Alexander Artikis
Gauthier Picard
Laurent Vercouter (Eds.)

LNAI 5485

Engineering Societies in the Agents World IX

9th International Workshop, ESAW 2008
Saint-Etienne, France, September 2008
Revised Selected Papers

Lecture Notes in Artificial Intelligence 5485

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Alexander Artikis Gauthier Picard
Laurent Vercouter (Eds.)

Engineering Societies in the Agents World IX

9th International Workshop, ESAW 2008
Saint-Etienne, France, September 24-26, 2008
Revised Selected Papers

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Alexander Artikis
National Centre for Scientific Research "Demokritos"
Institute of Informatics & Telecommunications
Software & Knowledge Engineering Laboratory
15310 Athens, Greece
E-mail: a.artikis@iit.demokritos.gr

Gauthier Picard
ENS Mines Saint-Etienne, Multi-Agent Systems Department
158 Cours Fauriel, 42023 Saint-Etienne Cedex 02, France
E-mail: picard@emse.fr

Laurent Vercouter
ENS Mines Saint-Etienne, Multi-Agent Systems Department
158 Cours Fauriel, 42023 Saint-Etienne Cedex 02, France
E-mail: vercouter@emse.fr

Library of Congress Control Number: Applied for

CR Subject Classification (1998): I.2.11, I.2, D.2, K.4, D.1.3, H.3.4

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-642-02561-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02561-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12699469 06/3180 5 4 3 2 1 0

Preface

The ninth annual international workshop Engineering Societies in the Agents' World was hosted by the École Nationale Supérieure des Mines de Saint-Étienne (ENSM-SE), France, in September 2008. The workshop was organized as a stand-alone event, running over three days. ESAW 2008 built upon the success of prior ESAW workshops: ESAW 2007 held in Athens, ESAW 2006 held in Dublin, and ESAW 2005 held in Kuşadasi, going back to first edition of the workshop which was held in Berlin in 2000. ESAW 2007 was attended by 60 participants from 12 different countries. Each presentation was followed by highly interactive discussions, in line with the ESAW spirit of having open discussions with fellow experts.

The ESAW workshop series started in 2000 to provide a forum for presenting highly interdisciplinary work on technologies, methodologies, platforms and tools for the engineering of complex artificial agent societies. Such systems have found applications in many diverse domains such as complex system engineering, P2P, e-business and ambient intelligence. Despite ESAW traditionally placing emphasis on practical engineering issues and applications, the workshop did not exclude theoretical and philosophical contributions, provided that they clearly documented their connection to the core applied issues.

Discussions coalesced around the following themes:

- Electronic institutions
- Models of complex distributed systems with agents and societies
- Interaction in agent societies
- Self-organization and emergence in agent societies
- Engineering social intelligence in multi-agent systems
- Trust and reputation in agent societies
- Collective and coordinated mutli-agent problem solving
- Analysis, design and development of agent societies

Two invited presentations underlined both the engineering aspects and the interdisciplinary nature of research on agent societies. The first invited talk was given by Klaus Fischer, from the German Reasearch Center for Artificial Intelligence (DFKI). In his talk, the contents of which appear in this volume as an invited submission, Dr. Fischer presented a model-driven approach and metamodel, PIM4Agents, for designing multi-agent and service-oriented applications for industrial purpose, within the Teletruck, Saarstahl and ATHENA projects. Dr. Fischer also made a demonstration of the used modeling tool during the demonstration session.

The second invited talk was given by Paul Bourgine, Vice Director of the CREA (Centre de Recherche en Epistémologie Appliquée) and president of the European Complex Systems Society. Dr. Bourgine presented several works from European projects (such as ONCE-CS and PATRES) on distributed problem solving in natural and artificial complex systems, and more precisely on the use of distributed and self-organizing

approaches to simulate and understand biological phenomena such as ontogenesis in computational biology.

We received 29 submissions for the workshop, and each paper was reviewed by at least three independant reviewers. During the workshop, 19 papers were presented. After the second reviewing phase, we accepted 13 extended papers and one invited long paper.

This edition also included an original demonstration session, organized by Jomi F. Hübner, during which nine agent-based technologies and works were shown. The session demonstrated concrete applications of agent societies to automatic B2B exchange or agent-based simulation, and toolkits for multi-agent engineering, such as LEIA, PRESAGE, POLAR, OperettA and CARTAGO.

The original contributions, the slides of the presentations and demonstrations, and more information about the workshop are available on the ESAW 2008 website (http://www.emse.fr/esaw08/). The present proceedings continue the series published by Springer (ESAW 2000: LNAI 1972, ESAW 2001: LNAI 2203, ESAW 2002: LNAI 2577, ESAW 2003: LNAI 3071, ESAW 2004: LNAI 3451, ESAW 2005: LNAI 3963, ESAW 2006: LNAI 4457, ESAW 2007: LNAI 4995). This volume contains extended and substantially revised versions of selected papers from ESAW 2008, and an invited contribution by Klaus Fischer.

The organization of ESAW 2008 would have not been possible without the financial help of:

- École Nationale Supérieure des Mines de Saint-Étienne
- General Council of Loire in Rhône-Alpes
- Saint-Étienne Metropole
- GEM (Groupe des Écoles des Mines) network
- Rhône-Alpes Region
- Loire Numérique cluster
- Upetec company

We would like to thank the Steering Committee for their guidance, the Program Committee and the additional reviewers for the insightful reviews, and the Local Organizing Committee for arranging an enjoyable event. We would also like to thank all the researchers who submitted a paper to the workshop. Finally, we would like to offer our thanks to Alfred Hoffman and the Springer crew for helping us realize these proceedings.

The next ESAW workshop will be hosted in The Netherlands by Utrecht University, in November 2009, with Huib Aldewereld, Virginia Dignum, and Gauthier Picard as organizers. We look forward to even more lively interactions, and a still higher level of originality and innovation.

February 2009

Alexander Artikis
Gauthier Picard
Laurent Vercouter

Organization

Organizers

Alexander Artikis	NCSR "Demokritos", Greece
Gauthier Picard	ENSM Saint-Étienne, France
Laurent Vercouter	ENSM Saint-Étienne, France

Steering Committee

Marie-Pierre Gleizes	IRIT, Université Paul Sabatier, France
Andrea Omicini	DEIS Università di Bologna, Italy
Paolo Petta	Austrian Research Institute for AI, Austria
Jeremy Pitt	Imperial College London, UK
Robert Tolksdorf	Free University of Berlin, Germany
Franco Zambonelli	Universita di Modena e Reggio Emilia, Italy

Local Organizing Committee

Philippe Beaune	ENSM Saint-Étienne, France
Yazid Benazzouz	ENSM Saint-Étienne, France
Gregoire Berthézene	ENSM Saint-Étienne, France
Noury Bouraqadi	École des Mines de Douai, France
Ségolène Courant	ENSM Saint-Étienne, France
Rosine Kitio	ENSM Saint-Étienne, France
Christelle Urtado	École des Mines d'Alès, France

Program Committee

Federico Bergenti	Università di Parma, Italy
Michael Berger	Siemens, Germany
Carole Bernon	IRIT Université Paul Sabatier, France
Holger Billhardt	Universidad Rey Juan Carlos, Spain
Guido Boella	Università degli Studi di Torino, Italy
Olivier Boissier	ENSM Saint-Étienne, France
Jeff Bradshaw	IHMC, USA
Monique Calisti	Whitestein Technologies, Switzerland
Davy Capera	UPETEC, France
Cristiano Castelfranchi	ISTC-CNR, Italy
Luca Cernuzzi	DEI, Universidad Católica, Paraguay
Rem Collier	University College Dublin, Ireland
Dan Corkill	University of Massachusetts at Amherst, USA

Mehdi Dastani	Utrecht University, The Netherlands
Paul Davidsson	Blekinge Institute of Technology, Sweden
Keith Decker	University of Delaware, USA
Oguz Dikenelli	Ege University, Turkey
Rino Falcone	ISTC-CNR, Italy
Paul Feltovich	IHMC, USA
Paolo Giorgini	University of Trento, Italy
Frank Guerin	University of Aberdeeen, UK
Salima Hassas	Université Claude Bernard Lyon 1, France
Mark Jelasity	University of Szeged, Hungary
Lloyd Kamara	Imperial College London, UK
Anthony Karageorgos	TEI of Larisa, Greece
René Mandiau	Université de Valenciennes, France
Fabien Michel	Université de Reims, France
Frédéric Migeon	IRIT Université Paul Sabatier, France
Simon Miles	King's College London, UK
Tim Miller	University of Melbourne, Australia
Pablo Noriega	IIIA, Spain
Julian Padget	University of Bath, UK
Juan Pavon Mestras	Universidad Complutense de Madrid, Spain
Alessandro Ricci	Università di Bologna, Italy
Juan Antonio Rodríguez-Aguilar	IIIA, Spain
Jaime Simão Sichman	University of São Paulo, Brazil
Leon Van der Torre	University of Luxembourg, Luxembourg
Wamberto Vasconcelos	University of Aberdeeen, UK
Mirko Viroli	Università di Bologna, Italy
Danny Weyns	Katholieke Universiteit Leuven, Belgium
David Wolpert	NASA, USA
Pinar Yolum	Bogazici University, Turkey

Additional Reviewers

Jan Broersen	Utrecht University, The Netherlands
Frédéric Armetta	Université Claude Bernard Lyon 1, France
Jomi F. Hübner	ENSM Saint-Étienne, France
Thomas Hubauer	Hamburg University of Technology, Germany

Table of Contents

IV Emergence and Self-organisation

V Simulation

Modeling TeleTruck: A Case Study

Klaus Fischer, Christian Hahn, and Stefan Warwas

German Research Center for Artificial Intelligence (DFKI) GmbH, Saarbrücken, Germany

Abstract. This article presents a use case for a modeling framework for multiagent systems (MAS) called Dsml4MAS. The modeling framework uses a metamodel (Pim4Agents) to describe the modeling artefacts and the relations among them. From this metamodel a tool chain is deduced that supports the graphical modeling of a MAS according to the specifications in the metamodel. The TeleTruck system, a system that was developed to support shipping companies to do online fleet management, is introduced as a use case. For the TeleTruck core functions concrete models are presented as an illustration of the abstract metamodel.

1 Motivation

Recently, associated with the increasing acceptance of agent-based computing as a novel software engineering paradigm, a lot of research addresses the identification and definition of suitable models and techniques to support the development of complex software systems with respect to agent-based computing. Agent-Oriented Software Engineering (AOSE) is a relatively young field - with its first workshop held in 2000 - that is concerned with how to engineer agent-based software systems.

The current state-of-the-art in developing multiagent systems (MASs) is to design the agent systems basing on an AOSE methodology and take the resulting design artifact as a base to manually code the agent system using agent-oriented programming languages (AOPLs). The gap between design and code may tend to the divergence of design and implementation which makes again the design less useful for further work in maintenance and comprehension of the system [1]. Furthermore, even if existing methodologies and modeling frameworks have different advantages when applied to particular problems, it seems to be widely accepted that a unique approach cannot be applied to every problem without some (minor) level of customization.

The main objective of this article is to present a use case for a modeling framework for MASs. A metamodel forms the core of this methodology. The metamodel describes individual concepts that can be introduced as artefacts into the design of a MAS and it defines the relations among these concepts. Once the metamodel is defined it can be used to derive a tool chain to get from usually graphical models, which specify the design of the MAS, to executable code. We use the TeleTruck system (cf. [2]) as an illustrating example to demonstrate how concrete models for a concrete MAS would look like. We chose

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 1–26, 2009.

TeleTruck because on the one hand it is a system that is capable of solving a complex scheduling problem and on the other hand TeleTruck was actually evaluated in a practical field study, which proves the practical relevance. The design of the problem solving core of the TeleTruck according to the presented modeling framework shows the usefulness of the modeling framework as well as of the core metamodel Pim4Agents.

2 The Architecture of the TeleTruck System

In this section, the application domain of the TeleTruck system is described and analyzed. Some of the properties are derived that make dealing with this domain so difficult. The application domain for the TeleTruck system is the planning and scheduling of transportation orders as performed by dispatchers in shipping companies. Many of the problems which must be solved in this area, such as the Traveling Salesman and related scheduling problems, are known to be NP-hard. The domain is highly dynamic, and decisions have to be made under a high degree of uncertainty and incompleteness.

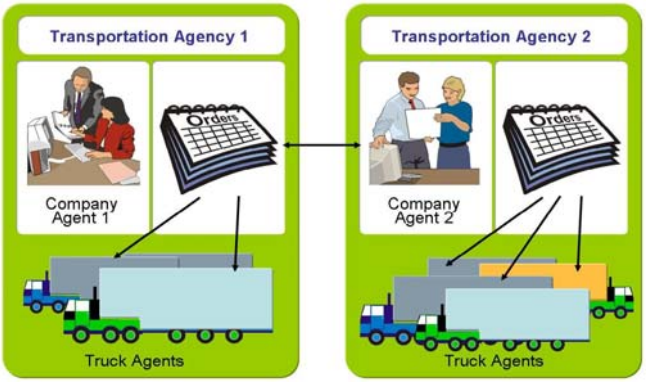


Fig. 1. TeleTruck: The domain of application

Cooperation and coordination are two very important processes that may help to overcome these problems. Using the TeleTruck system, several cooperation types such as the announcement of unbooked legs, order brokering, and different strategies for information exchange have been evaluated (see also [3]).

Corresponding to the physical entities in the domain, there are two basic types of agents in TeleTruck: *transportation companies* and *trucks*. Companies can communicate with their trucks and among each other. The user may dynamically allocate transportation orders to specific companies. Looking upon trucks as agents allows us to delegate problem-solving skills to them (such as route-planning and local plan optimization).

The *shipping company* agent (SCA) has to allocate orders to her[1] truck agents (TAs), while trying to satisfy the constraints provided by the user as well as local optimality criteria (costs). An SCA also may decide to cooperate with another company instead of having an order executed by her own trucks.

Each TA is associated with a particular shipping company from which he receives orders of the form "`Load amount` s `of good` g_1 `at location` l_1 `and transport` it to `location` l_2 `while satisfying time constraints` $\{c_{t_1}, \ldots, c_{t_n}\}$".

Interaction of the agents within one shipping company (called *vertical cooperation*) is totally cooperative. This means that a specific TA will accept deals (i.e., results of negotiation processes) with his SCA even if they are not locally profitable for him. We call such a setting an instance of a *cooperative task-oriented domain* (cf. [4]).

In the cooperation between SCAs we investigate in both a totally cooperative and a competitive setting (we call the latter setting an instance of a *competitive task-oriented domain*). If we assume a cooperative task-oriented domain, we are purely interested in the quality of the overall schedule which is emerging from the local problem solving done in the SCAs and TAs. A practical example for this setting is the cooperation among different, geographically distributed branches of one shipping company. On the other hand, in a competitive task-oriented domain among the SCAs, the overall schedule which is computed will be far from optimal. In this setting we investigate how a single SCA can maximize her profits and how she can avoid being tricked by other agents.

In the following we describe the multiagent approach underlying the TeleTruck system. The main idea of this article is to demonstrate the usefulness of the presented modeling framework for such a system. Therefore we provide a simplified description of TeleTruck. We concentrate on the situation within one shipping company and leave aside the idea of holonic structures that we introduced in [2]. The presented modeling framework would work for the more complex structures, too, but the limited space of this article does not allow to present this complete picture. We introduce the standard Contract Net Protocol (Section 2.1) and the simulated trading procedure as an iterative optimization mechanism which when they are combined are able to produce close to optimal solutions with reasonable computational effort for the complex task of online scheduling of transportation orders to a fleet of trucks.

2.1 Task Decomposition and Task Allocation

If an order o is announced to an SCA by a customer (which can also be another SCA), she has to compute a bid for executing the order. In order to determine the costs, she forwards the order to her TAs. Each TA a computes a bid

$$(a, cost(T_a \oplus o) - cost(T_a), w),$$

[1] We use 'she' to refer to shipping companies and 'he' to refer to trucks to resolve ambiguities.

where T_a is the current tour of a and w is the amount of the order a is able to transport. $cost(T_a \oplus o)$ denotes the additional costs for a when executing o given T_a. Let $\mathcal{O}^a = \{o_1^a, \ldots, o_n^a\}, n \in I\!N$ be the current set of orders for a. Constraints are derived from the information which is specified with the orders. Each solution to the resulting constraint solving problem is a valid tour which fulfills all constraints specified by $\mathcal{O}^a$. Then, a tries to find the best tour for $\mathcal{O}^a$ using a constraint solving and constraint optimization procedure.

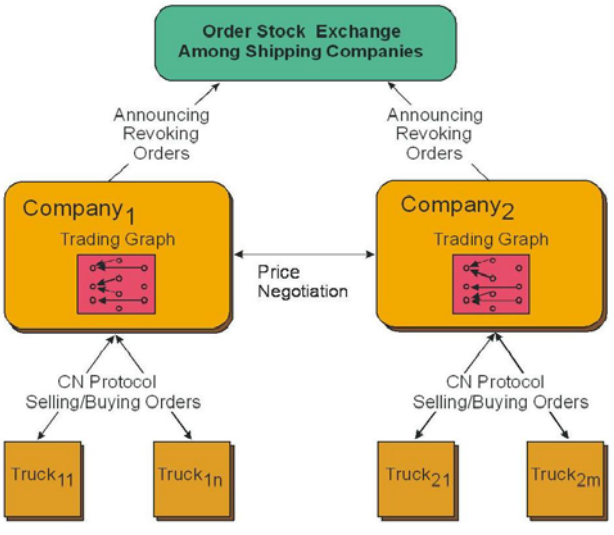


Fig. 2. Hierarchical organization of the agents in TeleTruck

For each order o announced by an SCA to her TAs, she receives a set of bids

$$\mathcal{B} = \{(a_1, c_1), \ldots, (a_n, c_n)\}, n \in I\!N$$

where c_i specifies the costs truck a_i will produce when executing order o, $1 \leq i \leq n$. The SCA selects

$$(a_{min}, c_{min}) \in \mathcal{B} \qquad \text{with} \qquad \forall (a, c) \in \mathcal{B} : c_{min} \leq c$$

and sends a grant to the TA a_{min}[2], notifying him that he will be granted the amount a_{min} provided that the SCA itself will actually receive a grant for o by the customer.

The procedure described so far is the well known Contract Net Protocol (CNP) [5]. Because the CNP provides time-out mechanisms it is easy to turn this communication protocol into an anytime algorithm (see for instance [6,7]), i.e., the system will produce a solution (at least with a high probability, only in cases where the

[2] If there is more than one truck who sent a bid with c_{min} the SCA can chose randomly among them.

constraints are so hard that only a low number of close to optimal solutions exist, the CNP might fail to produce these solutions) within a specified time t_0. The quality of the solution may be increased if more time for computation is available.

3 Simulated Trading: An Auction Mechanism for Dynamic Task Reallocation

Although the CNP presented in the last section is a very powerful and very popular task assignment mechanism, one cannot ignore the fact that the solutions the CNP provides can be quite far from optimal. The reason for this is that, although each of the individual task assignments in the CNP is a centralized optimal decision based on the current situation, a sequence of such decision is as a whole not optimal and can in some case actually turn out to be rather poor. The reason for this is that once a decision is done it is never reconsidered even when the situation changes by newly incoming orders.

To overcome this problem of the CNP we adopted the Simulated Trading (ST) [8] procedure. ST can be used for two different purposes:

- **Dynamic re-planning:** If a TA realizes that he cannot satisfy the time constraints of an order because of an unforeseen traffic jam, he can initiate an ST process leading to an order reallocation satisfying the time constraints.
- **Iterative optimization:** Starting from the initial CNP solution (see section 2.1), ST may be initiated to yield a better order allocation.

In the following, the principle of ST and its application in the TeleTruck system are explained.

3.1 Principles of ST

In [9], Bachem, Hochstättler and Malich present a parallel improvement heuristic for solving vehicle routing problems with side constraints. Their approach deals with the problem that n customers order different amounts of goods which are located at a central depot. The task of the dispatcher is to cluster the orders and to attach the different clusters to trucks which then in turn determine a tour to deliver the cluster allocated to them.

The solution to this problem is constructed using the Simulated Trading procedure. It starts with a set of feasible tours $T_1, \ldots, T_{t_0}$ which may e.g. be obtained by the CNP approach presented in Section 2.1. The tours are represented as an ordered list of customers that have to be visited. To guide the improvement of the initial solution, an additional processor, a stock manager is added to the system. The task of the stock manager is to coordinate the exchange of costumers orders between the different processors. To do this, it collects offers for buying and selling orders coming from the processors in the system.

A price system is introduced providing a quality criterion for order exchanges to the stock manager: If processor p sells an order i (i.e., an order from the depot

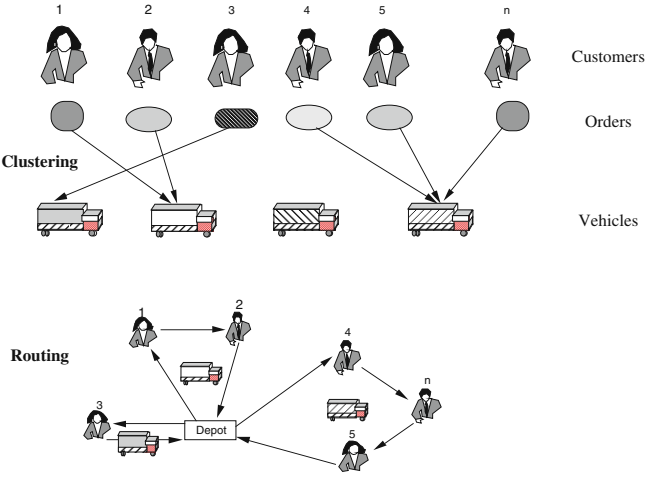


Fig. 3. The standard vehicle routing problem

to customer i), its cost should decrease. This saving of costs is associated as the price Pr to i, where

$$Pr \stackrel{\text{def}}{=} cost(T_p) - cost(T_p \ominus \{i\}) \text{ and } T_p \stackrel{\text{def}}{=} T_p \ominus \{i\}.$$

Here, the term $T_p \ominus \{i\}$ denotes the tour that evolves from T_p if customer i (or order i, respectively) is deleted from processor p's tour list. Accordingly, the price Pr for processor p buying a customer i is computed as the difference of costs for the old tour T_p and the costs for the new tour $T_p \oplus \{i\}$, which evolves from the insertion of costumer i in T_p, where

$$Pr \stackrel{\text{def}}{=} cost(T_p \oplus \{i\}) - cost(T_p) \text{ and } T_p \stackrel{\text{def}}{=} T_p \oplus \{i\}.$$

The exchange of orders is synchronized by the stock manager according to levels of exchange situations. At each level it asks each processor for a selling or buying order. Having done this, it updates a list of the offers and sends it to all tour managers. Each offer is associated with a quintuple (processor, Level, Selling or Buying, Costumer, Price).

The stock manager maintains a data structure, called *Trading Graph* whose nodes are the selling and buying offers of the processors. Furthermore, there exists an edge between vertices v_i = ($\mathrm{processor}_1$, l_i, Selling, c_i, Pr_i) and v_j = ($\mathrm{processor}_2$, l_j, Buying, c_i, Pr_j) (where $l_i < l_j$ with $l_i, l_j \in I\!N$) if $\mathrm{processor}_2$ wants to buy customer c_i from $\mathrm{processor}_1$. l_i and l_j indicate the levels of negotiation. The edge is weighted (or labeled) by the difference of the prices Pr_j - Pr_i, giving the global saving of an exchange of the order between these tours. In this graph the stock manager now looks for a so-called *trading match* i.e., a subset M of the nodes specifying admissible exchanges of orders between tours.

One problem here is, that with offering a selling of an order a processor believes that this order eventually will be bought by another processor, and it will base

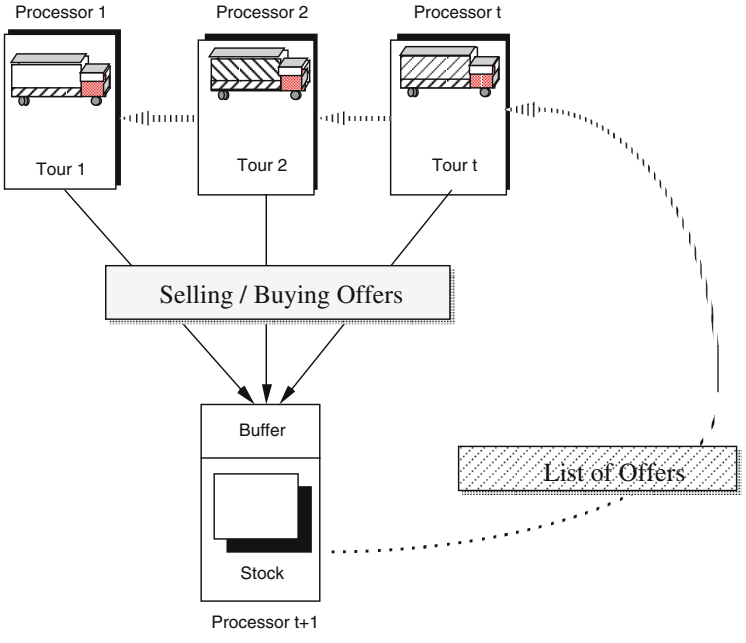


Fig. 4. Stock exchange for orders in the ST procedure

its future price calculations on its reduced tour. Thus, an admissible exchange must ensure, that with each node $v_i \in M$, all nodes of the processors selling or buying v_i and which have a smaller level than v_i have to be also in M.

The *gain* of the match is obtained by summing up the weights of the edges between nodes in M. A *trading match* is then defined to be an admissible match whose gain is positive.

3.2 Adapting ST for the TeleTruck System

The main idea is to let the SCA simulate a stock exchange where her TA can offer their current orders at some specific "saving price" and may buy orders at an "insert price". While getting sell and buy offers from her TAs the SCA maintains the trading graph and tries to find an order exchange that optimizes the global solution. A global interchange of k customers between all of the current tours of the TA corresponds to a match in the trading graph. The weight of the match is defined by the profit of this global interchange. Searching for a trading match is done by a complete enumeration of the trading graph. Though this requires exponential time in the worst case, it turned out to be feasible in practice since the trading graph normally does not have too many branches.

Important for the ST procedure are the decision criteria for the TA to decide which orders to sell or buy. This is done using heuristics like "buy nearest" and "sell farthest" combined with randomization techniques.

Note that simulated trading can only be active during a period of time when no new orders arrive at the SCA. Nevertheless, while the ST process is active the system maintains a valid solution because ST is done using a copy of the current plan of a TA and the current plan is replaced by the new one computed via the simulated trading procedure only if that was successful, i.e., a trading match was found which led to a new optimum. Thus, reactivity is ensured: when a new order arrives, the TA always uses the consistent original plan to compute a bid for the CNP. If a new order occurs while simulated trading is active, the procedure has to be aborted, unless the order fits into the TA's plan used for the ST process.

3.3 Using ST for Dynamic Re-planning

An important feature of the TeleTruck system is that TAs do not only *compute* plans: when time is up, they actually start *executing* the orders. Executing an order includes the steps of loading, driving, and unloading. Note, that even after the TA already has started the execution of his local plan, it is possible for him to participate in the CNP protocol. However, in the ST process the TA is not allowed to sell orders he has already loaded.

A problem in plan execution is that planning is done on statistical data which may be too optimistic. For instance, when the plan is actually executed the TA may get stuck in a traffic jam. Therefore, re-planning might be necessary because the TA may run into problems with respect to the time constraints which are specified for the orders. Fortunately, this situation can be nicely handled in our framework. We distinguish two cases:

Firstly, there are disturbances that can be resolved using local re-planning. In some cases, the TA can do this by selecting an alternative route to the next city where he has to deliver orders. This is done by computing the shortest path in a dynamically changing graph using Dijkstra's algorithm (this is actually what modern navigation systems do when they are equipped with TMC[3]). If this re-routing results in the violation of time constraints, the TA is forced to completely recompute his local plan using his local planning procedure. Even if the TA is able to successfully derive a new plan which satisfies all constraints, the quality of the plan may drop and thus, some orders may be sold within the next ST process. Therefore, restricted global rescheduling may occur already in this case.

Secondly, if the TA cannot fix the problem by local re-planning, the procedure depends on whether the order is already loaded on the TA or if it is not. In the latter case, the TA initiates a simulated trading process to sell the orders that he is no longer able to execute. If a trading match is found, this is a solution to the problem. If the simulated trading process does not find a valid solution for the situation, the TA has to report the problem and return the respective orders to his SCA. In this case the SCA herself can decide whether to sell the order to another SCA (see below) or to contact the customer, report the problem, and

[3] Traffic Message Channel.

try to negotiate about the violated constraints. In the worst case, the company has to pay a penalty fee.

If the orders that are causing trouble are already loaded on the TA, it is not possible to just return the order to the SCA or to sell it in a simulated trading process. In this case, the only chance for the TA is to report the problem to the SCA which then has to find a solution by contacting the client, trying to relax the constraints of the order. If a TA runs into this situation he is paralyzed in the sense that he cannot participate in the CNP or in the simulated trading process until he receives instructions from his SCA. Fortunately, the CNP and the simulated trading procedure can deal with this situation because they do not require participation of all TAs.

4 Platform Independent Language for Multiagent Systems

For designing MASs, we developed a platform-independent domain-specific modeling language for MAS called Dsml4MAS [10] in accordance to the language-driven initiative [11]. Like any language, Dsml4MAS consists of an abstract syntax, formal semantics and concrete syntax:

- The abstract syntax of Dsml4MAS is defined by a platform independent metamodel for MAS called Pim4Agents defining the concepts and their relationships. The details of Pim4Agents are discussed in more detail in Section 5.
- The formal semantics is expressed using the specification language Object-Z [12] which is a stated-based and object-oriented specification language. Object-Z is specialized on formalizing object-oriented specifications and bases on mathematical concepts (like sets, functions, and first-order predicate logic) that permits rigorous analysis and reasoning about the specifications. The denotational semantics of Dsml4MAS are defined by introducing additional variables, which are used to define the semantics and invariants in Object-Z classes. Operational semantics are specified in terms of class operations and invariants restricting the operation sequences. For a detail discussion on the semantics of Dsml4MAS, we refer to [13].
- The concrete syntax is defined as set of notations facilitating the presentation and construction of Dsml4MAS. It is specified using the Graphical Modeling Framework[4] (GMF) that provides the fundamental infrastructure and components for developing visual design and modeling surfaces in Eclipse. A detailed overview on the concrete syntax is given in [14].

Dsml4MAS is used to design MASs in a platform independent manner. However, to close the gap between design and implementation, we provide generic model transformations from Dsml4MAS on the platform independent level to two underlying execution platforms (i.e., Jack Intelligent Agents [15] or Jade [16]) on the platform specific level.

[4] http://www.eclipse.org/gmf/

5 Abstract Syntax: Platform Independent Metamodel for Agents

For describing the core building blocks of MAS in an adequate manner, we structured the core of Pim4Agents into different viewpoints briefly discussed in the remainder of this section.

- *Multiagent view* contains the core building blocks for describing MASs. In particular, the agents situated in the MAS, the roles they play within collaborations, the kinds of behaviors for acting in a reactive and proactive manner, and the sorts of interactions needed for coordinating with other agents.
- *Agent view* defines how to model single autonomous entities, the capabilities they have to solve tasks and the roles they play within the MAS. Moreover, the agent view defines to which resources an agent has access to and which kind of behaviors it can use to solve tasks.
- *Organization view* defines how single autonomous agents are arranged to more complex organizations. Organizations in Pim4Agents can be either an autonomous acting entity like an agent, or simple groups that are formed to take advantage of the synergies of its members, resulting in an entity that enables products and processes that are not possible for any single individual.
- *Role view* covers the abstract representations of functional positions of autonomous entities within an organization or other social relationships. In general, a role in Pim4Agents can be considered as set of features defined over a collection of entities participating in a particular context. The features of a role can include (but not be limited to) activities, permissions, responsibilities, and protocols. A role is a part that is played by an entity and can as such be specified in interactive contexts like collaborations.
- *Interaction view* focuses on the exchange of messages between autonomous entities. Thereby, two opportunities are offered: (i) the exchange of messages is described from the internal perspective of each entity involved, or (ii) from a global perspective in terms of agent interaction protocols focusing on the global exchange of messages between entities.
- *Behavior view* describes the vocabulary available to describe the internal behavior of intelligent entities. The vocabulary can be defined in terms of combining simple actions to more complex control structures or plans that are used for achieving predefined objectives or goals.
- *Environment view* contains any kind of Resource (i.e., Object, Ontology, Service etc.) that is situated in the environment and can be accessed and used by *Agents*, *Roles* or *Organizations* to meet their objectives.
- *Deployment view* describes the run-time agent instances involved in the system and how these are assigned to the organization's roles.

To lay the foundation for further discussions on how to use Pim4Agents for modeling the TeleTruck scenario, we focus on selected viewpoints in the remainder of this section.

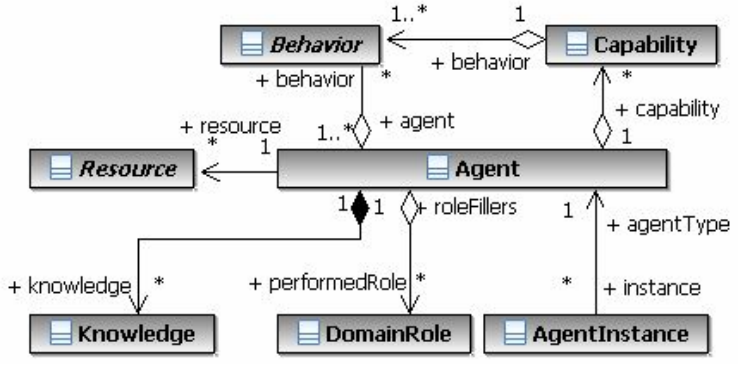


Fig. 5. The metamodel reflecting the agent aspect of Pim4Agents

5.1 Agent Aspect

The agent aspect (cf. Fig. 5) is centered on the concept of *Agent*, the autonomous entity capable of acting in the system. An *Agent* has access to a set of *Resources* which may include any kind of *Object* (e.g. Service) situated in the surrounding *Environment* that can be accessed by the *Agent*. Furthermore, the *Agent* can perform particular *DomainRoles* that define in which specific context the *Agent* is acting and *Behaviors* defining how particular tasks can be achieved by the *Agent*. These *Behaviors* may be grouped together into *Capabilities* the *Agent* may have available. Beside the *Capabilities* to achieve certain goals, the *Agent* may have additional *Knowledge* about the current state of the world which serve as input for *Behaviors* or *Capabilities*.

5.2 Organization Aspect

The organization aspect (cf. Fig. 6) describes how single autonomous entities cooperate within the MAS and how complex social structures can be defined. The *Organization* is a special kind of *Agent* (i.e., it is a recursive holonic structure) and can therefore perform *DomainRoles* and have *Capabilities*. In addition to the *Agent* properties, an *Organization* may require certain *DomainRoles* performed by its members and may have its own internal *Protocols* specifying (i) how the *Organization* communicates with other *Agents* (i.e., atomic *Agents* or complex *Organizations*) and (ii) how organizational members are coordinated. Which particular forms of collaboration inside an *Organization* exists is expressed by the concept of a *Collaboration* that defines how specific *Interactions* are used in terms of binding *Actors*—part of the *Interaction*—to *DomainRoles*—part of the *Organization*. In principle *Organizations* are statically defined structures. However, the *Organizations*/*Agents* that perform roles in an *Organization* can be dynamically assigned to the different roles defined for the *Organization*.

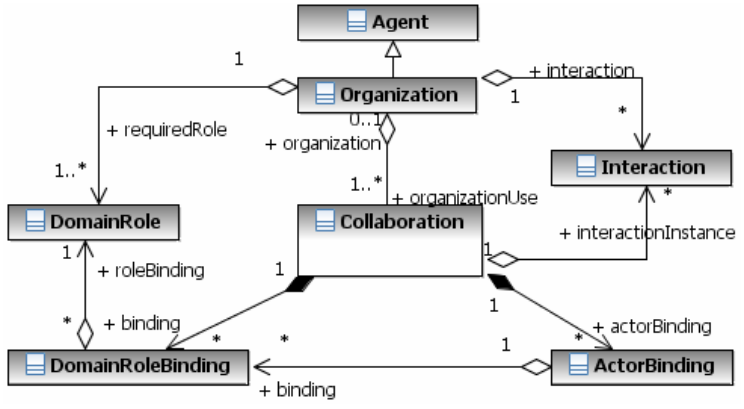


Fig. 6. The partial metamodel reflecting the organization aspect of PIM4AGENTS

5.3 Role Aspect

The role aspect (cf. Fig. 7) mainly deals with the different forms of roles and which kind of functionalities they should provide or be supported. A *Role* is an abstraction of the social behavior of the *Agent* in a given social context, usually a *Cooperation* or *Organization*. The *Role* specifies the responsibilities of the *Agent* in that social context. It refers to (i) a set of *Capabilities* that define the set of *Behaviors* it can possess and (ii) a set of *Resources* the *Role* has access to. An *Actor* can be considered as a generic concept as it either binds *AgentInstances* or *DomainRoles*. The *Actor* inherits from the *Role* and thus can have access to particular *Capabilities* and *Resources* that are necessary for exchanging messages. An *Actor* can be further partitioned in terms of a certain position within an

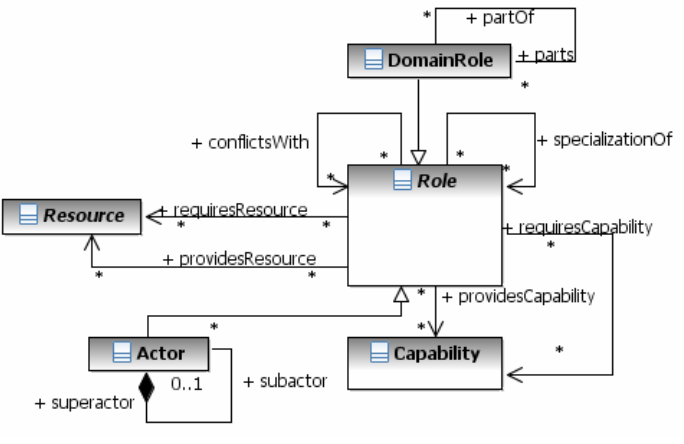


Fig. 7. The metamodel reflecting the role aspect of PIM4AGENTS

Protocol through the subactor reference. Subactors separate the agents that are grouped together in a given *Actor* into sub sets.

The reason why to distinguish between subactors will be getting more clear in Section 6.3 focusing on the graphical design of the Contract Net protocol and the Simulated Trading protocol.

