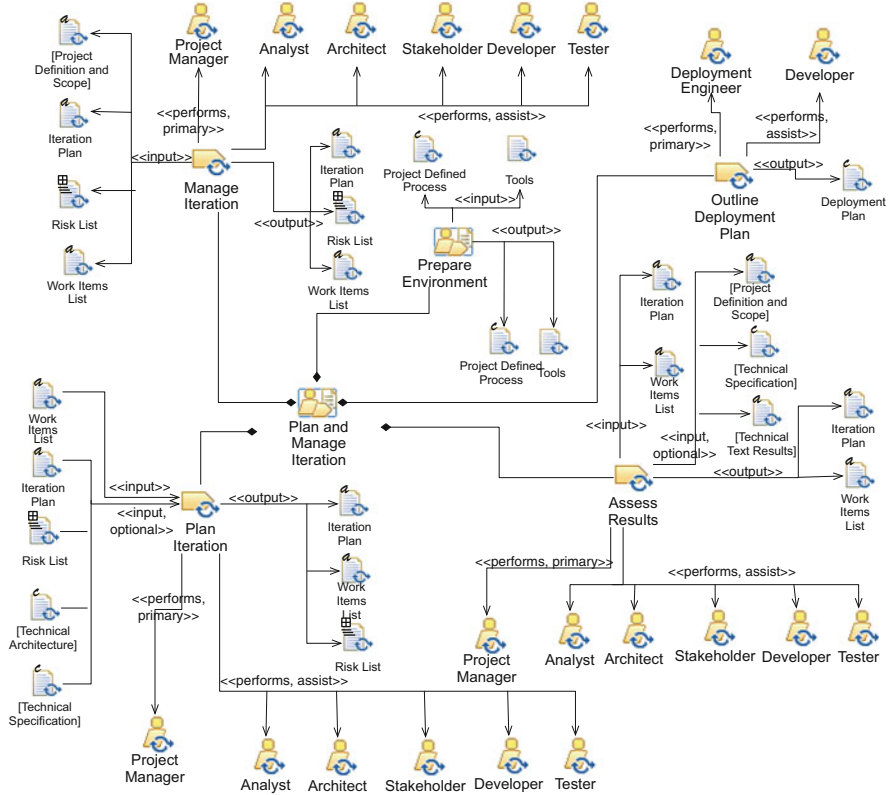


Fig. 39 The Plan and Manage Iteration activity described in terms of tasks, roles, and work products

3. Design the Solution
4. Implement Tests
5. Manage Iteration
6. Plan Iteration

Any Role

She/he is responsible for the following tasks:

1. Request Change

Architect

She/he is responsible for the following task:

1. Refine the Architecture

She/he assists in the following tasks:

1. Assess Results
2. Design the Solution

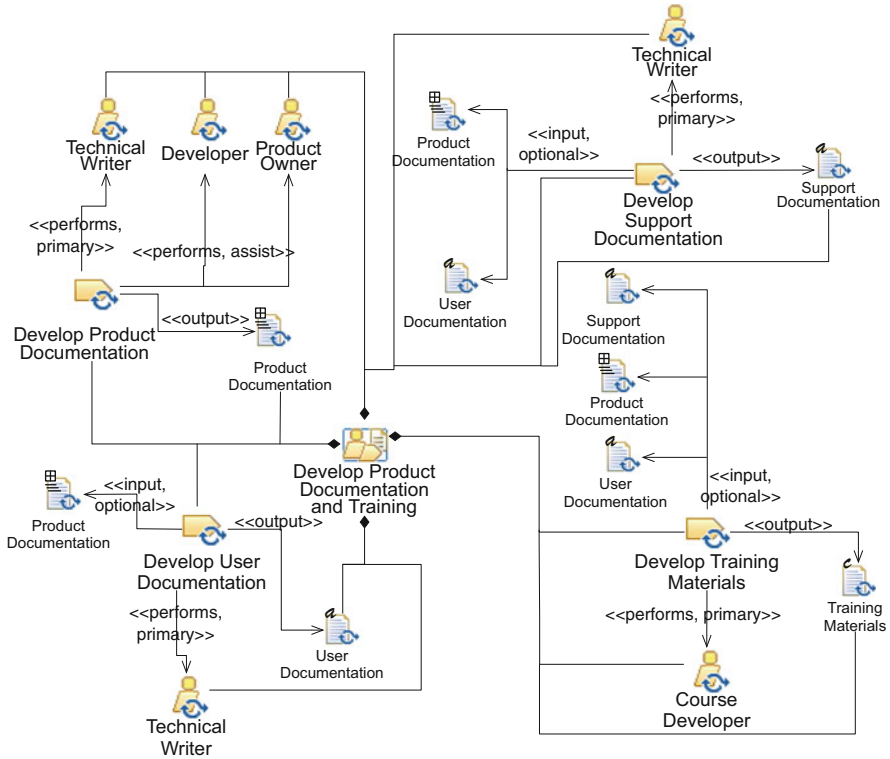


Fig. 40 The Develop Product Documentation and Training activity described in terms of tasks, roles, and work products

3. Detail System-Wide Requirements
4. Detail Use-Case Scenarios
5. Identify and Outline Requirements
6. Manage Iteration
7. Plan Iteration

Course Developer

She/he is responsible for the following tasks:

1. Develop Training Materials

Deployment Engineer

She/he is responsible for the following task:

1. Plan deployment

Developer

She/he is responsible for the following tasks:

1. Design the Solution
 2. Implement Developer Tests
 3. Implement Solution
 4. Integrate and Create Build
 5. Run Developer Tests
- She/he assists in the following tasks:
1. Assess Results
 2. Create Test Cases
 3. Detail System-Wide Requirements
 4. Detail Use-Case Scenarios
 5. Develop Product Documentation
 6. Identify and Outline Requirements
 7. Implement Tests
 8. Manage Iteration
 9. Outline Deployment Plan
 10. Plan Iteration

Process Engineer

She/he is responsible for the following tasks:

1. Deploy the process
2. Taylor the process

Product Owner

She/he assists in the following task:

1. Develop Product Documentation

Project Manager

She/he is responsible for the following task:

1. Assess Results
2. Manage Iteration
3. Plan Iteration

Stakeholder

She/he assists in the following tasks:

1. Assess Results
2. Create Test Cases
3. Design the Solution
4. Detail System-Wide Requirements
5. Detail Use-Case Scenarios
6. Identify and Outline Requirements
7. Implement Solution
8. Implement Tests
9. Manage Iteration
10. Plan Iteration

Technical Writer

She/he is responsible for the following task:

1. Develop Product Documentation
2. Develop Support Documentation
3. Develop User Documentation

Tester

She/he is responsible for the following tasks:

1. Create Test Cases
2. Implement Tests
3. Run Tests

She/he assists in the following tasks:

1. Assess Results
2. Design the Solution
3. Detail System-Wide Requirements
4. Detail Use-Case Scenarios
5. Identify and Outline Requirements
6. Implement Developer Tests
7. Implement Solution
8. Manage Iteration
9. Plan Iteration

Tool Specialist

She/he is responsible for the following tasks:

1. Set Up Tools
2. Verify Tool Configuration and Installation

2.3.2 Activity Details

The Construction phase includes six activities, which are described in the following subsections.

Plan and Manage Iteration

This activity is performed throughout the project lifecycle. The goal of this activity is to identify risks and issues early enough that they can be mitigated, to establish the goals for the iteration, and to support the development team in reaching these goals. The project manager and the team launch the iteration. The prioritization of work for a given iteration takes place. The project manager, stakeholders, and team members agree on what is supposed to be developed during that iteration. Team members sign up for the work items they will develop in that iteration. Each team member breaks down the work items into development tasks and estimates the effort. This provides a more accurate estimate of the amount of time that will be spent, and of what can be realistically achieved, in a given iteration. As the iteration runs, the team meets regularly to report status of work completed, the work to do next, and issues blocking the progress. In some projects, this status checking occurs in daily meetings, which allows for a more precise understanding of how

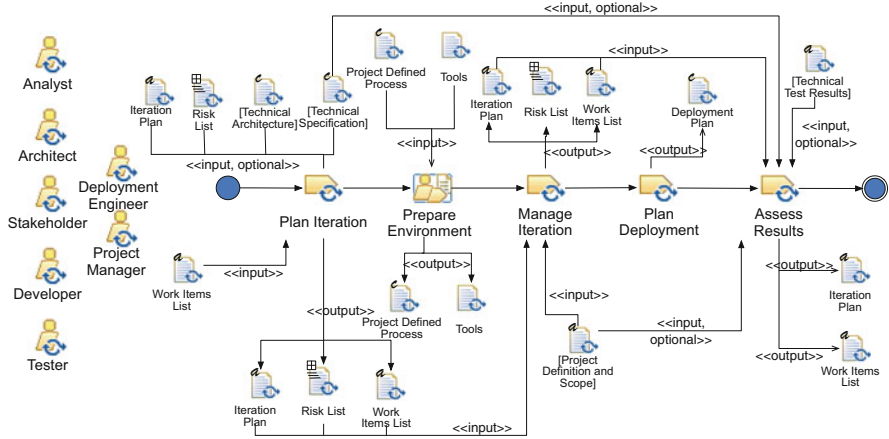


Fig. 41 The flow of tasks of the Plan and Manage Iteration activity

Table 14 Plan and manage iteration—the task description

Activity	Task	Task description	Roles involved
Plan and Manage Iteration	Plan Iteration	The purpose of this task is to identify the next increment of system capability, and create a fine-grained plan for achieving that capability within a single iteration.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).
Plan and Manage Iteration	Manage Iteration	Help the team meet the iteration objectives and keep the project on track. Manage stakeholders’ expectations as technical and practical discoveries are made during the project.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist). (perform), Domain Expert (assist)
Plan and Manage Iteration	Plan Deployment	If a deployment plan for the project already exists, update it to reflect the nature of this release. If this document does not exist, develop a deployment plan to indicate how this release will be rolled out to the production environment.	Deployment Engineer (perform), Developer (assist).
Plan and Manage Iteration	Assess Results	Demonstrate the value of the solution increment that was built during the iteration and apply the lessons learned to modify the project or improve the process.	Project Manager (perform), Analyst (assist), Architect (assist), Developer (assist), Stakeholder (assist), Tester (assist).

the work in an iteration is progressing. As necessary, the team makes corrections to achieve what was planned. The overall idea is that risks and issues are identified and managed throughout the iteration, and everyone knows the project status in a timely manner. During iteration assessments, the key success criterion is the demonstration that planned functionality has been implemented. Lessons learned are captured in order to modify the project or improve the process. If the iteration end coincides with the phase end, make sure the objectives for that phase have been met. The flow of tasks inside this activity is reported in Fig. 41, and the tasks are detailed in Table 14.

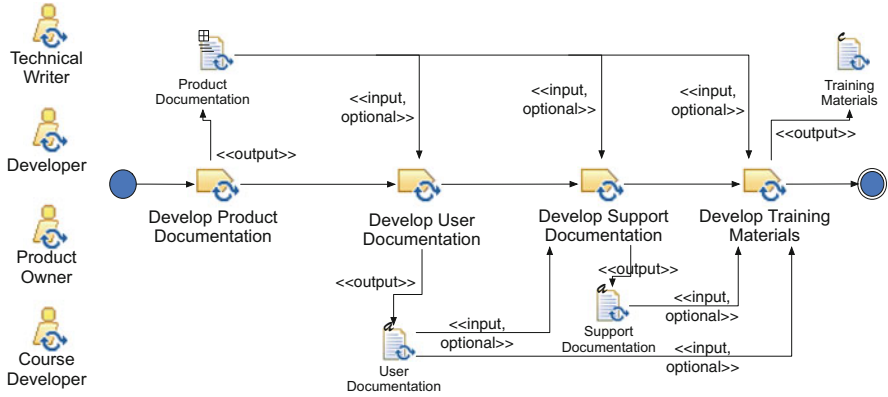


Fig. 42 The flow of tasks of the Develop Product Documentation and Training activity

Prepare Environment

See section “Prepare Environment” above.

Identify and Refine Requirements

See section “Identify and Refine Requirements” above.

Develop Solution Increment

See section “Develop Solution Increment” above.

Test Solution

See section “Test Solution” above.

Ongoing Tasks

See section “Ongoing Tasks” above.

Develop Product Documentation and Training

The goal of this activity is to prepare product documentation and training materials. The flow of tasks inside this activity is reported in Fig. 42, and the tasks are detailed in Table 15.

2.3.3 Work Products

The Construction phase generates four work products. Their relationships with the system meta-model elements are described in Fig. 43.

WorkProduct Kinds

Table 16 describes the work products of the Construction phase according to their kinds.

Table 15 Develop product documentation and training—the task description

Activity	Task	Task Description	Roles Involved
Develop Product Documentation and Training	Develop Product Documentation	The purpose of this task is to document enough information about the features that were developed in a particular release to be useful to customers throughout the life of the product.	Technical Writer (perform), Developer (assist), Product Owner (assist).
Develop Product Documentation and Training	Develop User Documentation	The purpose of this task is to provide useful information to end users of the product being released into production.	Technical Writer (perform), Developer (assist), Product Owner (assist).
Develop Product Documentation and Training	Develop Support Documentation	The purpose of this task is to ensure that the personnel who are tasked with supporting the system have enough information about the product to perform their jobs effectively after the product has been placed into production.	Technical Writer (perform)
Develop Product Documentation and Training	Develop training Materials	The purpose of this task is to enable adoption of the product and to encourage its proper use.	Course Developer (perform)

Product Documentation

Product documentation is created for the benefit of the marketing arm of an organization, the program manager, and those people who must assess the business value of a particular system. This is an often overlooked aspect of development team implementation. The team should consider that the customers of a particular release usually are not technical themselves, but do require a detailed enough understanding of how their product operates and how it meets stated business goals and needs

User Documentation

User documentation might include all or parts of user manuals (electronic or paper-based), tutorials, frequently asked questions (FAQs), on-line Help Files, installation instructions, work instructions, operational procedures, etc.

Support Documentation

Support documentation usually is developed for the three common tiers of a support organization. Tier 1 typically is the Help Desk where users call when they have a problem with a particular system. Tier 1 support personnel normally answer basic questions and, if necessary, log a ticket and escalate it to the appropriate Level 2 support desk.

Tier 2 support personnel may deal with more complex questions or issues regarding an application and might need to do some research on the characteristics of the system to provide an answer. If that person cannot resolve the issue, the ticket is escalated to Tier 3 support personnel who have a deeper understanding of the application’s code and the technology support the system’s architecture.

To properly convey the necessary information to each support tier, the application’s code base should be well commented and logically organized. This approach will facilitate the development of the support documentation. Support documentation typically includes

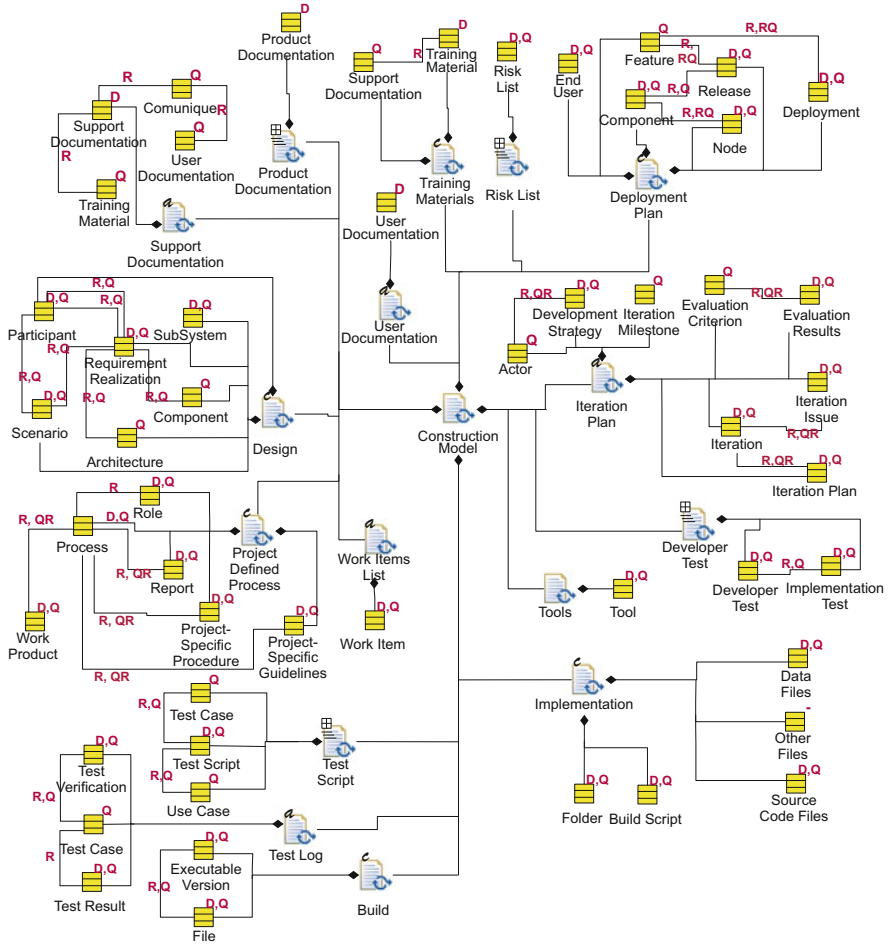


Fig. 43 The Construction phase documents structure

- User manuals with work instructions, process descriptions, and procedures
- Communications, training, and knowledge transfer deliverables
- Support and operations manuals
- Service information, including Help Desk scripts