5.4 Interaction Aspect

The interaction aspect of Pim4Agents (cf. Fig. 8) defines in which manner agents, organizations or roles interact. A *Protocol* is considered as a special form of an *Interaction*. Accordingly, the main concepts of a *Protocol* are *Actor*, *ACLMessage*, *MessageFlow*, *MessageScope* and *TimeOut*. In the deployment view, furthermore, the system designer can specify how *Protocols* are used within *Organizations*. This is done through the concept of a *Collaboration* that defines which organizational members (which are of the type *AgentInstance*) are bound to which kind of *Actor* as part of an *ActorBinding*. Beside a static binding, the designer may want to bind the *AgentInstances* at run-time, which can be done within a *Plan* using the *AssignRole* concept.

An interaction protocol as a pattern for conversation within a group of agents can be more easily described using generic placeholders like 'Initiator' or 'Participant' instead of describing the interaction between the particular agent instances taking part in the conversation. In Pim4Agents, this kind of interaction roles

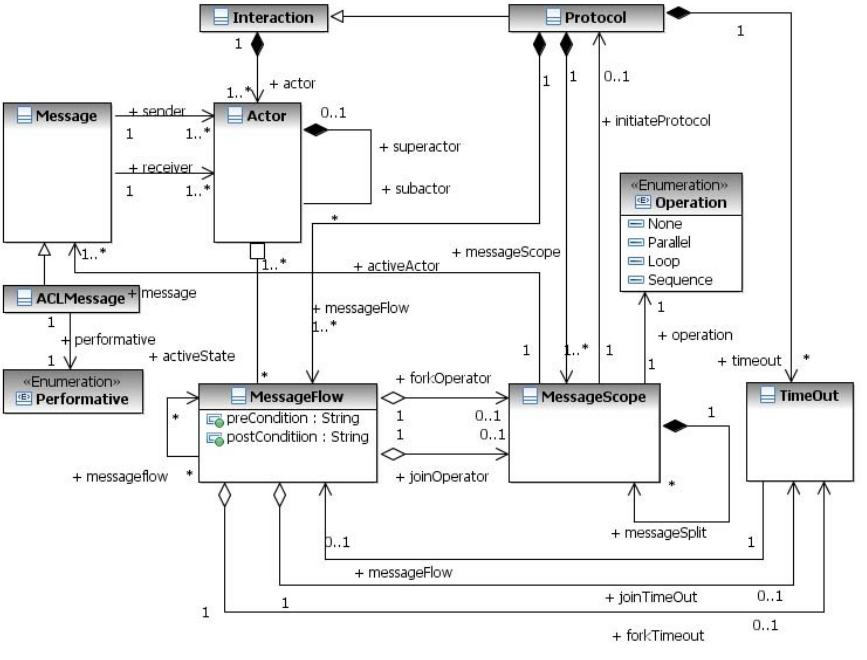


Fig. 8. The partial metamodel reflecting the interaction aspect of Pim4Agents

are called *Actors*. *Actors* represent named sets of role fillers at design time. We use *Actors* as a way to decouple *DomainRoles* of organizations from roles in interactions, meaning that the same protocol can be reused in different organizations. Actors as a specialization of *Role* can have subactors, where an *AgentInstance* bound to the parent actor must be bound to exactly one subactor. The actor subactor relationship is discussed in more detail in Section 6.3. Furthermore, *Actors* require and provide certain *Capabilities* and *Resources* defined in the role view of PIM4AGENTS.

Messages are an essential means for the communication between agents in MASs. In PIM4AGENTS, we distinguish between two sorts of messages, i.e., *Message* and *ACLMessage* which further includes the idea of *Performatives*. Messages have a content and may refer to an *Ontology* that can be used by the participating *Actors* to interpret the *Message* and its content. A *MessageFlow* defines the states of the AIP in which an *Actor* could be active. The main function of the *MessageFlow* is firstly to send and receive *Messages* which is done through the concept of a *MessageScope* and secondly to specify time constraints (i.e., the latest point in time) in which these *Messages* need to be sent and received through the *TimeOut* concept. A *TimeOut* defines the time constraints for sending and receiving messages and how to continue in the case of a *TimeOut* through the *messageFlow* reference.

A *MessageScope* defines the *Messages* and the order how these are sent and received. In particular this is achieved by connecting *Messages* to *Operations*. Beside *Messages* sent and received, a *MessageScope* may also refer to *Protocols* that are initiated at some specific point in time in the parent *Protocol*. This particular feature allows modeling of nested protocols. The order in which *Messages* are exchanged is defined by a so-called *Operation* featuring the following alternatives. A *Sequence* defines a sequencing of traces timely ordered. A *Parallel* denotes that several traces are executed concurrently and a *Loop* describes that a particular trace is executed as long as a particular *Condition* is satisfied. Finally, the *None* operation specifies that only a single *Message* is exchanged.

A combination of these *Operations* can easily be achieved by the *MessageScope's messageSplit* reference which allows to nest *Operations*. Beside *Operations*, further branching can be defined by specifying transitions between *MessageFlows* using their *messageFlow* reference. A *preCondition* and *postCondition* can be specified in order to define in which case the transition is triggered.

5.5 Behavioral Aspect

The behavioral aspect describes how plans are composed by complex control structures and simple atomic tasks and how information flows between those constructs. The core concepts of the behavioral aspect are depicted in Fig. 9.

A *Plan* can be considered as a specialization of the abstract *Behavior* to specify an agent's internal processes. An *Agent* can use several *Plans* which contain a set of *Activities* and *Flows* (i.e., *ControlFlow*, *InformationFlow*) that link *Activities* to each other. Furthermore, a *Plan* refers to a set of *MessageFlows* to ensure that

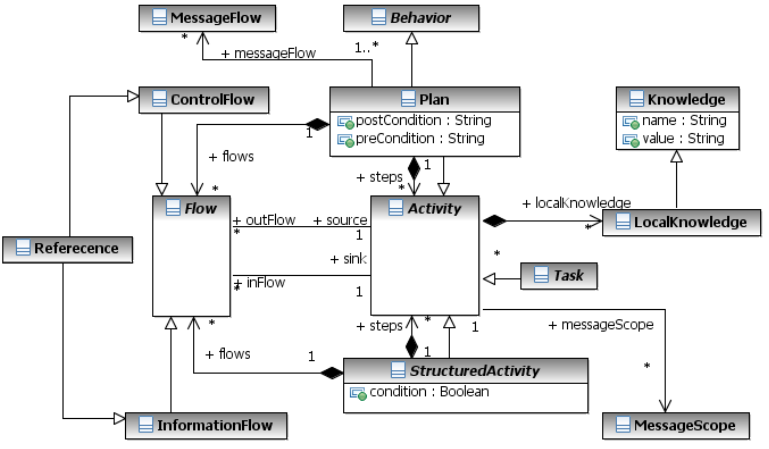


Fig. 9. The partial metamodel reflecting the core behavioral aspects of Pim4Agents

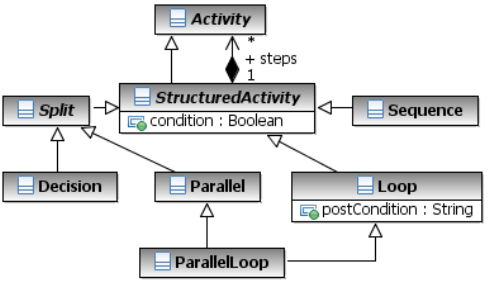


Fig. 10. The partial metamodel reflecting complex activities of the behavioral aspects

the message exchange (defined by the *Protocol*) is implemented by the particular *Plan* in an adequate manner.

The body of a *Plan* is mainly represented by the specializations of an *Activity*. A *StructuredActivity* (see Fig. 10) is an abstract class that introduces more complex control structures into the behavioral view. It inherits from *Activity*, but additionally owns a set of *Activities* and *Flows*.

A *Sequence* as a specialization of a *StructuredActivity* denotes a list of *Activities* to be executed in a sequential manner as defined by contained *ControlFlows* through their *sink* and *source* attributes. Beside using the concept of *Sequence*, a sequence of *Activities* can additionally be directly described by linking the particular *Activities* through *ControlFlows*. However, the concept of *Sequence* allows hiding the concrete trace which might be important when designing complex *Plans* as scalability is improved. A *Split* is an abstract class that defines a point in a *Plan* where a single thread of control splits into multiple threads of control. We distinguish between *Parallel* and *Decision* as specializations.

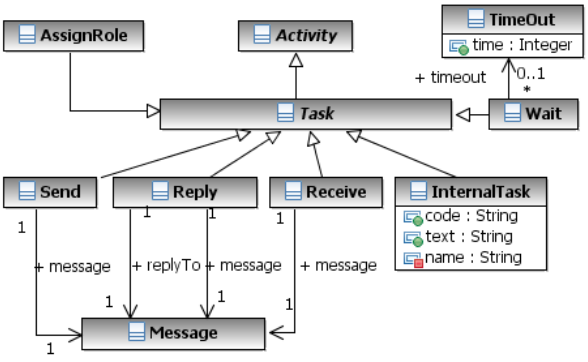


Fig. 11. The partial metamodel reflecting atomic activities of the behavioral aspects

A *Parallel* is a point in a *Plan*, where a single thread of control splits into multiple threads of control which are executed in parallel. Thus a *Parallel* allows *Activities* to be executed simultaneously or in any order. How the different threads are synchronized is defined by a *SynchronizationMode*. Feasible options are XOR (i.e., exactly one path is synchronized), AND (i.e., all paths are synchronized) and NofM (i.e., n of m paths are synchronized, where n and m are both natural numbers and $n \leq m$).

In contrast to a *Parallel*, a *Decision* in PIM4AGENTS is a point in a *Plan* where, based on a *Condition*, at least one *Activity* of a number of branching *Activities* must be chosen. A *Decision* can either be executed in an XOR or OR manner. In contrast, a *Loop* is a point in a *Plan* where a set of *Activities* are executed repeatedly until a certain pre-defined *Condition* evaluates to false. It allows looping that is block structured, i.e., patterns allow exactly one entry and exit point. A *ParallelLoop* as a specialization of *Loop* and *Parallel* allows specifying iterations in the form that each trace is executed in parallel.

Like a *StructuredActivity*, a *Task* (depicted in Fig. 11) is an abstract class that inherits from *Activity*. Unlike a *StructuredActivity*, a *Task* mainly focuses on atomic activities and thus does not contain any *Activities* or *Flows*. Fig. 11 depicts the partial metamodel of *Task*-related concepts. A *Send* activity specifies that the referred *Message* is sent, whereas the *Receive* activity denotes that the particular *Message* is received and *Reply* answers this *Message*. Furthermore, the *AssignRole* activity allows to assign *AgentInstances* to *Roles* (i.e., *Actors* and *Wait* to wait for a particular time or event. Finally, an *InternalTask* can be used to define code.

5.6 Deployment Aspect

The deployment aspect of PIM4AGENTS (cf. Fig. 12) deals with the kind of *AgentInstances* that exist in the running systems and how these instances are bound to the particular *DomainRoles* through the concept *DomainRoleBinding*. Even if using the deployment aspects is optional, i.e., the instances of agents could

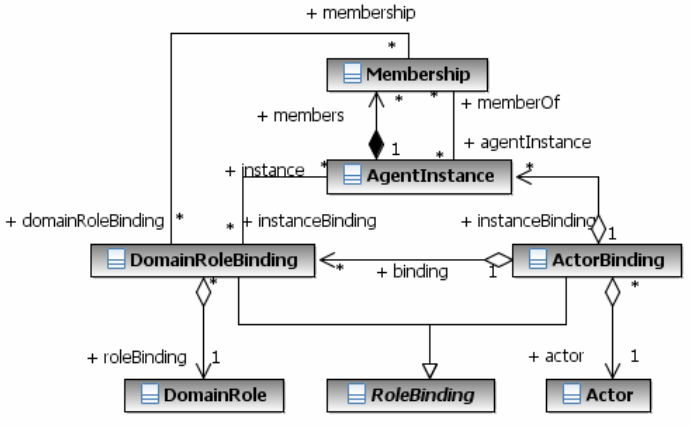


Fig. 12. The partial metamodel reflecting the deployment aspect of Pim4Agents

also be introduced on the particular agent platform selected to execute the design, for closed MASs it might already make sense to define the agent instance during design time. In this case, *AgentInstances* can be bound through the concept of *DomainRoleBinding* to the particular *DomainRoles* their *Agents* can perform. Through the concept of *ActorBinding*, these bound *AgentInstances* can then be assigned to *Actors* of an *Interaction*. An *AgentInstance* refers to its *Agent* type, which is either *Agent* or *Organization*. In the case of *Organization*, the *AgentInstance* refers to the concept *Membership* that includes all *AgentInstances* that are members.

6 Modeling TeleTruck

In this section, we demonstrate how to use the different aspects of Pim4Agents for designing the TeleTruck scenario. In particular, we illustrate how to model the collaborative settings, the interaction protocols to define the exchange of messages, as well as the agents' internal behaviors. We do not claim that the diagrams presented in this section cover really all aspects of the TeleTruck scenario. For example, there might be additional behaviors, collaborations, etc. that are not shown in this section. We picked a proper subset of the different aspects to give an insight how the Dsml4MAS language can be used for modeling the TeleTruck scenario.

6.1 Identification of Involved Agents

The agent diagram bases on the agent aspect of Pim4Agents (see Section 5.1) and is used to specify the different agent types of a system. Fig. 13 depicts how the two agent types TruckAgent and CompanyAgent can be modeled. The TruckAgent represents a company's trucks and performs the domain role TruckRole.

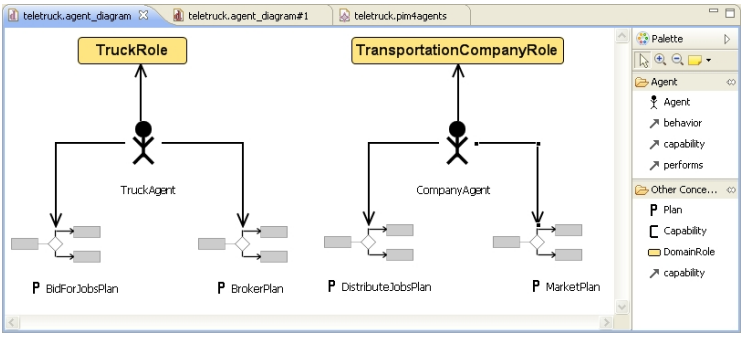


Fig. 13. Agent diagram of the TeleTruck scenario

Likewise, the CompanyAgent represents the controlling part of the transportation company and performs the TransportationCompanyRole.

The plans in Fig. 13 implement the agent's behaviors for the CNP and ST protocol (STP). For example, the DistributeJobsPlan implements the company's behavior in the CNP, whereas the BidForJobsPlan implements the opposite's behavior. Section 6.4 refines the BidForJobsPlan into more detail. Moreover, there might be additional plans for communicating with the customer and other tasks.

6.2 Modeling Organizations and Collaborations of Agents

After the agent types have been identified, the cooperation between these agents can be modeled. Pim4Agents provides the concept of *Organization* to specify the cooperation of agents (see Section 5.2). Modeling organizations with Dsml4MAS consists of two steps: (i) creation of the organization itself, its domain roles, plans, and utilized protocols and (ii) the specification of collaborations between domain roles inside the organization.

Fig. 14 shows how the TeleTruckOrganization can be modeled with the Dsml4MAS. The TeleTruckOrganization requires the domain roles TruckRole, TransportationCompanyRole, and Customer.

Moreover, Fig. 14 depicts that the TeleTruckOrganization utilizes the CNP and STP. The protocols will be refined later into more detail. The internal collaborations of the TeleTruckOrganization are modeled with the concept of *Collaboration* (see Section 5.2). Fig. 15 shows the TeleTruckCollaboration which describes the details of the collaboration between the TruckRole and TransportationCompanyRole.

The TeleTruckCollaboration contains the domain role binding TruckBinding and CompanyBinding which bind the domain roles to the collaboration and are visualized as ports. Actor bindings are used to specify the binding between a domain role binding and the actors of the utilized protocols. For example, the CompanyBinding is bound by the STPMarketBinding to the Market actor of the STP and by the CNPInitBinding to the Initiator actor of the CNP. This specifies that a role filler who performs the TransportationCompanyRole has to play the Market actor in the STP and the Initiator actor in the CNP.

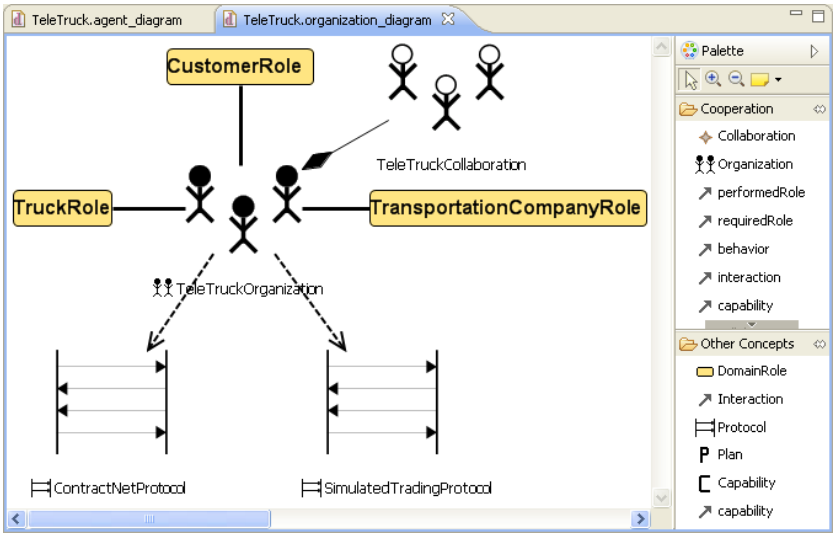


Fig. 14. Organization diagram of the TeleTruck scenario

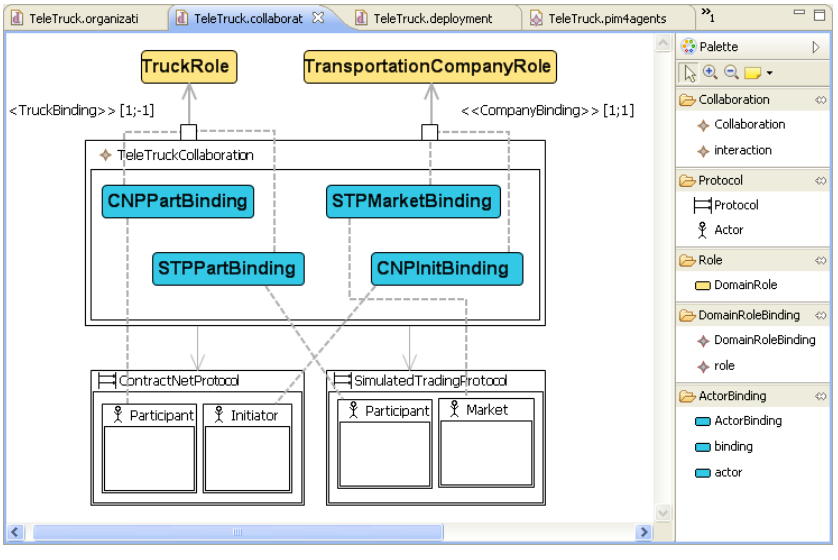


Fig. 15. Collaboration diagram of the TeleTruck scenario

6.3 Modeling Protocols

In the Dsml4MAS language, protocols are used to specify the message exchange between actors. The concept of actor allows us to handle named sets of role fillers at design time. A collaboration is the context in which the actors of the utilized

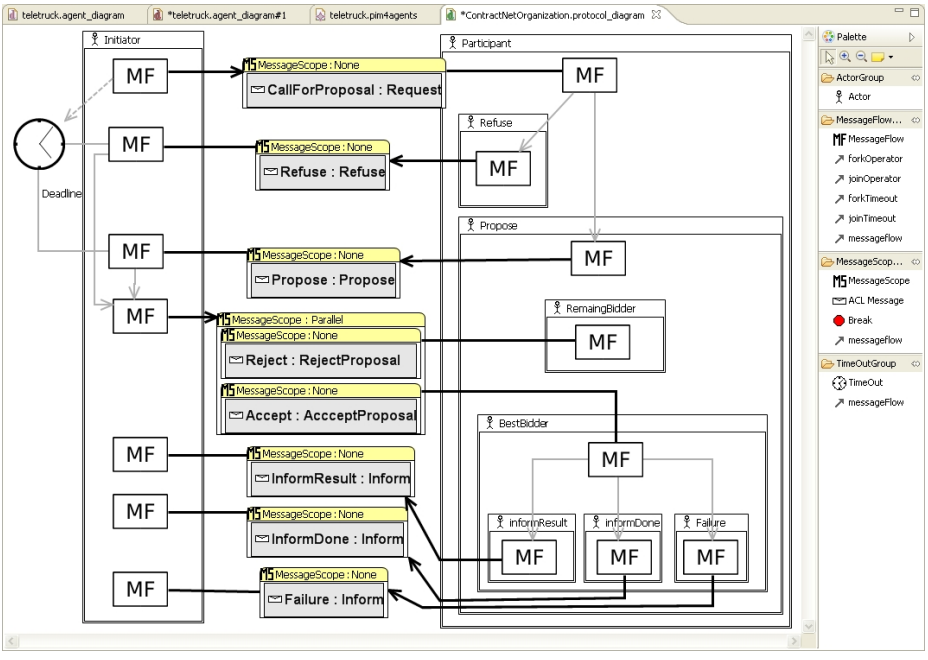


Fig. 16. Contract net protocol

protocols are bound to domain roles of an organization. Moreover, bindings are used at design time to specify the minimum and maximum number of role fillers that can be assigned to an actor at runtime.

The CNP belongs to the family of the cooperation protocols and is the most prominent protocol in DAI as it provides a solution for the connection problem, i.e., to find an appropriate agent to work on a given task. It bases on the contracting mechanism used by business to govern the exchange of goods and services and though uses a minimum of messages which makes it very efficient for task assignment. The protocol defines how an initiator sends out a number of calls for proposal to a set of participants. Some of these participants will refuse the call, while others may come up with a proposal. The initiator then evaluates the proposals and accepts the most adequate(s) and rejects the others. For those that are accepted, there are three different final results sent back to the initiator.

For designing the CNP using PIM4AGENTS, firstly, we introduce two actors called Initiator and Participant. The protocols starts with the first message flow of the Initiator that is responsible for sending the CallForProposal message of the performative type *cfp*. The CallForProposal message specifies the task as well as the conditions that can be specified within the properties view of the graphical editor. When receiving the CallForProposal, each agent instance performing the Participant decides on the base of free resources whether to Propose or Refuse. However, how this selection function is defined cannot be expressed in the protocol description, as private information are later on manually added to the automatically

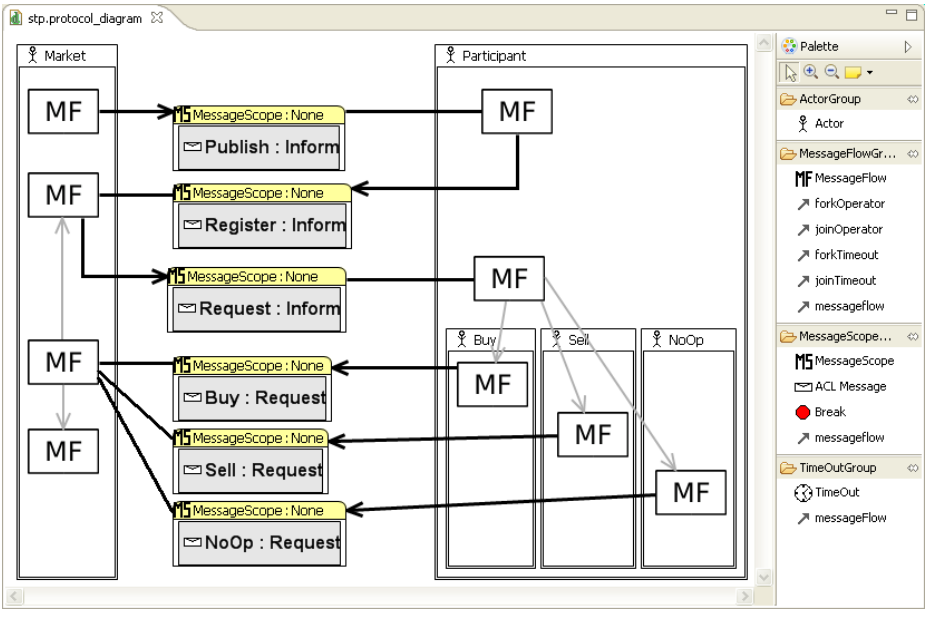


Fig. 17. Simulated trading protocol

generated behavior description. To distinguish between the alternatives, two additional actors (i.e., Propose and Refuse) are defined that are subactors of the Participant (i.e., any agent instance performing the Participant should either perform the Propose or Refuse actor). The transitions between the message flow within the Participant and the message flows of the Refuse or Propose actors through the *messageFlow* reference underline the change of state for the agent instance performing the Participant actor. However, the transitions are only triggered if a certain criterion is met. In the CNP case, the criterion is that the Initiator got all replies, independent of its type (i.e., Refuse or Propose). The *postConditions* of a *MessageFlow* can be defined in the properties view of the graphical editor. The message flows within the Refuse and Propose actors are then responsible for sending the particular messages (i.e., Refuse and Propose).

After the deadline expired (defined by the *TimeOut*) or all answers sent by the agent instances performing the Participant actor are received, the Initiator evaluates the proposals in accordance to a certain selection function, chooses the best bid(s) and finally assigns the actors BestBidder and RemainingBidder accordingly. Again, the selection function is not part of the protocol, but can be defined later on in the corresponding plan. Both, the BestBidder actor as well as the RemainingBidder actor—containing the agent instances that were not selected—are again subactors of the Propose actor. The Initiator sends an AcceptProposal message to the BestBidder and a RejectProposal message to the RemainingBidder in parallel.

After completing the work on the assigned task, the BestBidder reports its status to the Initiator either in the form of an InformResult or InformDone message

to report the successful completion or in the form of a Failure message in case the BestBidder fails to complete the task. Therefore, we distinguish again between InformResult, InformDone and Failure actors which are subactors of BestBidder.

Like the CNP, the STP can be described in a nice manner using DSML4MAS. For this purpose, we introduce five *Actors*, i.e., Market, Participant, Buy, Sell and NoOp. The latter three are subactors of the Participant *Actor*. The concrete message exchange is illustrated in Fig. 17.

6.4 Creation of Plans

Fig. 18 shows the TruckAgent's BidForJobsPlan which implements the behavior of the Participant actor of the CNP shown in Fig. 16. The basic behavior of the TruckAgent has already been described in Section 2.1.

The lower part of Fig. 18 shows the properties view of the ReceiveNewTask task. We can see that the task refers to the BidRequest message. The BidRequest message is sent by the CompanyAgent to all of its TruckAgents to offer a new task. As the TeleTruckOrganization utilizes the CNP to distribute tasks, the BidRequest message (type *Message*) refers to the CallForProposal message (type *ACLMessage*) of the CNP (see Fig. 16).

PIM4AGENTS distinguishes between *ACLMessages* that are used to specify the message sequences of a protocol and the actual *Messages* that are sent by plans. As protocols are reusable components, *ACLMessages* do not specify the resources that are transmitted by them. If a message shall be sent by a plan

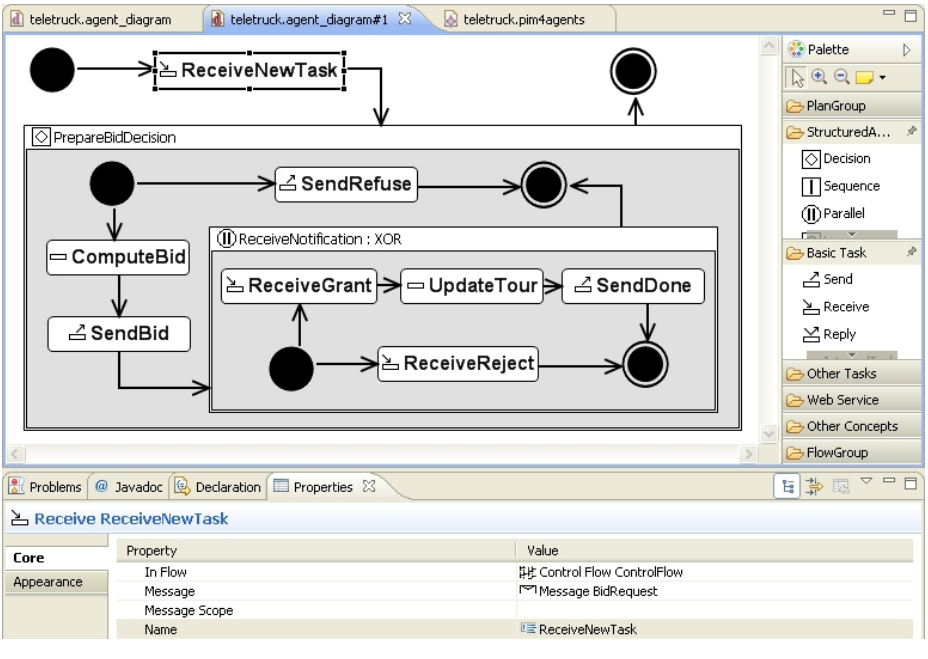


Fig. 18. BidForJobsPlan of the TELETRUCK scenario

(e.g. the BidRequest message from Fig. 18), we have to introduce a new *Message* that refers to a *ACLMessage* of a protocol (here the CallForProposal *ACLMessage*) and assign some application specific resources to it. In the TeleTruck example, the CompanyAgent sends information about the new order. Resources can be modeled in the environment diagram. Due to space restrictions, the resources are not shown here.

The TruckAgent either computes a bid or sends a refuse message to the CompanyAgent. A truck refuses new orders if it is currently not able to accept new ones. In this case the plan terminates. The computation of the bid is covered by the *InternalTask* ComputeBid. An *InternalTask* is like a black box behavior that is not further refined at the model level. Of course, a suitable algorithm has to be chosen and plugged into the generated plans.

The TruckAgent sends the computed bid to the CompanyAgent. If the bid is accepted, the TruckAgent updates its route and sends a notification to the CompanyAgent when the job is done. Otherwise, the plan terminates.

6.5 Modeling Deployment Aspects

In order to generate an executable multiagent system we have to model the deployment aspects of the TeleTruck system. For this purpose, the Dsml4MAS development environment offers the deployment diagram which bases on the deployment aspect of Pim4Agents (see Section 5.6). The deployment diagram allows us to model concrete instances of agents and organizations.

Fig. 19 depicts the organization instance TC1 of type TeleTruckOrganization, the agent instance CA1_1 of type CompanyAgent, and four agent instances of type TruckAgent which represent the company's trucks.

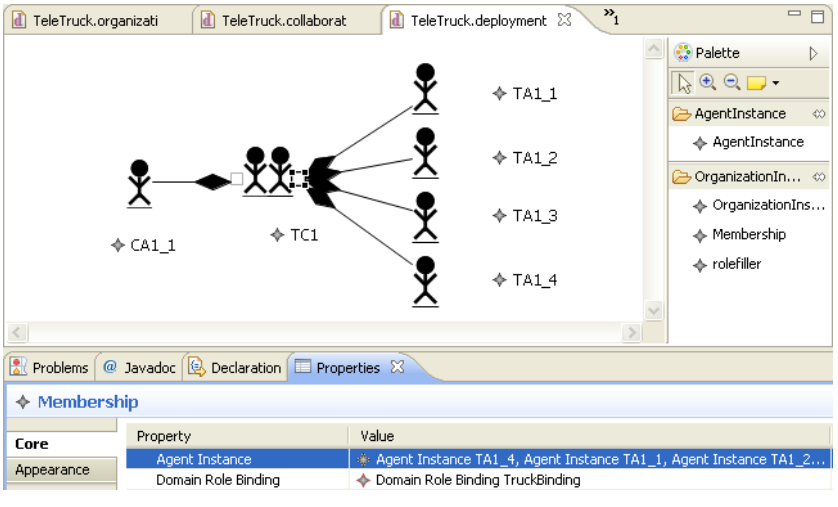


Fig. 19. Deployment diagram of the TeleTruck scenario

In order to assign the agent instances to the domain roles of the TC1 organization, we use the *Membership* concept. The lower part of Fig. 19 shows the properties view of the selected *Membership* concept. As we can see, the Membership concept refers to the TruckBinding of the TeleTruckCollaboration from Section 6.2. Therefore, it is exactly specified in which context the four truck agents are bound to the TC1 organization instance. The *Membership* concept specifies that the four trucks are now rolefillers of the TruckRole and play the Participant actor of the STP and CNP. Likewise, the company agent CA1_1 is bound to the CompanyBinding (see Fig. 15).

7 Related Work

Logistics management has been investigated in DAI and MAS research in the early nineties (e.g. [17]). A significant number of articles have been published till today which demonstrates the relevance of this application domain for DAI and MAS research. The workshop series on Agents in Traffic and Transportation shows that the research interest in this application domain is still active. Interested readers are referred to [18] for a more comprehensive overview of the work done so far. The work presented in this article builds on the results of [2].

In the MAS community, Agent UML (AUML) is the most prominent modeling language. AUML is an extension of the Unified Modeling Language (UML) to overcome the limitations of UML with respect to MAS development. AUML results from the cooperation between the Object Management Group (OMG) and the Foundation of Intelligent Physical Agents (FIPA), aiming to increase acceptance of agent technology in industry. However, even if AUML is the most cited agent-based modeling language, AUML's usage is mainly restricted to agent interaction protocols. Other aspects like agents, organizations or roles part of AUML are not often used.

Other, more methodology-oriented approaches like for instance Prometheus [19], Tropos [20] have demonstrated their usefulness in scenarios like the conference management system. However, it is unclear if complex scenarios like the TeleTruck scenario can be defined in an appropriate manner to generate executable code.

8 Conclusion and Future Work

This paper discusses a novel modeling framework for MAS called Dsml4MAS. It bases on a platform independent metamodel for multiagent systems (Pim4Agents) that defines the vocabulary for designing MASs in accordance to Dsml4MAS. We described the different aspects of Pim4Agents and demonstrated how to use this modeling framework when designing the TeleTruck scenario. In general, the design of the TeleTruck scenario can be described by the Pim4Agents in a nice and straightforward manner as Pim4Agents naturally supports the modeling of collaborative settings and complex interaction protocols like the Simulated Trading protocol or the Contract Net protocol.

However, the concrete algorithms have to be defined on the particular execution platform (e.g. Jack or Jade) target of the model transformation.

Our future work will comprise the development of further aspects that will mainly focus on the internals of an agent. Moreover, we are currently working on a model-driven methodology approach to develop DSML4MAS in an iterative manner.

References

1. Bordini, R.H., Dastani, M., Winikoff, M.: Current issues in multi-agent systems development (invited paper). In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS, vol. 4457, pp. 38–61. Springer, Heidelberg (2007)
2. Hans-Jürgen, Bürckert, Fischer, K., Vierke, G.: Holonic transport scheduling with teletruck. Journal of Applied Artificial Intelligence 14, 697–725 (2000)
3. Fischer, K., Kuhn, N., Müller, H.J., Müller, J.P., Pischel, M.: Sophisticated and distributed: The transportation domain. In: Müller, J.P., Castelfranchi, C. (eds.) MAAMAW 1993. LNCS, vol. 957, pp. 122–138. Springer, Heidelberg (1995)
4. Fischer, K., Müller, J.P.: A decision-theoretic model for cooperative transportation scheduling. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 177–189. Springer, Heidelberg (1996)
5. Davis, R., Smith, R.G.: Negotiation as a metaphor for distributed problem solving. Artificial Intelligence 20, 63–109 (1983)
6. Boddy, M., Dean, T.L.: Deliberation scheduling for problem solving in time-constrained environments. Artificial Intelligence 67, 245–285 (1994)
7. Russell, S.J., Zilberstein, S.: Anytime sensing, planning, and action: A practical model for robot control. In: Proceedings of IJCAI 1993, pp. 1402–1407. Chambery, F, Morgan Kaufmann Publishers Inc., San Mateo (1993)
8. Bachem, A., Hochstättler, W., Malich, M.: The Simulated Trading Heuristic for Solving Vehicle Routing Problems. Technical Report 93.139, Mathematisches Institut der Universität zu Köln (1993)
9. Bachem, A., Hochstättler, W., Malich, M.: Simulated Trading: A New Approach For Solving Vehicle Routing Problems. Technical Report 92.125, Mathematisches Institut der Universität zu Köln (1992)
10. Hahn, C.: A domain specific modeling language for multiagent systems. In: Padgham, L., Parkes, C.P., Mueller, J., Parsons, S. (eds.) Proceedings of 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008), pp. 233–240 (2008)
11. Cook, S., Jones, G., Kent, S., Wills, A.C.: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley, Reading (2007)
12. Smith, G.: The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers, Dordrecht (2000)
13. Hahn, C., Fischer, K.: The static semantics of the domain specific modeling language for multiagent systems. In: Proceedings of the 9th International Workshop on Agent-Oriented Software Engineering (AOSE 2008). Workshop at AAMAS 2008, 13.5.2008 (2008)
14. Warwas, S., Hahn, C.: The contrete syntax of the platform independent modeling language for multiagent systems. In: Proceedings of the Agent-based Technologies and applications for enterprise interOPerability (ATOP 2008) at AAMAS (2008)

15. Papasimeon, M., Heinze, C.: Extending the UML for designing JACK agents. In: Proceedings of the Australian Software Engineering Conference (2001)
16. Bellifemine, F., Bergenti, F., Caire, G., Poggi, A.: JADE - a java agent development framework. Multiagent Systems, Artificial Societies, and Simulated Organizations, ch. 5, vol. 15, pp. 125–147. Springer, Heidelberg (2005)
17. Fischer, K., Kuhn, N., Müller, H.J., Müller, J.P., Pischel, M.: Sophisticated and distributed: The transportation domain. In: Castelfranchi, C., Müller, J.P. (eds.) MAAMAW 1993. LNCS, vol. 957, pp. 122–138. Springer, Heidelberg (1995)
18. Perugini, D.: Agents for Logistics: A Provisional Agreement Approach. Ph.D thesis, The University of Melbourne (2006)
19. Padgham, L., Winikoff, M.: Prometheus: A pragmatic methodology for engineering intelligent agents. In: Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, pp. 97–108 (2002)
20. Susi, A., Perini, A., Giorgini, P., Mylopoulos, J.: The tropos metamodel and its use. Informatica 29, 401–408 (2005)

Part I

Organisations and Norm-Governed Systems

Specifying Open Agent Systems: A Survey

Alexander Artikis[1,2] and Jeremy Pitt[2]

[1] Institute of Informatics and Telecommunications,
National Centre for Scientific Research "Demokritos", Athens 15310
[2] Electrical & Electronic Engineering Department,
Imperial College London, SW7 2BT
a.artikis@acm.org, j.pitt@imperial.ac.uk

Abstract. Electronic markets, dispute resolution and negotiation protocols are three types of application domain that can be viewed as open agent systems. Members of such systems are developed by different parties and have conflicting goals. Consequently, they may choose not to, or simply fail to, conform to the norms governing their interactions. It has been argued that many practical applications in the future will be realised in terms of open agent systems of this sort. Not surprisingly, recently there is a growing interest in open systems. In this paper we review and compare four approaches for the specification of open systems, pointing out the extent to which they satisfy a set of requirements identified in the literature.

1 Introduction

Agents, although required to make decisions and act locally, operate in the context of multi-agent systems (MAS). A particular kind of multi-agent system is one where the members are developed by different parties and have conflicting goals. A key characteristic of this kind of MAS, that is due to the globally inconsistent sets of goals of the members, is the high probability of non-conformance to the specifications of the systems. A few examples of this type of MAS are electronic marketplaces, dispute resolution protocols, Virtual Organisations and digital media rights management protocols. Multi-agent systems of this type are often classified as 'open'. It has been argued that many practical applications in the future will be realised in terms of 'open agent systems' (OAS) of this sort.

For the purposes of this paper, we consider a multi-agent system as open if it exhibits the following characteristics:

- The internal architectures of the members are not publicly known.
- Members do not necessarily share a notion of global utility [38].
- The behaviour and the interactions of the members cannot be predicted in advance [24].

The first characteristic implies that an OAS may be composed of agents with different internal architectures. Therefore, we will treat OAS as *heterogeneous* ones. Moreover, there is no direct access to an agent's mental state and so we

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 29–45, 2009.

can only infer things about it. The second characteristic implies that the members of an OAS may fail to, or even choose not to, conform to the specifications (of that system) in order to achieve their individual goals. In addition to these characteristics, OAS are always subject to unanticipated outcomes in their interactions [24].

Often in the literature OAS are those where agents may enter or leave the system at any time (see, for example, [24,38,55,60]). Usually, agents enter a system after having successfully executed a *role-assignment protocol*. The specification of such a protocol is application-specific. Our definition of an agent system as open is irrespective of the protocols that specify the ways by which agents enter or leave the system.

Recently there is a growing interest in the MAS community in OAS. The aim of this paper is to present a critical survey of approaches for the specification of OAS. We review the following lines of work: (a) artificial social systems, (b) enterprise modelling, (c) commitment protocols, and (d) electronic institutions. We chose to review the aforementioned lines of work because they are well-known and illustrate different aspects of OAS. Clearly there are other prominent approaches for specifying OAS, including [3,4,5,6,8,18,22,28,32,40,43,54]. Given the space limitations, however, it is not possible to present here a review of all relevant lines of research.

Following [41] the review of each approach is divided into two parts: (i) the description of the approach, and (ii) our commentary on the approach. Consequently the paper may be read in two ways: its main aim is to describe and compare the various lines of research, but it can also be read as a catalogue of abstracts by ignoring the commentary included with each entry.

The commentary of each approach includes a discussion on whether a set of requirements identified in the literature for the specification of OAS are addressed. It has been argued, for example, that OAS may be viewed as instances of *normative systems* [26]. A feature of this type of system is that actuality, what is the case, and ideality, what ought to be the case, do not necessarily coincide. Therefore, it is essential to specify what is permitted, prohibited, and obligatory, and perhaps other more complex normative relations (such as duty, right, privilege, authority, ...) that may exist between the agents. Among these relations, considerable emphasis has been placed on the representation of 'institutional' (or 'institutionalised') power [27,30]. This is a feature of norm-governed systems whereby designated agents, when acting in specified roles, are empowered by an institution to create relations or states of affairs of special significance within that institution — *institutional facts* [39]. Consider, for example, the case in which an agent is empowered by an institution to award a contract and thereby create a bundle of normative relations between the contracting parties. According to the account given by Jones and Sergot [27], institutional power can be seen as a special case of a more general phenomenon whereby an action, or a state of affairs, A — because of the rules and conventions of an institution — counts, in that institution, as an action or state of affairs B [39] (such as when sending a letter with a particular form of words counts as making an offer).

It has also been argued that the semantics of the rules of an OAS must be, among other things, *formal*, *declarative* (the semantics should describe *what* rather than *how*), and *verifiable* (it should be possible to determine whether an agent is acting according to the rules of an OAS) [50,52]. In the commentary of each approach we will point out the extent to which these requirements are satisfied. Finally, the commentaries include a discussion on the types of computational task that may be performed on the system specification.

A short survey of interaction protocols for MAS was recently presented in [31]. The criteria against which each protocol was evaluated are different from the ones we are concerned with. In [31] emphasis was placed on the issue of dynamic protocol composition. We do not exclude from our survey protocols that are not dynamically composable; moreover, unlike the survey of [31], we are concerned with the normative relations that are expressed by each approach for specifying OAS, whether the approach has a verifiable semantics, and the supported computational tasks.

2 Artificial Social Systems

Moses, Tennenholtz and Shoham [33,34,47,48,53] present in various papers an approach to the design of multi-agent systems, called artificial social systems. An artificial social system is based on a set of restrictions on the agents' behaviour, called *social laws*. These laws allow the agents to co-exist in a shared environment and pursue their goals. Social laws are determined at design-time, that is, before the commencement of the agent activities, and enable the members of a system to create their own plans at run-time.

Moses and Tennenholtz [34] define artificial social systems first by defining multi-agent systems in general, and then by defining *normative systems* and *social systems*.

A multi-agent system $\mathcal{M}$ is represented as a tuple of the form $\mathcal{M} = (N, W, K_1, \ldots, K_n, A, Able_1, \ldots, Able_n, I, T)$, where:

- $N = \{1, \ldots, n\}$ is a set of agents.
- W is a set of possible worlds (states).
- $K_i \subseteq W \times W$ are accessibility (equivalence) relations that capture the knowledge of each agent i ($i \in N$).
- A is a set of primitive individual actions.
- $Able_i : W \rightarrow 2^A$ are functions that determine the set of actions that each agent i is physically capable of performing ($i \in N$).
- I is a set of possible external inputs for the agents. The elements of this set are intended to capture any messages that an agent may receive from outside the system.
- $T : W \times (A \times I)^n \rightarrow W \cup \{\perp\}$ is a state transition function that determines the state immediately after the current one as a function of the actions that each agent performs, and the input each agent receives in the current state.

A normative system is a multi-agent system associated with a set of social laws. Tennenholtz [53] structures the states of a system in order to define social laws.

Given disjoint sets of states $S_1, \ldots, S_n$, the set of system states W is defined as the set of all tuples $(s_1, \ldots, s_n)$ where $s_i \in S_i$ for each agent i. Given a system state $w \in W$, w_i is the i-th element of w.

Given a set of system states W, a first-order language $\mathcal{L}$ (with an entailment relation $\models$), and a set of actions A, a social law is a set of constraints of the form (α_j, φ_j) where $\alpha_j \in A$ and $\varphi_j \in \mathcal{L}$, at most one for each $\alpha_j \in A$.

The language $\mathcal{L}$ is used to describe what is true and false in different system states. Intuitively, an action a is *prohibited* for agent i at state w by the constraint (a, φ) if and only if $w_i \models \varphi$. $\varphi \in \mathcal{L}$ is the most general condition that prohibits the performance of action a.

Every physically possible action that is not prohibited by a social law at a particular state is considered *allowed* (according to that law) in that state. The $Legal_i : W \rightarrow 2^A$ functions represent the set of actions that agent i is allowed to perform in state $w \in W$, according to the system's laws. The $Legal_i$ functions satisfy the three following properties:

1. Each agent knows what actions it is allowed to perform (*epistemological adequacy*).
2. The set of actions that an agent is allowed to perform are physically possible for that agent (*physical adequacy*): $\forall i \in N,\ \forall w \in W,\ Legal_i(w) \subseteq Able_i(w)$.
3. In every state, there is at least one action an agent is allowed to perform (*non-triviality*): $\forall i \in N,\ \forall w \in W,\ Legal_i(w) \neq \emptyset$.

A normative system $\mathcal{N}$, extending a multi-agent system $\mathcal{M}$, is defined as a pair $\mathcal{N} = (\mathcal{M}, \{Legal_i\}_{i \in N})$.

In a normative system the state transition function takes as an argument not only the current state and the actions performed by the agents but also the social laws that are in force (which are associated with the performed actions). Given such input, the state transition function produces a prediction about the set of possible next states, provided that all agents comply with the laws.

A social system is regarded as an instance of a normative system. All possible states in a social system must be 'socially acceptable'. This will happen if the initial states are 'socially acceptable' and as long as every agent obeys the laws of the system. Moreover, each agent should always be able to attain a set of 'socially acceptable' goals — these are the goals that the social system allows the agents to achieve.

The set of 'socially acceptable' states is represented by the W_{soc} set ($W_{soc} \subseteq W$) whereas the set of 'socially acceptable' goals is represented by the G_{soc} set. An agent's goal g is associated with a set of states $W_g \subseteq W$; these are the states in which the goal has been achieved. The W_{soc} set is intended to capture the 'safety' and 'fairness' of the system while the G_{soc} set is intended to capture the 'liveness' of the system. The definition of both of these sets is application-specific.

Social systems are defined in terms of the agents' *legal plans*. A plan for an agent i is defined as a function $Ch_i : W \rightarrow A$ that satisfies the following property:

– Each chosen action must be physically possible:
$\forall i \in N,\ \forall w \in W,\ Ch_i(w) \in Able_i(w)$.

The above property is strengthened in order to define a *legal* plan:

– Each chosen action must be allowed according to the system's laws: $\forall i \in N,\ \forall w \in W, Ch_i(w) \in Legal_i(w)$.

A social system $\mathcal{S}$ for a multi-agent system $\mathcal{M}$, consistent with W_{soc} and G_{soc}, is a normative system extending $\mathcal{M}$ that satisfies the following:

1. A state $w \in W$ is *legally reachable* only if it is 'socially acceptable', that is, $w \in W_{soc}$.
2. For every legally reachable state w, if the goal of agent i in w is 'socially acceptable', that is $g \in G_{soc}$, then there is a legal plan for i that, starting in w, will attain g as long as all other agents act according to the laws of the social system.

Fitoussi and Tennenholtz [16,17] define two properties of social laws. These properties are called *minimality* and *simplicity*. The rationale for defining minimal social laws is that a minimal law provides agents more freedom in choosing their behaviour (that is, it prohibits fewer actions) while ensuring that agents conform to the system specification. The rationale for defining simple social laws is that a simple law relies less on the agents' capabilities rather than a non-simple one. Learning a simple law is likely to be faster and its representation is likely to be more succinct.

As already mentioned, work on artificial social systems focuses on the design-time specification of laws that allow agents to achieve their goals at run-time. Sometimes it is not feasible to design social laws at design-time. For example, it might be the case that all the characteristics of a system are not known at design time. In such a case the members of a system will *converge* to social laws. In a complementary (to the artificial social systems approach) work, Shoham and Tennenholtz [46,49] examine the case in which laws emerge at run-time. The issue of law emergence is beyond the scope of this paper; therefore, we do not present here this line of work.

Commentary

The work on artificial social systems has been very influential in the field of multi-agent system specification. We make the following comments with respect to the suitability of this approach for OAS specification. First, social laws only refer to the permissions of the agents, regulating their behaviour. No other normative relations are considered. This is probably due to the fact that the applications that have been studied in the context of the artificial social systems approach (for example, mobile robots moving along a two-dimensional grid [48, Section 2] or a circular automated assembly line [17, Section 3.1]) can be mainly described in terms of physical actions rather than communicative actions, and *brute facts* rather than institutional facts [39]; being in physical possession of an object, for example, is a brute fact (it can be observed), whereas being the owner of that object is an institutional fact.

Second, in the artificial social systems approach it is assumed that all members comply with the social laws. However, as Shoham and Tennenholtz [47, p.281] mention, the members of agent systems (and especially OAS) do not always act according to the social laws. There is a need, therefore, to consider and develop mechanisms that deal with non-conformance to the laws.

The semantics of social laws are formal and declarative. Regarding the verification of compliance to social laws, it is necessary that the laws make no assumptions about the internal architectures of the agents, that is, ϕ in law (a, ϕ) does not refer to the mental states the agents — recall that there is no access to the agents' internal states in an OAS. Finally, to the best of our knowledge, there is no discussion about a software implementation allowing for reasoning about the social laws of a system (thus, informing the decision-making of the agents by computing legal plans, allowed actions, and so on).

3 Enterprise Modelling

In this section we review the work of Fox, Grüninger and colleagues [19,23] on modelling computational enterprises/organisations. In this approach an organisation is viewed as a set of agents playing roles in which they are acting to achieve specific goals according to various constraints defining the 'rules of the game'. More precisely, an organisation consists of: *divisions* and *subdivisions* (recursively defined), a set of *organisation agents* (members of divisions), a set of *roles* that members play in the organisation, and an *organisation goal tree* that specifies the goal (and the sub-goals) that members are trying to achieve.

A role R is associated with, among other things, a set of: (i) *goals* that each agent occupying R is obliged to achieve, (ii) *processes*, that is, activity networks defined to facilitate the achievement of the goals, (iii) *policies* constraining the performance of R's processes, and (iv) *skills* that are required to achieve the goals of R. An element of a role on which considerable emphasis is placed is that of *authority*. Depending on the type of authority agent Ag_1 has over agent Ag_2, due to the roles Ag_1 and Ag_2 occupy:

- Ag_1 may assign a set of goals to Ag_2, thus making Ag_2 obliged to achieve these goals.
- Ag_1 may assign a set of new roles to Ag_2.
- Ag_1 may *empower* Ag_2 to perform a set of actions. Empowerment — a key feature of this approach on enterprise modelling — may be necessary in order to achieve the set of goals associated to the roles one occupies; for instance, an agent may manipulate/consume a set of resources only if it is empowered to access these resources.

The constraints on the members of an organisation ('rules of the game'), such as the policies of each role, are expressed with the use of a dialect of the Situation Calculus [36]. Moreover, a logic programming implementation of the Situation Calculus formalisation has been developed supporting the following computational tasks:

– Planning; what sequence of actions must be performed to achieve a particular goal? In order to minimise the cycle time for a product, for example, we may compute a plan maximising the concurrency of actions.
– Narrative assimilation (temporal projection); what are the effects of the occurrence of a set of actions? For instance, how are the obligations and powers of a set of agents affected by the actions performed so far? Or what will happen if we move one task ahead of schedule and another behind schedule?

Commentary

Fox and colleagues, in their model of computational enterprises/organisations, identify and represent several normative relations of the member agents, such as obligation, authority and empowerment. This is in contrast to the artificial social systems approach in which only permissions are considered. This difference is not surprising as the modelling of computational organisations, unlike the applications studied in the context of the artificial social systems approach, requires the representation of institutional facts and communicative actions. The representation of 'empowerment', however, is not very clear. In some cases 'empowerment' seems to coincide with the concept of institutional power presented in the introduction, in the sense that an empowered agent may create a set of institutional facts, such as the establishment of a contract with a third party, while in other cases 'empowerment' seems to coincide with permission, in the sense that performing a permitted action will not be penalised.

Like the artificial social systems approach, Fox and colleagues do not consider mechanisms for dealing with non-compliance to the specifications (for instance, violating obligations).

The semantics of the organisational constraints, expressed by means of a version of the Situation Calculus, are formal, declarative and verifiable. Finally, there is no discussion on the complexity of the computational tasks supported by the logic programming implementation of the Situation Calculus formalisation — for instance, can verification of compliance be performed in computational systems composed of hundreds of agents?

4 Commitment Protocols

The concept of *commitment* has been frequently used in the literature for specifying protocols for OAS. The term 'commitment' (or 'social commitment') has been used in the MAS field to refer to a wide variety of different concepts [42]. A common usage of this term is in the sense of an obligation directed from one agent to another. In this paper we review the work of Singh and colleagues [7,9,11,12,51,52,56,59] on the so-called *commitment protocols*.

A commitment is viewed as a four-place relation involving a proposition (p) and three agents (x, y and G). More precisely, $c = \mathbf{C}(x, y, G, p)$ denotes a commitment from x toward y in the context of G and for the proposition p. In this case x is the debtor, y is the creditor, G is the context group and p is the

discharge condition of commitment c. The context group is viewed as the adjudicating authority that resolves disputes between the debtor and the creditor. *Explicit commitments*, unlike *implicit* ones, are explicitly represented by one or more agents. Implicit commitments are assumed to be common knowledge, or mutually believed in a multi-agent system. Commitments can also be divided into *base-level commitments* and *meta-commitments*. A commitment c is a meta-commitment if the proposition p refers to another commitment; c is a base-level commitment if p does not refer to other commitments. Regarding agent societies, Singh states that "[m]etacommitments create a society. The metacommitments are the norms of the society" [51, p.106].

Singh [51] has discussed the relationship between his interpretation of commitment and related concepts from deontic logic. For instance:

- A *pledge* is an explicit commitment.
- Singh claims that the classical approaches in deontic logic that express *ought* are unsuitable because 'ought' is inherently contextual. Therefore, he defines 'ought' to be relativised to the context group:
  $\mathbf{Ought}(x, G, p) \equiv (\exists G : x \in G \ \& \ c = \mathbf{C}(x, G, G, p))$. Notice that 'ought' is directed to the context group, not to a particular agent.
- *Obligation* has two main readings: one close to pledge and one close to 'ought'.
- A *convention* or a *custom* is considered to be an implicit meta-commitment relativised to a debtor and a context group.
- A *power* refers to the ability of an agent to force (if it desires) the alteration of a legal relation in which the another agent participates. Power is expressed as follows: $\mathbf{Power}(x, y, r) \equiv \mathbf{C}(G, x, G, request(x, G, r) \Rightarrow perform(G, r))$ where r is an operation on commitments whose creditor or debtor is an agent y and the context group is G. '$\Rightarrow$' represents strict implication.

Commitment protocols have been formalised in various ways, giving different types of semantics to the concept of commitment. Here we will briefly present three such formalisations.

Yolum and Singh [58,59] have specified commitment protocols with the use of a subset of Shanahan's 'full version' of Event Calculus (EC) [44]. All the 'operations' on commitments — *creating*, *discharging*, *cancelling*, *releasing*, *delegating*, *assigning* commitments — are formalised with the use of EC axioms. Consider, for example, the 'release' operation: $Release(E(y), c)$ releases the debtor x from the commitment $c = \mathbf{C}(x, y, p)$ (here, c is a 'base-level' commitment — in the interest of clarity we omit the context group G from the representation of c). The creditor y performs an action denoted by $E(y)$ that 'terminates' the debtor's commitment to bring about p [58]:

$$Release(E(y),\mathbf{C}(x, y, p)) \equiv \\ Happens(E(y), t) \wedge Terminates(E(y),\mathbf{C}(x, y, p), t) \quad (1)$$

Yolum and Singh have employed an abductive EC planner [45] in order to compute planning queries regarding the EC specifications of commitment protocols.

Given an initial state of a commitment protocol, a final state and the specification of the protocol, the planner will compute the protocol runs (if any), that is, the sequences of actions, that will lead from the initial state to the final one. Consequently, agents may use the EC planner to compute protocol runs that lead to a 'desirable' outcome. In the case where agents are not capable of processing a commitment protocol specification, however, a subset of a commitment protocol specification may be compiled to a finite state machine where no commitments are explicitly mentioned [57].

Singh and colleagues have also used the $C+$ action language [21] — a formalism with *explicit* transition systems semantics — to express commitment protocols (see [7, 11, 12] for a few recent papers). Like EC, $C+$ has been employed to express all operations on commitments as well as the effects of the agents' actions. The Causal Calculator [21], a software implementation using satisfiability solvers to compute planning and narrative assimilation queries regarding $C+$ formalisations, allows for the execution of the commitment protocol specifications.

In a line of work primarily aimed at protocol composition [9,10], commitment protocols have been formalised with the use of the Web Ontology Language (OWL) [1] and the Semantic Web Rule Language (SWRL) [25]. SWRL has been used for formalising protocol rules that include instances of OWL-P, an OWL ontology for protocols expressing, among other things, the concept of commitment. Protocol composition is beyond the scope of this paper; therefore, we do not expand our presentation of this line of research.

Commentary

The notion of commitment is very important when specifying OAS, that is, systems in which the members may have competitive individual goals and there is no guarantee about their behaviour. It is difficult to see, however, how an interaction protocol for an OAS (such as, for example, a protocol for e-commerce, negotiation, argumentation or voting) can be specified simply in terms of commitments in this sense. Other normative relations are at least as important when specifying a protocol and the meaning of protocol actions. For example, when a contract is established between two parties by means of accepting an offer, a party may be interested in finding out the circumstances in which it has the 'legal capacity' or institutional power to initiate proceedings against the other party. In an auction house it is important to know the circumstances in which an agent has the institutional power to place a bid and, consequently, change the current bidding price. Similarly there are other examples in which more normative relations than simply commitments need to be considered.

Like the other two reviewed approaches (artificial social systems and enterprise modelling), the formalisations of commitment protocols do not cater for mechanisms dealing with non-compliance with the system specifications (commitments, in this case).

The semantics of the aforementioned formalisations of commitment protocols are formal and declarative. Moreover, no assumptions are made about the

internal architectures of the agents, allowing for the verification of compliance in OAS.

A comparison of the Event Calculus (EC) and the $C+$ language, that were used for expressing commitment protocols, and a comparison of EC and the Situation Calculus, employed by Fox and colleagues for enterprise modelling, may be found in [35], for example. (See [2] for a comparison of EC and $C+$ with respect to specifying OAS.) To the best of our knowledge, the relationship between the aforementioned action languages, and SWRL and OWL-P, that have also been employed for specifying commitment protocols, has not been discussed.

The abductive EC planner and the Causal Calculator can be used, in principle, for the provision of both design-time services — for instance, proving protocol properties — and run-time services — for example, calculating the commitments current at each time for the benefit of the agents or their designers. These implementations, however, can become inefficient when considering large OAS. Moreover, it is assumed that the domain of each variable is finite and known from the outset (for instance, we need to know all possible propositions that an argumentation protocol participant may claim). Consequently, due to these issues, the use of these implementations for the provision of run-time services may be limited.

5 Electronic Institutions

Esteva and colleagues [13,14,15,20] have developed a systematic approach to the design and development of multi-agent systems which incorporate aspects of conventional behaviour and organisational structures in a formal specification of structured interactions. Such a specification of a multi-agent system is called an Electronic Institution (EI).

The core notions defining an EI are [14]:

- *Roles*. A named role defines dialogic capabilities, that is, an agent must occupy a role in order to make a communicative action. Roles also determine the organizational structure, through the definition of a role hierarchy to indicate, for example, subsumption and exclusivity between roles.
- *Dialogic framework*. The dialogic framework of an EI defines the acceptable message format of communicative actions, by specifying a named list of illocutions (a name for each type of communicative action), the content language (the language of propositions embedded in the illocution), and the ontology (a vocabulary of terms). In the specification of an EI, the dialogic framework also includes the roles and the organizational structure. The dialogic framework determines the space of illocutions that can be exchanged, the pertinent exchanges are defined by scenes.
- *Scenes*. A scene is directed graph specifying a structured exchange of messages between roles. Essentially a scene is a finite state diagram defining a communication protocol, with each state transition labelled by an illocution scheme, itself composed from elements of the dialogic framework (an illocution, two agent/role pairs (the sender/receiver), an expression of the content language, and a time stamp).

– *Performative structure.* A directed graph of scenes defines the performative structure(s) of the EI. The specification of a performative structure describes how agents can move between scenes by satisfying certain pre-conditions, specifically related to roles they occupy and communicative actions they make.
– *Normative rules.* Normative rules in an EI are expressions which impose obligations or prohibitions on communicative actions within scenes, which constrain or regulate the behaviour of agents which occupy the roles that are the subject of the rule.

In [20] two 'sorts' of normative rule are given. One sort concerns the so-called 'strict' obligations to perform communicative actions according to the scene specification. A second sort are (what is called) 'less strict' normative rules which express other permissions, obligations and prohibitions with a conditional or temporal constraint.

A 'strict' normative rule is of the form:

$$(\bigwedge_{i=1}^{m}\{illocutions\}_i \wedge \bigwedge_{j=1}^{n}\{conditions\}_j) \rightarrow \bigwedge_{k=1}^{m'}\{illocutions\}_k \wedge \bigwedge_{l=1}^{n'}\{conditions\}_l$$

with the intuitive meaning that if the illocutions $i, 1{\leq}i{\leq}m$ have been communicated ('uttered') in the appropriate scene states, and the expressions $j, 1{\leq}j{\leq}n$ are satisfied, then the illocutions $k, 1{\leq}k{\leq}m'$ satisfying the expressions $l, 1{\leq}l{\leq}n'$ 'must be' communicated in the appropriate scene states. If the right-hand side is $\perp$ instead of a conjunction of illocutions and conditions, then uttering the illocutions i is said to be a violation.

Note that no deontic operators are used in a 'strict' normative rule. Such operators are employed to express the 'less strict' normative rules [20]:

OBLIGED	*illocution*	PERMITTED	*illocution*
BEFORE	*illocution*	BETWEEN	(*illocution, illocution*)
IF	*condition*		

Here, the first expression contains a conditional obligation to perform a certain action (utter an illocution) prior to a certain other action, subject to certain other conditions (which might include performance of past actions). For example, in an auction scene, a buyer might be obliged to pay for an item before the auctioneer closes the auction, if the buyer performed the action (uttered the illocution) that was the winning bid. The second expression denotes a permission in an interval of time: for instance, in a voting scene, a voter might be permitted to vote after the chair has called for votes, and before the chair closes the ballot.

Two types of mechanism have been devised in order to deal with the possibility of non-compliance with the normative rules. In [37] devices called 'interagents' were employed to *enforce* the rules of an auction house to the buyer and seller agents. In this setting, forbidden actions were physically blocked by interagents. In [20] sanctions were applied in the case of rule violation; failure to comply with an obligation, for example, could result in a monetary penalty.

Software tools for computational support of the EI specification languages have been developed. Islander [13], for instance, is an integrated development environment for specifying EIs. It offers a graphical user interface which allows the user to edit performative structures, scenes and illocutions. A verification module is implemented which, it is claimed, checks for 'integrity' (whether all elements of an EI are defined), protocol correctness (standard properties, such as liveness, reachability and termination), and 'norm consistency'. In [13] verification of 'norm consistency' is performed only over the so-called 'strict' normative rules. Another tool for computational support of the EI specification languages is presented in [20]; this is a rule-based system for executing a set of normative rules, including the 'less strict' normative rules, with the aim of providing run-time services, such as the computation of the permissions and obligations of the agents current at each state.

Commentary

Electronic Institutions (EI) are, in some sense, a result of an almost ethnographic study of the fishmarket auction extended to other (human) institutions. The five core notions of EI can be used to give order and structure to MAS intended to support similar types of activity (e-commerce, negotiation, and so on). As such, it is a specification formalism whose analytic and formal basis lends itself well to computational support.

We note though that the normative rules concern only the permissions and obligations of the agents — this is similar to the work on commitment protocols where the focus was on directed obligations. It has been argued that other normative relations are at least as important as those when specifying the meaning of protocol actions.

The general strategy of designing mechanisms to force compliance and eliminate non-permitted behaviour — *regimentation* [26] — is rarely desirable (it results in a rigid system that may discourage agents from entering it), and not always practical (violations may still occur due to, say, a faulty regimentation device). For all of these reasons, allowing for sanctioning mechanisms may be a better option than relying exclusively on regimentation devices (interagents, in this case).

The semantics of the EI specification languages are formal and declarative. Moreover, verification of compliance has been performed in settings where there is no access to the internal architectures of the agents. However, the representation of the agents' actions and their effects is not as expressive as those of enterprise modelling, where the Situation Calculus was used, and commitment protocols, where the Event Calculus and the $C+$ language were used.

To the best of our knowledge, a precise equivalence between (fragments of) the formalism used for verification of 'norm consistency' with that used for supporting run-time activities has not been established. Consequently, it is not possible to, say, verify a system specification for 'norm consistency' and then translate that specification to an equivalent for the provision of run-time services.

The verification module for 'norm consistency' operates under the assumption that the domain of each variable is finite and known from the outset, while the

complexity is exponential to the number of variables and their possible values. Consequently, like the Event Calculus planner and the Causal Calculator that were employed in the context of commitment protocols, it may not be possible to use this module in large systems.

6 Discussion

We reviewed four approaches for the specification of multi-agent systems exhibiting the features open systems; no access to the code of the agents, no guarantee of benevolent behaviour, and no possibility of predicting the agent interactions. Most reviewed approaches focused on the representation of the permissions and obligations (or commitments) of the agents. These normative relations may be adequate for expressing the system rules when a system can be mainly described in terms of brute facts. When this is not the case, however, it is necessary to explicitly represent the concept of institutional (or institutionalised) power in order to express the circumstances in which an agent may create a set of institutional facts. Jones and Sergot state the following with respect to this issue:

> "[W]e need to make it explicit at the outset that 'empowering' is not an *exclusively* legal phenomenon, but is a standard feature of any norm-governed organisation where selected agents are assigned specific roles (in which they are empowered to conduct the business of that organisation). Thus although it is perhaps legal examples, [..], which come most immediately to mind, it is clear that illustrations of essentially the same sort of phenomenon also occur frequently in other than legal contexts: the Chief Librarian is empowered to waive lending restrictions; the Head of Department alone is empowered to assign duties to members of the department; [..]. An adequate account of the mechanisms by means of which an organisation conducts its affairs will have to incorporate, one way or another, the phenomenon of *institutionalised power*, as we shall here choose to call it." [27, p.430]

The work on enterprise modelling included a representation of institutional power, although the distinction between institutional power and permission was not always clear. The other reviewed approaches did not explicitly represent institutional power.

Although all reviewed approaches specified rules regulating the behaviour of agents (in terms of permissions, obligations, commitments), only the work on Electronic Institutions developed mechanisms for dealing with rule violation.

All reviewed approaches specified rules with a formal, declarative and verifiable semantics. They all employed different formalisms to express the rules, though, thus the expressiveness of each formalisation varied. Details about the relationship between some of the employed formalisms for expressing the rules of a system can be found in the papers cited in earlier sections.

The support for computational tasks regarding the system specification, offering design-time and run-time services, is crucial for the realisation of open systems. All reviewed approaches apart from the work on artificial social systems

included software implementations executing the system specifications with the aims of proving properties of the specifications, checking whether agents comply with the rules, or informing the decision-making of the agents by computing plans that lead to a desired state, or by computing their current permissions, obligations, and so on. There was no demonstration of an implementation framework, however, providing all of the above services at run-time in a large multi-agent system.

References

1. OWL Web Ontology Language: Overview. W3C Recommendation (2004), http://www.w3.org/TR/owl-features/
2. Artikis, A., Sergot, M., Pitt, J.: Specifying norm-governed computational societies. ACM Transactions on Computational Logic 10(1) (2009) (retrieved July 6, 2008), http://www.acm.org/pubs/tocl/accepted/304artikis.pdf
3. Bandara, A.: A Formal Approach to Analysis and Refinement of Policies. Ph.D thesis. Imperial College London (2005)
4. Boella, G., van der Torre, L.: Security policies for sharing knowledge in virtual communities. IEEE Transactions on Systems, Man and Cybernetics 36(3), 439–450 (2006)
5. Boella, G., van der Torre, L.W.N.: The ontological properties of social roles in multi-agent systems: Definitional dependence, powers and roles playing roles. Artificial Intelligence and Law 15(3), 201–221 (2007)
6. Bradshaw, J., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M., Acquisti, A., Benyo, B., Breedy, M., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M., Van Hoof, R.: Representation and reasoning about DAML-based policy and domain services in KAoS. In: Rosenschein, J., Sandholm, T., Wooldridge, M., Yoko, M. (eds.) Proceedings of Conference on Autonomous Agents and Multi Agent Systems (AAMAS), pp. 835–842. ACM Press, New York (2003)
7. Chopra, A., Singh, M.: Contextualizing commitment protocols. In: Proceedings of Conference on Autonous Agents and Multi-Agent Systems (AAMAS), pp. 1345–1352. ACM, New York (2006)
8. Dastani, M., Dignum, V., Dignum, F.: Role-assignment in open agent societies. In: Proceedings of Conference on Autonomous Agents and Multi-Agent Systems, pp. 489–496. ACM, New York (2003)
9. Desai, N., Mallya, A., Chopra, A., Singh, M.: Interaction protocols as design abstractions for business processes. IEEE Transactions on Software Engineering (TSE) 31(12), 1015–1027 (2005)
10. Desai, N., Mallya, A., Chopra, K., Singh, M.: OWL-P: A methodology for business process modeling and enactment. In: Kolp, M., Bresciani, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2005. LNCS, vol. 3529, pp. 79–94. Springer, Heidelberg (2006)
11. Desai, N., Singh, M.: A modular action description language for protocol composition. In: Proceedings of Conference on Artificial Intelligence, AAAI (2007)
12. Desai, N., Singh, M.: Checking correctness of business contracts via commitments. In: Proccedings of International Conference on Autonomous Agents and Multi-Agent Systems (2008)
13. Esteva, M., de la Cruz, D., Sierra, C.: Islander: an electronic institutions editor. In: Castelfranchi, C., Johnson, L. (eds.) Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 1045–1052. ACM Press, New York (2002)

14. Esteva, M., Padget, J., Sierra, C.: Formalizing a language for institutions and norms. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS, vol. 2333, pp. 348–366. Springer, Heidelberg (2002)
15. Esteva, M., Rodríguez-Aguilar, J., Sierra, C., Vasconcelos, W.: Verifying norm consistency in electronic institutions. In: Proceedings of the AAAI 2004 Workshop on Agent Organizations: Theory and Practice, pp. 8–14 (2004)
16. Fitoussi, D., Tennenholtz, M.: Minimal social laws. In: Proceedings of Conference on Artificial Intelligence (AAAI) and Innovative Applications of Artificial Intelligence (IAAI), pp. 26–31. AAAI Press/The MIT Press, Menlo Park (1998)
17. Fitoussi, D., Tennenholtz, M.: Choosing social laws for multi-agent systems: minimality and simplicity. Artificial Intelligence 119(1-2), 61–101 (2000)
18. Fornara, N., Colombetti, M.: Formal specification of artificial institutions using the event calculus. In: Multi-Agent Systems: Semantics and Dynamics of Organizational Models. IGI Global (2008) (to appear)
19. Fox, M., Barbuceanu, M., Grüninger, M., Lin, J.: An organizational ontology for enterprise modeling. In: Prietula, M., Carley, K., Gasser, L. (eds.) Simulating Organizations: Computational Models for Institutions and Groups, pp. 131–152. AAAI Press/The MIT Press (1998)
20. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.: Implementing norms in electronic institutions. In: Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 667–673. ACM Press, New York (2005)
21. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artificial Intelligence 153(1–2), 49–104 (2004)
22. Grossi, D., Dignum, F., Meyer, J.-J.C.: A formal road from institutional norms to organizational structures. In: Durfee, E., Yokoo, M., Huhns, M., Shehory, O. (eds.) Proceedings of Conference on Autonomous Agents and Multi-Agent Systems, pp. 616–623 (2007)
23. Grüninger, M., Fox, M.: The role of competency questions in enterprise engineering. In: Proceedings of the IFIP WG5.7 Workshop on Benchmarking-Theory and Practice (1994)
24. Hewitt, C.: Open information systems semantics for distributed artificial intelligence. Artificial Intelligence 47, 79–106 (1991)
25. Horrrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and ruleML. W3C Submission (2004), http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/
26. Jones, A., Sergot, M.: On the characterisation of law and computer systems: the normative systems perspective. In: Deontic Logic in Computer Science: Normative System Specification, pp. 275–307. J. Wiley and Sons, Chichester (1993)
27. Jones, A., Sergot, M.: A formal characterisation of institutionalised power. Journal of the IGPL 4(3), 429–445 (1996)
28. Kagal, L., Finin, T.: Modeling communicative behavior using permissions and obligations. Journal of Autonomous Agents and Multi-Agent Systems 14(2), 187–206 (2006)
29. Lomuscio, A., Sergot, M.: Deontic interpreted systems. Studia Logica 75(1), 63–92 (2003)
30. Makinson, D.: On the formal representation of rights relations. Journal of Philosophical Logic 15, 403–425 (1986)
31. McGinnis, J., Miller, T.: Amongst first-class protocols. In: Artikis, A., O'Hare, G.M.P., Stathis, K., Vouros, G. (eds.) ESAW 2007. LNCS, vol. 4995. Springer, Heidelberg (2008)

32. Minsky, N., Ungureanu, V.: Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. ACM Transactions on Software Engineering and Methodology (TOSEM) 9(3), 273–305 (2000)
33. Moses, Y., Tennenholtz, M.: On computational aspects of artificial social systems. In: Proceedings of Workshop on Distributed Artificial Intelligence (DAI), pp. 267–284 (1992)
34. Moses, Y., Tennenholtz, M.: Artificial social systems. Computers and Artificial Intelligence 14(6), 533–562 (1995)
35. Mueller, E.: Commonsense Reasoning. Morgan Kaufmann, San Francisco (2006)
36. Pinto, J., Reiter, R.: Temporal reasoning in logic programming: a case for the situation calculus. In: Warren, D. (ed.) Proceedings of Conference on Logic Programming, pp. 203–221. MIT Press, Cambridge (1993)
37. Rodriguez-Aguilar, J., Martin, F., Noriega, P., Garcia, P., Sierra, C.: Towards a test-bed for trading agents in electronic auction markets. AI Communications 11(1), 5–19 (1998)
38. Rosenschein, J., Zlotkin, G.: Rules of Encounter: Designing Conventions for Automated Negotiation among Computers. MIT Press, Cambridge (1994)
39. Searle, J.: What is a speech act? In A. Martinich, editor. In: Martinich, A. (ed.) Philosophy of Language, 3rd edn., pp. 130–140. Oxford University Press, Oxford (1996)
40. Serban, C., Chen, Y., Zhang, W., Minsky, M.N.: The concept of decentralised and secure electronic marketplace. Journal of Electronic Commerce Research (2008)
41. Sergot, M.: The representation of law in computer programs: a survey and comparison. In: Bench-Capon, T. (ed.) Knowledge Based Systems and Legal Applications. Academic Press, London (1990)
42. Sergot, M.: A computational theory of normative positions. ACM Transactions on Computational Logic 2(4), 522–581 (2001)
43. Sergot, M., Craven, R.: The deontic component of action language $nC+$. In: Goble, L., Meyer, J.-J.C. (eds.) DEON 2006. LNCS (LNAI), vol. 4048, pp. 222–237. Springer, Heidelberg (2006)
44. Shanahan, M.: The event calculus explained. In: Veloso, M.M., Wooldridge, M.J. (eds.) Artificial Intelligence Today. LNCS (LNAI), vol. 1600, pp. 409–430. Springer, Heidelberg (1999)
45. Shanahan, M.: An abductive event calculus planner. Journal of Logic Programming 44, 207–239 (2000)
46. Shoham, Y., Tennenholtz, M.: Emergent conventions in multi-agent systems. In: Proceedings of Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 225–231 (1992)
47. Shoham, Y., Tennenholtz, M.: On the synthesis of useful social laws for artificial agent societies. In: Swartout, W. (ed.) Proceedings of Conference on Artificial Intelligence (AAAI), pp. 276–281. The AAAI Press/ The MIT Press (1992)
48. Shoham, Y., Tennenholtz, M.: On social laws for artificial agent societies: off-line design. Artificial Intelligence 73(1-2), 231–252 (1995)
49. Shoham, Y., Tennenholtz, M.: On the emergence of social conventions: modeling, analysis and simulations. Artificial Intelligence 94(1-2), 139–166 (1997)
50. Singh, M.: Agent communication languages: rethinking the principles. IEEE Computer 31(12), 40–47 (1998)
51. Singh, M.: An ontology for commitments in multiagent systems: towards a unification of normative concepts. Artificial Intelligence and Law 7(1), 97–113 (1999)

52. Singh, M.: A social semantics for agent communication languages. In: Dignum, F., Greaves, M. (eds.) Issues in Agent Communication. LNCS, vol. 1916, pp. 31–45. Springer, Heidelberg (2000)
53. Tennenholtz, M.: On computational social laws for dynamic non-homogeneous social structures. Journal of Experimental and Theoretical Artificial Intelligence 7, 379–390 (1995)
54. Uszok, A., Bradshaw, J., Lott, J., Breedy, M., Bunch, L., Feltovich, P., Johnson, M., Jung, H.: New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS. In: Proceedings of Workshop on Policies for Distributed Systems and Networks, pp. 145–152. IEEE Computer Society, Los Alamitos (2008)
55. van Eijk, R., de Boer, F., van der Hoek, W., Meyer, J.-J.: Open multi-agent systems: agent communication and integration. In: Jennings, N., Lesperance, Y. (eds.) ATAL 1999. LNCS, vol. 1757, pp. 218–232. Springer, Heidelberg (2000)
56. Venkatraman, M., Singh, M.: Verifying compliance with commitment protocols. Journal of Autonomous Agents and Multi-Agent Systems 2(3), 217–236 (1999)
57. Yolum, P., Singh, M.: Commitment machines. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, pp. 235–247. Springer, Heidelberg (2002)
58. Yolum, P., Singh, M.: Flexible protocol specification and execution: applying event calculus planning using commitments. In: Castelfranchi, C., Johnson, L. (eds.) Proceedings of Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 527–535. ACM Press, New York (2002)
59. Yolum, P., Singh, M.: Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. Annals of Mathematics and Artificial Intelligence 42(1-3), 227–253 (2004)
60. Zambonelli, F., Jennings, N., Wooldridge, M.: Organisational rules as an abstraction for the analysis and design of multi-agent systems. International Journal of Software Engineering and Knowledge Engineering 11(3), 303–328 (2001)

From Multi-Agent to Multi-Organization Systems: Utilizing Middleware Approaches

Matthias Wester-Ebbinghaus, Daniel Moldt, and Michael Köhler-Bußmeier

University of Hamburg, Department of Informatics
Vogt-Kölln-Straße 30, D–22527 Hamburg
{wester,moldt,koehler}@informatik.uni-hamburg.de

Abstract. Modern software systems share with social organizations the attributes of being large-scale, distributed and heterogeneous systems of systems. The organizational metaphor for software engineering has particularly been adopted in the field of multi-agent systems but not entirely exploited due to an inherent lack of collective levels of action. We propagate a shift from multi-agent to multi-organization systems that we rest upon an organization theoretically inspired reference architecture. We further suggest to utilize agent-oriented technology as a means for realization. We draw upon the wide variety of organizational modelling and middleware approaches and establish a best fit between different approaches and requirements for different architectural levels.