Training Materials

Training materials that can be used to train end users and production support personnel might consist of

- Presentation slides
- Handouts
- Job aids
- Tutorials

Table 16 Construction phase—work product kinds

Name	Description	Work product kind
Architecture Notebook	See sections “Architecture Notebook” and “Work Product Kinds”.	
Build	See sections “Build” and “Work Product Kinds”.	
Deployment Plan	See sections “Deployment Plan” and “WorkProduct Kinds”.	Composite
Design	See sections “Design” and “Work Product Kinds”.	
Developer Test	See sections “Developer Test” and “Work Product Kinds”.	
Glossary	See sections “Glossary” and “Work Product Kinds”.	
Implementation	See sections “Implementation” and “Work Product Kinds”.	
Iteration Plan	See sections “Iteration Plan” and “Work Product Kinds”.	
Product Documentation	Information about a specific product that has been captured in an organized format.	Structured
Project Defined Process	See sections “Project Defined Process” and “Work Product Kinds”.	
Project Plan	See sections “Project Plan” and “Work Product Kinds”.	
Risk List	See sections “Risk List” and “Work Product Kinds”.	
Support Documentation	Documents used by members of a production support team that provide information about how to service and support a specific product.	Free Text
System-Wide Requirement	See sections “System-Wide Requirements” and “Work Product Kinds”.	
Test Case	See sections “Test Case” and “Work Product Kinds”.	
Test Log	See sections “Test Log” and “Work Product Kinds”.	
Test Script	See sections “Test Script” and “Work Product Kinds”.	
Tools	See sections “Tools” and “Work Product Kinds”.	
Training Materials	This work product represents all the materials needed to train end users and production support personnel on the features and inner workings of a product for a particular release.	Composite.
Use Case	See sections “Use Case” and “Work Product Kinds”.	
Use-Case Model	See sections “Use Case Model” and “Work Product Kinds”.	
User Documentation	Documents that can be utilized by the end users of a particular system or product. This type of documentation typically is written in a way that enables system users to easily find information they need to use the product.	Free Text.
Vision	See sections “Vision” and “Work Product Kinds”.	
Work Items List	See sections “Work Items List” and “Work Product Kinds”.	

- On-line demos
- Video vignettes
- Lab exercises
- Quizzes
- Workshop materials, etc.

2.4 The Transition Phase

The Transition phase is composed by the Inception iteration as described in Fig. 44. The process flow within the iteration is detailed in Fig. 45.

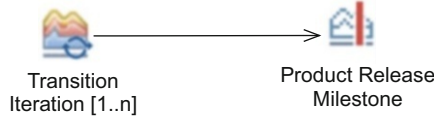


Fig. 44 The Transition iteration inside the Transition phase

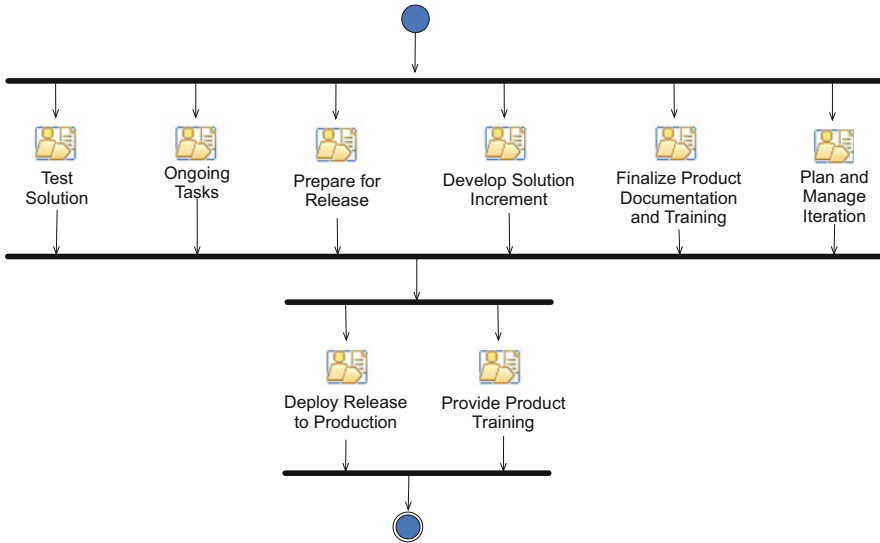


Fig. 45 The Transition phase flow of activities

The workflow inside each activity will be detailed in the following subsections (after the description of process roles). The Transition phase involves thirteen different process roles, seventeen work products as described in Fig. 46.

The phase is composed of eight activities each of them composed of one or more tasks as described in the following. Activities are Plan and Manage Iteration, Develop Solution Increment, Test Solution, Finalize Product Documentation and Training, Prepare for Release, Package the Release, Ongoing Tasks, Provide Product Training. The description of the Plan and Manage Iteration activity, Develop Solution Increment activity and Test Solution activity in terms of tasks, roles, and work products can be found respectively in Figs. 12, 27, and 28.

Details of new activities are reported in Figs. 47, 48, 49, and 50.

2.4.1 Process Roles

Analyst

She/he assists in the following tasks:

1. Assess Results
2. Design the Solution

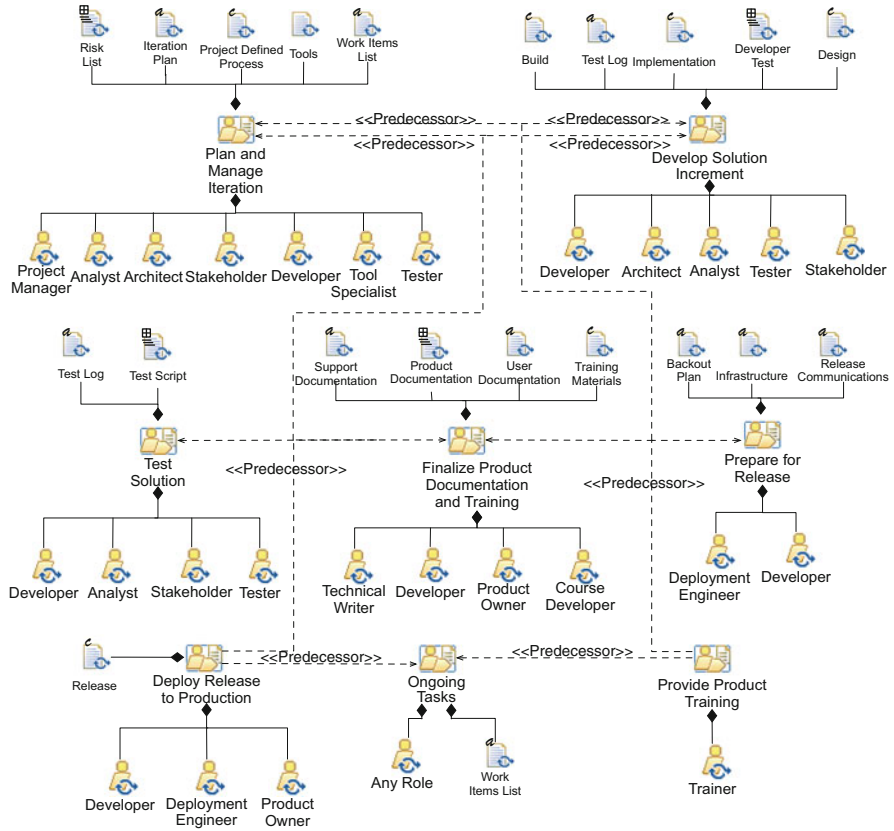


Fig. 46 The Transition phase described in terms of activities, output work products and involved

3. Implement Tests
4. Manage Iteration
5. Plan Iteration

Any Role

She/he is responsible for the following tasks:

1. Request Change

Architect

She/he assists in the following tasks:

1. Assess Results
2. Design the Solution
3. Manage Iteration
4. Plan Iteration

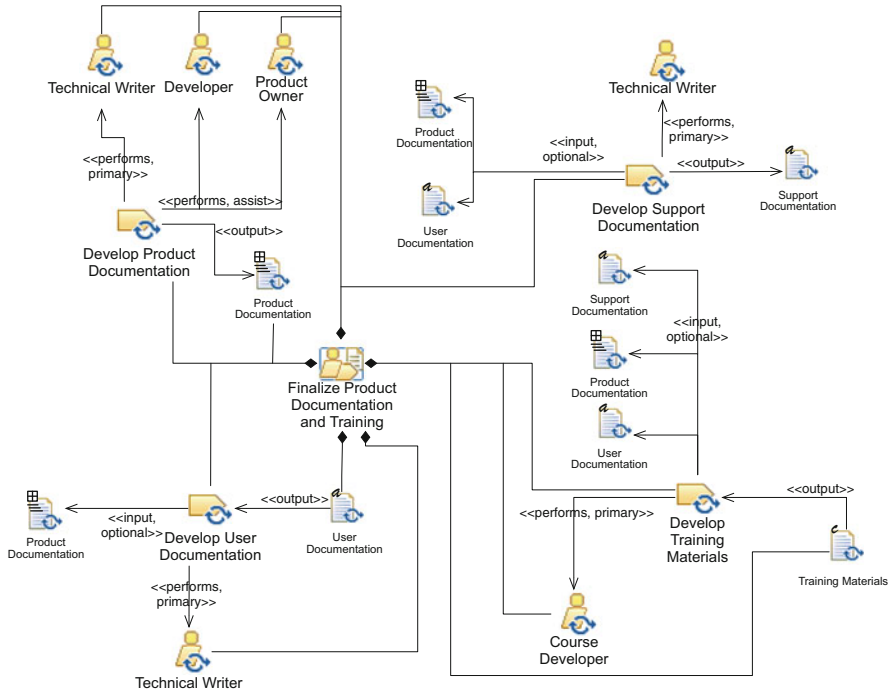


Fig. 47 The Finalize Product Documentation and Training activity described in terms of tasks, roles, and work products

Course Developer

She/he is responsible for the following tasks:

1. Develop Training Materials

Deployment Engineer

She/he is responsible for the following tasks:

1. Deliver Release Communications
2. Develop Release Communications
3. Execute Backout Plan (if necessary)
4. Execute Deployment Plan
5. Install and Validate Infrastructure
6. Verify Successful Deployment
7. Develop Backout Plan

She/he assists in the following tasks:

1. Package the Release

Developer

She/he is responsible for the following tasks:

1. Design the Solution
2. Develop Backout Plan

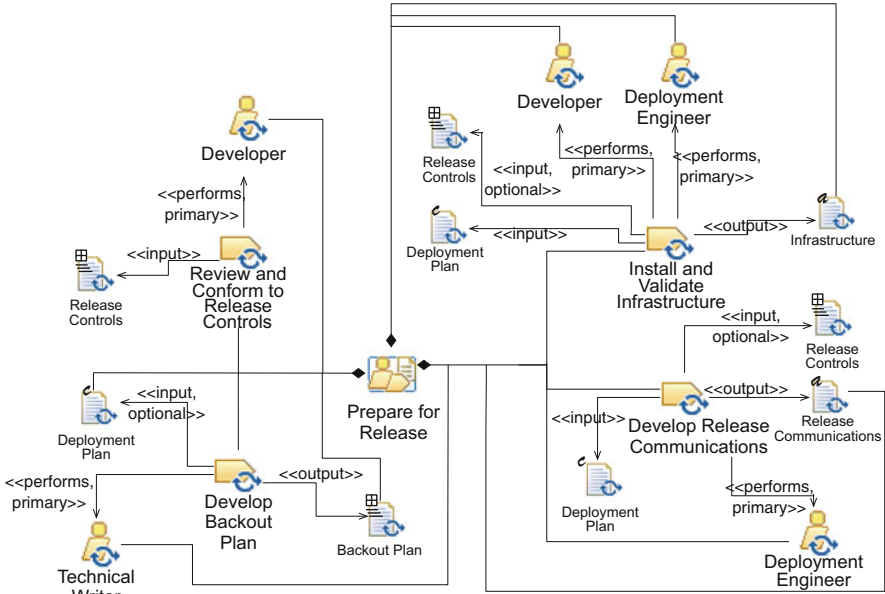


Fig. 48 The Prepare for Release activity described in terms of tasks, roles, and work products

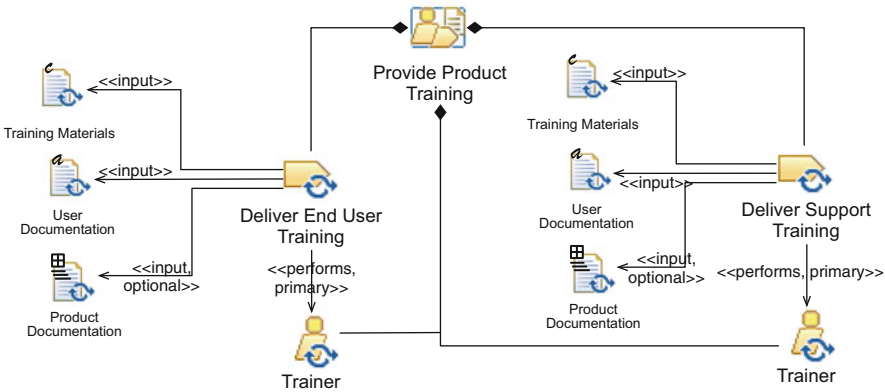


Fig. 49 The Provide Product Training activity described in terms of tasks, roles, and work products

3. Implement Developer Tests
4. Implement Solution
5. Install and Validate Infrastructure
6. Integrate and Create Build
7. Package the Release
8. Review and Conform to Release Controls
9. Run Developer Tests

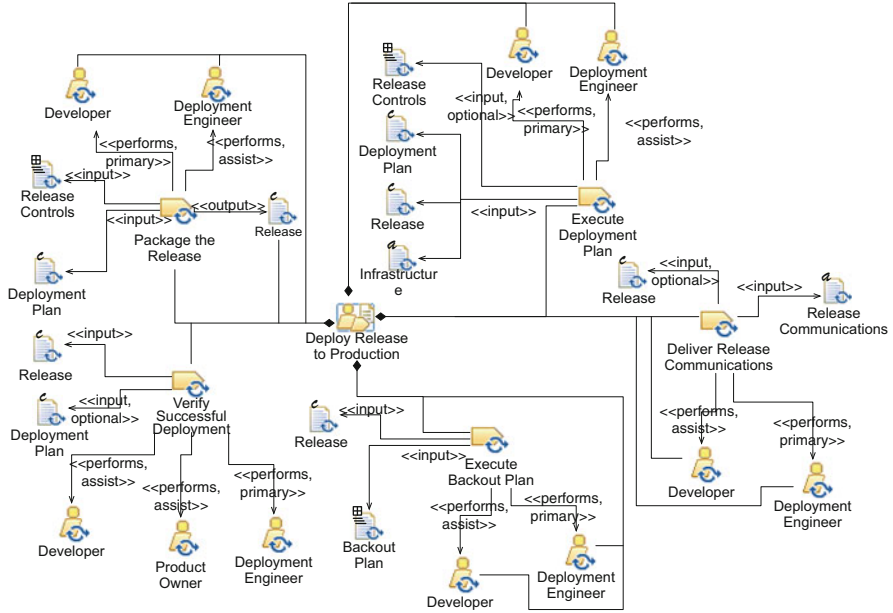


Fig. 50 The Deploy Release to Production activity described in terms of tasks, roles, and work products

She/he assists in the following tasks:

1. Assess Results
2. Deliver Release Communications
3. Develop Product Documentation
4. Execute Backout Plan (if necessary)
5. Execute Deployment Plan
6. Implement Tests
7. Manage Iteration
8. Outline Deployment Plan
9. Plan Iteration
10. Verify Successful Deployment

Process Engineer

She/he is responsible for the following tasks:

1. Deploy the Process
2. Tailor the Process

Product Owner

She/he assists in the following tasks:

1. Develop Product Documentation
2. Verify Successful Deployment

Project Manager

She/he is responsible for the following tasks:

1. Assess Results
2. Manage Iteration
3. Plan Iteration

Stakeholder

She/he assists in the following tasks:

1. Assess Results
2. Design the Solution
3. Implement Solution
4. Implement Tests
5. Manage Iteration
6. Plan Iteration

Technical Writer

She/he is responsible for the following tasks:

1. Develop Product Documentation
2. Develop Support Documentation
3. Develop User Documentation

Tester

She/he is responsible for the following tasks:

1. Implement Tests
2. Run Tests

She/he assists in the following tasks:

1. Assess Results
2. Design the Solution
3. Implement Developer Tests
4. Implement Solution
5. Manage Iteration
6. Plan Iteration

Tool Specialist

She/he is responsible for the following tasks:

1. Set Up Tools
2. Verify Tool Configuration and Installation

Trainer

She/he is responsible for the following tasks:

1. Deliver End User Training
2. Deliver Support Training

2.4.2 Activity Details

The Transition phase includes eight activities, which are described in the following subsections.