1 Introduction

The characterization of modern software systems as *application landscapes* [1], *ultra-large-scale (ULS) systems* [2] or *software cities* [3] carries the comprehension as large-scale, inherently distributed and heterogeneous *systems of systems*. Such characteristics are typically not intended but evolve out of necessity or even "at random". The various parts of the overall system are embedded in complex and dynamic environments, where they co-evolve with various other systems (software systems, other technical systems, social systems) and consequently have to be appropriately versatile themselves. These co-evolutionary forces account for continual growth and rising complexity even considering formerly small and simple software systems. In this respect, the challenges in dealing with modern software systems resemble the challenges that organizations in social societies face. Despite being inherently complex, distributed and heterogeneous themselves and despite existing in potentially highly complex and dynamic environments, organizations have to draw and manage their boundaries in a way that properly supports their goals and purposes. According to Hannan and Carrol [4], the main capacities of organizations in comparison to other social collectivities are precisely their *durability, reliability* and *accountability*.[1]

Thus, it is not surprising that multiple software engineering approaches have been brought forth in recent years that take an organization-oriented perspective

[1] It will not be discussed in this paper that each of these capacities is a double-edged sword.

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 46–65, 2009.

on software systems. Many of these approaches are rooted within the multi-agent system (MAS) paradigm of software engineering as the underlying metaphors are already socially inspired and thus provide an ideal breeding context for organization-oriented ideas. A thorough overview of recent and current work in this field can be found in [5].

While MAS research has made significant contributions to establishing an organizational metaphor for software engineering, when relating these contributions to organization theory it becomes obvious that the true potential of the organizational metaphor is not entirely exploited. Organization theories are frequently classified along the dimension of *analysis level*. On the on hand, organizations can be regarded as systems composed of individuals for whom they provide technical and social contexts. This conception is what MAS research has focussed on so far. On the other hand, Scott [6] points out that one cannot comprehend the importance of organizations in human societies if they are only regarded as contexts for individual actors. Instead, organizations are collective actors in themselves. They carry out actions, utilize resources, enter contracts and own property. Indeed, they are the primary social actors of today's society. With respect to ever larger and more complex software systems, it becomes obvious that this conception of organizations as collective actors is equally vital for software engineering. However, it has only been taken into account tentatively by MAS research.

One might argue that this was just a matter of extrapolating the concepts for the individual level (agents as actors) to the collective level (organizations as actors). Approaches for *agentifying* multi-agent systems [7] and holonic multi-agent systems [8] pave the way for this advancement. We realize these possibilities but consider them mainly as *technical* means to realize collective agency. To arrive at a *conceptual* basis, we insist that the transition from individual to collective agency should more heavily rely on results that organization theory as a discipline with a long tradition in investigating organizations, their internals, and their environments already has to offer. The rationale behind this is to learn from the adaptivity, robustness, scalability and reflexiveness of social (multi-)organization systems and to translate their building principles in effective information technology.

Consequently, the aim of this paper is the provision of a software development proposal that builds upon and extends the MAS approach in order to account for the true potential of the organizational metaphor. Central to our proposal is the advancement from the individual agent to the organization as a software engineering metaphor of higher granularity. In [9] Ferber advances the distinction between ACMAS (agent-centred multi-agent systems) and OCMAS (organization-centred multi-agent systems). We consider our approach as one further step in this shift of paradigm and term the systems introduced by our approach MOS (multi-organization systems). We do not claim to supplant MAS philosophy by introducing an entirely new paradigm. The approach we present completely rests on an underpinning of MAS. But we do claim to take an entirely fresh look at the game of organization-oriented software architectures.

In Section 2 we present the ORGAN-model (**Org**anizational **A**rchitecture **N**ets) as a "thinking model" to comprehend systems of systems under a multi-organization perspective. We use real-world inspirations to derive guidelines for building software systems accordingly. In Section 3 we compare three distinct organizational models for MAS that rely on middleware approaches for deployment. We investigate the prospects of the middleware philosophy with respect to the engineering of large-scale systems following ORGAN. We conclude our results in Section 4 and provide an outlook on open issues and future work.

2 Engineering Systems of Systems: The ORGAN-Model

In this section, we consider characteristics and problems of large-scale software systems and present the ORGAN-model as an organization theoretically inspired comprehension model.

2.1 Software Systems in the Large

The ever growing size of modern software systems marks the basis for the request for new software engineering perspectives and paradigms in [2]. However, it is necessary to go beyond the concept of size alone in order to get a grasp of the nature and effects of software systems of the category ULS system [2]. Their parts are most often derived from structures and processes of real-world systems. These include physical and mechanical systems but above all social systems.[2] Social systems of a certain size are inherently distributed and decentralized. This condition accounts for the fact that the accompanying software systems are not monolithic (which might also yield an enormous size) but rather *systems of systems* as characterized in [10]. The several system parts are typically not only geographically distributed but also independently acquired, deployed and useful. The overall system is not intentionally formed and developed but evolves gradually and exhibits a considerable degree of emergent behaviour.

In [2] an analysis of the problems resulting from these characteristics is carried out. The inherent geographical distribution in combination with managerial and operational independence inevitably lead to decentralization. Besides data management and operation this affects the evolution of the overall system, its control and observability. It is not possible to stop the system or to take out uniform rollouts and release changes. Instead, the functional range of the system is extended, adapted or restricted simultaneously to its operation. Thus, the evolution of the overall system is continuous rather than following well-defined phases. Consequently, a perspective is taken on that differs from the traditional perspective on software engineering relying on a top-down design for the realization of one or a few applications. Governance mechanisms that assume a comprehensive knowledge of system-wide parameters, the possibility of solving

[2] However, the topic of software systems as a dimension of *socio-technical systems* and its specific implications will not be addressed thoroughly in this paper. We just assume that it is a continuous source of reciprocal adaptions.

conflicts uniformly, and the effective adoption of a central authority become mostly obsolete. Instead, one has to deal with decentralized control, the impossibility of global observability and assessment and difficult estimation of the effects of perturbations.

Organization-oriented MAS engineering addresses many of the just mentioned problems. In particular, they deal with autonomous and potentially heterogeneous system parts subsumed under joint superstructures. In Section 3 we will elaborate on some related concepts. Nonetheless, one drawback of MAS approaches to systems of systems engineering becomes obvious. The sheer size of the software systems addressed in this paper entails the necessity to distinguish different levels of abstraction. Degree of abstraction relates to the granularity of the system units that are studied at each respective level. The core metaphor of MAS research is the individual agent, which is of rather small granularity. As mentioned in the introduction, one common distinction regarding organization theoretical studies is whether an individual or a collective level of analysis is chosen. Collective level issues like the integration of multiple agents into a joint system and the governance of this system in a way that allows corporate agency of the system at higher levels of abstraction are not inherently rooted in the agent paradigm. Like stated in the introduction, recent comprehensions of MAS put a stronger focus on the system/organization level and some approaches even address the topic of multiple agents acting "as a whole" and thus exhibiting corporate agency. These efforts narrow the gap between the agent paradigm and the demand for collective level perspectives. However, there remains the risk of a mismatch between the applied core metaphor and the required conceptual level, at which application problems have to be addressed.

For this reason, we advocate a temporary departure from the agent paradigm at this point. Instead, we introduce the ORGAN reference model for comprehending systems of systems under an organization-oriented perspective.

2.2 Universal Model of an Open, Controlled System Unit

The underpinning of ORGAN is the universal model of an open and controlled system unit from Figure 1 that is applied at *all* system levels.[3] Three types of internal system units and system processes are distinguished respectively. Internal system units are categorized as *operational units, integration units* and *governance units*. It is important to note that operation, integration and governance are in the first place *analytical aspects* of system units. However, they are carried and backed up by certain (individual or collective) actors, hence the classification of internal system units. Although analytically distinct, the three aspects are intrinsically interwoven

[3] This model has an underlying *reference net* semantics [11]. Reference nets carry some extensions compared to ordinary coloured Petri nets. They implement the nets-within-nets concept where a surrounding net can have references to other nets as tokens. Synchronous channels allow for bi-directional synchronous communication between net instances. In [12], we show how the approach of modelling system units with reference nets allows for straightforward prototyping and simulation.

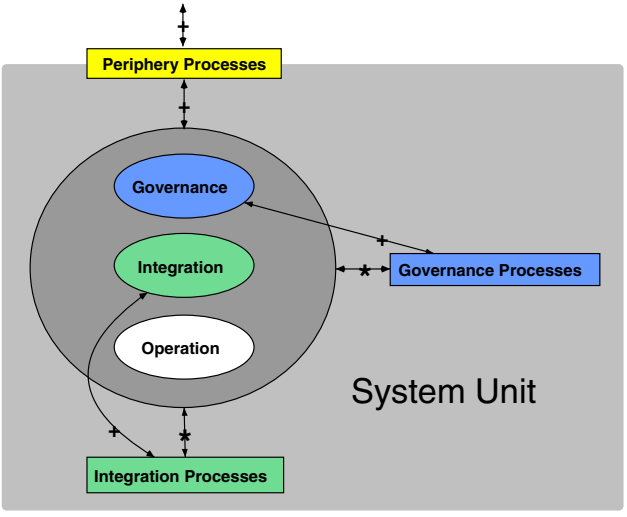


Fig. 1. Universal model of an open, controlled system unit

and interdependent. The three sets of internal system units do not even have to be disjoint. Instead, particular system units might fulfil multiple analytical roles.

Integration units together with operational units represent the "here and now" of the system unit in focus. The operational units are so to say the intrinsic units and carry out the system's primary activities. They are dependent on the integration units which offer a *technical frame* via intermediary, regulation, and optimisation services in the course of integration processes. The governance units represent the "there and then" of the system unit in focus. They offer a *strategic frame* via goal/strategy setting, boundary management, and transmitting their decisions to the other internal system units in the course of governance processes.

Integration and governance units embody a two-level control structure. Integration attenuates oscillation between governance and operation of the system and thus has a "calming effect" on the system as a whole. Governance units are responsible for defining and adapting system rules while integration units are responsible for applying these very rules to the system's operation. Goals, strategies and performance standards are transmitted to the operational units via the reroute of the integration units. These in turn typically carry out a refinement or concretion of the strategic guidelines in the form of subgoals, operation plans or schedules before these are filled with life by the operational units.

Each system unit is a *Janus-faced* entity. It embeds system units as internal system units and is itself embedded in other system units as one of their internal system units. Thus, besides the already mentioned internal frames (technical and strategic) each system unit in focus is *externally framed* by surrounding system units to which the system unit in focus (by means of its internal system units) relates via periphery processes.

To conclude, we take a recursive, self-similar nesting approach, borrowed from Koestler's concept of a *holon* [13] that we extend with a generic reference model for control structures at each level. We arrive at a modular approach to comprehend systems of systems. Each system part may be regarded under a platform perspective and under, where it offers technical and strategic guidelines for its inhabitants. Furthermore the same part may be regarded under a corporate agency perspective, where it collectively acts as a holistic entity in the context of a higher-level system part. This provides a conceptual basis to systematically study and implement different modes of coupling, both horizontally and vertically.

2.3 Reference Architecture

The universal model of system unit from Figure 1 itself is too abstract and generic to provide a meaningful guideline for engineering large-scale software systems. With respect to software architectures, a selective distinction of different system levels has to be carried out. Here, we advance the ORGAN reference architecture for multi-organization systems.

Overview. The core unit now is the organization instead of the agent. This directly leads to three mandatory architectural levels, the organization itself, its internals and its environment. As an organization might have different conceptual environments (e.g. different domains within which it operates) and the architecture targets at the inclusion of multiple organizations we have to deal with multiple environments. Thus, the need for a fourth architectural level that acts as a system closure arises.

This distinction of architectural levels resulting from rather technical considerations is confirmed by analysis levels that organizational theories target at according to Scott's classification [6]. At the *socio-psychological level* the internals of an organization in terms of relationships between its individual members are studied. The *organization structure level* introduces the perspective on an organization as a holistic, identifiable entity. It studies structures and processes that characterize an organization in terms of its parts (divisions, work groups, project teams) and its analytical components (specialization, communication network, hierarchy). The *ecological level* focuses on characteristics and actions of organizations as corporate actors operating in an even wider network of relations. For this level, further distinctions are possible. Among them, the level of an *organizational field* bears the most comprehensive concept of an organization's environment. It consists of organizations that, in the aggregate, constitute a recognized area of institutional life (key suppliers, consumers, regulatory agencies, etc.). Finally, the *society* integrates all fields under a common body of law and uses the fields as mediators for decrees that actually target at individual organizations.

Consequently, we distinguish four types of system levels for organization-oriented architectures and denominate them as departments, organizations,

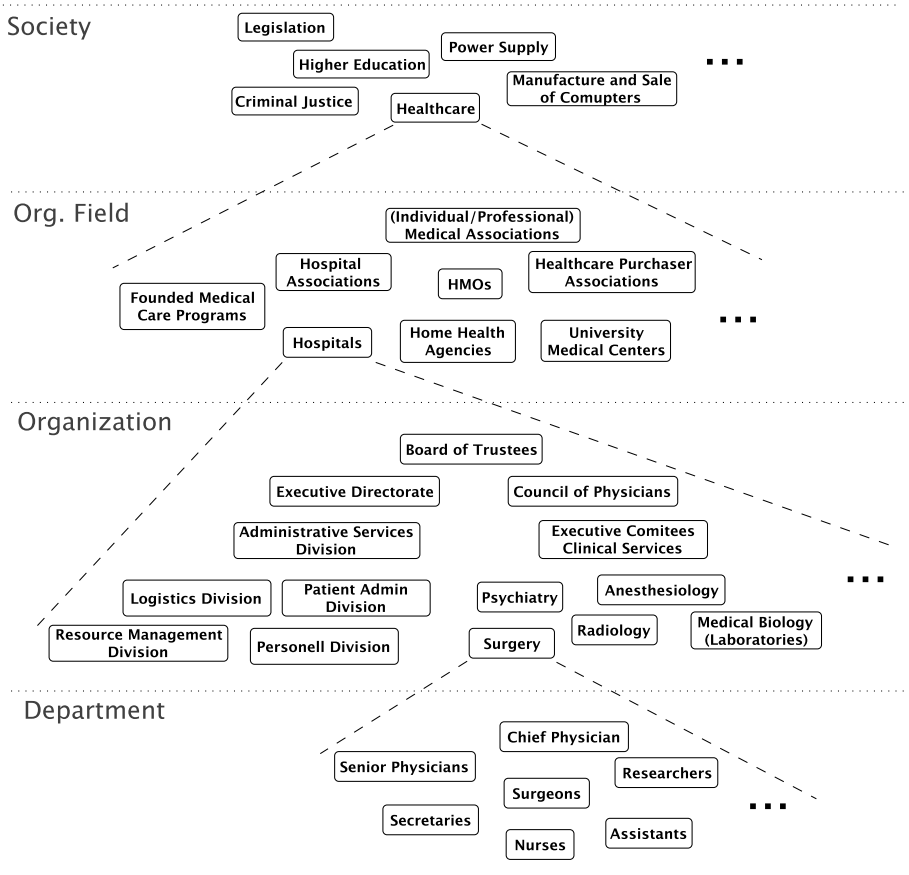


Fig. 2. Actors on different levels of action in a healthcare setting

organizational fields and the society. In order to exemplify, what types of actors operate at each level in a real-world setting, we have assembled the illustration in Figure 2 for the particular case of healthcare from several sources [6,14,15].

For the remainder of the paper we want to regard departments, organizations, organizational fields and the society as particular incarnations of the universal model of system units from Figure 1 and specifically term them *organizational units*. Figure 3 shows an illustration of this approach. In taking on this perspective, we study for each of the four types of organizational units what characteristics operation, integration and governance take on respectively (cf. [16] for further details). The overarching purpose is to investigate what implications this analysis yields for each type of organizational unit under a *software engineering perspective*. We want to let real-world inspirations guide our approach to build organization-oriented software systems accordingly. We consider the architectural role of each type of organizational unit and investigate further implications like network topology or degree of

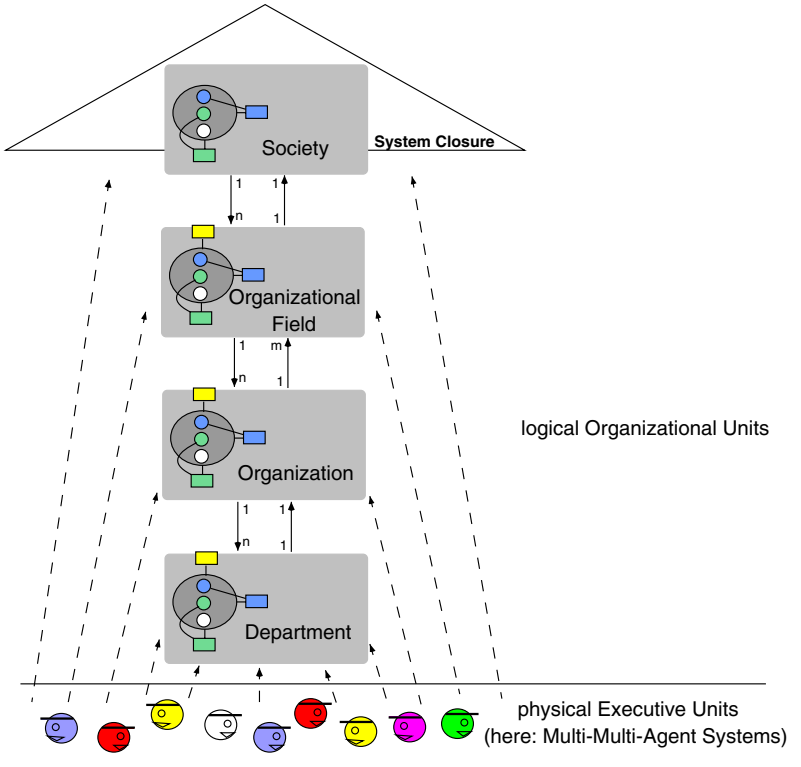


Fig. 3. ORGAN reference architecture

autonomy and uniformity of embedded units.[4] Of course, none of the implications that we propose here can be taken as mandatory. The possible variety for each type of organizational unit is simply too vast. However, we want to define some *archetypical* characteristics for each type.

Society. There exists one organizational unit of the type society (but see the conclusion). The society represents the upper closure of the system. Each software system serves a certain purpose (or certain purposes). This is mirrored by the architectural role of the society. The society supports a specific extract of reality from which the purpose of the overall system is derived. The *operational* aspect is embodied by different scopes of the overall system (e.g. healthcare, power supply, e-commerce). For each system scope the society embeds one organizational field. These fields also make up the *integrational* aspect by

[4] To be precise, we would have to distinguish between operational, integration and governance units when studying autonomy and uniformity. Governance units are clearly more unique and exhibit a higher degree of autonomy than integration and especially operational units. Furthermore, they are one of the main sources for restricting autonomy of and causing uniformity between other units. In our investigation we mainly look at autonomy of and uniformity between operational units at each level.

managing transitions and interactions between each other. *Governance* is taken care of by specific fields that constitute the society's government. It specifies system laws, some of which have a system-wide validation and some of which only hold for certain fields. For this purpose the society's government guards over these laws and utilizes governance structures at the field level as caretakers at those system levels where the laws actually have impact.

Consequently, in a software system we expect the organizational unit of society to be a loosely coupled network of fields with the only archetypical characteristic of being somewhat centralized by means of the fields that constitute the government. Organizational fields as embedded units have a considerable degree of autonomy. They have to obey to societal laws but these are quite general in nature, the more detailed and elaborated legal and regulatory bodies develop at the field level itself. In this respect, fields are very different from each other and share a low degree of uniformity as each field embodies a distinct scope of the overall system.

Organizational Field. Each organizational field provides a consistent and largely self-contained picture of a particular part of the overall system ("living space"). Fields represent the immediate environments for organizations. Participating organizations as the *operational* part of a specific field have some functional interest in common and face each other as discrete and co-equal entities (cooperating or competing out of self-interest). However, they are embedded in the common environmental frame of the field. Here we can distinguish the material-resource features as an *integrational* aspect from the institutional features with both *integrational* and *governance* aspects. Material-resource features capture factors directly related to the demand, supply and exchange of products and services on the field. Thus, they are to a great part inherently constituted by the participating organizations themselves, but also encompass organizations that enable and mediate organizational interchange in the first place or offer auxiliary services (e.g. industry associations, market organizers, assurance companies, banks, consultant offices). Material-resource features always rest on institutional foundations that are made of practices and symbolic constructions constituting the organizing principles of the field. Thus, we find organizations for setting (e.g. regulatory agencies, professional associations) and implementing (e.g. market oversight or facilities for licensing and certification) the institutional logics of a field. These institutional logics are available for participants to elaborate, which fosters a more efficient design of organizations.

In a software system, we expect an organizational field unit to be a network of organizations that is broadly organized in three layers that quite canonically correspond to the analytic distinction between operation, integration and governance. We find several sets of organizations that share a great degree of uniformity (organizational *populations*) because of operating under very similar and specific field-wide circumstances. Organizations have a very high degree of autonomy. Besides being able to exist on multiple fields and thus dimnishing the dependence from each single field, an organization buffers its internals from direct field access. Fields can only exert indirect control on their embedded organizations by means of sanctions and exclusion from field processes.

Organization. Just like agents in agent-oriented software engineering, organizations in the context of the ORGAN-architecture are the central modelling units that determine, at which level of abstraction the whole system is regarded. Organizations operate on one or more organizational fields, depending on the variety of domains, to which they relate. Organizations are composed of departments, which mirror the complexity that the organizations face in their environments. Different needs and functions of the organization with regard to their environmental embedding are mapped onto different departments. Contrary to organizational fields that host distinct and co-equal entities, the departments of an organization are dependent on the unique organization they belong to and exist on their behalf. They are fused into a joint superstructure under a joint strategy to pursue and achieve joint goals. Many departments fulfil an *operational* as well as an *integrational* role. One particular department might act as an integration unit for departments that it groups according to the organization's superstructure. At the same time, it might act as an operational part towards another department to whose grouping it belongs. The superstructure is typically hierarchical in nature but may be augmented by various vertical ties. Finally, the *governance* aspect is taken care of by departments that constitute the dominant coalition (including at least high-ranked managers, but potentially encompassing further actors from within and even outside of the organization) of the organization. It sets the organization's goals and strategies.

In a software system, we expect an organization to be a department network that is basically hierarchic in nature with many departments fulfilling multiple superstructural roles. Of course, such vertical hierarchies are typically augmented by various horizontal ties and linkages up to the point of fully-fledged matrix organizations. Nevertheless, the main purpose of an organization's superstructure is the successive grouping of its departments (grouping by multiple criteria like authority, function, market, region, etc.). Departments exhibit a low degree of autonomy and uniformity. They fully depend on and exist on behalf of their organization and fulfil specific functions (uniformity may come into being for the sake of economies of scale however).

Department. The departments are the actuators of the overall system. They represent the final implementation means for all higher-level system activities. Each department exists on behalf of an organization and this organization determines the department's characteristics concerning structure and processes. Here, a continuum of possibilities opens up, from rather *bureaucratic* structures (high degree of standardization) to rather *organic* structures (low degree of standardization). Each department is *governed* by its management and its *operational* members are *integrated* by their respective positions into the department's context.[5]

[5] Having an immaterial concept like an organizational position as an integration unit at the department level may seem odd at first glance. However, formalizing position characteristics for social organizations is a first step in reifying positions. For software systems, this may be carried even further as will be shown in Section 3 where we consider different middleware approaches for organization-oriented MAS engineering.

In a software system, we expect a department unit to consist of a strongly coupled network of individual actors (agents) that is highly intermeshed. The management causes centralization to some degree, but most connections and relationships are inseparably tied to work processes where the distinction between the analytic aspects of operation, integration and governance shifts from scenario to scenario. Uniformity between actors highly depends on the specific department. Autonomy is very much restricted by position characteristics and thus mostly depends on whether the department is rather organic or bureaucratic in nature.

To conclude, the ORGAN-model for organization-oriented software architectures provides a conceptual thinking model for large-scale software systems. Contrary to an agent-oriented perspective, collective levels of action are inherent to a truly organization-oriented perspective. Different modes of vertical as well as horizontal coupling result. However, when it comes to actually realizing multi-organization systems according to the ORGAN-model, we are of the opinion that current MAS technology provides an ideal starting point. This opinion is illustrated in Figure 3 where the organizational units are considered as *logical* constructs that have to be incarnated by means of *physical* executive units, which is accompanied by a *change in paradigm*. In the following Section 3 we elaborate on our proposal to utilize and combine MAS (middleware) approaches in order to realize multi-organization systems.

3 Utilizing Multi-Agent Middleware Approaches

Organization-oriented approaches to multi-agent system design employ the mechanism of *formalization* borrowed from social organizations. Formalization in this context refers to the extent, to which expectations on behaviour are explicitly and precisely specified, and to the extent, to which these specifications are independent from the particular occupants of social positions [6]. In this respect, rationality resides in the social structure itself, not in the individual participants. Adopting this principle for multi-agent system engineering allows for separation of concerns. Organizational specifications and the agents that fill these specifications with life may be designed separately. The aim is to combine local agent autonomy with the assurance of global system characteristics.

Some approaches carry separation of concerns to the implementation level. Instead of just resting "in the heads" of the participating domain agents, organizational specifications are encapsulated and managed by an explicit middleware and thus software technically *reified*. This allows arbitrarily heterogeneous agents to participate in the organization as the middleware acts as an intermediary. Furthermore, it is extremely useful in open multi-agent environments where agents belonging to different stakeholders continuously enter and leave and the organizational specifications have to be buffered against potentially harmful influences.

The specific characteristics of a given middleware approach depend on the characteristics of the organizational specifications that are to be supported. Here, we examine three distinct approaches. Afterwards, we analyse how the

benefit of separation of concerns can even be extended with respect to not only separating organizations and domain agents but also different system levels of software systems in the large. We specifically consider the levels introduced by the ORGAN-architecture from Section 3.

3.1 Middleware Approaches

To distinguish different modelling approaches we adopt the approach taken in [17] where distinctions are made based on different *organizational dimensions* that are supported. A qualitative comparison of the approaches including their middleware philosophy is delayed until Section 3.3.

$\mathcal{M}$OISE$^+$/$\mathcal{S}$-$\mathcal{M}$OISE$^+$. The $\mathcal{M}$OISE$^+$ modelling language [18] incorporates a structural, functional and deontic dimension. In the *structural dimension*, roles and groups are specified. Roles are related to one another via inheritance relationships. Groups consist of a set of roles where for each role the minimal and maximal cardinality is specified. In addition to inheritance, additional links (compatibility, authority, communication, acquaintance) with either an intra-group or inter-group scope may exists between roles.

The *functional dimension* consists of a set of social schemes. A scheme is a goal decomposition tree where the root is the initial organizational goal the scheme targets at. Groups may be linked to schemes. Roles belonging to the group are then assigned to coherent sets of the scheme's goals (so called missions) via permission or obligation links in the *deontic dimension*.

$\mathcal{S}$-$\mathcal{M}$OISE$^+$ [19] is the middleware for supporting $\mathcal{M}$OISE$^+$ specifications in open multi-agent systems. Figure 4 gives an overview of its architecture. Domain agents connect to the organization via proxies called *OrgBoxes* that offer an API representing the agents' possibilities to act as members of the organization. All requests for changes in the state of the organization (e.g. role adoption, group creation, mission commitment) are routed to a central *OrgManager*. Requests are only fulfilled if no organizational constraints are violated. The OrgManager can also act proactively, for example by informing agents (mediated by their

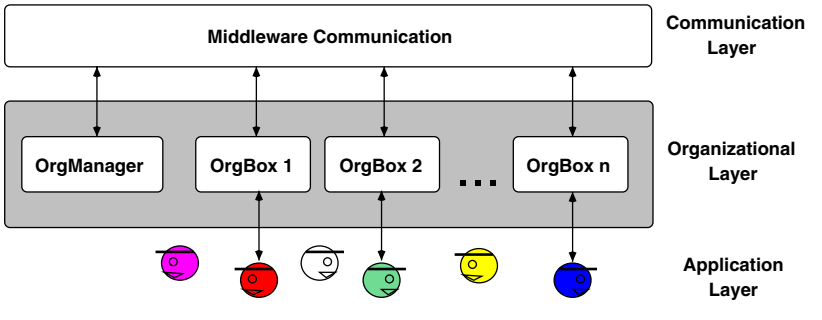


Fig. 4. $\mathcal{S}$-$\mathcal{M}$OISE$^+$ middleware approach (adapted from [19])

OrgBoxes) of missions they are enforced to commit to or goals that have become ready to pursue. Agents have access to the organizational specification and are free to interpret it to optimise their organization-aware reasoning.

ISLANDER/AMELIE. ISLANDER [20] introduces a conceptual shift from organizations to institutions. Instead of being constructive in terms of goal/task trees ($\mathcal{M}$OISE$^+$/SONAR) to achieve certain organizational objectives in a divide-and-conquer style, ISLANDER focuses on the regulative character of institutions by declaratively specifying what agents are permitted or forbidden to do in certain institutional contexts and what the consequences of their actions are.

ISLANDER supports a structural, dialogical/interactional, and a normative dimension. The *structural dimension* differs from the $\mathcal{M}$OISE$^+$ structural dimension in that it does not relate roles and groups but so called scenes. Scenes represent the *interactional dimension*. A scene is basically a collection of roles in interaction with each other following a well-defined interaction protocol. Each scene embodies a largely self-contained collective activity of the overall system. In the structural dimension, relationships among scenes are established by a so called performative structure. It specifies the network between scenes and defines transitions between scenes. Transitions define which agents playing which role under which circumstances can move from one scene to another and whether new instances of scenes have to be brought up upon firing transitions.

An additional *normative dimension* specifies consequences of agent actions. A norm defines, which obligations hold after certain communicative acts have (or have not) been uttered in certain scenes and certain side conditions hold.

AMELIE [21] is the middleware for supporting ISLANDER specifications. Figure 5 gives an overview of its architecture. Just like in $\mathcal{S}$-$\mathcal{M}$OISE$^+$, agents do not interact directly but connect to middleware mediators, in the case of AMELIE these are the *governors*. Contrary to $\mathcal{S}$-$\mathcal{M}$OISE$^+$ there exist different roles in managing the institution. An *institution manager* is in charge of the institution as a whole. I starts the institution, authorises agents to enter and manages the creation of new scene executions. Each scene execution is managed by a separate

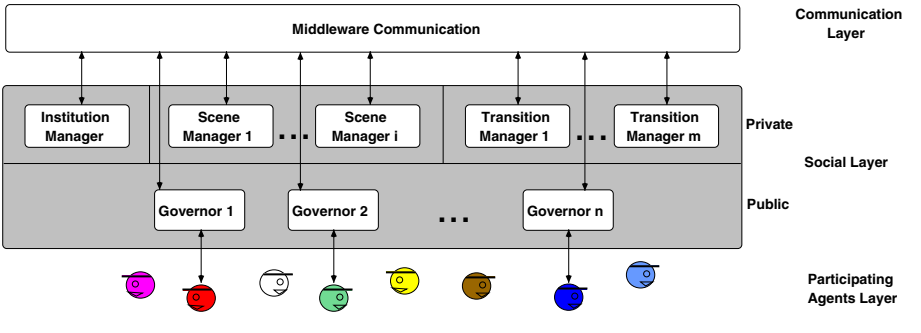


Fig. 5. AMELIE middleware approach (adapted from [21])

scene manager overseeing the execution according to the associated scene protocol. *Transition managers* are in charge of managing agent transitions between scenes. The different management parts communicate with each other and with the governors. Each governor manages multiple conversations with its respective connected agent, one conversation for each scene and transition participation. In addition, it keeps track of the norms that concern its associated agent.

SONAR. SONAR [22] is a mathematical model of multi-agent organizations based on Petri nets. It has a structural, functional, and interactional dimension. However, *structural and functional dimension* are inseparable. The functional part consists of a Petri net, where each place models a task and each transition models the execution of the task associated with the unique place in the transition's preset. Task executions may introduce subtasks, which results in a multi-tree structure with multiple roots associated with core tasks. This functional specification is enriched with structural information by partitioning the Petri net by means of so called organizational positions. Consequently, each position is responsible for the execution of some tasks and possibly permitted to delegate (sub-)tasks to other positions.

The *interactional dimension* comes into being by relating each place with a set of roles and each transition with an interaction protocol. Thus, tasks correspond to the implementation of roles and task executions to the interactions that have to be carried out between these roles. Subtasks correspond to the refinement of roles into subroles whose implementation is further delegated.

The SONAR middleware approach is illustrated in Figure 6. It differs from the two already presented ones. It does introduce a middleware layer, but this layer is not *physically* distinguishable from the application layer. Instead, the

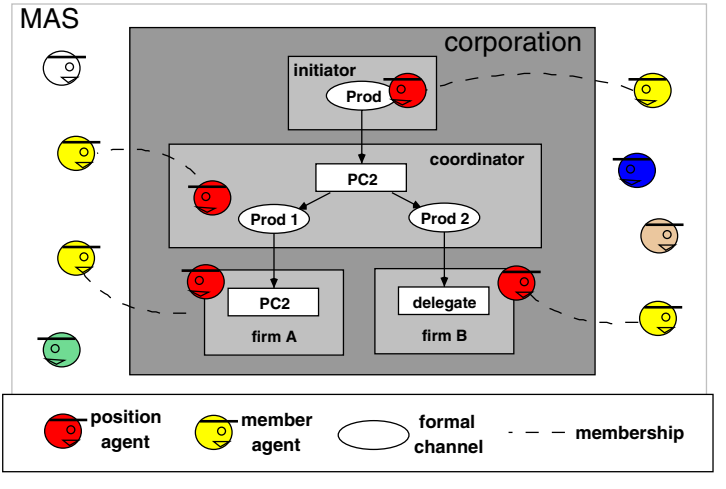


Fig. 6. SONAR middleware approach (adapted from [23])

layer *logically* consists of *position agents*, one for each organizational position. Position agents are launched on behalf of the organization. Domain agents have to connect to them in order to act as members of the organization. Position agents mediate interactions and take care that the organizational specifications are honoured. But contrary to $\mathcal{S}$-$\mathcal{M}$OISE$^+$ and AMELIE, SONAR advocates complete distribution. Each position agent only knows its local context ("upwards" and "downwards" delegation partners, own tasks, own task executions, associated roles and interactions patterns). There is no central facility (or some central facilities) that keeps a global picture and exploits centralized coordination.

3.2 Multiple Layers of Middleware

The benefit of separation of concerns fostered by middleware approaches basically aims at separating domain agents and organizational concerns. However, the "agent neutrality" of middleware layers can be carried forward to separate the engineering of different system levels in the context of large-scale software systems. Figure 7 shows an illustration of this idea. To realize it, both organizational modelling languages and their supporting middleware frameworks would have to incorporate a *Janus-faced* character. In the case of "looking downwards", not much changes. In case of "looking upwards", organizational modelling languages would have to take into consideration processes that are not entirely enclosed by the modelled system in focus. The accompanying middleware implementations need to be able to connect to other middleware layers themselves and to mediate activity not only horizontally but also vertically.

The concepts of a Janus-faced character and periphery processes were also stressed in the context of the ORGAN model in Section 2.2. Thus, we consider the approach from Figure 7 as a promising realization strategy to engineer software

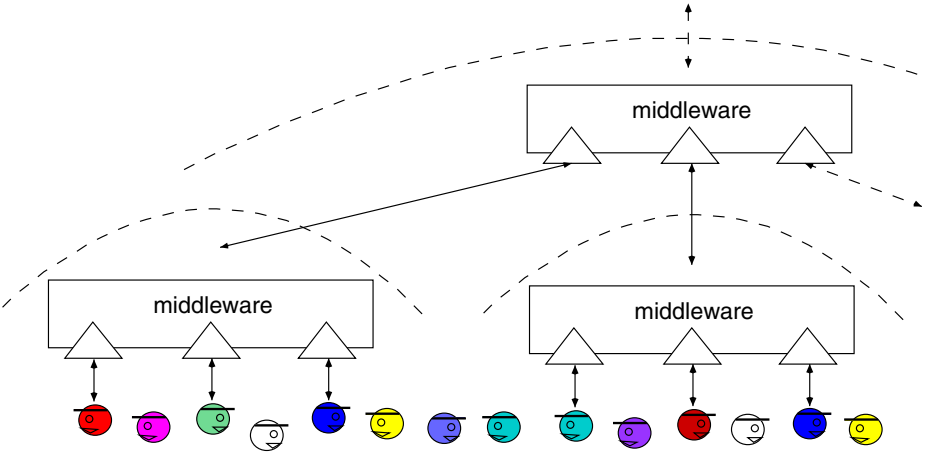


Fig. 7. Iterating middleware layers

systems in the large according to ORGAN. The whole idea of recursive, self-similar nesting is by no means new to multi-agent system research and is for example incorporated by the concept of *holonic multi-agent systems* [8]. In our opinion, nested middleware layers foster such an approach in the context of large-scale systems where there exists a strong need for the modular design, creation, and maintenance of distinct system parts.[6]

3.3 Pooling of Competencies

Like stated in Section 2, different levels of a system call for different modes and paradigms (not only degrees) of coupling and organizing. Here we may avail ourselves of the broad spectrum of already existing organizational/institutional multi-agent approaches supported by middleware. We exemplify this potential by relating the three approaches from Section 3.1 to the different layers of the ORGAN-architecture from Section 2.2. A primary distinction was already mentioned. While $\mathcal{M}$OISE$^+$ and SONAR are rather constructive in terms of achieving certain organizational purposes, ISLANDER is rather declarative in focussing on regulations. For this reason, we consider ISLANDER ideally suited for the level of organizational fields and the other two approaches best suited for the level of departments and organizations.

Department Level. For the department level, a high degree of detail is necessary. Ultimately, departments are both source and sink for all activities. It is not only necessary to model activities but also under which circumstances and conditions they come into being. $\mathcal{M}$OISE$^+$ is very rich when it comes to model relationships between actors in terms of role inheritance, various types of additional links (compatibility, authority, acquaintance, communication) between roles and clustering of roles into groups. For this reason, $\mathcal{M}$OISE$^+$ is ideally suited for the level of departments. Even the lack of an interactional dimension in some way suits the department level. Social schemes might be considered as abstract programs but how these programs are actually followed for particular instances might be open to mutual adjustment (only possible if all participating roles shared mutual communication links).

In addition, the centralized management of the $\mathcal{S}$-$\mathcal{M}$OISE$^+$ middleware fits the department level. Departments exhibit the notion of locality in the sense that the network of interdependencies is quite tight and highly intermeshed. Thus obtaining a global picture of what is going on is very useful. The drawbacks of centralized solutions like risk of bottleneck and high response times should pose no severe problems as number of participants as well as network distances are typically at small or medium scale for departments.

[6] The middleware approaches from Section 3.1 tend to establish anonymity between domain actors. As the explanations from Section 2.2 should have made clear, this conception is not adequate for software systems according to the ORGAN-model where there may exist a quite high degree of penetration and visibility between certain organizational units. It is a matter of designing organizational/institutional processes in an appropriate way to establish the desired degrees of coupling and acquaintance.

Organization Level. SONAR models on the other hand are more abstract and much better suited for the organizational level where not individuals but departments are related to one another. The core unit of abstraction in SONAR is the position rather than the role. Each position is unique like it is typically the case for departments in an organization. The level of detail $\mathcal{M}$OISE$^+$ offers for relating roles to one another (e.g. inheritance, compatibility, cardinality) is not necessary when relating departments instead of individuals. The most frequently adopted view when regarding organizations nowadays is a business process perspective. Entities related to those processes are characterized by abstract service interfaces that define which functionality each respective entity contributes. SONAR exactly offers such a process-centric perspective. Each position is responsible for the execution of several tasks and may delegate (sub-)tasks to other positions. This can be considered as the abstract service description of each position ("offers", "uses") and clearly defines what each position has to contribute to business processes. An explicit interactional dimension is vital as business processes may relate entities that are quite depart from one another (locally as well as conceptually), which mostly prohibits mutual adjustment. Starting business processes and actually supplying services is a matter of the underlying $\mathcal{M}$OISE$^+$ departments.

The completely distributed middleware approach taken by SONAR perfectly matches organizations as large and widely distributed entities. Maintaining a centralized comprehensive picture of a whole organization would be very costly and induce an enormous information overhead. It is not even necessary in the first place. Networks at the organizational level are less intermeshed than it is the case for members at the department level. Organizations are better characterized by (partly overlapping) clusters of frequent interaction.

Organizational Field and Society Level. Ascending to the level of organizational fields, there no longer exist explicit notions of joint goals or strategies. Instead, organizations cooperate and compete as co-equals in order to serve their respective distinct purposes. At the same time, governance structures enforce common institutional logics that field participants have to adhere to. Here lies the core competency of ISLANDER. It specifies an environmental framework that regulates the behaviour of participants by specifying what one is allowed, forbidden or obliged to do in certain institutional contexts. The inherent goal is to let participants pursue their respective goals, but on the basis of well-defined practices and symbolic constructs. ISLANDER's performative structure and scene protocols are open for participants to elaborate in order to fit their individual behaviour into the institution. With ISLANDER applied to the field level, SONAR organizations are to adjust their business process specifications in a way to act in and travel between scenes.

Each organizational field might be represented by one single scene or by a set of scenes. As ISLANDER's performative structure provides transitions between scenes it scales in order to include an arbitrary number of fields and thus also addresses the level of society.

Fortunately, AMELIE offers a distributed middleware approach that is needed for the field and society level. However, scene managers might become overloaded if a field was embodied by one or just a few scenes.

4 Conclusion and Outlook

We have presented a proposal to progress from multi-agent to multi-organization systems, which we consider a necessary transition in the face of software systems of ever growing size and complexity. We see one main contribution in the explicit identification and differentiation of collective level aspects in our conceptual framework ORGAN. It features different modes of collective action that are adequate for different contexts. The accompanying concepts and principles are deeply rooted in organization theory. The rationale behind this approach is to learn from the adaptivity, robustness, scalability and reflexiveness of social (multi-)organization systems and to translate their building principles in effective information technology.

In addition, we have made a suggestion of how to utilize agent-oriented technology as a realization means. One particular point is to take advantage not only of current middleware implementations but also of the variety of underlying modelling approaches in order to establish a most adequate fit between them and the requirements of different system levels. Of course, this proposal is somewhat preliminary. The current state-of-the-art of the presented middleware approaches does not match the requirements of our nested middleware proposal and it is far from clear how to connect and integrate the quite different modelling approaches that were proposed for the different system levels on a conceptual basis. Nonetheless, we consider the proposal as a promising vantage point.

Another issue concerns the limitations of the 4-layer approach taken for ORGAN. One might argue that there exist system levels above the society and that there exist further sublevels between the ones presented here. Our idea to soften the 4-layer restriction of ORGAN is somewhat different from introducing *new* levels. Instead, we aim at keeping the four levels presented in this paper but at the same time following a multi-perspective approach. An organizational unit might be able to occupy multiple architectural roles in multiple instances at the same time. This puts stronger emphasis on the *relations* between units instead of the units themselves. Depending on relation, units may take on different manifestations in terms of architectural roles. This allows for example for federations (organizations embedded in organizations as departments), subfields (fields embedded in fields as organizations) or societies of societies (societies embedded in societies as fields). The universal basis of all organizational units in form of the model of a system unit from Figure 1 fosters such a multi-perspective conception.

References

1. Lankes, J., Matthes, F., Wittenburg, A.: Softwarekartographie: Systematische Darstellung von Anwendungslandschaften. Wirtschaftsinformatik 2005 (2005)
2. Northrop, L.: Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute, Carnegie Mellon (2006)

3. Hess, A., Humm, B., Voss, M., Engels, G.: Structuring software cities - a multi-dimensional approach. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 122–129 (2007)
4. Hannan, M., Carroll, G.: An introduction to organizational ecology. In: Hannan, M., Carroll, G. (eds.) Organizations in Industry: Strategy, Structure and Selection, pp. 17–31. Oxford University Press, New York (1995)
5. Wester-Ebbinghaus, M., Moldt, D., Reese, C., Markwardt, K.: Towards Organization–Oriented Software Engineering. In: Züllighoven, H. (ed.) Software Engineering Konferenz 2007 in Hamburg: SE 2007 Proceedings. LNI, vol. 105, pp. 205–217. GI (2007)
6. Scott, W.R.: Organizations: Rational, Natural and Open Systems. Prentice-Hall, Englewood Cliffs (2003)
7. Rölke, H.: Modellierung von Agenten und Multiagentensystemen. Logos Verlag, Berlin (2004)
8. Fischer, K., Schillo, M., Siekmann, J.H.: Holonic multiagent systems: A foundation for the organisation of multiagent systems. In: Mařík, V., McFarlane, D.C., Valckenaers, P. (eds.) HoloMAS 2003. LNCS, vol. 2744, pp. 71–80. Springer, Heidelberg (2003)
9. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: An organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
10. Maier, M.: Architecturing principles for systems-of-systems. Systems Engineering 1(4), 267–284 (1999)
11. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
12. Wester-Ebbinghaus, M., Moldt, D.: A janus-faced net component for the prototyping of open systems. In: Proceedings of the 15th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2008 (2008)
13. Koestler, A.: The Ghost in the Machine. Henry Regnery Co. (1967)
14. Johannessen, J.A.: Systemics applied to the study of organizational fields. Kybernetes 25(1), 33–50 (1996)
15. Mintzberg, H.: Structure in Fives: Designing Effective Organizations. Prentice-Hall, Englewood Cliffs (1983)
16. Wester-Ebbinghaus, M., Moldt, D.: Structure in threes: Modelling organization-oriented software architectures built upon multi-agent systems. In: Proceedings of the 7th International Conference an Autonomous Agents and Multi-Agent Systems (AAMAS 2008), pp. 1307–1311 (2008)
17. Boissier, O., Hübner, J.F., Sichman, J.S.: Organization oriented programming: From closed to open organizations. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS, vol. 4457, pp. 86–105. Springer, Heidelberg (2007)
18. Hübner, J.F., Sichman, J.S., Boissier, O.: A model for the structural, functional and deontic specification of organizations in multiagent systems. In: Bittencourt, G., Ramalho, G.L. (eds.) SBIA 2002. LNCS, vol. 2507, pp. 439–448. Springer, Heidelberg (2002)
19. Hübner, J.F., Sichman, J.S., Boissier, O.: S-moise: A middleware for developing organised multi-agent systems. In: International Workshop on Organizations in Multi-Agent Systems: From Organizations to Organization-Oriented Programming (OOOP 2005), pp. 107–120 (2005)
20. Esteva, M., Padget, J., Sierra, C.: Formalizing a language for institutions and norms. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS, vol. 2333, pp. 348–366. Springer, Heidelberg (2002)

21. Esteva, M., Rodriguez-Aguilar, J., Rosell, B., Arcos, J.: An agent-based middleware for electronic institutions. In: Sierra, C., Sonenberg, L., Tambe, M. (eds.) Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), pp. 236–243 (2004)
22. Köhler, M.: A formal model of multi-agent organisations. Fundamenta Informaticae 79(3–4), 415–430 (2006)
23. Köhler, M., Wester-Ebbinghaus, M.: Closing the gap between organizational models and multi-agent system deployment. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS, vol. 4696, pp. 307–309. Springer, Heidelberg (2007)

Part II

Privacy and Security

RBAC-MAS and SODA: Experimenting RBAC in AOSE

Ambra Molesini[1], Enrico Denti[1], and Andrea Omicini[2]

[1] Alma Mater Studiorum—Università di Bologna
viale Risorgimento 2, 40136 Bologna, Italy
ambra.molesini@unibo.it, enrico.denti@unibo.it

[2] Alma Mater Studiorum—Università di Bologna a Cesena
via Venezia 52, 47023 Cesena, Italy
andrea.omicini@unibo.it

Abstract. Role-Based Access Control models are currently considered as the most effective approach for engineering access control systems. In this paper we experiment their application in the context of Multi-Agent Systems (MAS), by discussing the design of an access control system with an agent-oriented methodology such as SODA. In particular, we show how a clear separation between mechanisms and policies can be achieved by organising the access control system along two layered subsystems, and discuss the advantages of such an approach.

1 Introduction

According to Anderson, "security engineering is about building systems to remain dependable in the face of malice, error or mischance" [1]. Security requirements are often still considered among the non-functional system requirements—something that can be taken into account at a later point of the software development process. Yet, fitting security mechanisms into an existing design leads to design problems and software vulnerabilities: security should rather be considered as a key issue throughout the whole development process, to be defined and explored in the requirements specification phase—i.e., among the functional requirements [2]. So, software engineering methodologies should provide developers with models and processes able to effectively capture the security concerns.

In principle, the agent-oriented paradigm seems a good candidate for capturing security issues in software systems, since the intrinsic agents' features – such as autonomy, intentionality and sociality – make it possible to express the security requirements at the proper abstraction level at early stages of the engineering process, mapping them onto suitable security mechanisms in subsequent stages. A software system can be then conceived as a Multi-Agent System (MAS) where autonomous entities – the agents – interact with each other in order to achieve their goals: such an agent "ensemble" is often represented as an agent *society* at design time. Thus, software engineering techniques should specifically be inspired by – and tailored to – the agent paradigm, so as to fully exploit the agent

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 69–84, 2009.

as well as the agent society metaphors: according to that, several agent-oriented methodologies have been proposed in the last years [3,4,5].

However, most of them still fall short in providing a full-fledged security-oriented approach for agent-oriented systems, although some recent research work is opening the way especially with respect to the requirements analysis [6,7]. Indeed, research on security for MAS has been mainly focused on the solution of individual security problems, such as attacks from an agent to another, from a platform to an agent, and from an agent to a platform [2]: instead, adequately integrating security and systems engineering into a coherent agent-oriented development process is seemingly a more complex task. In particular, in the engineering of interaction of complex software systems, security is strictly related to two other dimensions—coordination and organisation [8,9,10]. In fact, both coordination and security establish laws and rules for constraining the space of interaction—that is, the system dynamics; organisation is their static counterpart, in that it specifies on the one side the agents and the roles they play, as well as the agent-role and inter-role relationships between them. The connection between organisation and security is quite apparent in Role-Based Access Control (RBAC) models & architectures, currently considered as the most promising approach in the engineering of security of complex information systems [11].

Generally speaking, access control is aimed at allowing authorised users to access the system resources they need, while preventing unauthorised users to do the same. Today, access control is typically designed by clearly separating the definition of a suitable *access policy* – i.e., the set norms for granting / refusing access to resources – from the hardware & software *mechanisms* used to implement and enforce it: this separation guarantees independence between the protection requirements to be applied and the way they are applied. Different access policies can thus be easily compared independently from their actual implementation, and changed with no impact over the system; in its turn, the underlying mechanism can support different, multiple policies over time.

More recently, the RBAC technique has been introduced in the context of MAS infrastructures, integrating a role-based security approach with agent-based coordination and organisation, where the role abstraction is already at play [8]. This choice helps facing the typical MAS heterogeneity and openness, since the security properties can be specified in terms of RBAC general concepts: for instance, participating agents can adopt heterogeneous computational models (from purely reactive to cognitive ones), as well as enter/ leave/ change role in the organisation as needed, according to the system policies.

In this paper we first discuss the application of RBAC models in the MAS context (Section 2). Then, we focus on the RBAC requirements to be addressed for engineering an RBAC system (Subsection 2.2), and show how SODA, an agent-oriented methodology, addresses those requirements (Section 3) Finally, as a case study, we apply our approach to the control of the accesses to a university building (Section 4), and show the benefits of a clear separation between mechanisms and policies. Discussion and comparison with some relevant related work are reported in Section 5.

2 Access Control in Multi-Agent Systems

Access control is aimed at enabling (only) the authorised users to access the system resources in a controlled and supervised way. A key aspect is the clear separation between the rules used to decide whether access to a resource should or not be granted for a given user – the *access policy* – and the hardware & software *mechanisms* actually enforcing such rules. Such a separation is useful for two main reasons: first, to uncouple the definition of a policy from its implementation, so that the latter is not affected by policy changes; then, to more easily identify the basic properties that any access control system should satisfy—such as complete mediation, default deny, minimum privilege, etc.

With respect to previous access control models, such as Discretionary Access Control (DAC) and Mandatory Access Control (MAC), RBAC – a NIST standard [12] – specifies security policies in terms of organisational abstractions (users, roles, objects, operations, permissions, and sessions) and their relationships [13]. Users are assigned to roles, and roles to permissions. A *role* is understood as a job function within the context of an *organisation* with some associated semantics regarding the authority and responsibilities conferred to the user that plays the role at a given time. A *permission* is an approval to perform an *operation* on some protected *objects*: the exact semantics of "operation" and "object" depends on the specific case. A session is a mapping between a given user and the subset of its currently active roles: so, each session is associated with a single user, while a user can be associated to one or more sessions.

Organisation rules are defined in terms of relationships between the above elements—namely, between roles and permissions, and between roles and users; inter-role relationships are also introduced to specify *separation of duties*. More precisely, *static separation of duty* (SSD) is obtained by enforcing constraints on the assignment of users to roles, while *dynamic separation of duty* (DSD) is achieved by placing constraints on the roles that can be activated within or across the given users' session(s). Accordingly, introducing RBAC in the context of MAS coordination models and infrastructures essentially amounts at mapping roles, sessions, and policies onto suitable runtime issues of the MAS organisation, dynamically manageable via infrastructural services.

2.1 RBAC for Multi-Agent Systems

RBAC-MAS [9,14] is a model for an RBAC-like organisation of MAS, where RBAC general concepts are tailored to MAS specificities (Fig. 1).

Generally speaking, agent-oriented methodologies often exploit MAS role-based organisational models just as analysis & design tools [15]: instead, applying an RBAC-like model into MAS shifts the focus on runtime aspects, making roles, sessions, and policies the key runtime issues of a MAS organisation. In particular:

- RBAC users are represented in RBAC-MAS by *Agent Classes*;
- the behaviour of each role (agent) is defined in RBAC-MAS in terms of *Actions* and *Perceptions* used, respectively, to affect and perceive the computational environment where the agent is situated;

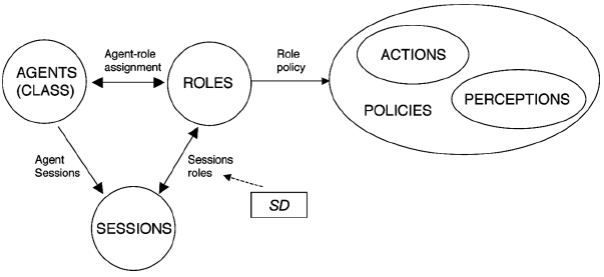


Fig. 1. RBAC-MAS reference model [9]

- *Policies* constrain the admissible interaction histories of an agent playing a specific role, and are used to explicitly model the organisational rules: at runtime, they are enforced by the underlying MAS infrastructure;
- while RBAC equips agents with a default role-set, in RBAC-MAS the agents' session starts with no activated roles: roles can be subsequently activated on a step-by-step basis, according to the specified activation/deactivation policies. The dynamics of role activation is constrained by the DSD rules.

More generally, RBAC approaches split some security issues which were previously faced by individual applications between the design and the infrastructure levels. Analogously, the factorisation in terms of agents of some security issues that frequently emerge in the engineering of a distributed system involves both the extension of MAS infrastructures with suitable services, and the improvement of the methodological support towards the design of complex secure systems.

2.2 Requirements for Engineering an RBAC-MAS System

In order to support the analysis and design of an access control system, a methodology should properly support the analysis and design of all the abstract entities adopted by the access control technique of choice.

In the specific case of RBAC-MAS, a methodology should allow engineers to model and design roles, organisations, objects, policies, operations, as well as static and dynamic constraints. Although at a first sight this requirement might seem somehow obvious, and possibly even superficial, a more detailed analysis leads to highlight some interesting aspects, that we will discuss below. Moreover, the separation between policy and mechanism introduces further constraints: in fact, while such two sub-systems can be designed separately, they are indirectly coupled by the representation language of the access policies, since these are designed by one sub-system, but enforced by the other. So, while it is not necessary to know the specific policy during the mechanism design phase, knowing how the policy is represented is relevant to choose the most appropriate storage and to decide the most adequate enforcing implementation.

It is important to note that in the original RBAC-MAS formulation operations and objects collapse to the same entity "action" [14], i.e., an operation over a given object. However in the following we prefer to characterise both actions – the operation over the object – and objects, given that, from a software engineering viewpoint, the object entity hides all the environment abstractions and structures. The knowledge of the environment and of its topological structure is instead crucial when we deal with the engineering of a new system specially in the context of MAS engineering where is often necessary to design new environment abstractions in order to support the achievement of the agent's tasks/goals. Obviously, the same distinction is not so crucial in the context of MAS infrastructures – where RBAC-MAS was developed – because there environment is already analysed and designed in terms of the infrastructures themselves. Finally, the distinction between actions and objects is very important also in the engineering of the interactions between agents and environment.

In turn, abstract entities add their own requirements, which can be outlined as follows:

Role — This entity implies that the methodology should support the modelling and design of both the user roles and the administrative roles which are required for the system management.

Organisation — This entity implies that the methodology should support the modelling and design of agent societies and of the rules that govern them.

Object —This entity hides a lot of complexity: in fact, the ability of modelling the "system's objects"(i.e., the system "resources") requires that the methodology is able to model the environment of the MAS. In the case of the mechanism sub-system, this means that the methodology should be able to both model and design the environment, since the mechanism has to provide the physical and logical control to prevent unauthorised access. In addition, the notion of MAS environment, as suggested in [16], implies both the modelling of the environment abstractions – entities of the environment encapsulating some functions – and of the topology abstractions – entities of MAS environment that represent the (either logical or physical) spatial structure. In fact, enabling system requirements to determine the topological structure of the system is necessary in order to capture the wide range of access control systems (e.g., from controlling the access to a file, to a room in a building, etc). Summing up, topology constraints should be considered since the earliest requirement analysis phase. So, supporting the object entity requires that a methodology enables engineers to model and design both the topological structure of the environment and the resources that populate it.

Action and Perception — These entities imply that the methodology should support the modelling and design of the actions that roles can perform over the objects and of the perceptions of the environment—as highlighted in the Subsection 2.1.

Policy — This entity leads to the design of rules concerning the abstractions. More precisely, these policies represent rules that involve roles, objects and actions, and rules over role activations and default roleset.

Finally, the mechanism sub-system should manage the association between users and roles, and should do that in a dynamic way: indeed, policies could change over time and at any time, and the sub-system should be able to support and implement such changes with no need to be stopped or reset.

3 SODA and RBAC-MAS

3.1 The SODA Methodology

SODA (Societies in Open and Distributed Agent spaces) [17,18] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts meta-model (A&A) [19], and introduces a *layering principle* as an effective tool for scaling with the system complexity, applied throughout the analysis and design process [20].

The SODA *abstractions* – explained below – are logically divided in three categories: *i)* the abstractions for modelling/designing the system active part (task, role, agent, etc.); *ii)* the abstractions for the reactive part (function, resource, artifact, etc.); and *iii)* the abstractions for interaction and organisational rules (relation, dependency, interaction, rule, etc.). In its turn, the SODA *process* is organised in two phases, each structured in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. Each sub-phase models (designs) the system exploiting a subset of 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 active part, one for the reactive part, and another for interaction and organisational rules. Fig. 2 shows an overview of the methodology structure: as we will show in the case study (Section 4), each step is practically described as a set of relational tables (listed in Fig. 2 for the sake of completeness).

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

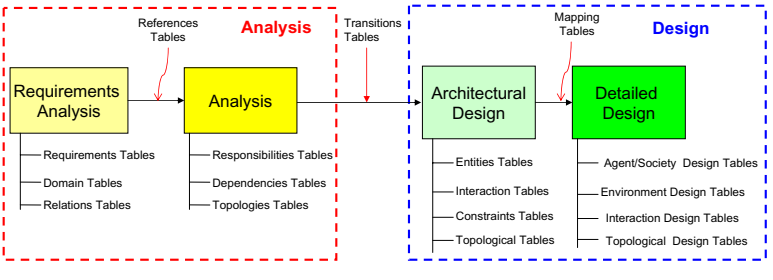


Fig. 2. An overview of the SODA process

external-environment notion 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 modelled in terms of a suitable *relation*.

Analysis. The Analysis step expresses the 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 here the starting point for the definition of *dependencies* among such abstract entities. The structure of the environment is also modelled in terms of *topologies*, i.e., topological constraints over the environment.

Architectural Design. The main goal of this stage is to assign responsibilities of achieving tasks to *roles*, and responsibilities of providing functions to *resources*. In order to attain one or more tasks, a role should be able to perform *actions*; analogously, the resource 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, while rules enable and bound the entities' behaviour. Finally, the topology constraints lead to the definition of *spaces*, i.e., conceptual places structuring the environment.

Detailed Design. Detailed Design is expressed in terms of *agents*, agent *societies*, *composition*, *artifacts* and *aggregates* and *workspaces* for the abstract entities, while the interactions are expressed by means of *uses*, *manifests*, *speaks to* and *links to* concepts. More precisely, agents are intended here autonomous entities able to play several roles, while a society can be seen as a group of interacting agents and artifacts when its overall behaviour is essentially an autonomous, proactive one. The resources identified in the previous step are here mapped onto suitable artifacts, while aggregates are defined as a group of interacting agents and artifacts when its overall behaviour is essentially a functional, reactive one. The *workspaces* take here 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.

3.2 RBAC Requirements in SODA

In Subsection 2.2, two major requirement categories were outlined: one about the representation language of the access policies, the other concerning the RBAC-MAS abstract entities. With respect to the former issue, SODA is conceptually orthogonal to any possible representation language, since at the moment no hypothesis is made about the language adopted for compiling its relational tables. With respect to the latter issue, RBAC-MAS abstract entities are captured by suitable SODA abstractions as follows:

Role — this RBAC-MAS entity is directly mapped onto SODA *role* concept in the Architectural Design phase, namely in the Entities Tables.[1]

Organisation — this entity is mapped by SODA *societies* in the Detailed Design phase (see the homonymous tables); in turn, the rules governing such societies are embodied in the social artifacts expressed by the Environment Design Tables.

Object — since this RBAC-MAS entity is used to represent the environment of the system, and given that topology constraints need to be considered since the earliest requirement analysis phase, its impact spreads through all SODA phases. More precisely, since the environment is composed of both environment abstractions and topological abstractions, in the analysis phase such aspects are captured via the *legacy system* and *function* abstractions, and the *topology* abstractions, respectively. Then, in the Design phase (which is particularly relevant for the mechanism sub-system), environment abstractions take the form of *resources* (in the Architectural Design step) and *artifacts* (in the Detailed Design step); in the same way, topology abstractions take the form of *spaces* (in Architectural Design) and *workspaces* (in Detailed Design).

Action and Perception — these entities are natively supported by SODA: *actions* and *use* map action in the Architectural Design and in the Detailed Design respectively, while *manifest* maps perception in the Detailed Design.

Policy — this entity finds it specific counterpart in the *Rule* abstraction (Constraints Tables) of SODA's Architectural Design, since in the Analysis phase they are simply considered as relations and dependencies; such rules are then mapped onto suitable (individual or social) *artifacts* in the Detailed Design step.

Considering how SODA supports RBAC-MAS requirements with respect to the three abstractions categories outlined above in this Section, it is worth noting that in the design of the mechanism sub-system only the reactive abstractions are involved, while in the design of the policy sub-system only the interactions and rules abstractions are used: the active abstractions, instead, are just modelled – not designed –, as in this kind of system the corresponding roles, from the RBAC design perspective, are just an input of the system, defined in the policy sub-system requirements. This is no longer true if a new system is being designed from scratch, where it is unlikely to have such roles as inputs of the whole system: rather, in such cases these roles should likely be first designed—and only then used as inputs in the design of the policy sub-system.

Summing up, all the key RBAC-MAS issues discussed above are quite well captured in SODA: such aspects are naturally taken into account when the

[1] The term "Entities" in SODA tables is used with a different, and more specific, semantics than in RBAC-MAS: Entities Tables, in fact, refer to roles, resources, actions, and operations, while RBAC-MAS uses that term to refer to a wide range of abstractions—from roles to organisations, objects, operations, permissions and constraints.

methodology is applied, and take the form of suitable requirements and specifications in the SODA tables, as we will show in the case study below.

4 The Case Study

The case study we consider is the management of the access control to a university building: for the sake of brevity, we set the core layer quite at a high abstraction level, and only a limited set of the SODA tables will be reported. As highlighted in Section 2, the design of mechanisms (Subsection 4.1) is kept separate from the design of policies (Subsection 4.2). Here we present the key system aspect, that is, the topological structure of the environment, leaving the discussion of the application roles involved to the sub-systems design.

The topological structure, shared between the sub-systems, is established during the requirements analysis, since it derives from the physical structure of the building. More precisely, following the hierarchial view in Figure 3, layer *a* represents the whole building, layer *b* represents the spatial organisation of the building in terms of classrooms, administration, departments, and library, layer *c* represents the spatial organisation inside each department – composed of administration offices, professors' offices and the library – and administration – made of offices; finally, layer *d* represents the spatial organisation inside the administration department, which is, again, made of offices. Such a hierarchical representation simplifies the design of both mechanisms and policies, since the same mechanism can be replicated in the access points of the building, while finer-grained policies can be expressed for each space.

The general description above summarises the analysis of the topological constraints for both the mechanism and policies. As far as the mechanisms are concerned, this analysis suggests that the designer organises the environment following the topological structure, nesting spaces and workspaces, and mapping spaces onto workspaces: this approach simplifies the design of the mechanisms, which can be structured along different control levels. On the other hand, the policy designer can express accurate policies, defining whether each role can either access or not the building, along with its access privileges, for each access point—i.e., on a fine-grained basis.

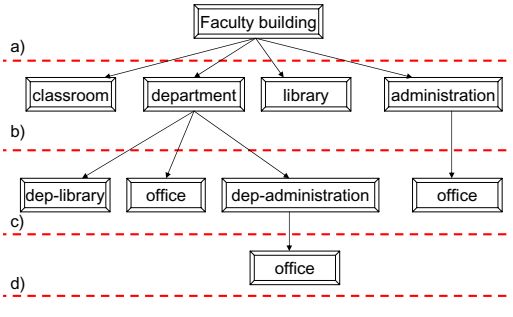


Fig. 3. The physical structure of the university building

4.1 Designing the Mechanism

The topological structure of the environment explained above is captured by two different SODA tables in the Architectural Design phase (Figure 4): the Space table ($(L)S_t$) describes the logical space of the system, while the Space-Connection table ($(L)SC_t$) shows the relation between spaces. Then, the workspaces in the Detailed Design step naturally follow from the spaces defined in the Architectural Design step.

According to this topological structure, the global mechanism can be conceived as composed of two complementary sub-mechanisms, one for the access to the whole building (Figure 5, top) and another for the access to a single room/office department—simply "room" in the following (Figure 5, bottom). Both are based on "Interface Artifacts" that represent the wrappers to the hardware resources capturing the user credentials: in the case of the whole building, there is an Interface Artifact for each hardware device that monitors a specific physical access point, while each room has its own Interface Artifact.

We assume that Interface Artifacts generate an event whenever a user enters the building (room); we also assume that such events are perceived by a suitable "(Room-)Access Manager" agent, whose task is to check whether such an access can be authorised. For this purpose, the Access Manager exploits the "User(-room) Artifact" to check if the user can access the building (room) and, if so, modifies the state of the "Building-State Artifact" accordingly. Room access, instead, must be granted also to users that are not permanently authorised, provided that they have an appointment: this is performed via the "Appointment Artifact".

The "User(-room) Artifact" stores all the roles permanently qualified to access the building (room), along with their access privileges. This artifact provides two different sets of functionalities: those needed to check the access authorisations, and those required by special users—such as administrators – for management

Space	Description
Faculty	the whole building
Classroom	the student space
Library	the faculty library
Department	the research centre
Administration	the faculty bureaucracy centre
Dep-Library	the department library
Dep-Administration	the department bureaucracy centre
Office	the rooms for employees

Space	Connection
Faculty	Classroom, Library, Department, Administration
Administration	Office
Department	Dep-Library, Dep-Administration, Office
Dep-Administration	Office

Fig. 4. Topological Structure in top down order: $(L)S_t$, $(L)SC_t$

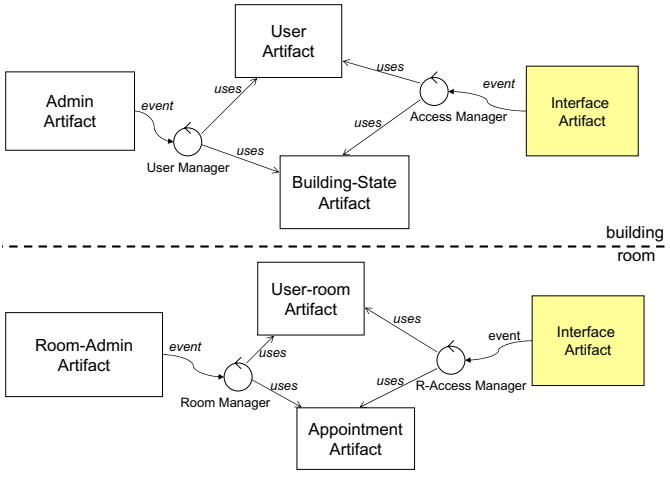


Fig. 5. The mechanisms for the access control for the whole building (top) and the single room (bottom)

purposes – such as adding or deleting roles, modifying the users' privileges, and so on. The "Building-State Artifact" traces the people inside the building: when a user exits the building, an event is generated, and the Access Manager modifies the artifact again. The "Appointment Artifact" manages the users' appointments, storing the list of the appointments for a given room: so, it is obviously shared by all the people that work in that room. The stored data include the time and the people involved in each appointment, enabling the policies designer to express several different policies—for instance, deciding whether the access should be denied or authorised if the people involved are not in office.

Users are managed by the "User Manager" agent, while users authorised to enter a room are managed by the "Room Manager": both such managers perceive the events generated by the "Admin Artifact" (respectively, by the

Artifact	**Usage Interface**
Interface Artifact	enter_role, exit_role
Admin Artifact	new_role, canc_role, update_role, get_roles, new_user, canc_user, update_user, can_pol, new_pol, update_pol
User Artifact	check_access, new_role, canc_role, update_role, get_roles, new_user, canc_user, update_user, can_pol, new_pol, update_pol
Building-State Artifact	update_in, update_exit, get_roles
Room-Admin Artifact	new_role, canc_role, update_role, get_roles, new_user,canc_user, update_user, can_pol, new_pol, update_pol, insert_appointment, modify_appointment, delete_appointment, get_appointments
User-room Artifact	check_access, new_role, canc_role, update_role, get_roles, new_user, canc_user, update_user, can_pol, new_pol, update_pol
Appointment Artifact	check_appointment, insert_appointment, modify_appointment delete_appointment, get_appointments

Fig. 6. Artifact-UsageInterface table

"Room-Admin Artifact"). In turn, this represents the interface between the human administrator and the mechanism itself, and is used by the system administrator to introduce or delete roles and, more generally, to edit the policies over time (in the case of the building) or to handle appointments (in the case of rooms).

These functionalities are represented in the Artifact-UsageInterface table (AUI_t) of the Detailed Design (Figure 6): the usage interface represents the set of operations provided by an artifact. For space reason, only the names of the artifacts' operations have been reported, by omitting operations' parameters.

4.2 Designing RBAC Policies

RBAC policies are designed during SODA's architectural design phase: more precisely, the constraints that shape the role interaction spaces drive the design of the organisational rules.

In our case, the environment needs not – and actually can not – be explicitly designed, as it is already represented in/by the above mechanism as specified in Subsection 3.2: all we need is to model it in the analysis phase, so as to identify the relationships and the interactions between the two sub-systems. Then, the mechanism's artifacts will enforce the policies designed here, while, conversely, the roles (agents) defined here will interact with the mechanism. For the same reason, also the topological structure is implicit in the mechanism: so, the spaces/workspaces identified in Subsection 4.1 are the same here, too. As a result, we now focus only on the design of the interaction and organisational rule entities.

From the viewpoint of sub-system requirements, our scenario, in its simplest version, involves six different roles: *Student*, *Professor*, *Technician*, *Administrative staff*, *Guide* and *Visitor*. Professors, Technicians, and Administrative staff can freely access the building at any time. Students, instead, can access the building – in particular, classrooms and library – only during the regular opening hours; in addition, to access the Administrative staffs' and Professors' offices, they must have an appointment. Finally, Visitors cannot access the building without a Guide, who is a member of the University – Professor, Technician, Administrative staff – that escorts visitors inside the building.

Role	Action
Visitor	enter, exit, ask_appointment
Student	enter, exit, ask_appointment
Professor	enter, exit, canc_appointment, set_appointment, change_policy insert_role, canc_role
Administrative staff	enter, exit, canc_appointment, set_appointment, change_policy insert_role, canc_role
Technician	enter, exit, canc_appointment, set_appointment, change_policy insert_role, canc_role
Guide	enter, exit
System administrator	enter, exit, change_policy, insert_role, canc_role

Fig. 7. Role-Action Table ($(L)RA_t$)

Beyond these roles, the user management activity highlights the need of a new "service" role – the *System administrator* – for modifying the access privileges and managing the users' credentials. This is not surprising, since during SODA's Architectural Design new roles are often identified that complete and support the activities of the roles directly deducted from the requirements.

So, there are seven different roles in all, each potentially able to perform the actions depicted in Figure 7—we say "potentially" because the role will actually be enabled to do such actions only if/when authorised to. Rules derive from the desired policies, and are listed in Figure 8; the corresponding association to roles is given in Figure 9.