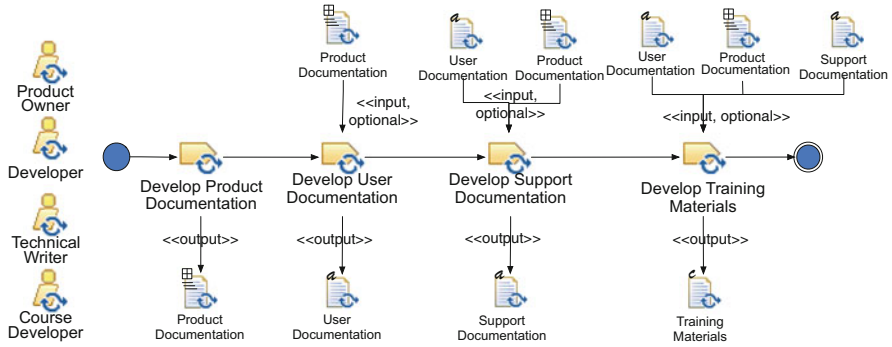


Fig. 51 The flow of tasks of the Finalize Product Documentation and Training activity

Plan and Manage Iteration

See section “Plan and Manage Iteration” above.

Develop Solution Increment

See section “Develop Solution Increment” above.

Test Solution

See section “Test Solution” above.

Finalize Product Documentation

This activity prepares product documentation and training materials. The flow of tasks inside this activity is reported in Fig. 51, and the tasks are detailed in Table 17.

Prepare for Release

This activity prepares product documentation and training materials. The flow of tasks inside this activity is reported in Fig. 52, and the tasks are detailed in Table 18.

Provide Product Training

This activity provides product training. The flow of tasks inside this activity is reported in Fig. 53, and the tasks are detailed in Table 19.

Ongoing Tasks

See section “Ongoing Tasks” above.

Deploy Release to Production

This activity results in the release of a set of integrated components into the production environment. The flow of tasks inside this activity is reported in Fig. 54, and the tasks are detailed in Table 20.

Table 17 Finalize product documentation and training—the task description

Activity	Task	Task description	Roles involved
Finalize Product Documentation	Develop Product Documentation	<p>Development team members sometimes take documentation for granted, or do not give it enough consideration. However, after a product is delivered, customers who pay for the system and for support often do not have enough information to effectively manage the product. If a technical writer is made available to a development team, that role often takes the burden off the team for developing the formal product documentation and for ensuring that it is in the correct format and business language. If a technical writer is not available, the development team and product owner must make every effort to create enough documentation to ensure that the features that have been developed for each release are understood and can be communicated effectively by the paying customer to their stakeholders. Delivering a professionally developed product requires that a development team provide the customer with accurate, detailed, and comprehensive product documentation. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Identify features of current release 2. Document each feature 3. Review product documentation with stakeholders 4. Update product documentation as necessary 5. Deliver product documentation 	Technical Writer (perform), Developer (assist), Product Owner (assist).
Finalize Product Documentation	Develop User Documentation	<p>User documentation might include all or parts of user manuals (electronic or paper-based), tutorials, frequently asked questions (FAQs), on-line Help Files, installation instructions, operational procedures, etc. User documentation often is used as the basis for training materials - if the documentation is of poor quality, the training materials might not be any better. Without good user documentation, a system might be well developed by a development team but might not meet the End User's expectations because they will not be able operate the application effectively. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Determine user documentation contents 2. Leverage product documentation 3. Leverage other materials 4. Write user documentation content 5. Perform quality review 6. Deliver user documentation 	Technical Writer (perform)
Finalize Product Documentation	Develop Support Documentation	<p>Support documentation often is the most overlooked aspect of a documentation effort. Anyone who has had the opportunity to provide end user support for a particular application can appreciate how important effective, well-written support documentation can be. This documentation very often is technical in nature and differs significantly from user or product documentation, which normally is written for the lay person. The development team should do its best to make sure that personnel who perform an IT support role have the right amount and the relevant type of information necessary to support the application, whether they provide Tier 1, Tier 2, or Tier 3 support. Support documentation often is developed based on these three different support categories. How effectively the code base</p>	Technical Writer (perform)

(continued)

Table 17 (continued)

Activity	Task	Task description	Roles involved
		<p>is commented and the ease with which those comments are found and understood contributes to the quality and quantity of support documentation. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Determine support documentation contents 2. Leverage available materials 3. Write support documentation 4. Perform quality review 5. Deliver support documentation 	
Finalize Product Documentation	Develop Training Materials	<p>Having the correct amount and type of materials available to adequately train end users and supporters of an application is necessary to promote usability and to achieve the desired business value. If a course developer is available, they can assume most of the burden of creating the training materials, lab exercises, and workshops for delivery of those courses to either end users or support personnel. If a course developer is not available, development team members should take the time to properly develop a suite of training materials for the feature set developed during a release. Although different parts of training materials should be created during feature development sprint/iterations, the bulk of the work (and the integration of the materials) usually is reserved for the release sprint/iteration that occurs immediately before a scheduled release. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Determine scope of training materials 2. Develop user training materials 3. Develop support training materials 4. Perform quality review 5. Perform dry run 6. Deliver training materials 	Course Developer (perform)

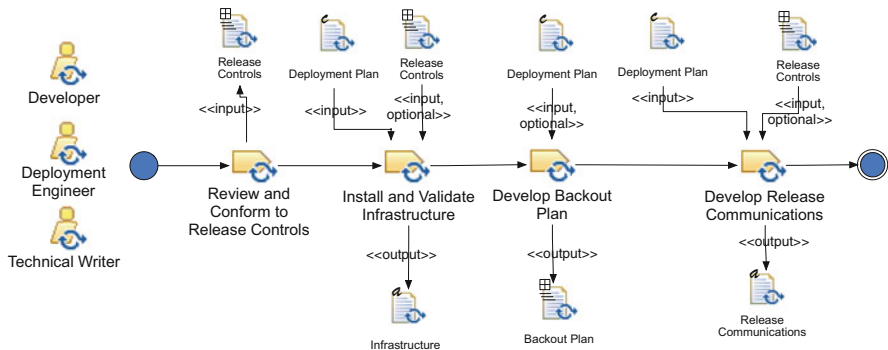


Fig. 52 The flow of tasks of the Prepare for Release activity

2.4.3 Work Products

The Transition phase generates nine work products. Their relationships with the system meta-model elements are described in Fig. 55.

Table 18 Prepare for release—the task description

Activity	Task	Task description	Roles involved
Prepare for Release	Review and Conform to Release Controls	<p>Release controls describe the minimum number of requirements that a software package must adhere to before being released into production. This is especially important if a development team is new or emerging, because they might not be aware of the great responsibilities a deployment manager has. In fact, a deployment manager is responsible to senior management for ensuring that nothing is placed into production that does not conform to the rigid controls designed to protect the IT organization’s ability to successfully deliver IT services to internal and external customers. Release controls typically consist of</p> <ul style="list-style-type: none"> • Release or deployment plan • Backout plan • Release component definitions • Release package integrity verification • References to configuration items (CIs) • Customer approval • Ready for transfer to operations and support staff <p>This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Locate release controls 2. Review release controls 3. Ensure the team release conforms to the controls 	<p>Technical Writer (perform), Developer (perform)</p>
Prepare for Release	Install and Validate Infrastructure	<p>A release package cannot be deployed to production if the environmental infrastructure within which the release will be run is not sufficiently built or tested. Whether the release is deployed as a “push” (where the application is deployed from a central point and proactively delivered to target locations) or a “pull” (where the application is made available at central point and pulled by a user at a time of their choosing), the infrastructure needed to support the application must be considered and implemented. Some key aspects of installing and/or validating the desired infrastructure:</p> <ul style="list-style-type: none"> • Identify the requirements and components of the environment configuration • Determine the lead times required to establish the infrastructure environments • Procure and install the infrastructure components that are not yet available • Test the newly installed infrastructure components • Test the integration of newly installed components with the rest of the environmental configuration • Validate other aspects of the infrastructure including <ul style="list-style-type: none"> – Security components and their integration – Database connectivity and security – License management, as appropriate – Configuration management, in terms of configuration items (CIs) <p>This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Identify infrastructure needs 2. Procure components 3. Schedule components for installation 4. Install and test components 5. Validate other component aspects 	<p>Deployment Engineer, Developer (perform)</p>