For the sake of clarity, Figure 8 is organised in different sets of rules. The first set (Guide-Rule and Visitor-Rule) reports the DSD and SSD constraints over the corresponding roles (shown in Figure 9) that are enforced by the "User Artifact". In particular, the *Guide* role is dynamically incompatible with any other role during a session (DSD constraint), since the Guide cannot abandon visitors, who are not allowed to move alone inside the building. Similarly, the *Visitor* role is incompatible with any other because a visitor cannot cover any position inside the university: this is an SSD constraint, since this incompatibility holds permanently (it is not related to a temporary status in the session). The second set of rules represents the constraints over the administrative operations, enforced by the the "(Room-)Admin Artifact". The two other sets express, respectively, the constraints over the access to the building (third set) and to

Rule	Description
Guide-Rule	Guide cannot be activated together other roles (DSD constraint)
Visitor-Rule	Visitor cannot be activated together other roles (SSD constraint)
Admin-Rule	The Administrator can modify the access rules for the whole building but cannot modify the access rules for the offices
Prof-Admin-Rule	The Professor can modify the access rules for his/her office
Staff-Admin-Rule	The Administrative staff can modify the access rules for their office
Visit-Rule	Visitor can access the building only together a Guide
Building-Rule	The access to the building is possible only when the building is open to the public
Uni-Build-Rule	Professor, Technician, Administrative staff and System administrator can always access the building
App-Rule	The access to an office is granted only if the Student has an an appointment and the Professor/Administrative staff is in the office
Administration-Rule	The access to the staff office is possible only when the office is open to the public
ClassRoom-Rule	The access to a classroom is not granted during a lecture
Library-Rule	The access to the library is possible only when the library is open to the public
Lab-Rule	The access to the laboratory is possible only when the laboratory is open to the public
Department-Rule	The access to the department is possible only if the destination room grants the access

Fig. 8. Rule table ($(L)Ru_t$)

Role	Rule
Visitor	Visitor-Rule, Visit-Rule, Building-Rule, Administration-Rule ClassRoom-Rule, Library-Rule, Department-Rule
Student	Building-Rule, App-Rule, Administration-Rule, Lab-Rule ClassRoom-Rule, Library-Rule, Department-Rule
Professor	Uni-Build-Rule, Administration-Rule, Lab-Rule Library-Rule, Department-Rule, Prof-Admin-Rule
Administrative staff	Uni-Build-Rule, Administration-Rule, Lab-Rule Library-Rule, Department-Rule, Staff-Admin-Rule
Technician	Uni-Build-Rule, Administration-Rule,Library-Rule, Department-Rule
Guide	Guide-Rule, Uni-Build-Rule, Administration-Rule, Lab-Rule ClassRoom-Rule, Library-Rule, Department-Rule
System administrator	Uni-Build-Rule, Administration-Rule, Admin-Rule Library-Rule, Department-Rule

Fig. 9. Role/Rule association table $(L)RoRu_t$

each room (fourth set), and are enforced, respectively, by the "User Artifact" and by the "User-room Artifact" & "Appointment Artifact" pair.

5 Conclusions and Related Work

To the best of our knowledge, this is the first attempt to support the design of an RBAC system via an agent-oriented methodology such as SODA. In fact, other works in the literature (e.g. [21,22]) exploit MAS for realising an RBAC system, but in the context of specific domain applications: thus, they lead to ad-hoc solutions which are not easily reusable in other contexts, due to the lack of separation between the "static part" of the system – the mechanism – and the "dynamic part" – the policies. In addition, these works delegate the enforcing of policies to agents, while our approach is that such an enforcing should more properly be done by suitable environmental abstractions [9].

Moving from the "university building access" case study, this paper aims at showing the benefits of a clear separation between mechanism and policies, so as to split the design of an access control system in two separate aspects: our SODA-based approach leads to design such aspects as two sub-systems, exploiting the agent paradigm. In particular, the mechanism sub-system is designed as general as possible, since its structure is basically stable and reusable as is in other applications: artifacts wrap the physical resources, and a society of agents reacts to the events occurring in the environment. From this viewpoint, SODA's intrinsic support for both environmental abstractions – artifacts – and topology abstractions – workspaces – makes it possible to support the whole design process of the environment, including its spatial structure, in a uniform way.

Policies, on the other hand, are generally tied to the application domain, so they typically have to be re-designed each time: as highlighted in Subsection 4.2, the design of this sub-system is focused on the definition of roles and their access privileges, which are the core of any RBAC system. Again, SODA natively supports both roles and access privileges, which can be easily expressed in terms

of rules. So, only one sub-system needs to be redesigned in response to any application or policy change—the other sub-system remains untouched, unlike what would happen with a monolithic system.

Future work will be mainly devoted to improve the methodology in several directions: *i)* to support the design of secure agent-oriented systems since the earliest Requirement Analysis step; *ii)* to develop a language for SODA rules which could be able to capture all the relevant RBAC permissions and constraints, and *iii)* to more deeply study the access control issues related to artifacts.

References

1. Anderson, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd edn. Wiley Computer, Chichester (2001)
2. Mouratidis, H., Giorgini, P.: Secure tropos: A security-oriented extension of the tropos methodology. International Journal of Software Engineering and Knowledge Engineering 17, 285–309 (2007)
3. Henderson-Sellers, B., Giorgini, P. (eds.): Agent Oriented Methodologies. Idea Group Publishing, Hershey (2005)
4. Bergenti, F., Gleizes, M.P., Zambonelli, F. (eds.): Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Kluwer Academic Publishers, Dordrecht (2004)
5. Bernon, C., Cossentino, M., Pavón, J.: An overview of current trends in european AOSE research. Informatica 29, 379–390 (2005)
6. Liu, L., Yu, E., Mylopoulos, J.: Analyzing security requirements as relationships among strategic actors. In: 2nd Symposium on Requirements Engineering for Information Security (SREIS 2002), Electronic Proceedings, Raleigh, NC, USA (2002)
7. Yu, E., Cysneiros, L.M.: Designing for privacy and other competing requirements. In: 2nd Symposium on Requirements Engineering for Information Security (SREISí 2002), Electronic Proceedings, Raleigh, NC, USA (2002)
8. Omicini, A., Ricci, A., Viroli, M.: RBAC for organisation and security in an agent coordination infrastructure. Electronic Notes in Theoretical Computer Science 128, 65–85 (2005); 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo 2004), Proceedings (August 30, 2004)
9. Viroli, M., Omicini, A., Ricci, A.: Infrastructure for RBAC-MAS: An approach based on Agent Coordination Contexts. Applied Artificial Intelligence, Special Issue: State of Applications in AI Research from AI*IA 2005 21, 443–467 (2007)
10. Johnson, M., Feltovich, P.J., Bradshaw, J.M., Bunch, L.: Human-robot coordination through dynamic regulation. In: IEEE International Conference on Robotics and Automation, ICRA 2008, Pasadena, California, May 19–23, 2008, pp. 2159–2164. IEEE Computer Society, Los Alamitos (2008)
11. Sandhu, R.S., Coynek, E.J., Feinsteink, H.L., Youmank, C.E.: Role-based access control models. IEEE Computer 29, 38–47 (1996)
12. RBAC: American National Standard 359-2004 (Role Base Access Control – home page) (2004), http://csrc.nist.gov/rbac/
13. Ferraiolo, D., Kuhn, R., Sandhu, R.: RBAC standard rationale: Comments on a critique of the ANSI standard on Role Based Access Control. IEEE Security & Privacy 5, 51–53 (2007)

14. Omicini, A., Ricci, A., Viroli, M.: An algebraic approach for modelling organisation, roles and contexts in MAS. Applicable Algebra in Engineering, Communication and Computing, Special Issue: Process Algebras and Multi-Agent Systems 16, 151–178 (2005)
15. Ricci, A., Viroli, M., Omicini, A.: An RBAC approach for securing access control in a MAS coordination infrastructure. In: Barley, M., Massacci, F., Mouratidis, H., Scerri, P. (eds.) 1st International Workshop Safety and Security in MultiAgent Systems (SASEMAS 2004), AAMAS 2004, Proceedings, New York, USA, pp. 110–124 (2004)
16. Molesini, A., Omicini, A., Viroli, M.: Environment in Agent-Oriented Software Engineering methodologies. Multiagent and Grid Systems, Special Issue on Environment Engineering for MAS 5 (2009)
17. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: A Roadmap to Artefacts. In: Dikenelli, O., Gleizes, M.-P., Ricci, A. (eds.) ESAW 2005. LNCS, vol. 3963, pp. 49–62. Springer, Heidelberg (2006)
18. SODA: Home page (2008), http://soda.apice.unibo.it/
19. Omicini, A.: Formal ReSpecT in the A&A perspective. Electronic Notes in Theoretical Computer Sciences 175, 97–117 (2007); 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA 2006), CONCUR 2006, Post-proceedings, Bonn, Germany (August 31, 2006)
20. Molesini, A., Omicini, A., Ricci, A., Denti, E.: Zooming Multi-Agent Systems. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 81–93. Springer, Heidelberg (2006)
21. Drouineaud, M., Lüder, A., Sohr, K.: A role based access control model for agent based control systems. In: Unland, R., Ulieru, M., Weaver, A.C. (eds.) 1st IEEE International Conference on Industrial Informatics (INDIN 2003), Banff, Alberta, Canada, pp. 307–311 (2003)
22. Yamazaki, W., Hiraishi, H., Mizoguchi, F.: Designing an agent-based RBAC system for dynamic security policy. In: IEEE 13th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2004), 9th International Workshop Enterprise Security (ES 2004), Modena, Italy, pp. 199–204. IEEE Computer Society, Los Alamitos (2004)

Sensitive Data Transaction in Hippocratic Multi-Agent Systems

Ludivine Crépin[1,3], Yves Demazeau[1], Olivier Boissier[2], and François Jacquenet[3]

[1] Laboratoire d'Informatique de Grenoble - CNRS
{Ludivine.Crepin,Yves Demazeau}@imag.fr
[2] Centre G2I
Ecole Nationale Supérieure des Mines de Saint-Etienne
Olivier.Boissier@emse.fr
[3] Laboratoire Hubert Curien (UMR CNRS 5516)
Université Jean Monnet, Saint-Etienne, France
Francois.Jacquenet@univ-st-etienne.fr

Abstract. The current evolution of Information Technology leads to the increase of automatic data processing over multiple information systems. The data we deal with concerns sensitive information about users or groups of users. A typical problem in this context concerns the disclosure of confidential identity data. To tackle this difficulty, we consider in this paper the context of Hippocratic Multi-Agent Systems (HiMAS), a model designed for the privacy management. In this context, we propose a common content language combining meta-policies and application context data on one hand and on the other hand an interaction protocol for the exchange of sensitive data. Based on this proposal, agents providing sensitive data are able to check the compliance of the consumers to the HiMAS principles. The protocol that we propose is validated on a distributed calendar management application.

Keywords: Privacy, Sensitive Data Transaction, Confidentiality, Multi-Agent Systems, Interaction Protocol.

1 Introduction

With the use of multiagent technologies, the sensitive data transmission problem in Multi-Agent Systems (MAS) is all the more present since users delegate their sensitive data to an autonomous agent (the interaction is an essential feature of Multi-Agent Systems). Spread of sensitive data over the Internet using autonomous entities becomes an important risk that requires to be considered nevertheless this problem has not received enough attention by the researchers in the domain until now.

We have proposed in [1] the model of Hippocratic Multi-Agent Systems (HiMAS) that takes into account this data sensitivity regarding moral issues and not legal aspects. This model defines the concept of *private sphere* for an agent

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 85–101, 2009.

or a user to structure and to represent the data involved in the management of privacy, and the nine principles that should govern the functioning of a HiMAS so that privacy is preserved in the Multi-Agent Systems. In order to engineer agents societies according to this conceptual framework we focus in this article on a precise objective of the design of such a system: sensitive data protection during sensitive data transaction. Such a transaction represents a sensitive data transaction between two agents. To tackle this problem, we propose a sensitive data transaction protocol inspired by [2,3], with an associated content language in the HiMAS context. This protocol is our first step for the implementation of a HiMAS. To illustrate this protocol, we have chosen the distributed calendar management application presented in [4].

The next section briefly presents the model of Hippocratic Multi-Agent Systems in order to draw the global context in which we place our present work. Section 3 focuses on the definition of the content language and the associated semantics used in the protocol that we propose in section 4. We present an application of our sensitive data transaction protocol in section 5. Finally we talk about related work in section 6 and conclude with some perspectives on the future work.

2 Foundations: Hippocratic Multi-Agent Systems (HiMAS)

As introduced in the previous section, the HiMAS model proposed in [1] is composed of two main components: the private sphere representation and some hippocratic principles that we present in the following sections. The reader interested in more information about this model and the private sphere, may refer to [1].

2.1 Private Sphere, Consumer and Provider

The private sphere contains information that an agent considers as sensitive, represented by sensitive data, and all the associated management rules. For instance, in the context of calendar management [4], sensitive data is the user's slots of time or meetings that are delegated to an agent. The agent's private sphere represents all this kind of data and all the rules defining the conditions of its disclosure, its use or its sharing for example.

To define the private sphere dimensions, we are inspired by many researches in social science [1]. The first one focus on the **ownership rights** of sensitive data. They are only assigned to agents concerned by this data [5]. Moreover the private sphere is also **personal** [6,7], **personalizable** (the agent chooses what its private sphere contains) [8,9,10] and **context-dependent** [11,12].

To represent the possible positions of an agent with respect to the private sphere, we define three roles represented in the Figure 1. The **consumer** role characterizes the agent which asks for sensitive data and uses it. The **provider**

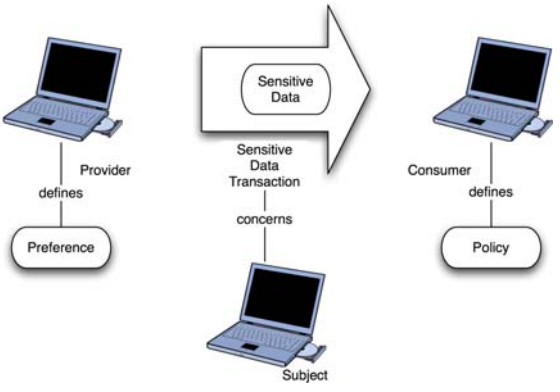


Fig. 1. Agents role in relation with a sensitive data transaction

role characterizes the agent which discloses sensitive data[1]. The last role, the **subject**, describes the agent from whom origined sensitive data.

With this definition of the agent's private sphere we install a provider-centred view on the management of sensitive data. This is due to the fact that we mainly have a user-centred view on privacy preserving: user should be confident in the management of the sensitive data they delegate to their personal agent.

2.2 Nine Principles for HiMAS

The HiMAS model is inspired by the Hippocratic Databases [14]. In order to preserve privacy, a HiMAS must respect the nine principles described below.

1. **Purpose specification:** The provider must know the objectives of the sensitive data transaction. Therefore it can evaluate the transaction consequences.
2. **Consent:** Each sensitive data transaction requires the provider's consent (and the subject's consent if it is not the same agent).
3. **Limited collection:** The consumer commits to cutting down to a minimum the amount of data for realizing its objectives.
4. **Limited use:** The consumer commits to only use sensitive provider's data to satisfy the objectives that it has specified and nothing more.
5. **Limited disclosure:** The consumer commits to only disclose sensitive data to reach its objectives. Moreover it must disclose it the least number of times possible and to the least number of agents.
6. **Limited retention:** The consumer commits to retain sensitive data only for the minimum amount of time it takes to realize its objectives.
7. **Safety:** The system must guarantee sensitive data safety during storage and transactions.
8. **Openness:** The transmitted sensitive data must remain accessible to the subject and/or the provider during the retention time.

[1] We can notice that this vision is the opposite of the centered service vision like for example [13], regarding the consumer and the provider.

9. **Compliance:** Each agent should be able to check the obedience to the previous principles.

3 Content Language for Sensitive Data Transaction

In order to integrate the HiMAS principles in the interaction protocol that we propose, we have chosen to define the semantics of these principles. This study also leads us to determine the different links between these principles. The first step of this work is to group together these principles according to their purpose into the HiMAS agent's reasoning: during the sensitive data transaction; during the other interactions; and in relation to the system implementation. After the study of this semantics, we propose a representation of the required principles in a content language. These two steps are the foundations of the sensitive data transaction protocol that we propose.

3.1 Content Language Semantics

Let us consider the principles that play a part in a sensitive data transaction. In such a context, the provider defines a **policy** and the consumer a **preference** to define their desires regarding the sensitive data manipulations.

The consumer's policy and the provider's preference are similar to the policy and the preference defined in [2]: these concepts are composed of the transaction objectives[2], the deletion time of collected data, a broadcasting list and the data format (required references).

In order to map a policy to a preference, a sensitive data transaction groups together required sensitive data with the consumer's policy and the provider's consent and preference.

Seven of the nine HiMAS principles play a part in sensitive data transactions:

- **1. Purpose specification:** The consumer asks for provider's sensitive data in order to realize required tasks. Since the consumer must declare his purpose, these tasks should be used to define its objectives. The consumer must send them to the provider.
- **3. Limited collection:** With the definition of its objectives, a consumer can select the sensitive data that is only required for the realization of its objectives.
- **4. Limited use:** The consumer can then determine the possible uses of the collected sensitive data by virtue of its objectives.
- **5. Limited disclosure:** The objectives enable the consumer to determine which agents are allowed to receive the collected sensitive data.
- **6. Limited retention:** The specification of the objectives defines also the sensitive data retention time for the consumer.
- **8. Openness:** The openness implies that the provider and/or the subject are in the broadcasting list.

[2] The objectives are close to the concept of goal, like for example in BDI model [15] or [16].

– **2. Consent:** The mapping between a policy and a preference represents the consent principle that is made after the respect of the principles previously presented.

Principles must be also considered in the different interactions that could take place in the system. We should insure that the consumer respect the **9. Compliance** principle in these interactions.

The last principle, **7. Safety**, has not to be considered in the agents reasoning since it relates to the system design and is therefore not included in the formalization presented in this article.

The semantics of the principles playing a part in the sensitive data transaction. During sensitive data transaction, the central principle for the agent's reasoning is **1. Purpose specification** (Figure 2).

Table 1. Concept representing HiMAS principles

Principle	Associated Concept
1. Purpose specification	*Purpose* composed by a set of *Objective*
3. Limited Collection	*Collection* composed by a set of *Data*
4. Limited Use	*PossibleUses* composed by a set of *Use*
5. Limited disclosure	*BroadcastingList* composed by a set of *Agent*
6. Limited retention	*RetentionTime*
7. Openness	*Subject* and *Provider* included in *Agent*
2. Consent	*Consent*

For each principle (and for the notion of format[3] that is required in our approach) we define an associated concept in a conceptual graph [17] (refer to Table 1 and to Figure 2). Each principle and the notion of format is represented by a concept linked to another according to a semantic relationship. In order to define these, we use an existential positive conjunctive fragment of the first order logic that allows us not to obtain contradictory logical information. We represent each concept by an atomic predicate and each relationship by a binary predicate. The formal description of the conceptual graph presented in Figure 2 is described in Table 2.

The implementation of this conceptual graph is made by using an OWL file [18]. Figure 3 presents an example of our implementation. We have chosen to present the instantiation of the relationship *isComposedBy* for the concepts *Collection* and *Data*. This approach uses an extensible knowledge representation language, RDF and RDFS. Each associated concept is represented by a RDFS class and each semantic relationship by an OWL property. RDFS gives a vocabulary to RDF that instantiates RDFS classes and properties. So each instantiation (application context-dependent) of these concepts and these semantic links is in a RDF structure in relation to the vocabulary defined by the RDFS.

[3] All the required references.

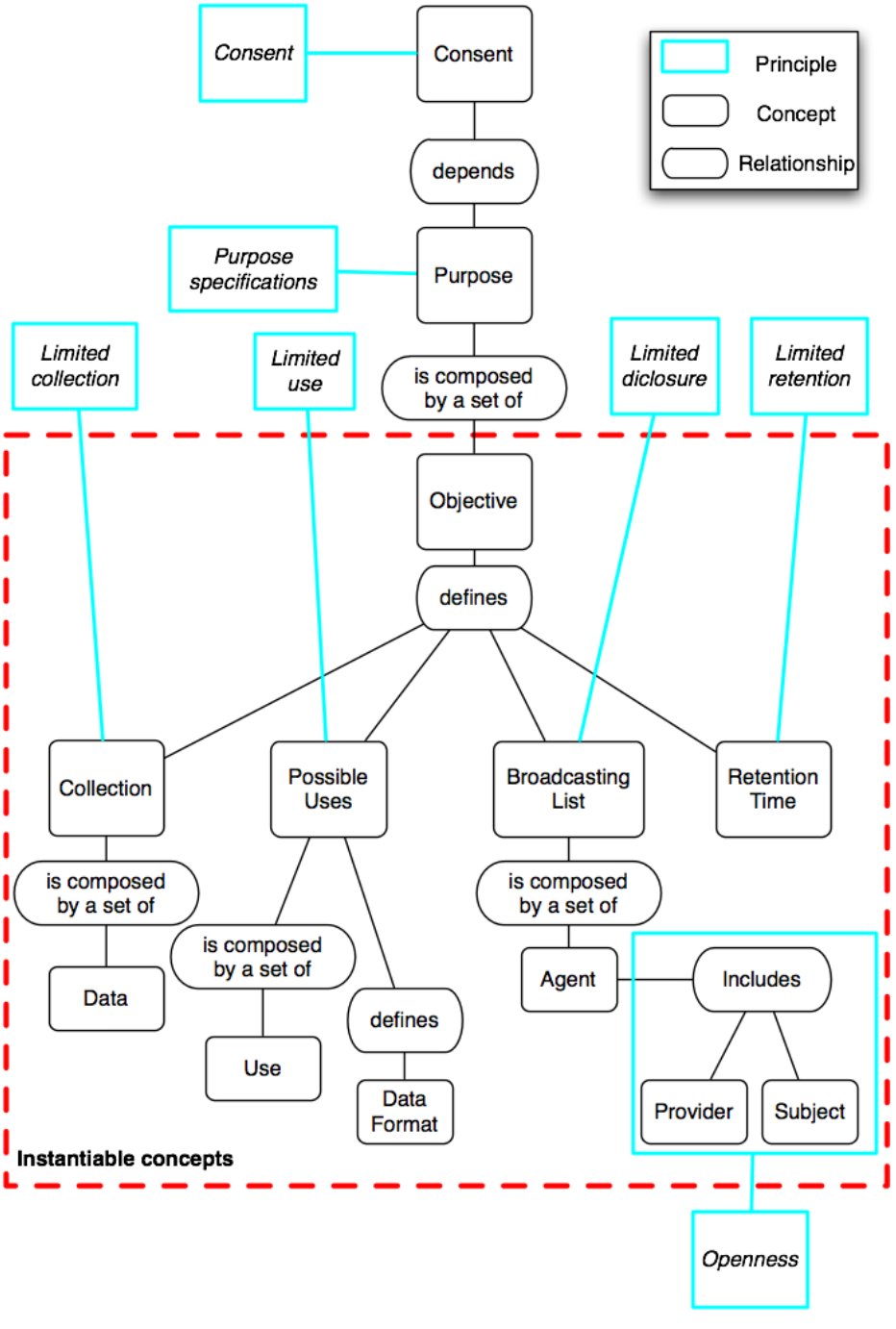


Fig. 2. Conceptual graph representing the semantics of HiMAS principles

Table 2. Principles formalization

	Principle: 1. Purpose Specification
$\forall p\ Purpose(p)$	$\rightarrow\ \exists x\ composedBy(p,x) \wedge Objective(x)$
	Principle: 3. Limited collection
$\forall x\ Objective(x)$	$\rightarrow\ \exists y\ defines(x,y) \wedge Collection(y)$
$\forall y\ Collection(y)$	$\rightarrow\ \exists z\ composedBy(y,z) \wedge Data(z)$
	Principle: 4. Limited use
$\forall x\ Objective(x)$	$\rightarrow\ \exists y\ composedBy(y,z) \wedge PossibleUses(y)$
$\forall y\ PossibleUses(y)$	$\rightarrow\ \exists z\ composedBy(y,z) \wedge Use(z)$
$\forall y\ PossibleUses(y)$	$\rightarrow\ \exists z\ defines(y,z) \wedge Format(z)$
	Principle: 5. Limited disclosure
$\forall x\ Objective(x)$	$\rightarrow\ \exists y\ defines(x,y) \wedge BroadcastingList(y)$
$\forall y\ BroadcastingList(y)$	$\rightarrow\ \exists z\ composedBy(y,z) \wedge Agent(z)$
	Principle: 8. Openness
$\forall z\ Agent(z)$	$\rightarrow\ \exists w\ includes(z,w) \wedge Subject(w)$
$\forall z\ Agent(z)$	$\rightarrow\ \exists w\ includes(z,w) \wedge Provider(w)$
	Principle: 6. Limited retention
$\forall x\ Objective(x)$	$\rightarrow\ \exists y\ defines(x,y) \wedge RetentionTime(y)$
	Principle: 2. Consent
$\forall c\ Consent(c)$	$\rightarrow\ \exists x\ depends(c,p) \wedge Purpose(p)$

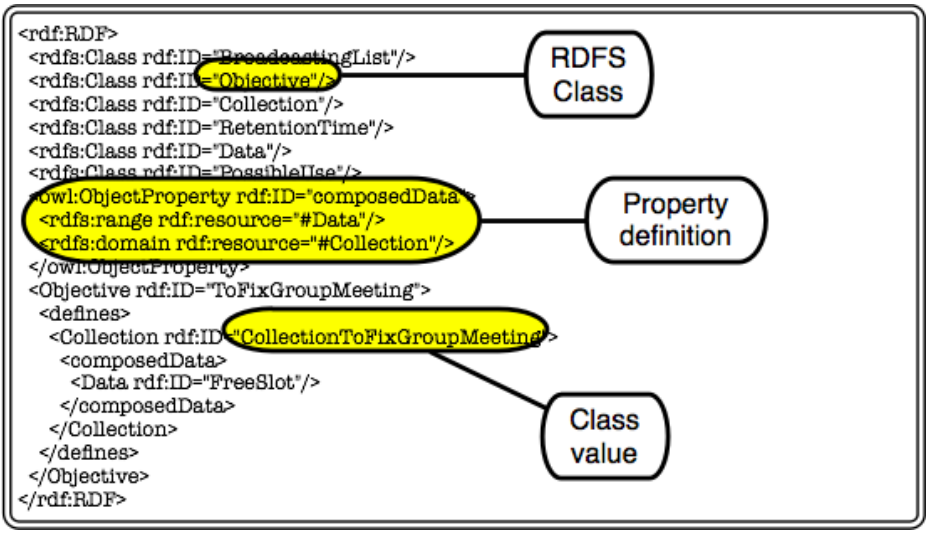


Fig. 3. Example of the conceptual graph implementation

Taking the context of the application into account. HiMAS principles define generic constraints that the agency must satisfy to preserve the private sphere. The previous study semantics that we have just presented, must be linked to the HIMAS application context because of the context-dependent characteristic of

the private sphere. An example of the introduction of the context is presented in more details is section 5.

For this integration, we need to instantiate the defined conceptual graph by giving all the possible values for each concept according to the application context and by linking these values (see the dotted block in Figure 2). These values are represented in a RDF structure (see Figure 3).

We have chosen to not instantiate the possible values of two concepts: consent and purpose. Indeed the value of the consent concept can be true or false. Therefore it can be represented by a boolean and we need only to define the semantic links of this concept for the agents' reasoning. We indicate just that the provider must give or not its consent according to the consumer's purpose. This last concept is composed by a set of objectives. Therefore, by defining all the possible values for the objective concept, we define also all the possible values for the purpose.

3.2 Content Language Syntax

We sum up first all the requirements for sensitive data transaction in a HiMAS represented in Figure 4. Then we present the syntax of such a transaction.

In [1], we have shown that HiMAS agents have to determine risk-taking for a sensitive data transaction. During sensitive data transaction, the consumer (resp. provider) builds its policy (resp. preference) according to its intention. Before building such a transaction, the HiMAS agents pass a judgement on the other HiMAS agents regarding their reliability. For example, this function can be implemented by a processus of trust management like in [19]. If the consumer and the provider are reliable, then the transaction can begin.

We begin the description of the content language elements according to the chronological order of a sensitive data transaction: the design of the policy, the sensitive data transaction and the design of the preference.

Policy. A policy must contain the objectives, the retention date, the broadcasting list and the data format for each asked data (Figure 4 and Table 3).

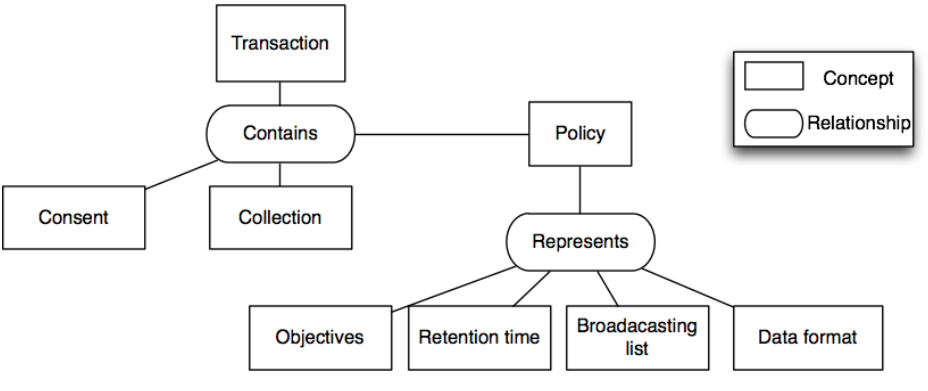


Fig. 4. Sensitive data transaction representation

Table 3. Policy formalization

$\forall y\ policy(y)$	$\rightarrow$	$\exists z\ represents(y,z) \wedge objective(z)$
$\forall y\ policy(y)$	$\rightarrow$	$\exists z\ represents(y,z)\ \wedge format(z)$
$\forall y\ policy(y)$	$\rightarrow$	$\exists z\ represents(y,z)\ \wedge broadcastingList(z)$
$\forall y\ policy(y)$	$\rightarrow$	$\exists z\ represents(y,z)\ \wedge retentionTime(z)$

Once the consumer has determined its objectives and the concepts representing them, it builds a policy syntactically (using an XSD schema) and semantically valid (using an OWL file).

Sensitive data transaction. We have defined in [1] such a transaction set up a policy, a preference, the provider's consent and the sensitive data requested by the consumer. Notice that the formalization presented in Figure 4 does not refer to the provider's preference. Indeed a preference and a policy are based on the same concepts and we represent the provider's preference by the modifications that the provider induces from the consumer's policy if there is no agreement on the constraints defined in the policy.

All values for all elements of the transaction are defined in the content language that allows the consumer to build a valid transaction with regard to the privacy preservation.

Table 4. Sensitive data transaction formalization

$\forall y\ transaction(y)$	$\rightarrow$	$\exists z\ contains(y,z) \wedge consent(z)$
$\forall y\ transaction(y)$	$\rightarrow$	$\exists z\ contains(y,z) \wedge collection(z)$
$\forall y\ transaction(y)$	$\rightarrow$	$\exists z\ contains(y,z) \wedge policy(z)$

In order to build a sensitive data transaction that is syntactically valid, we use the same approach as for the policy. We formally define such a transaction in Table 4 and in Figure 4.

4 Sensitive Data Transaction Protocol

In this section, we present a sensitive data transaction protocol based on the content language previously defined. This approach also allows us to provide a guideline about the design of the policy and preference for the HiMAS agents.

In our content language, the consumer's objectives are semantically linked to the principles playing a part in a sensitive data transaction. This content language includes all the possible values for each class representing one HiMAS principle. The consumer can therefore know if it violates the private sphere or not by verifying that the elements contained in its policy are included in the content language and by verifying that it respects the semantic links between these elements.

The sensitive data transaction protocol that we propose is presented in Figure 5. The content language implementation must be common to all the HiMAS agents so

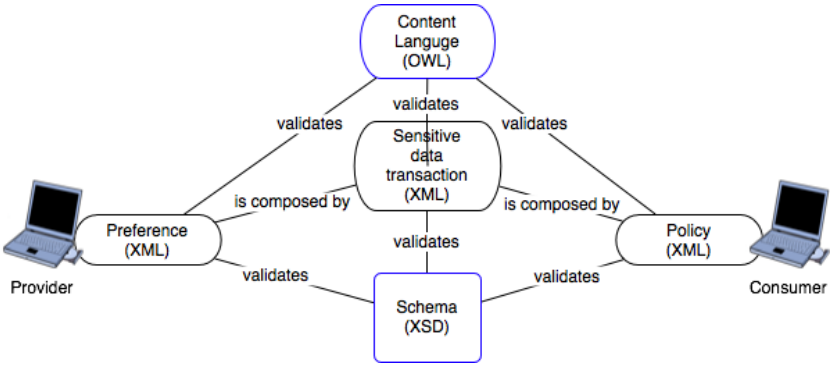


Fig. 5. Sensitive data transaction protocol

that each agent can base its reasoning on the same vocabulary and the same semantics. We have chosen to represent this as external to the agents and available for the consultation by the agency. With this approach, many HiMAS can refer to the same language if their application context is the same. Moreover, this technique allows us to consider the openness between many HiMAS having the same context. At a design level, the possible modifications for this language require only one control entity and there are no propagation problems.

Each consumer and each provider validate their policy and their preference using the content language previously presented in order to build and to execute a sensitive data transaction.

4.1 Steps of the Interaction Protocol

We present now the three steps of the interaction protocol that we propose in a chronological order: the design of the policy, the sensitive data transaction and the design of the preference. These steps are represented in Figure 6.

Design of the policy. A consumer builds its policy according to its objectives by using the content language. In this way, it can be understood by the other agents. Moreover the consumer's behavior respects the private sphere if its policy validates the content language.

A first constraint of our protocol imposes that the XSD file validates the XML file to ensure the syntax of such a transaction.

A second constraint of our protocol imposes that the values of the XML file must be included in the conceptual graph previously defined (see Figure 2) to ensure the semantics.

Sensitive data transaction. Once the consumer has defined and validated its policy, the sensitive data transaction can begin.

To inform the provider about its request, the consumer must build a sensitive data transaction. This transaction contains its policy and must be validated by the content language.

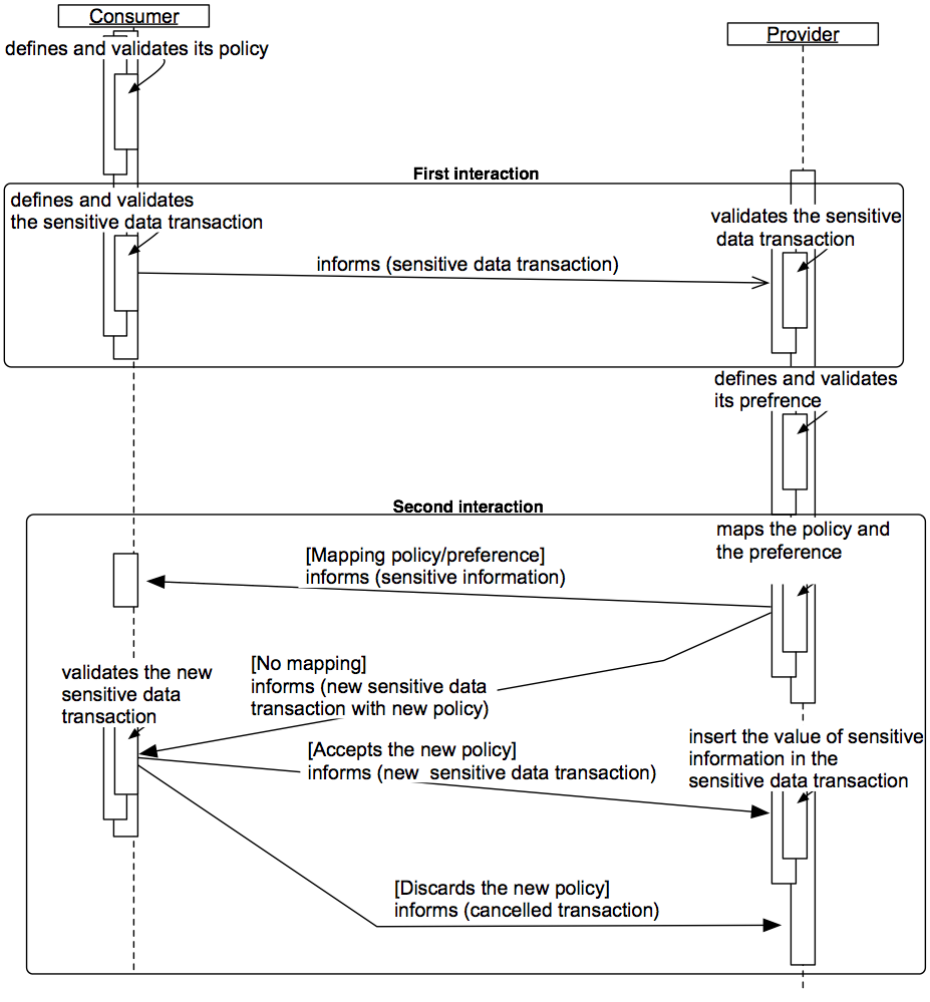


Fig. 6. Sensitive data transaction protocol

Once the sensitive data transaction file built and validated, the consumer can send it to the provider in order that the provider could know its request. This step is represented by the first interaction of Figure 6.

Design of the preference. From the management rules of its private sphere, a provider establishes the conditions of the use, the disclosure, the retention of its sensitive data. Once it received a sensitive data transaction, these rules allow it to accept or not the consumer's policy.

Before analyzing the consumer's policy, the provider must first verify the transaction validity at a syntactic and semantic level, using the content language. These two validations allow to determine if a consumer has a malicious

behavior on the limitations imposed by HiMAS principles and on the sensitive data transaction protocol.

If the sensitive data transaction is validated, then the provider can make a mapping between its preference and the consumer's policy. If no mapping is found, the provider can propose to the consumer some adaptations of its policy.

Once the consumer and the provider have agreed on the policy, the provider completes the transaction with the values of requested sensitive data. If no agreement is found, the transaction is canceled and the provider can not answer to the consumer's request. The second interaction of Figure 6 represents these steps.

4.2 Synthesis

One of the first advantages of this approach is the possibility to verify the constraints defined by the principles of the HiMAS thanks to the content language. The consumer (resp. provider) can design its policy (resp. preference) with respect to the constraints defined by HiMAS principles. This obedience is made by the semantic links between the concepts representing the HiMAS principles.

Each transaction between the consumer and the provider can be represented by the "inform" communicative act of FIPA [20]. Indeed, these two agents exchange only one specific data: a sensitive data transaction that will be completed during such a transaction.

This protocol is provider-centred and is opposite to all the most of transaction protocols that are in general service-centred. It defines the same principles as the P3P [2] and sensitive data transaction as an interaction in ISLANDER [21]. In order to preserve completely the private sphere, this protocol must be integrated in a secure communication medium (principle **7. Safety**) which is not purpose in this paper.

5 Application

In order to illustrate and to implement the HiMAS model and the sensitive data transaction, we consider a decentralized calendar management application [4]. In this context, each user is represented by an agent in charge of the scheduling of events, either tasks or meetings. Timetables can be shared with other agents. When agents do not share their timetables, a negotiation system is necessary to fix the meetings.