(continued)

Table 18 (continued)

Activity	Task	Task description	Roles involved
Prepare for Release	Develop Backout Plan	A rollback might be needed for a variety of reasons, including corruption of the production code base, inoperable components, an unplanned undesirable effect of the release on other production systems, an unhappy customer, etc. The Development team should provide the production support organization with a specific plan and decision criteria made available to them to avoid or minimize service interruptions. This task includes the following steps: 1. Determine if backout plan exists 2. Develop the backout plan (if applicable) 3. Update the backout plan (if applicable)	Developer (perform), Deployment Engineer (assist)
Prepare for Release	Develop Release Communications	When a release is pushed to production, all the stakeholders of that product should be notified that the event has happened and what the release means to each of the stakeholders. Often, the output of this task does not need to be created from scratch; for products that plan multiple releases, just updating the communicate details for each release might be enough. However, if any of the stakeholder groups change, or there is a significant difference in the product distribution, more significant content might need to be developed. A development team can develop high quality software, but if messaging to the stakeholders is conducted poorly or not at all, the end user experience might be degraded. By simply answering the questions “who, what, when, where, why, and how” in a format appropriate for each stakeholder group, a product release can become a more satisfying experience for all those involved. This task includes the following steps: 1. Identify stakeholders for this release 2. Draft communique for each stakeholder group 3. Provide communiques to deployment manager	Deployment Engineer (perform)

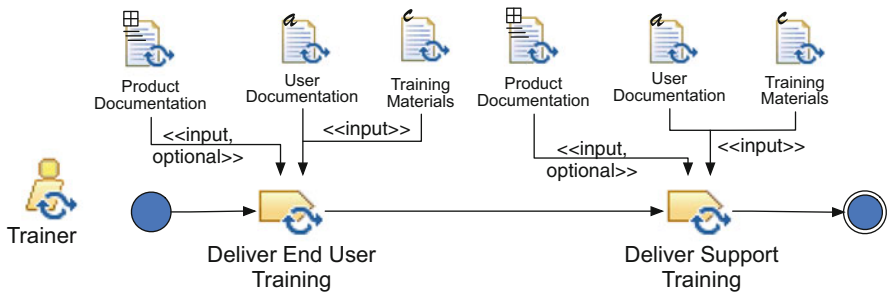


Fig. 53 The flow of tasks of the Provide Product Training activity

WorkProduct Kinds

Table 21 describes the work products of the Construction phase according to their kinds.

Table 19 Provide product training—the task description

Activity	Task	Task description	Roles involved
Provide Product Training	Deliver end user Training	Often, a trainer will deliver training to end users; rarely will the development team deliver training because they are busy developing additional features for this or other systems. If a trainer is not available, the product owner might have to train the end users. End user training usually consists of presentation slides, job aids, hands-on labs and exercises, or workshops that integrate these methods into an environment that the end users can understand and relate to. This task includes the following steps: <ol style="list-style-type: none"> 1. Validate user training logistics 2. Prepare for user training delivery 3. Deliver user training and gather feedback 4. Provide feedback to the program level 	Technical Writer (perform)
Provide Product Training	Deliver Support Training	Because a release ultimately will have to be supported by Help Desk or other technical personnel, a set of training materials designed to convey key concepts to those individuals must be developed and delivered by the development team or by a technically oriented trainer. Delivery of this training might include presentation slides, job aids, lab exercises, or workshops designed to provide real-world scenarios to accelerate learning and retention. In addition, the training materials might be different for each level (tier) of support. This task includes the following steps: <ol style="list-style-type: none"> 1. Validate support training logistics 2. Prepare for support training delivery 3. deliver support training and gather feedback 4. Provide feedback to the program 	Trainer (perform)

Backout Plan

The purpose of this work product is for the development team to provide, in one document, all the information needed by the production support organization to determine if a rollback is needed, who will authorize it, how it will be performed, etc. While someone on the development team normally authors a draft version of the Backout Plan, the Deployment Engineer is ultimately responsible for its contents and existence. A backout plan typically answers the following questions:

- Under what circumstances will a rollback be required? Or conversely, under what circumstances will the deployment be considered a success?
- What is the time period within which a rollback can take place?
- Which authorizing agent will make the decision to revert?
- Who will perform the rollback and how soon after the decision has been made will the rollback be performed?
- What procedures (manual and automated) will be followed to execute the rollback?
- What other contingency measures or available workarounds should be considered?

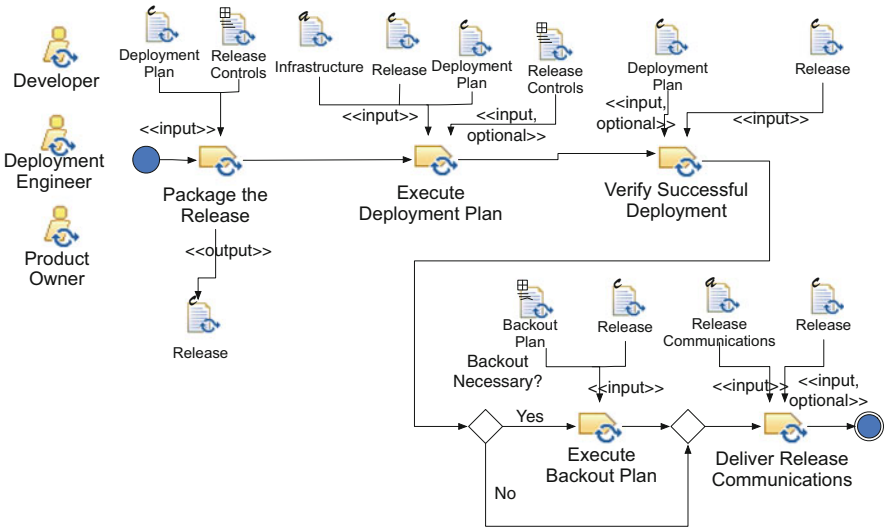


Fig. 54 The flow of tasks of the Deploy Release to Production activity

- What is the expected time required to perform a reversion?
- What are the communication procedures required in the event of a backout?
- Has the Backout Plan been successfully tested?

Infrastructure

Infrastructure normally is defined as anything that supports the flow and processing of information in an organization. The infrastructure needed to support a release package normally includes

- Software, including
 - Operating systems and applications for servers and clients
 - Desktop applications
 - Middleware
 - Protocols
- Hardware
- Networks, including
 - Routers
 - Aggregators
 - Repeaters
 - Other transmission media devices that control movement of data and signals
- Facilities

Table 20 Deploy release to production—the task description

Activity	Task	Task description	Roles involved
Deploy Release to Production	Package the Release	<p>The key activities normally used to package a release:</p> <ul style="list-style-type: none"> • Assemble the components and integrate them through a normal (i.e., continuous integration) or release build script • Install the release package in one or more test environments and verify its integrity • Tag the elements of the release package in the code base to create a baseline • Package appropriate documentation to accompany the release: <ul style="list-style-type: none"> – Deployment plan – Build plan, procedures, and scripts – Backout plan – Relevant licensing information – Relevant infrastructure information – Release communiques <p>This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Assemble components 2. Test the release 3. Tag source code repository 4. Package release documentation 5. Deliver release package 	Developer (perform), Deployment Engineer (assist)
Deploy Release to Production	Execute Deployment Plan	<p>This task is straightforward: follow the procedures in the Deployment Plan for the rollout of a specific product release. If the deployment plan does not exist or it is poorly constructed, this task might be much more difficult.</p> <p>The main point here is that to achieve a high probability of success, the development team should have previously developed a detailed plan that organizes and articulates all the unique instructions for deploying that particular release. Because an experienced deployment engineer normally executes this task, they might be able to overcome any missing deployment procedures or content. However, that is not an excuse for a development team to not develop the plan's contents. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Review deployment plan 2. Release code 	Deployment Engineer (perform), Developer (assist)
Deploy Release to Production	Verify Successful Deployment	<p>Using the success criteria documented either in the deployment plan or in the backout plan, the deployment engineer, in collaboration with the development team, will determine whether the rollout can be declared a success or not. If the deployment is successful, the previously prepared release communiques should be delivered. If the deployment is unsuccessful, then the backout plan should be invoked. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Test production release 2. Run manual tests 3. Determine if release should be reversed 	Deployment Engineer (perform), Developer, Product Owner (assist)
Deploy Release to Production	Execute Backout Plan (if necessary)	<p>Assuming a backout plan is available for this release, the deployment engineer (or development team) will follow the instructions for reversing the installation of the product into production, if there is a problem. While the plan might have been written with good intentions, sometimes key procedures are</p>	Deployment Engineer (perform), Developer (assist)