In this scenario, the private sphere is managed by the user that delegates his sensitive data to an agent. Users indicate to their agent the basis of the policy for each sensitive data transaction when the agent is the consumer. In the other way, when the agent is the provider, it defines its preference for the sensitive data transaction thanks to rules that are given by the user at the beginning if the experimentations. This aspect is not developed in this article, we focus only on the interaction between agents for a sensitive data transaction.

We have chosen a simple example for the illustration of the sensitive data transaction protocol: a consumer wants to fix a group meeting with a provider

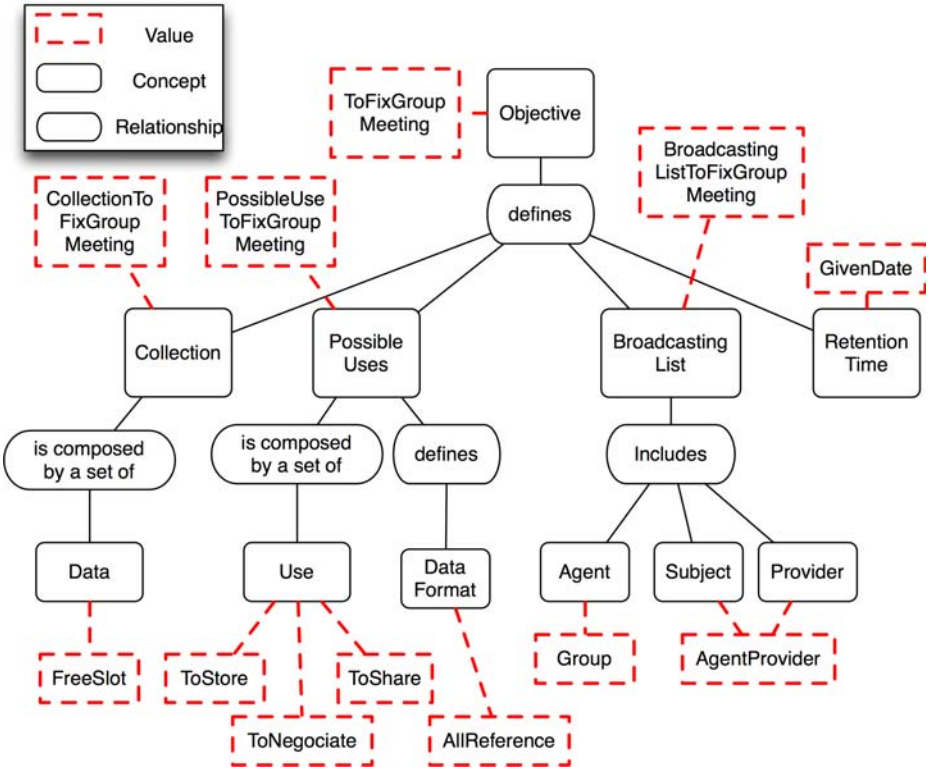


Fig. 7. Content language in context of calendar management and objective "to fix a group meeting"

and other agents (group G) in a given period of time (interval between two slots of time). We consider as sensitive data the free and occupied slots of time in users' calendar. Figure 7 represents this example.

In order to fix such a meeting, we define the following constraints:

- The sensitive data that the consumer can collect is the free slots of time for a given period.
- The consumer can disclose this sensitive data to the group G and it must guarantee that the provider is able to access to this data.
- If the sensitive data was disclosed, all the possible references can be disclosed.
- The consumer can not retain collected data after a given time.
- The possible uses of the collected sensitive data are storage, negotiation and sharing.

The implementation of this HiMAS is made by instantiating the classes of Figure 3 with the values of Figure 7. For example, the class *Objective* is instantiated by the value "ToFixMeetingGroup" and this value defines the value

"PossibleUseToFixGroupMeeting" (composed by the values "ToStore, ToNegociate, ToShare" linked to the class *Use*) for the class *PossibleUses*.

Once the content language is defined, the consumer and the provider can build, according to their intentions, a sensitive data transaction, regarding the privacy preservation. In this scenario, the user gives to his agent the objective of the sensitive data transaction and the agent guides him for the policy creation in terms of required data, retention time, broadcasting list end references set with regards to the privacy preservation.

For this, the consumer agent builds its policy by parsing the content language. It first finds the objective corresponding to the goal "to fix a group meeting". After it chooses the values of its policy among the values proposed in the content language for its objective and sends a sensitive data transaction to the provider. These values are chosen in function of the user's needs.

The provider check the policy thanks to the content language in order to verify the consumer intentions. If it agrees with this policy, it informs the consumer of the required sensitive data. Else it can modify the consumer's policy by other values of the content language, according to its preference, and it informs the consumer of its modification. In this case, the consumer accepts or not this new policy.

6 Related Work

The principles playing a part in the sensitive data transaction allow HiMAS agents to define their policy and their preference. This vision can be associated with the policy about policy that are the metapolicies. We propose in this section a global vision of this notion in order to present its main aspects.

Metapolicies are a notion introduced by Hosmer in [22,23] that describe this like a set of policies about policies. These metapolicies are used in order to define a set of rules and assumptions about the policies of security in a given system for the policies interaction coordination.

Some other works use this notion like Kühnhauser [24] that uses metapolicies for the interfacing and the cooperation of complex policies, and for conflict resolution between the security policy. An other kind of work is the PONDER system [25,26], where metapolicies are used in order to describe the security policies and to resolve the conflicts.

Generally the main objective of metapolicies is to define and to manage a set of policies of security for a given system regarding to the resolution of conflicts.

HiMAS principles define guidelines for the agents' reasoning about their policy and preference. These principles represent metapolicies for the agents behavior in relation to the communication and the manipulation of sensitive data. However the policy in our study case is not the same as in the work about security. HiMAS principles allow the agents to reason about a set of behavior constraints and do not allow to manage the set of agents' policies. We may link these principles to the notion of metaknowledge introduced by Pitrat [27].

7 Conclusion and Perspectives

Our sensitive data transaction protocol allows us to apply seven HiMAS principles: **1. Purpose specifications**, **2. Consent**, **3. Limited collection**, **4. Limited use**, **5. Limited disclosure**, **6. Limited retention** and **8. Openness**. This protocol is generic and can be personalizable according to the kind of sensitive information that is exchanged.

The obedience to these principles consists in the consideration of our protocol at two levels. The first one is the definition of the content language. These principles are semantically and syntactically defined in a content language. The second one represents the use of the content language by the agents to build a sensitive data transaction.

The semantic links between the HiMAS principles allow us to determine in a content language, the maximal set of the sensitive data processing that a consumer can do on the collected data. A provider can also verify if a consumer respects the principles that limit the collection, the use, the disclosure and the retention, by referring to the content language. To ensure that all the principles are taken into account, we also formalize the sensitive data transaction that contributes to the malicious agent detection (agents that do not adhere to this formalization).

The content language of our protocol solves the main problem of the P3P [28]. Indeed, the mapping between a policy and a preference based on the same content language, a provider is able to understand the consumer's intention contrary to the P3P where this mapping is not guaranteed. Another advantage is the possibility to define the limitations imposed by HiMAS principles.

As a perspective, we want to focus on the principle of **9. Compliance** which is related to the problem of the interaction between agents. A first hint would be to implement a social order [29] in relation to the judgment function of HiMAS agents. We plan to implement this function using some trust management technics. The formalization and the implementation of this principle aim us to take temporal aspects into account. Indeed, the social order that we propose is based on the protocol presented in this article and allows us to study the interactions dynamic with regards to the privacy preservation.

Acknowledgments. This work is supported by Web Intelligence project, financed by the ISLE cluster of Rhône-Alpes region. We thank France Telecom R&D for supporting the research related to trust mentioned in this paper.

References

1. Crépin, L., Vercouter, L., Jaquenet, F., Demazeau, Y., Boissier, O.: Hippocratic multi-agent systems. In: Proceedings of the 10th International Conference of Entreprise Information Systems, pp. 301–308 (2008)
2. W3C: Plateform for privacy preferences (2002), http://www.w3.org/p3p/
3. Cranor, L.F.: Web Privacy with P3P. O'Reilly, Sebastopol (2002)

4. Demazeau, Y., Melaye, D., Verrons, M.-H.: A decentralized calendar system featuring sharing, trusting and negotiating. In: Ali, M., Dapoigny, R. (eds.) IEA/AIE 2006. LNCS, vol. 4031, pp. 731–740. Springer, Heidelberg (2006)
5. Thomson, J.J.: The right of privacy. Philosophy and Public Affairs 4, 295–314 (1975)
6. Demeulenaere, P.: Difficulties of private life characterization from a sociologic point of view. In: Privacy in Information Society, vol. 11 (2002)
7. Baase, S.: A Gift of Fire: Social, Legal, and Ethical Issues in Computing. Prentice-Hall, Englewood Cliffs (2003)
8. Westin, A.F.: Special report: legal safeguards to insure privacy in a computer society. Commun. ACM 10(9), 533–537 (1967)
9. Warren, S.D., Brandeis, L.D.: The right to privacy. Wadsworth Publ. Co., Belmont (1985)
10. Lessig, L.: Code and Other Laws of Cyberspace. Basic Books, New York (2000)
11. Bellotti, V., Sellen, A.: Design for privacy in ubiquitous computing environments. In: Proceedings of the European Conference on Computer Supported Cooperative Work (ECSCW), pp. 77–92. Kluwer Academic Publishers, Dordrecht (1993)
12. Palen, L., Dourish, P.: Unpacking "privacy" for a networked world. In: Proceedings of the 2003 Conference on Human Factors in Computing Systems, pp. 129–136. ACM, New York (2003)
13. Rezgui, A., Ouzzani, M., Bouguettaya, A., Medjahed, B.: Preserving privacy in web services. In: Chiang, R.H.L., Lim, E.P. (eds.) Proceedings of the Workshop on Web Information and Data Management, pp. 56–62. ACM, New York (2002)
14. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Hippocratic databases. In: Proceedings of the International Conference Very Large Data Bases, pp. 143–154. Morgan Kaufmann, San Francisco (2002)
15. Bratman, M.E.: Intention, plans, and practical reason. O'Reilly, Harvard University Press, Cambridge (1987)
16. Sichman, J.S., Demazeau, Y.: Exploiting social reasoning to deal with agency level inconsistency. In: Proceedings of the First International Conference on Multiagent Systems, pp. 352–359. MIT Press, Cambridge (1995)
17. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading (1984)
18. W3C: Owl web ontology language (2004), http://www.w3.org/tr/owl-features/
19. Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: P2P-based collaborative spam detection and filtering. In: Proceedings of 4th International Conference on Peer-to-Peer Computing, pp. 176–183. IEEE Computer Society, Los Alamitos (2004)
20. FIPA: Fipa communicative act library specification (2002), http://www.fipa.org/specs/fipa00037/index.html
21. Esteva, M., de la Cruz, D., Sierra, C.: Islander: an electronic institutions editor. In: Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems, pp. 1045–1052. ACM, New York (2002)
22. Hosmer, H.H.: Metapolicies I. ACM SIGSAC Data Management Workshop 10(2-3), 18–43 (1991)
23. Hosmer, H.H.: Metapolicies II. In: Proceeding of the 15th National Computer Security Conference, pp. 369–378. Elsevier Advanced Technology Publications, Amsterdam (1992)
24. Kühnhauser, W.E.: A paradigm for user-defined security policies. In: Symposium on Reliable Distributed Systems, pp. 135–144 (1995)

25. Lupu, E., Sloman, M., Dulay, N., Damianou, N.: Ponder: Realising enterprise viewpoint concepts. In: Proceeding of the 4th International Enterprise Distributed Object Computing Conference, pp. 66–75. IEEE Computer Society, Los Alamitos (2000)
26. Twidle, K., Lupu, E.C.: Ponder2 - policy-based self managed cells. In: Bandara, A.K., Burgess, M. (eds.) AIMS 2007. LNCS, vol. 4543, p. 230. Springer, Heidelberg (2007)
27. Pitrat, J.: Métaconnaissance, Futur de l'Intelligence Artificielle. Hermés (1990)
28. Thibadeau, R.: A critique of P3P: Privacy on web (2000), dollar.ecom.cmu.edu/p3pcritique/
29. Castelfranchi, C.: Engineering social order. In: Omicini, A., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2000. LNCS, vol. 1972, pp. 1–18. Springer, Heidelberg (2000)

Part III

Agent-Oriented Software Engineering

ADELFE Design, AMAS-ML in Action

A Case Study

Sylvain Rougemaille, Jean-Paul Arcangeli, Marie-Pierre Gleizes, and Frédéric Migeon

IRIT - SMAC
118, route de Narbonne
F-31062 Toulouse Cedex 9
{sylvain.rougemaille,jean-paul.arcangeli,marie-pierre.gleizes,
frederic.migeon}@irit.fr

Abstract. The complexity of engineers tasks leads us to provide means to bring the Adaptive Multi-Agent Systems (AMAS) design to a higher stage of automation and confidence thanks to Model Driven Development (MDD). This paper focuses on a practical example and illustrates the modifications that have been done to the ADELFE methodology. In the *Design* phase, we propose to use a Domain Specific Modeling Language (DSML) for the specification of cooperative agents. We also, add a Model Diven *Implementation* phase using model transformation, DSMLs and code generation. These phases carry out a model centric process to produce and partially generate the system code. We present the use of our MD process applied to a simple, but very illustrative example: the foraging ants simulation.

1 Introduction

Our team works both on adaptation and Multi-Agent Systems, the result is that we propose paradigms to manage adaptation at different conceptual levels. We propose an approach which introduces adaptation following three independent axes [1]. The first one differentiates system level adaptation, achieved according to AMAS principles [2], from agent adaptation, allowed by a flexible agent architecture [3]. The second axis distinguishes functional adaptation (which concerns the system expected functionality, i.e. the service performed) and operational adaptation (which concerns execution mechanisms, i.e. means to perform services independently of the functionality itself). Finally, the third one concerns adaptation time. Adaptation is qualified as dynamic when it occurs at runtime and static when it occurs at design time. As the system is designed to provide a function for the user and that it is responsible for that, system level adaptation deals with means to preserve dynamically the adequacy between the function the system offers and user requirements. Concerning agent adaptation, it is important to notice that, as long as agents execute, they may encounter various operating systems configurations. Therefore, flexible agent architecture is a way of defining and maintaining agent skills up-to-date in order to keep it playing its role.

The combined capacities of these approaches, AMAS principles and flexible agent architecture, enable to deal with systems which can be characterized as complex, due to the complexity of the domain (coupling with the environment, numerous interacting entities) or the one coming from the execution layer. Our proposal is to ease the

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 105–120, 2009.

design of such systems by combining different adaptation kinds (system/agent, functional/operational) within a tool that would assist the engineers all along the design. This assistant would reduce domain complexity by automating the implementation of the system, letting engineers focus on business concerns. Moreover, complexity of the execution support would be totally hidden thanks to generative tools.

This is the goal of our research, in which we try to combine several software technologies such as reflection, aspect orientation, components, software architectures for implementation issues, as well as AMAS which ease to handle system complexity. In order to make all these technologies cooperate, we use a model driven approach that allows us to integrate modelling and implementation tasks in a common environment, such as Eclipse. All these ”good practices” and principles are specified and gathered in a methodology called ADELFE, which is a development process based on the RUP (Rational Unified Process) and specialised for AMAS developing.

In this paper, we present a practical example of the join use of both AMAS and flexible agent principles within the ADELFE *Design* and *Implementation* phases applied to a simple, but very illustrative example: the foraging ants simulation. The following of the paper is organised as follows. First is presented the context of this paper: section 2 for the ADELFE methodology and its adaptation to a MD approach and section 3 for the case study. Thereafter, the paper focuses in section 4 on the several phases where model transformations and code generations are used. In section 5, we analyse the work presented according to engineers points of view. Finally, we discuss some related works and lastely we conclude.

2 ADELFE 2.0

ADELFE[1] is an agent-oriented methodology for designing Adaptive Multi-Agent System (AMAS) [2]. The MAS developed according to ADELFE provides an emergent global function [4]. What we call the global function is the function the system is in charge of, whereas what we call local function is one provided by one agent. The global function is qualified as emergent because it is not coded inside the agent. The agents are not aware of this global function. Let’s take the example of the robot transportation application developed with ADELFE [5] where agents have to transport boxes from a room to another one by passing through narrow corridors (agents cannot pass each other). The agents have to move in an environment containing 2 rooms, 2 corridors, boxes, walls, others robots. Each agent’s local behaviour consist in avoiding collision and in trying to be cooperative. Being cooperative means for the agent maximising its utility in the system. Therefore, it tries to avoid situations of concurrency, uselessness, ambiguity and other kind of conflicts. The global phenomena not coded inside the agent is that a traffic direction emerges. To obtain this emergent behaviour, the system follows

[1] ADELFE is a French acronym for ”Atelier de Développement de Logiciels à Fonctionnalité Emergente”. It was a French RNTL-funded project (2000-2003) which partners were: ARTAL Technologies (http://www.artal.fr) and TNI-Valiosys (http://www.tni-valiosys.com) from industry and IRIT (http://www.irit.fr/SMAC) and L3I (http://www-l3i.univ-lr.fr) from academia. See http://www.irit.fr/ADELFE

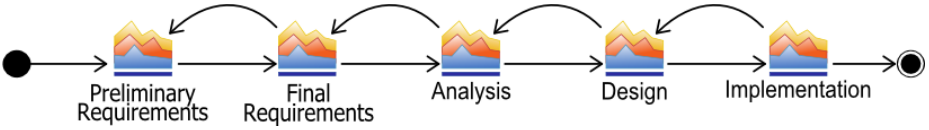


Fig. 1. The ADELFE 2.0 phases

the AMAS theory [6] in which the agents are endowed with the ability to autonomously and locally modify their interactions in order to react to changes in their environment. These alterations transform their collective function i.e. the global function performed by the MAS they belong to. This system is self-organising and is able to adapt to its environment. According to the AMAS principles, interactions between agents depend on their local view and on their ability to ”cooperate” with each other. Every internal part of the system (agent) pursues an individual objective and interacts with agents it knows by respecting cooperative techniques which lead to avoid Non Cooperative Situations (NCS) like conflict, concurrence etc. Facing a NCS, a cooperative agent acts to come back to a cooperative state and permanently adapts itself to unpredictable situations while learning on others.

2.1 Adelfe 1.0

The ADELFE agent-oriented methodology aims at guiding AMAS designers through a development process based on the RUP (Rational Unified Process) [7], a standard process of object-oriented methodology. ADELFE covers the phases of usual software design from the requirements to the design; it uses UML notation and extension of UML already done in AUML, in particular the AIP (Agent Interaction Protocols) notations [8]. Our aim is not to add one more methodology to existing ones but to work on some aspects not already taken into account such as complex environment, dynamics, and adaptation. As this methodology concerns only applications designed following the AMAS principles, some activities or steps have been added to the RUP in order to be specific to adaptive multi-agent systems. In the preliminary and final requirements, the environment modelling and the expression of the situations that can be “unexpected” or “harmful” for the system have been added. In the analysis phase, two activities are dedicated to the AMAS. First, ADELFE helps the designer to decide if the use of the AMAS principles is required to implement his application. ADELFE provides also guides to identify cooperative agents among all the entities defined during the final requirements. Concerning the design phase, three activities are added. The first concerns the relationships between agents. The second is about the agent design. In this activity, the cooperation failures are defined. Then, a fast prototyping activity helps to build and verify the agent behaviour.

2.2 Extending Adelfe 1.0

Rationale. The design phase of ADELFE was previously carried out using UML1.4 profile, to take into account cooperative agents and their specificity. Moreover, the AUML AIP has been extended to integrate cooperation failure. However, since its last

version, UML2.0 [9] has integrated many of the desired features of the FIPA for AUML, making a step further in the AIP direction (adding the concepts of Combined Fragments to the sequence diagram, for instance). As a consequence, the profiles based on the previous UML version was kind of "deprecated notations". In the purpose of updating the ADELFE methodology, we begun a metamodeling process to characterise as precisely as possible the concepts involved in the AMAS principles and mandatory for ADELFE. With this metamodel, we made the choice of developing our own DSML (Domain Specific Modeling Language) [10], considering that the AMAS approach constitutes a domain on its own. We called this language AMAS-ML (AMAS-Modeling Language). All along the metamodeling process, we had in focus that this specific language would be used in the ADELFE methodology for the purpose of specific design. Besides this fact, the abstraction and the concepts that it brought have been used to initiate a model driven implementation phase.

It is important to notice what are the advantages of using a DSML (whether it is a profile of UML or obtained from Model-Driven approach). The main benefit is about semantics. What can be expressed by designer becomes closer to the concepts of the domain whereas the use of a general-purpose language (like UML or a OO programming language) introduces a gap between ideas and description. Another advantage of Model-Driven Engineering lies in automation. By extracting the information the designer has already given in previous diagrams, model transformations allow not only to speed up development but especially to reduce design complexity, which is inherent to the systems we deal with. For example, we are able to separate behavioural or functional concerns form operational ones which will be implemented transparently, keeping designer focused on business concerns (see figure 2).

In the ADELFE V.2, the design phase has been improved and an implementation one has been added.

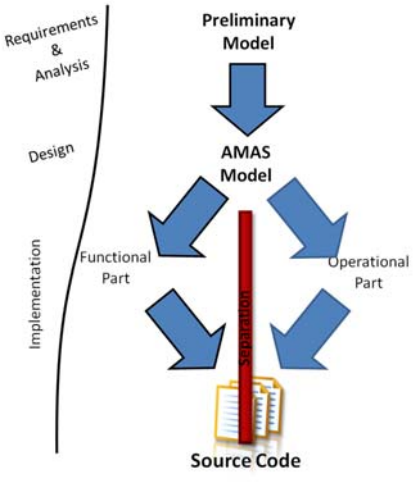


Fig. 2. Separation of concerns

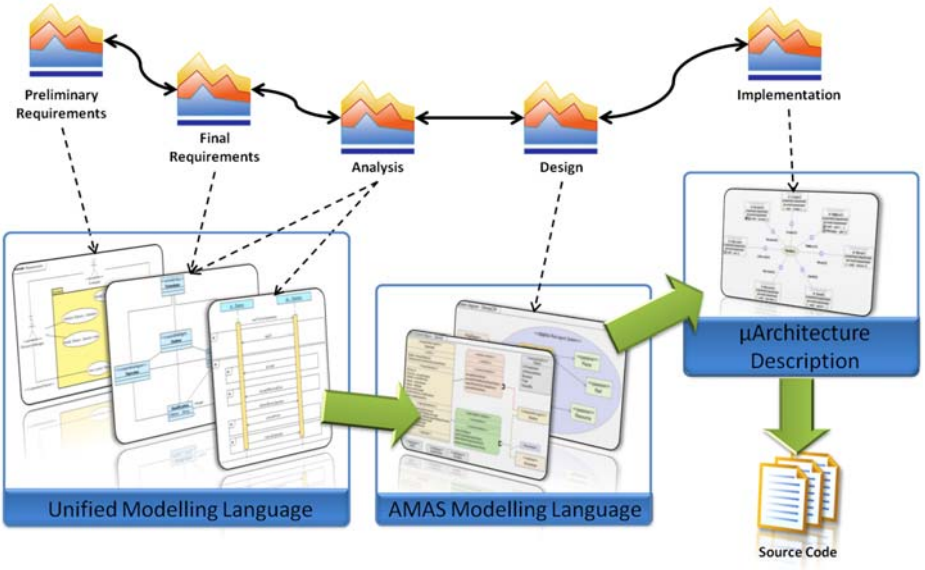


Fig. 3. Phases and diagrams in ADELFE

Design. AMAS-ML is used in several steps of the design phase, from the detailed agent structure design to the definition of the agent cooperative behaviour. This is done thanks to specialised diagrams:

- The agent diagram: it is used to model the cooperative agent structure, as well as its relationship with environmental entities. It defines all the specific features of a cooperative agents, its *Representations*, *Characteristics*, *Skills* and *Aptitudes*.
- The behavioural rules diagram: it allows the specification of rules involved in the decision process of an agent. It is expressed with the cooperative agents features. Based on their representations and perceptions, agents have to decide next actions to lead. These actions may be done in the purpose of NCS recovering (*CooperativeRule*) or not (*StandardBehaviorRule*).
- The interaction diagram: for the moment it corresponds to the UML 2.0 sequence diagram. We have defined a transformation which allows us to integrate the protocols and messages defined in the UML model into our AMAS-ML model. However, we are studying the interest of developing our own diagram editor.

The next implementation phase takes as input the result of the design, that is the AMAS-ML model (see figure 3).

Implementation. As we have presented it in [1], this phase is guided by one main idea: the separation of concerns. More precisely, we want to separate all that constitutes the "operating" concerns (basic mechanisms of the agent), from all specific behaviour concerns (the way agents use their tools to achieve their goals). To do so, we based this phase on a specific tool which we have developed: MAY (Make Agents Yourself).

It allows developper to describe agent micro-architecture (operating mechanisms) thanks to a DSML: μADL (micro-Architecture Description Language). The architectural style of the micro-component assembly and the MAY generation process give a kind of "abstract agent machine" (we could say a application-dedicated API) which can be used by the developer as an abstract layer to implement the behaviour of agents. This phase involves several generation or transformation steps which are illustrated in figure 4 with SPEM 2.0 (Software and system Process Engineering Metamodel) [11]:

- *Micro-Architecture Extraction* : this is the first model to model transformation, from AMAS-ML to μADL, which has been implemented with ATL [12]. It eases the *Agent Architecture Analyst* tasks by creating a μADL model from the AMAS-ML model elements that we consider as "operating mechanisms" (see further section 4.3).
- *Abstract Micro-Architecture Code Generation* : this first step of code generation gives code skeletons. Once the architecture is sufficiently refined and consistent, the *Java Developer* may implement micro-components services.

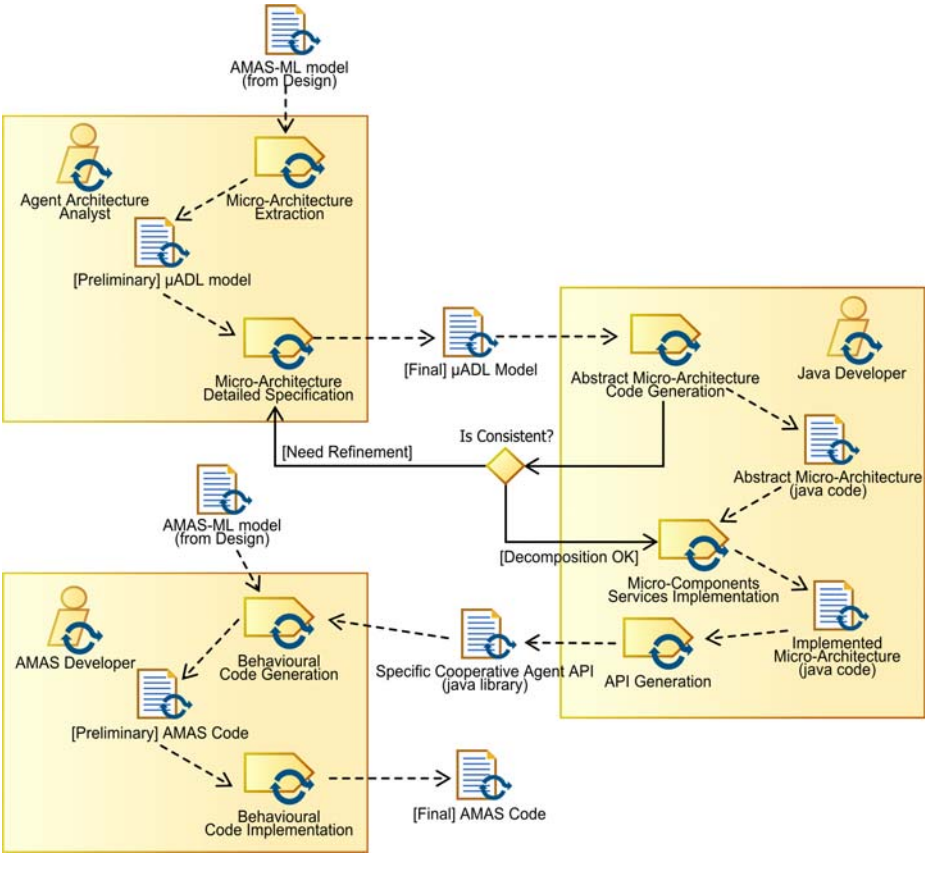


Fig. 4. The ADELFE 2.0 Implementation phase in SPEM 2.0

– *API generation* : at this point, MAY generates the whole API, that is, tools to execute, create and deploy the specific agent models.
– *Behavioural code generation* : from the behavioural rules expressed in the design phase with AMAS-ML, we proposed to generate a code skeleton to ease the task of the AMAS developer. The aim is to provide some hints to achieve the decision process of each agent in the system.

3 Case Study: The Foraging Ants

ADELFE has been used to develop a simulation of foraging ants, on one hand, for providing a tool for ethologists and on the other hand, for testing that cooperative ants following AMAS approach provide correct results. The application was chosen because the behaviour of foraging ants is quite simple and allows to focus on development techniques. The environment is composed of the nest, some obstacles, pheromone, patches of food and ants. The pheromone self-evaporates during time and can be accumulated when several ants drop pheromone at the same place. The foraging ants have several characteristics. They have different degrees of perception for obstacles, other ants, food, and pheromone. They always know where their nest is located. They can carry a given quantity of food. They go out of the nest for a given duration and at the end of this duration, they go back to the nest and rest in the nest for an amount of time. The foraging ant behaviour consists first in exploring the environment. When it encounters an obstacle, it avoids it. When it encounters food, it can harvest it. When it is loaded, it goes back to nest in dropping a given quantity of pheromone on the ground. By consequence, tracks of pheromone appear in the environment. During its exploration, an ant is attracted by pheromone and leads to follow pheromone track. This behaviour implies a reinforcement of the existing tracks.

4 Applying Adelfe 2.0

This section depicts the way the application described above can be implemented thanks to ADELFE and its model driven implementation phase. Thus, we focus mainly on the last steps of the methodology, the first ones are summarised as they do not constitute a new proposition.

4.1 Preliminary Steps: Requirements

These steps are devoted to the establishment of requirements and are usual in software development methodology. They consist in a description of the problem domain as it is demanded to be solved, as well as a specification of the final user needs. The first phase, namely the *Preliminary requirements* phase, has already been completed. Although it is not formalised, the brief description of the section 3 could be considered as its result. It constitutes an overview of the requirements (user needs, key-words and limits).

Concerning the next phase, *Final requirements*, it is involved in the description of the system environment and in the identification of the different elements which populate it. From the requirements previously established we determine the following entities:

- Passive Entities (resources for the system): the pheromone, obstacles, the food and the nest,
- Active Entities (entities that could act autonomously): foraging ants.

Furthermore, from the requirements already presented we have characterised the environment of the system as:

- Accessible: its state is known by the system (simulation purpose);
- Non-deterministic: ants actions could have different results;
- Discret: as a simulated environment it is defined as a grid;
- Dynamic: ants actions modify continuously its state.

From a more common point of view the *Use cases* identified for the system are all related to the management of a simulation tool: configuration of the ants parameters and observation of the results, and so on. Thus they are not extensively exposed in this paper.

4.2 Analysis

After we have described the requirements, we proposed a first analysis which is intended to allow us to determine whether an AMAS approach is convenient or not. Fortunately it does! In fact, from global to local point of view and focusing on the ants activity observation, we can say that:

- There is no obvious way to define the task of the colony;
- The global task (food gathering) is performed by a great number of interacting entities;
- These entities are conceptually distributed;
- The environment is evolving during time;
- Each ant possesses a limited perception of its environment, as well as a limited range of actions. Moreover they have to adapt themselves to an ever changing context.

By analysing these few sentences, it seems that the AMAS approach is particularly well adapted to our problem. It also seems obvious that the agent in the previously identified entities could be none but the ant, in fact:

- it is the only entity possessing an autonomous activity and trying to reach a personal goal (harvest food);
- it has a partial vision of its environment, which moreover is evolving;
- it has to deal with other entities and thus with potential cooperation failures.

To sum-up the results obtained at this phase, we have determined that AMAS approach is appropriate to the problem we want to solve and thanks to the requirements too, we have identified the agent within the AMAS: the foraging ant. Since this point, and for the following steps we are focussing on the design and implementation of the cooperative agent. To do so, we have adopted a model driven approach rather than a code-centric one.

4.3 Model Driven Design and Implementation

The beginning of the ADELFE methodology is based upon the RUP which is intrinsically bound to the UML notation. Information contained by the model resulting of the preliminary steps of the method is necessary for the following phases. But, we also assume that UML models aren't as specific as we want models to be for the design of cooperative agents. To cope with this lack of specialisation, we have proposed our own DSML (Domain Specific Modeling Language) based on an AMAS meta-model and called AMAS-ML (AMAS-Modeling Language) [1]. However, this choice does not prevent us for bridging UML preliminary models with AMAS-ML, in fact, we gather their information to feed our AMAS specific model thanks to transformations. Furthermore, we use UML 2.0 sequence diagrams [9] to specify agent protocol [13] as well as entities interaction, we extract from it the relevant information thanks to model transformations to. The next section presents the models which have been defined in the scope of the foraging ants simulation tool design.

Agent Diagram. For the precise design of the agents, we use the AMAS-ML agent diagram (see figure 5 for details). According to the cooperative agent paradigm, an agent is made up with several parts and modules. They represent its main abilities or specificities such as : *Representations*, *Characteristics*, *Skills* and *Aptitudes*; they also represent the way it interacts with its environment: *action module*, *perception module*, *action*, *perception* and the means it involves *actuators* and *sensors*. In our example, an ant does not use direct communication; it only deposes pheromone which could be sensed by other ants (stigmergy), that is why no communication action could be noticed in the figure 5. The perception consists in filling the different *representations* with fresh values. For instance, the *food* array corresponds to the position where food has been perceived. With these gathered positions and its skills (*favour()*, etc.), the ant agent has to determine the better way to go. To do so, it fills the *interpretedSurroundings* grid with integer values (the more appealing a position is, bigger is the integer value). This decision process consists in the choice of the favourite positions. It is expressed as rules and is described in the next section. As it has been presented in section 3, the ants have to depose pheromone tracks in order to communicate the place where food has been discovered. This task involved the use of a specific action *dropPheromon()* which is fulfilled thanks to the *ExocrynGland* actuator. This information is useful for the "extraction" of the agent architecture, it indicates which part of the agent is responsible for the performing of an action (see section 4.3). The other specific features of the *ForagingAnt* cooperative agent are shown in the figure 5 those features are used in the next design step.

Behavioural rules diagram. We distinguish two kinds of rules the *Standard behaviour* which constitutes the local function or goal assigned to an agent; and the *Cooperative behaviour* rules which are intended to manage *Non Cooperative Situations*. At this step in the methodology, we are designing these rules as being triggered from an agent state, which is itself characterized by a logical expression over the *Representations*, *Characteristics* and *Skills* of the agent. A rule result in a set of actions, or skills that have to be accomplished in order to reach a local goal or to recover a cooperative state. The figure 6 shows an example of a graphical representation of those rules. The left

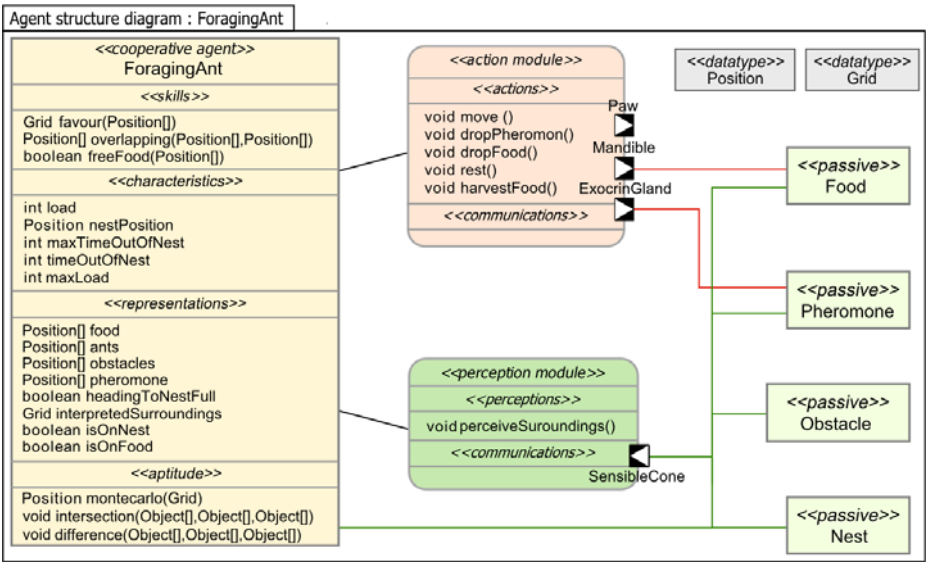


Fig. 5. AMAS-ML Agent Diagram: foraging ant cooperative agent detailed design

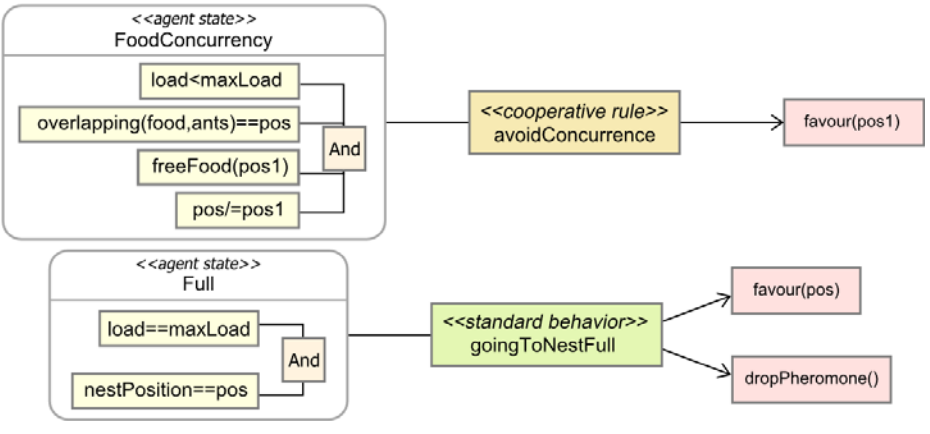


Fig. 6. AMAS-ML Behaviour Rules Diagram: concurrence avoidance cooperative rule and returning

hand side presents states that triggered the actions at the right hand side. The rule binds a state with actions and is labelled by a rectangle (in the middle of the figure) which is also used to specify the kind of behaviour it is related to, cooperative or standard. Actually, the figure 6 presents the avoidance of concurrence that could appear when ants are lusting for the same food patch and the standard ant behavior that consist in going back to the nest while deposing a pheromone track when food has been collected (in fact, destinations are only favoured as a Monte-Carlo algorithm introduces non-determinism in moves).