(continued)

Table 20 (continued)

Activity	Task	Task description	Roles involved
		<p>missing or have not been thought out. The team backing out the release should be aware that blindly following the backout plan might not be the best approach. It is best to consider the unique circumstances within which the deployment has failed and rely on common sense and experience when executing the backout plan. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Identify release problem(s) 2. Backout the release 3. Determine if the backout was successful 4. Communicate the backout 	
Deploy Release to Production	Deliver Release Communications	<p>This task represents the distribution of communiques that were prepared beforehand as part of the release communications artifact. Although the development team is responsible for preparing the communications, the responsibility for sending the communiques normally is assigned to the deployment engineer, if that is the organizational protocol. This task includes the following steps:</p> <ol style="list-style-type: none"> 1. Validate the communiques 2. Send release communications 3. Validate communications were received 	Deployment Engineer (perform), Developer (assist)

Release Communications

This artifact should specify the different groups to which communications are directed, the method of communication (e.g., email, text or pager message, bulletin, newsletter, phone message, etc.). All communiques should be prepared in advance so that it is a matter of disseminating information when the release to production has been determined to be successful.

Also included in this artifact is a listing of the responsible parties who will execute the communications when a successful release has been declared (normally the Deployment Engineer), as well as the timing and dependencies of the communiques.

While there is no prescribed format for the release communications artifact, each communique should indicate the preferred delivery mechanisms (e.g., beeper notification, telephone calls, a posting to an internal release website, live or pre-recorded presentations by senior management, etc.) and generally answer the following questions:

- Who are the parties (stakeholders) that are interested in knowing that a release to production has taken place?
- What specifically (features, functions, components) has been placed into production?

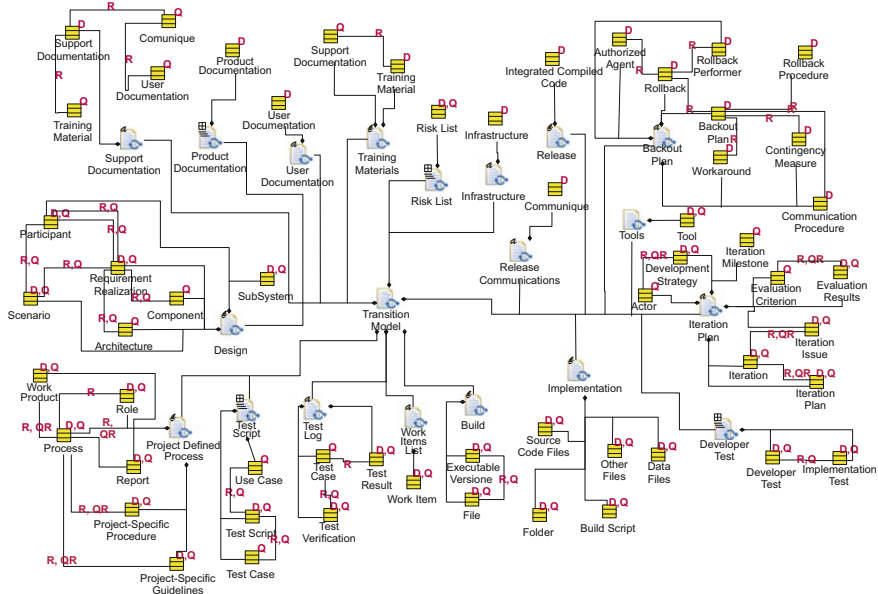


Fig. 55 The Transition phase documents structure

- Why is this release valuable to stakeholders and what business purpose does it serve?
- Where is the product available (on which platforms, geographical locations, business units, etc.)?
- How can the stakeholders access the system and under what circumstances?
- When was the product released (or when will it be released if the release date is in the future)?

Release Controls

Some common release controls are:

- A release or deployment plan must be documented and reviewed with the Deployment Manager (or the release organization). This plan must address how risks, issues, or code deviations are to be handled during the key timeframe leading up to the actual release.
- The components of each release package must be defined, integrated, and compatible with one another.
- The integrity of each release package must be verified and maintained.
- References to the configuration items (CIs) that the release package represents, if applicable.

Table 21 Transition phase—work product kinds

Name	Description	Work product kind
Backout Plan	A backout plan defines the criteria and procedures to be followed if a release into production needs to be rolled back.	Structured
Deployment Plan	See sections “Deployment Plan” and “WorkProduct Kinds.”	Composite
Infrastructure	In reference to a release sprint, infrastructure refers to all the hardware, software, and network facilities necessary to support a deployed release. The purpose of this work product is to provide the underlying capabilities within which an application can be run as designed.	Free Text
Product Documentation	See sections “Product Documentation” and “WorkProduct Kinds.”	
Release Communications	When a release is pushed to production, all the stakeholders of that product should be notified that the event has happened and what the release means to each of the stakeholders. Often, the output of this task does not need to be created from scratch; for products that plan multiple releases, just updating the communicate details for each release might be enough. However, if any of the stakeholder groups change, or there is a significant difference in the product distribution, more significant content might need to be developed. In any case, communicating effectively to the end user community is important. A development team can develop high quality software, but if messaging to the stakeholders is conducted poorly or not at all, the end user experience might be degraded. By simply answering the questions “who, what, when, where, why, and how” in a format appropriate for each stakeholder group, a product release can become a more satisfying experience for all those involved.	Free Text
Release Controls	The purpose of this work product is to identify all the requirements to which a release package must conform to be considered “deployable.”	Structured
Support Documentation	See sections “Support Documentation” and “WorkProduct Kinds.”	
Training Materials	See sections “Training Materials” and “WorkProduct Kinds.”	
User Documentation	See sections “User Documentation” and “WorkProduct Kinds.”	

- The customer for which the application is being developed must approve the release, indicating that the user community (or a specific subset) is ready to receive and use the requisite capabilities of the release.
- Each release package must be capable of being backed out of production without negatively impacting the remaining production environment.
- The contents of each release package must be transferred to operations and support staff with sufficient documentation and knowledge transfer so that those organizations can effectively support the released capabilities in production.

3 Work Product Dependencies

Figure 56 describes the dependencies among the different work products and work product slots (see Sect. 1). The list of work products fulfilling the different work product slots is reported in Table 22.

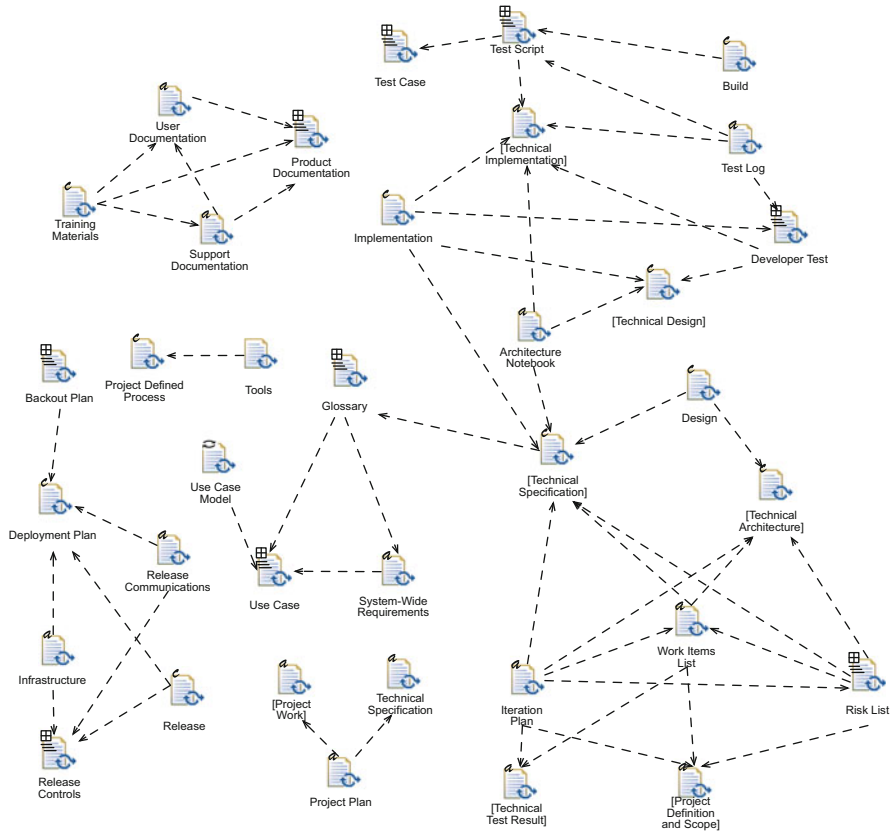


Fig. 56 The Work Product Dependency diagram

Table 22 OpenUp work product slots and the fulfilling work products

Work product slot	Fulfilling work product
[Project Definition and Scope]	Project Plan
[Project Work]	Iteration Plan, Work Items List
[Technical Architecture]	Architecture Notebook
[Technical Design]	Design
[Technical Implementation]	Build, Implementation
[Technical Specification]	Glossary, System-Wide Requirements, Use Case, Use-Case Model, Vision
[Technical Test Results]	Test Log

References

1. Eclipse Foundation: The OpenUP Website. Online at: <http://epf.eclipse.org/wikis/openup/>. Accessed on 15 Jan 2013
2. Eclipse Foundation: Work Product Slot. Online at: http://epf.eclipse.org/wikis/mam/core.mdev.common.base/guidances/concepts/work_product_slot_D5B44CE7.html. Accessed on 15 Jan 2013

3. Object Management Group (OMG): Unified Modeling Language (UML), V2.4.1, Infrastructure Specification. Doc. number: formal/2011-08-05 (2011)
4. Object Management Group (OMG): Unified Modeling Language (UML), V2.4.1, Superstructure Specification. Doc. number: formal/2011-08-06 (2011)
5. Kroll, P., Kruchten, P.: The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison-Wesley Professional, Boston (2003)
6. Software Process Engineering Metamodel. Version 2.0. Final Adopted Specification ptc/07 03-03. Object Management Group (OMG) (March 2007)
7. Seidita, V., Cossentino, M., Gaglio, S.: Using and extending the SPEM specifications to represent agent oriented methodologies. In: AOSE, pp. 46–59 (2008)

Index

A

Action model, 276, 282
Action modeler, 275
Actor(s), 260, 463, 466
Adaptive multi-agent system (AMAS), 20
ADELFE, 3, 20
Agency domain, 11
Agent(s), 45, 258, 293, 298, 302–303, 334, 364–368
Agent class model, 261, 278
Agent class modeler, 275
Agent interaction protocol (AIP), 293
Agent-oriented methodology, 115
Agent-Oriented Software Engineering (AOSE), 2
Agile, 491
AOSE. *See* Agent-Oriented Software Engineering (AOSE)
Architecture design, 274–276
Artifact(s), 181, 333, 407
ASPECS, 3, 65
aT3 process editor (APE), 254

B

BDI, 466
Business model, 33

C

Capability(ies), 258, 466
Capability model, 281
Communication, 293, 308
Communication acts, 52–53
Construction, 492, 493
Construction phase, 391
Contract template, 333, 334, 336, 361–362
Cooperative agent, 26
Cooperative behaviour, 24, 28, 54–55
Coordination, 407

D

Design phase, 261
Design process, 10
Design process documentation template (DPDT), 8–10, 13, 15
Distilled state chart, 119
Domain model, 260, 261, 266, 269–271
Domain modeler, 264
Dynamical dimension, 179

E

Early requirements analysis, 464
Elaboration, 382, 492, 493
ELDAMeth, 3, 115
ELDATool, 133
Entity, 26, 53
Environment, 143, 181, 407
 conditions, 186
 dimension, 181
Event-driven lightweight distilled state charts agent, 115

F

FIPA, 8
 design process documentation and fragmentation, 9
 protocol, 314
 standard specification, 2
Functional dimension, 177
Functional objectives, 178

G

Gaia, 3
GMoDS model. *See* Goal Model for Dynamic Systems (GMoDS) model
Goal, 466
Goal model(s), 261, 265–269, 466

Goal modeler, 264
 Goal Model for Dynamic Systems (GMoDS)
 model, 261, 265, 266, 267
 GORMAS, 3

H

Hierarchical complex systems, 66
 Holons, 66

I

IEEE-FIPA design process documentation
 and fragmentation (FIPA-DPDF), 2
 IEEE FIPA SC00097B specification, 11
 IEEE-FIPA standard design process
 documentation template, 9
 Implement agent, 60
 Implementation agent, 295, 320–322
 Implementation task, 295, 320–322
 Inception, 492, 493
 Inception phase, 377
 INGENIAS, 3, 219–250
 INGENIAS agent framework (IAF), 220,
 371
 INGENIAS-Agile, 3
 INGENIAS development kit (IDK), 220, 371
 INGENIAS methodology, 371
 Interaction, 408
 Inter-agent issues, 408
 Iterations, 493
 Iterative approach, 491

J

JADE, 288
 JANUS, 69

L

Layering, 407
LivenessProperty, 145
 Low-level design, 274, 276

M

MAS meta-model, 118
 Metamodel, 22–29, 291–296, 374
 Metamodeling, 11
 Method base, 8
 Method construction guidelines, 254
 Method engineering, 8
 Method fragment, 8
 Micro-increments, 492

Mission analysis, 175
 Model driven development (MDD), 220, 372
 Model driven engineering (MDE), 11
 Multi-agent system, 116

N

Nominal behaviour, 24, 28, 53–54
 Non-cooperative situations, 27
 Norm(s), 182, 334, 336
 Normative context, 331, 355–362

O

O-MaSE, 3
 O-MaSE metamodel, 258, 277
 Ontology, 293, 307–308, 311–312
 Open multiagent systems, 331
 OpenUp, 3
 Organization(s), 141, 333, 347–355
 Organizational abstractions, 141
 Organizational agents (OAs), 260
 Organizational design, 175
 Organizational dimensions, 204
 Organizational mission, 186
 Organizational model, 261, 271
 Organizational norms, 356–358
 Organizational patterns, 156
 Organizational rules, 141
 Organizational structure, 141
 Organizational unit (OU), 176, 334
 Organizational Unit Technology, 192
 Organization-Centered Multi-agent Systems
 (OCMAS), 173
 Organization dynamics design, 175
 Organization model, 265, 266
 Organization modeler, 264
 Organization Model for Adaptive
 Computational Systems (OMACS),
 258
 Organization Technology, 191–192

P

PASSI. *See* Process for Agent Societies
 Specification and Implementation
 (PASSI)
 Plan, 466
 Plan model, 261, 276, 282
 Plan modeler, 275
 Policies, 260
 Policy model, 261, 276, 279–281
 Policy modeler, 275
 Preparation phase, 222, 225

Problem analysis, 264–265
Problem domain, 11
Process documentation template, 10
Process for Agent Societies Specification
and Implementation (PASSI), 4,
287–329
Products, 177
Protocol(s), 143, 260, 309, 314
Protocol model(s), 261, 276, 278–279
Protocol modeler, 275

R

Regulated open multiagent systems, 331
Requirement(s), 31–32, 297, 300–302
Requirement analysis, 261, 263–274
Requirement engineer, 264
Requirement gathering, 264
Reward system, 215
Role(s), 176, 258, 293, 299, 303–304,
309–310, 313, 334, 349
Role model, 261, 266, 271–274
Role modeler, 264
ROMAS, 4
RUP, 29

S

SafetyProperty, 145
Scrum, 220, 221
Self-organizing multi-agent systems, 20
Service(s), 145, 178, 334, 339, 341, 355, 362,
363
Service analysis, 175
Simulation, 116
Situational method engineering (SME), 8
Social contracts, 356, 358–360
Social relationships, 177, 333
SODA, 4
Software process engineering metamodel 2.0
(SPEM 2.0), 9, 12, 289
Solution analysis, 265–266
Solution domain, 11
SpeADL, 58, 59

SPEM 2.0. *See* Software process engineering
metamodel 2.0 (SPEM 2.0)
Sprint phases, 222, 232
Stakeholders, 178
Structural dimension, 176
Sub-organizations, 145
System as it is, 464
System description specification, 266
System-to-be, 464

T

Tables, 409
Task, 293, 299, 304–305
Topology, 145
Transition, 492, 493
Tropos, 4, 463

U

UML, 290, 300–304, 314, 328
UML2, 29
Unified development process (UDP), 372
Unified process, 491
Use cases, 38, 296

V

Virtual organization, 174
Virtual organization model (VOM), 175

W

Work flow and technological interdependence,
192
Work product(s)
content diagram, 13–14
dependencies, 16
kind, 12
Workspaces, 181

Z

Zoom, 414