μADL. From the design phase and the AMAS-ML agent diagram, we propose to generate automatically an abstract agent architecture. This architecture is made up with micro-components which specification could be edited and modified with the μADL language. The result of this step is called the "Agent model"; it is used to generate a specific API which is given to the developer in order to complete the following stages of this phase. In the figure 7, the ant cooperative agent is shown as the result of a model to model transformation from the AMAS-ML model of the figure 5. From this point, another model driven tool is used to proceed the last generation step, which is called MAY (Make Your Agents Yourself); it is described in the next section.

MAY results. MAY generates an dedicated API providing the agents modelled thanks to μADL. In our case the agent model has been extracted from the results of the AMAS-ML design (see 4.3). The abstract agent architecture has to be implemented, by reusing micro-components or developing new ones. In our case, we choose to re-use micro-components devoted to the interaction with a grid with a graphical representation which was developed for a previous project. Once this task is completed, MAY can generate the agent specific API that will be used for the development of the agent behaviour. Of course, designer may also combine generation steps and manual modifications. Compiler makes uses of Java interfaces to specify architcture while implementation is done with classes. MAY generates all interfaces features and ask the user to decide which classes are implementation in order to avoid conflicts.

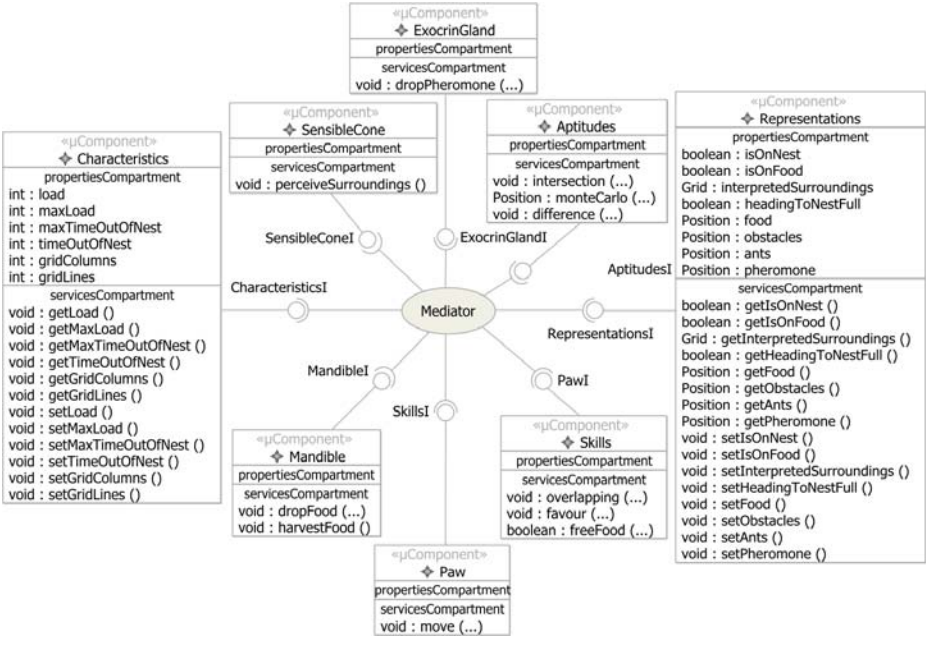


Fig. 7. μADL diagram of the Ant cooperative Agent

4.4 Application Code

At this stage, the work that still have to be done by the developer is the implementation of the ant agent behaviour. We already possess some interesting information concerning this behaviour which is contained by the AMAS-ML model (see section 4.3). There is another code generation step which allows us to generate code skeletons and hints for the implementation of the decision module. Concerning our example, the ant decision module consists in detecting patterns which modify the value of the interpretation grid. This grid is given to the *monteCarlo* aptitude, which selects the next position to be, thanks to a randomised algorithm. Thus the decision of the ant can be summarised as the ponderation of this grid as well as the positioning of the *headingToNestFull* boolean, thereafter the ant moves to the randomly determined position dropping pheromone if necessary. We implemented the *decide* and the *act* methods wich are called by the *LifeCycle* micro-component.

5 Experiments Analysis

From the example presented here, and even if it does not constitute a "real world" or industrial software development experiment, we can draw some conclusions at different levels.

5.1 From the Designer Point of View

We have not presented here, for space saving convenience, the detailed design phase as it has been done for the implementation phase. However, the use of the AMAS-ML diagram has shown its interest in the cooperative agents design. Actually, the expression of behaviour as rules over the agent knowledge and characteristics has naturally induced an incremental and iterative process in the precise design of agents features and behaviour. Thus these two tasks benefit one from the other. For example, while expressing the cooperative behaviour of an agent a designer could need some new useful skills. Conversly the adding of elements to the agent during detailled design could lead to new states that need to be handled by new behavioural rules.

5.2 From the Developer Point of View

The introduction of a model driven phase has brought a higher level of automation to the AMAS development. Developers profit from model driven tools which help them in the production of agent oriented software, domain they are probably not familiar with. In fact, MAY offers to developers a way to produce their own agent oriented API with a minimum of effort. In the mean time, this code generation process can still be manually conducted by a MAS expert who would control every part of his/her code. The implementation phase is a real model-driven process which keeps models and code consistent.

To give concrete values,we can emphasize the following results. It took 3 days to develop the entire prototype shown in figure 8 where only half a day was spent for behavioural part. Ant API is only 53ko weight, with 17 classes and 9 interfaces. For environment, 29 classes were designed for a total of 69ko. Finally, behaviour and main are contained in 2 classes (6ko).

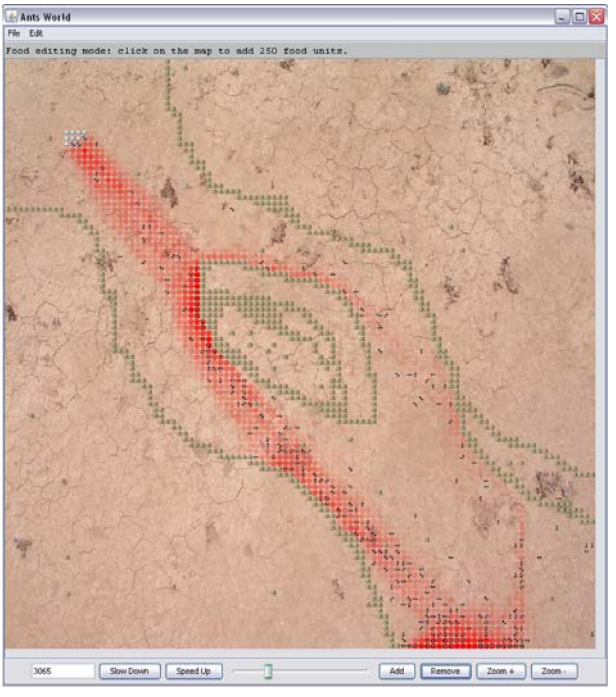


Fig. 8. Ant simulation prototype

5.3 From the Method Engineer Point of View

We advocate that the implementation phase, thanks to its model driven approach, only depends on the input "domain" model (AMAS-ML model in our case). Consequently, it could be considered as a method fragment [14], parameterized by the domain model and of course the associated transformation. One can object that this transformation could be a problem, but we assume that MDD already offers means to assist its definition[2]. Furthermore, the target model, namely μADL, offers a reduced set of concepts that could be mapped easily. However, transformation generation still constitutes a challenging issue in the MD world.

6 MAS and MDE Related Works

Currently, some existing agent-based methodologies INGENIAS [15], PASSI [16], and TROPOS [17] use model transformations in order to design MAS. These methods and the associated tools coming from MDE are reviewed and briefly analysed in this section. Few works on MAS engineering have involved the use of tools coming from MDE, and the most advanced are: MetaDIMA [18], INGENIAS, TROPOS and SODA [19].

[2] http://www.eclipse.org/gmt/amw/

MetaDIMA helps the designer to implement MAS on the DIMA platform using MetaGen which is a MDE tool dedicated to the definition of metamodels and models. DIMA is a development and implementation platform developed in Java where agents are seen as a set of dedicated modules (perception, communication, etc.). MetaDIMA provides a set of metamodels and a set of knowledge-based systems on top of DIMA to ease the design of MAS by providing languages more specific than Java code.

INGENIAS proposes to transform the MAS expressed in the INGENIAS metamodel in code dedicated to a given platform using the modelling language of INGENIAS and the implementation model of the platform. Its main originality consists in providing evolutionary tools. Because tools used for transforming specification in code are based on metamodels, if the metamodel specifications evolve, the tools can also evolve. Moreover, these transformations are expressed as templates which also can be tuned for specific purposes.

In TROPOS, all the phases use different models which are described by metamodels; it also uses UML notation and automatic transformations. For example, it translates plan decomposition into a UML 2.0 activity diagram by using a transformation language based on the following three concepts: pattern definition, transformation rules and tracking relationships.

Molesini et al. [19] propose to fill the gap between methodologies and infrastructures by using metamodelling for mapping the abstractions at the AOSE methodology level onto the abstractions at the infrastructure level. They provide guides for mapping SODA concepts onto three different infrastructures: TuCSoN, CArtAgO and TOTA.

Our work pursues the same objective as the works described previously although it addresses adaptation issue from both system and agent points of view. In fact, we aim at taking it into account and providing design and generation tools to implement such adaptive systems. For this purpose, we propose to generate an adapted execution platform for AMAS, using MDE tools and principles as well as the flexibility provided by MAY.

7 Conclusion

In this paper, we have presented an example of the practical use of Domain Specific Languages, model transformations, and code generation in the scope of a dedicated methodology. That is to say, a whole Model Driven Engineering process devoted to the implementation of an AMAS. The benefits of such an approach have been analysed from several points of view, and even if some technical works still have to be done to integrate this approach in a specific tool, we assume that the results are quite satisfying. The implementation phase process still needs some adjustments, nevertheless the experience gained from its further practical application should bring us useful material to do so. Moreover, the ADELFE v.2 methodology has been applied to other projects (for example a Manufacturing Control Simulation) from which we already gather interesting information about the implementation phase. Finally, our team is leading works on the definition of specialised micro-architectures and micro-components that are going to enrich the MAY library and thus favour reusability.

References

1. Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M.P.: Model Driven Engineering for Designing Adaptive Multi-Agent Systems. In: Artikis, A., O'Hare, G.M.P., Stathis, K., Vouros, G. (eds.) ESAW 2007. LNCS, vol. 4995, Springer, Heidelberg (2008), http://www.springerlink.com (online)
2. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 172–202. Idea Group Pub. (2005); ISBN: 1-59140-581-5
3. Leriche, S., Arcangeli, J.P.: Adaptive Autonomous Agent Models for Open Distributed Systems. In: International Multi-Conference on Computing in the Global Information Technology (ICCGI), Guadeloupe, March 4-9, 2007, pp. 19–24. IEEE Computer Society, Los Alamitos (2007), http://www.computer.org
4. Georgé, J.P., Edmonds, B., Glize, P.: Making self-organising adaptive multiagent systems work. In: Bergenti, F., Gleizes, M.P., Zambonelli, F. (eds.) Methodologies and Software Engineering for Agent Systems, pp. 319–338. Kluwer Academic Publishers, Dordrecht (2004)
5. Picard, G., Gleizes, M.P.: Cooperative Self-Organization to Design Robust and Adaptive Collectives. In: International Conference on Informatics in Control, Automation and Robotics (ICINCO), Barcelona, Spain, September 14-17, 2005, pp. 236–241. INSTICC Press (2005), http://www.insticc.net/
6. Capera, D., Georgé, J.P., Gleizes, M.P., Glize, P.: The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In: TAPOCS 2003 at WETICE 2003, Linz, Austria, June 09-11, 2003, IEEE CS, Los Alamitos (2003)
7. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
8. Odell, J., Parunak, H., Bauer, B.: Representing Agent Interaction Protocols in UML. Springer, Heidelberg (2000)
9. Object Management Group, Inc.: Unified Modeling Language (UML) 2.0 Superstructure Specification, Final Adopted Specification (2003)
10. France, R.B., Rumpe, B.: Domain specific modeling. Software and System Modeling 4(1), 1–3 (2005)
11. Object Management Group, Inc.: Software & Systems Process Engineering Metamodel Specification v2.0. Omg edn. (2007)
12. Jouault, F., Kurtev, I.: Transforming models with ATL (atlas transformation language). In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
13. Bauer, B., Odell, J.: UML 2.0 and agents: how to build agent-based systems with the new UML (unified modeling language) standard. Engineering Applications of Artificial Intelligence 18(2), 141–157 (2005)
14. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardisation to research. Int. J. of Agent-Oriented Software Engineering 1, 91–121 (2007)
15. Pavón, J., Gómez-Sanz, J.J.: Agent oriented software engineering with INGENIAS. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS, vol. 2691, pp. 394–403. Springer, Heidelberg (2003)
16. Cossentino, M., Gaglio, S., Sabatucci, L., Seidita, V.: The PASSI and Agile PASSI MAS Meta-models Compared with a Unifying Proposal. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS, vol. 3690, pp. 183–192. Springer, Heidelberg (2005)

17. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
18. Jarraya, T., Guessoum, Z.: Towards a Model Driven Process for Multi-Agent System. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS, vol. 4696, pp. 256–265. Springer, Heidelberg (2007)
19. Molesini, A., Denti, E., Omicini, A.: From AOSE methodologies to MAS infrastructures: The SODA case study. In: Artikis, A., O'Hare, G.M.P., Stathis, K., Vouros, G. (eds.) ESAW 2007. LNCS, vol. 4995, pp. 300–317. Springer, Heidelberg (2008), http://www.springerlink.com (online)

Exception Handling in Goal-Oriented Multi-Agent Systems

Ibrahim Cakirlar, Erdem Eser Ekinci, and Oğuz Dikenelli

Ege University, Department of Computer Engineering,
35100 Bornova, Izmir, Turkey
icakirlar,erdemeserekinci@gmail.com,
oguz.dikenelli@ege.edu.tr

Abstract. Cooperative, autonomous and distributed properties of multi-agent systems deduce the dynamic capabilities of multi-agent system applications. On the other hand, these suitable features increase the error proneness of these applications. In this paper, we propose an exception handling approach to make multi-agent system applications more reliable and robust. And also we classify multi-agent exceptions and have implemented our approach on SEAGENT goal-oriented multi-agent development framework[1].

1 Introduction

One of the most important properties that make a software technology applicable to industrial settings is error proneness. Traditional technologies, such as object oriented paradigm, provide an infrastructure to develop reliable software applications that have great degree of error proneness maturity. Robust technologies can detect, diagnose and recover from failures and uncertain situations [6,11]. To recover failures and uncertain situations, initially it's clear that there is a requirement of identifying errors and deciding what should be done to recover. Exception handling approach should be enriched with primitive properties of robust technologies, exactly recovery. According to the general characteristic of agents, multi-agent systems are expected to be behaving robustly. But for the reason of deficient definition of exception handling mechanism, multi-agent systems (MASs) are far away from robustness behaviour. This problem is an important obstacle that frustrates MASs to meet the industry.

Nowadays, developed software applications are trying to satisfy the robustness requirement by extending exception handling mechanisms, which are provided by programming languages [4]. Exception handling is based on the exception concept that is defined as an error occurred during execution flow of a program [4,9,12,20]. Programming languages give chance of recovering exceptions by their exception handling mechanisms. When an exception occurs, programming language manages the deviation from the normal execution flow of the program to

[1] These research is supported by The Scientific and Technological Research Council of Turkey, Electric Electronic and Information Research Group with 106E008 project.

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 121–136, 2009.

the handler, which is implemented by developer. To make MASs robust, such an exception handling approach must be redesigned according to the general characteristics of the agents.

As known, multi-agent systems consist of cooperative, distributed and autonomous software entities, named agents [19]. These primitive properties prevent direct usage of classical exception handling approaches in MASs [13]. Distributed but cooperative work of agents mean spreading the program over agent organization and execution flow of the program is managed by separate agents at different times. Handling an exception of this distributed execution flow requires to reconstruct the classical handling approach. Moreover, it must be considered that how the autonomy of agents is effected by implementation of such a distributed handling approach.

In this research we aim to develop an exception handling approach for goal-oriented MASs. In this type of MAS, cooperation and internal intents of agents are modelled with goals. In other words, goals are used to define distributed execution flow, spread over agents, and standalone execution flow of an agent. In our handling approach, we advocate modelling exception handlers with specific goals to recover exceptions. In addition, exception of an agent may not be recovered by that agent and this exception may hamper whole agent organization to achieve organizational goals. For handling such an exception, cooperation of agents may be required. In [18], Lamsweerde and Letier also claim using goals to model exception handlers at the requirement analysis phase of goal-oriented MAS development. But at design time and runtime only goals are not enough for handling exceptions of goal-oriented MASs. Plans are another important artifact of goal-oriented MASs that is used to define detailed internal execution flow of agents to achieve agent goals. So, exceptions of plans should be considered. In this research, an infrastructure for handling plan exceptions is also defined.

This paragraph defines how the rest of this paper is organized. In section 2, we determined our approach. In sub-sections of section 2, we classify exceptions of goal-oriented MAS and conveniently, an abstract architecture for handling these types of exceptions is designed. The next section contains development details of this architecture which is implemented within SEAGENT Goal-Oriented Multi-Agent System. Section 4 touches on a case study that shows how efficient the implemented architecture works. In section 5, we criticize other exception handling techniques for MASs. Finally we conclude the paper in Section 6.

2 Exception Handling Mechanism for Goal-Oriented Multi-Agent Systems

To explicate proposed MAS exception handling approach, first we have to classify multi-agent exceptions then we must define an approach to handle the instances of these classifications. Exceptions are generally specified as deviations from the regular execution flow of a program. In MASs, execution flow is expressed in terms of cooperation and individual. So, to classify MAS exceptions that cause deviations, entities directly used in different types of execution flows have to be specified.

In a goal-oriented multi-agent system, execution flow of agents (individual and cooperation) is designed by system goals, agent goals and plans as directly used executable entities. System goals are used to define execution flow of MAS from the cooperation perspective. System goals model the cooperation of participant agents to achieve organizational aims by defining roles, goals, and communication protocols between these goals. System goals also composed of agent goals. Agent goals are used to define internal execution flow of an agent. Although goals are used to specify agent and MAS execution flow, they are not enough for defining detailed execution flow on their own inside. At this point, plans are required to specify execution details of an agent to achieve agent goals.

Our MAS exception handling approach is inspired with the idea of modelling exception handling mechanism on the simplicity principle [3]. Conveniently to this principle, we claim that each exception should be handled with the executable entities that are same level with the exception occurred in. If it can not be handled in the same level, it hampers the execution flow of one upper level. So, it should be handled in the upper level. In other words, if individual execution flow of an agent crashes, initially it must be tried to fix internally by agent. If it can not be fixed and causes crash of cooperative execution flow, it has to be compensated cooperatively by participant agents to provide robustness of MAS.

As adverted previously, an agent's internal execution flow is designed with agent goals and plans. In particular, plans define the execution flow more specifically than agent goals and agent goals defines what to do for agent in more generic perspective. According to our approach, if a plan crashes, it must be handled by another plan. If a plan level exception is not handled, the goal that the agent want to achieve by executing this plan also can not be achieved. Similarly, exceptions of agent goals may be handled via other agent goal(s).

Through agent organization perspective, execution flow of cooperation that is designed by system goals can be broken down for the reason of irregular behaviours of participant agents. In this circumstance agent organization tries to make new decisions to achieve its organizational goals. These exceptions can only be handled via cooperative handling.

2.1 Exception Levels

We categorized exceptions of multi-agent systems due to hierarchy of executable entities defined above. In our approach, according to the executable entities of MAS, exceptions are classified in three levels; plan level, agent level and MAS level exceptions. In[13], Platon also defines a classification for MASs. In his research, there are two classification levels; agent level and code level. This classification is extended by adding one more upper level of MAS level exceptions. These entities and our exception levels are shown in Table-1 and following paragraphs define these levels in detail.

– **Plan Level Exceptions:** Agent performs plans to achieve designed agent goals in its life cycle. Along life cycle of an agent, plans can crash for the reason of the logical, implementation, system, communication or knowledge-base errors. In detail, a plan may be implemented inadequately according

Table 1. Classification of MAS Exceptions

Exception Level	Entity	Standalone	Cooperative
MAS Level	System Goal	-	X
Agent Level	Agent Goal	X	X
Plan Level	Plan	X	-

to domain requirements or programming requirements such as unexpected parameter types or values. System that executes the plans, planner of the agent in our case, may not work properly and cause exceptions. Additionally un-reached or improper messages according to the communication protocol also make a plan crashes. Finally, the knowledge that is used during execution of a plan can cause an exception for the reason of inconsistent operations with the knowledge-base. All these types of exceptions are classified in Plan Level Exceptions for the reason of deviations in the execution flow of plans.

- **Agent Level Exceptions:** Agents try to achieve agent goals corresponding to their objectives. An agent may not be able to achieve its goals for several reasons. For example, an agent may decide to achieve an agent goal. Along life cycle of agent knowledge-base errors, unhandled plan exceptions and system errors may frustrate the agent to achieve the specified agent goal. Occurrence of several exceptional situations hampers the execution of agent goal. For example occurrence of an uncertain situation in agent beliefs, unreached agent messages, agent system failures generate exceptional situations. All these type of exceptions are classified in Agent Level Exceptions for the reason of deviations in the execution flow of agent goals.
- **Multi-Agent Level Exceptions:** In MAS, agents cooperate to achieve system goals and these system goals are modelled in a specific order. During the cooperation for achieving a system goal, an agent goal can be broken down for the reason of exceptions occurred lower levels. Erroneous execution of one of the participant agent can prevent the achieving of the organizational goal. All these type of execution errors are classified in Multi-agent Level Exceptions and represent upper level exceptions that are occurred during this cooperation between agents.

2.2 Abstract Exception Handling Architecture of MAS

To determine exception handling mechanism for agents, the autonomy characteristic should be considered. Agents may have choices to recover exceptions according to their preferences in their knowledge-base. So, the exception handling architecture has to be supported with the decision making mechanism of an agent. To define an exception handling architecture, how a goal-oriented MAS works to make decisions should be determined. In goal-oriented MAS, agents cooperate to achieve a system goal conveniently with their agent goals. So, execution of a system goal is started soon after one of the participant agents, called as initiator agent, desires to fulfil it's agent goal that is also designed as sub-goal

of the system goal. After deciding which agent goal to take in execution, plans that can achieve the agent goal should be matched. Concurrently, other participant agents of the system goal starts to work when the initiator agent's request reaches to cooperate. Participant agent also do the same decision making actions with initiator one.

In the life cycle of the agent, agents' planner should take place in exception handling process as well as decision making process. The handling process should spread over three phase of agent cyclic execution model; perception, reasoning and action [14]. In perception phase, planner of the agent should observe the execution of plans and agent goals, which are in execution. After detection of an exception for an agent goal or plan, it should decide what to do for handling this exception. It may find proper plans instead of crashed ones by reasoning. If there are no implemented plans to compensate the execution flow, the agent planner should select a new agent goal instead of one with crashed plan. All these actions are related to adjust the internal execution flow of the agent. If an agent can not succeed to correct the internal execution flow, the cooperation for achieving organizational (system) goal can be frustrated for the reason of one of the participant agent's crash. In this circumstance, the agent with the crashed plan should take in execution of the other system goal related with crashed one.

To provide robustness of MAS application within above mentioned exception handling process, developers of the MAS application should define the alternative goals and plans as well as ordinary goals and plans. Figure 1 illustrates a partial view of goal-oriented MAS meta-model that focuses on exception handling. As seen in figure, we extend the general meta-model of MAS by adding special relationships to the semantics of goal and plan concepts. These special relationships can be listed as *sameAs*, *inverseOf* and *exceptional*. The agent planner should make decision by watching out these semantic relations when an unexpected state occurs. At such state, the planner firstly should follow existing *exceptional* relations that are defined for recovering agent from exceptional state. If there is no designed exceptional goal or plan for exceptional state, it should try to find alternative goals by querying *sameAs* relations instead of crashed one. Defined *sameAs* goal aims the same objective with the crashed one. So the exceptional state of the agent can be disappeared by trying the same objective

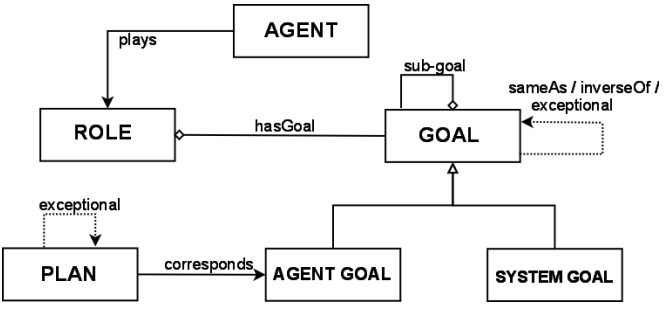


Fig. 1. Executable Entities Meta model

with another goal. Lastly *inverseOf* goal should be queried to roll back the unexpected state. These inverse goals correspond opposite objective with the crashed one for rolling back the effects.

3 Implementation of Proposed MAS Exception Handling Approach

Previously defined abstract exception handling architecture is implemented in SEAGENT Multi-Agent Development Framework [1]. To clarify how the implemented exception handling architecture works, execution life-cycle of SEAGENT planner and used artifacts at run-time must be defined in detail. The primitive properties of SEAGENT are semantic web support, goal orientation and HTN based plan execution. SEAGENT planner [2] supports these features using semantically defined artifacts (with OWL ontologies) that are stored in the knowledge-base of agents.

In detail, SEAGENT planner uses semantically described goal and HTN ontologies. The HTN ontology resembles to the HTN structure presented in Sycara et. al [16]. In the HTN formalism, plans, which are composed to achieve a predefined agent goal, consist of two types of tasks: complex tasks called as behaviours and primitive tasks called as actions. Each plan has a root task, which is a behaviour itself consisting of sub-tasks. On the other hand, actions are primitive tasks, which are directly executable. Next, main concepts of goal ontology are system goal and agent goal. System goal specify the organizational aims and agent goals define the intents of agents.

During the decision making process, all SEAGENT planner internal modules use aforementioned concepts defined in goal and HTN ontology. Life-cycle of decision making process is shown in Figure 2. The decision making process starts with an objective that may be sent from outside of the agent or generated by listening internal events. After an objective is reached, then the planner's goal resolution module queries the ontologies in the knowledge-base to decide what to do. Soon after deciding on a goal, the plan resolution takes place in the planner and queries the knowledge-base to find proper plan for achieving the found goal. Nextly, the planner generates a graph that represents the execution flow of goals and plans in the reduction phase. After the graph generation, it binds the graph to the current execution process and executes the nodes (represent goals and plans) of graph.

Exceptions that are occurred during the execution of the graph are directed to the planner as recover objective. Initially, plan resolution module queries the knowledge-base to find the proper handling plan to recover exceptional state. Found plan that is defined exceptional in plan ontology is selected for the handling process. The reduction module generates graph of the recovery plan and this graph is bound to the execution dynamically. Otherwise, if there is no defined handling plan for recovering exceptional plan (that means the intended goal by the plan is also crashed), then the goal resolution module seeks an alternative goal for handling the crashed goal, which is aimed to fulfil with the

Fig. 2. SEAGENT Planner Decision Making Life-cycle

crashed plan. At this time, if the goal resolution module finds an alternative goal for handling, reduction module converts found goal to graph at reduction cycle and deviated inner model replaced with the new one to fulfil exception handling by dynamic binding.

The goal and HTN ontologies are extended to support reasoning ability of SEAGENT planner for exception handling process. To define handling exceptions that are occurred in plans, an exception property is added to HTN ontology. Figure 3 shows partial view of HTN ontology that focuses on exceptions. Agent plans consist of tasks, and also tasks consist of sub-tasks. *Exceptional* property makes the definition of handling task between possible agent plan's tasks.

```
<owl:class rdf:about="behaviour">
 <rdfs:subclassof rdf:id="task"/>
</owl:class>
<owl:class rdf:about="action">
   <rdfs:subclassof rdf:id="task"/>
</owl:class>
<owl:objectproperty rdf:about="subtask">
   <rdfs:range rdf:id="task"/>
   <rdfs:domain rdf:id="behaviour"/>
</owl:objectproperty>
<owl:objectproperty rdf:about="exceptional">
   <rdfs:range rdf:id="task"/>
   <rdfs:domain rdf:id="task"/>
</owl:objectproperty>
```

Fig. 3. HTN Ontology

On the other hand, into the SEAGENT goal ontology, *exceptional*, *sameAs* and *inverseOf* properties are added for defining the exceptional handling goals. The ontology is illustrated in Figure 4. *Exceptional* property is used for defining exception handling goals. The *SameAs* property is sub-property of *owl:sameAs*, and used for defining agent goals that achieves same objective with the corrupted one. And *inverseOf* property is sub-property of *owl:inverseOf* and expresses agent goals achieving opposite objective to roll back effects of the crashed one.

```
<owl:Class rdf:about="Goal">
  <rdfs:subClassOf rdf:ID="MasEntity"/>
</owl:Class>
<owl:Class rdf:about="AgentGoal">
  <rdfs:subClassOf rdf:ID="Goal"/>
</owl:Class>
<owl:Class rdf:about="SystemGoal">
  <rdfs:subClassOf rdf:ID="Goal"/>
  <rdf:type rdf:resource="owl-Class"/>
</owl:Class>
<owl:ObjectProperty rdf:about="Exceptional">
  <rdfs:domain rdf:resource="Goal"/>
  <rdfs:range rdf:resource="Goal"/>
  <rdf:type rdf:resource="owl-ObjectProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverseOf">
  <rdfs:subPropertyOf rdf:resource="inverseOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="sameAs">
  <rdfs:subPropertyOf rdf:resource="sameAs"/>
</owl:ObjectProperty>
```

Fig. 4. Goal Ontology

Details of exception handling in graph structure are specified on an example in Figure 5. The example based on a system goal SG1 corresponds to an organizational objective of Role1 and Role2. SG1 consist of agent goals AG1 and AG2 related with aforementioned roles. B1 is the root behaviour of the plan corresponds to the agent goal AG1. The right side of the figure illustrates graph model of the B1. The model includes the normal and exceptional execution flow of the B1. The dotted arc between A1 and A3 sub-tasks of B1 emphasize the exceptional execution flow. This link has specific semantic that shows reduction of the behaviour on exceptional cases. Although the link relates two HTN actions in this case, it can be defined on any type of tasks.

When an exception occurs, during the execution of the A1, the execution of whole model, for B1, is aborted to start the recovery process. Plan resolution module selects A3 plan for handling the exception. Reduction module converts A3 to graph and dynamic linking module replace A2's graph model with A3's one. The execution of the recovered graph model for B1 has to be continued from the deviation point to fulfil the handling task.

The extended planner determines the handling level from lower to higher. If the exception perception is about for an erroneous plan, the plan resolution module queries agent knowledge-base to find a proper plan for the recovery of the aborted one. But, if there is no way to fix the crashed plan, that means intended goal crashed, then the goal resolution module queries agent knowledge-base to find a proper agent goal to recover the aborted one. This process is

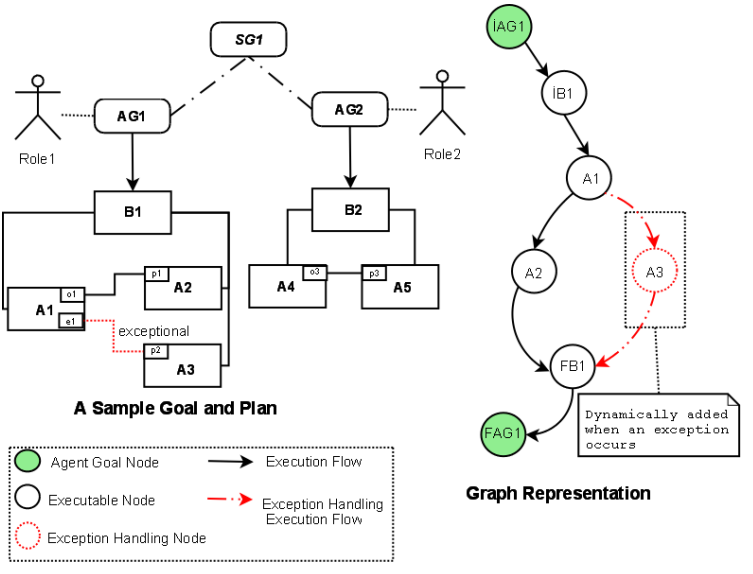


Fig. 5. Planner Executable Entities Sample

executed by reasoning *exceptional*, *sameAs* and *inverseOf* in sequence. The goal resolution module firstly queries agent goals that handle the exception, defined in goal ontologies with *exceptional* property. Afterwards the agent goal for the same objective with crashed one, defined with *sameAs* property tries to be found. And lastly the agent goal achieves rolling back the effects of crashed one, defined with *inverseOf*, tries to be reasoned. Thereafter, the discovery of finding the proper agent plan for selected agent goal is started. As a result of goal resolution, a proper agent plan for the exception handling agent goal, is converted to graph model. The deviated inner model of the aborted agent goal is replaced with the new one.

For example agent goals AG1 and AG2 are sub-goals of the SG1 system goal as shown in Figure 5. When an exception occurs during the execution of the graph model for AG1, execution of the whole graph is aborted to start recovery process. If the goal resolution module can not find a proper agent plan, for recovering B1, tries to find a proper agent goal instead of AG1. As a result of reasoning process an agent goal that is defined as *exceptional*, *sameAs* or *inverseOf* with AG1 recovers the AG1 from the corruption. Proper agent plan for handling goal is converted to the graph and dynamically replaced with the AG1s' graph model.

At higher level, if the exception perception is about for an exceptional agent goal, goal resolution module queries agent knowledge-base to find a proper system goal for the recovery of the aborted one. System goals related with the crashed one is queried with the *exceptional*, *sameAs* or *inverseOf* order previously defined. As a result of reasoning, selected system goal's sub-goals is started to handle the occurred exception during cooperation. Figure 5 depicts the

composition of the system goal SG1. If an exception occurs during the execution of the SG1 sub-goal AG1, execution of the whole graph for AG1 is aborted to start recovery process. If goal resolution module can not find a proper agent goal for AG1, tries to find a proper system goal for SG1. As a result of reasoning process a system goal that is defined as exceptional, sameAs or inverseOf with SG1, recovers both the AG1 and AG2 from the corruption. The exception handling sub-goals related with each role of SG1 is about to be started to start the recovery process of SG1.

4 Case Study

An electronic barter application is implemented with SEAGENT Multi-Agent Framework as a case study. In this application, base scenario is achieved by the *Customer*, *Barter* and *Cargo* roles assigned to the agents. Every agent is named via active role that is playing in the application. Customer agents are responsible for adding, evaluating barter proposals. The Barter agent manages all trades in the system. This agent is responsible for collecting barter proposals, matching proper barter proposals and tracking the bargaining process between Customer agents. After finalization of bargaining, Customer agents send engagement message to the Barter agent. Then, the Barter agent notifies the Cargo agent for transporting barter products between Customer agents. This scenario is completed by the acceptance of all participant agents. The goal model of this scenario is shown in Figure 6.

As seen in the goal model, main goals of the *ExchangeBarterProducts* system goal is illustrated. ExchangeBarterProducts system goal consists of five main agent goals; *prepareBarterProposal*, *evaluateBarterProposal*, *tradeBarterProposal*, *matchBarterProposals* and *organizeBarterProductTransport* goals, which are assigned to the *Customer*, *Barter* and *Cargo* roles. To achieve *ExchangeBarterProducts* goal, initially Customer role achieve its *prepareBarterProposal* goal to request trade from Barter role. Barter role collects these proposals with *tradeBarterProposal* and tries to find proper trades with *matchBarterProposals* goals. After finding convenient trades, the Barter agent notifies the related agents that play the

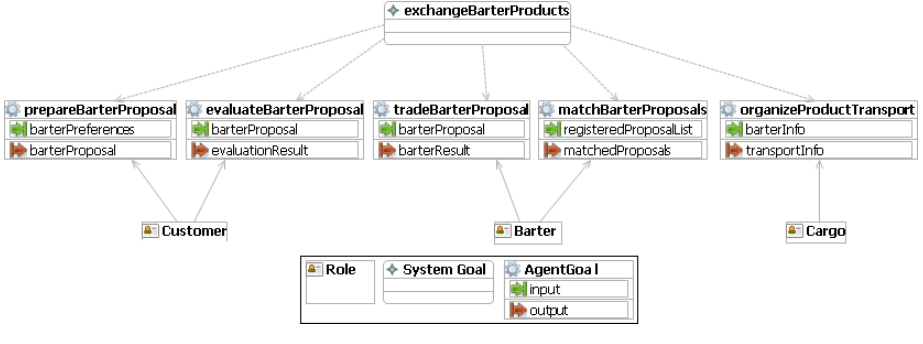


Fig. 6. Case Study Goal Model

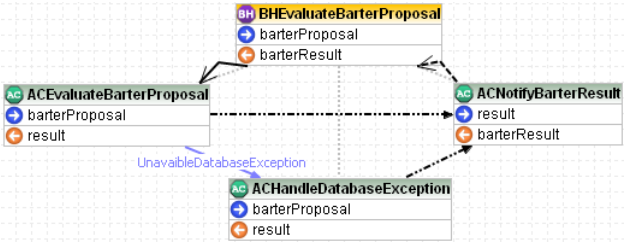


Fig. 7. Plan Level Exception Definition

Customer roles to start bartering. Then agents playing Customer roles communicates with each other to achieve *evaluateBarterProposal* goals. Then Barter role communicates with Cargo role for the transport of exchanged barter products. The transport information is notified by all participants and the execution of *Exchange-BarterProducts* system goal is completed by engagement messages of Customer roles.

During trading process, Barter agent matches proper barter proposals and start bargaining between Customer agents. Figure 7 illustrates agent plan that corresponds *evaluateBarterProposal* agent goal. During evaluation process, existing product database can be unavailable to satisfy the requests. Here we implement a user-defined java exception, named *UnavaibleDatabaseException,* that corrupts the execution of the *BHEvaluateBarterProposal* plan if database is unavailable. To cause the *UnavaibleDatabaseException*, our planner modules do not commit the changes in the knowledge-base that is performed by the crashed task. After this operation over knowledge, the recovery task that handles the occurred exception, *ACHandleDatabaseException*, is dynamically added to the model to provide the robustness of the plan. As a result of execution, database becomes available by opening the connection to the database. At the end of exception handling process, the plan is recovered from the exceptional situation then agent normally continues executing the plan and accepts or refuses the barter proposal.

Let us assume that the same exception**,** *UnavaibleDatabaseException*, occurred during the execution of *evaluateBarterProposal* agent goal and not handled at plan level. If the exception is forced to be thrown, Customer agent tries to handle occurred exception within plan level initially. But it can not find proper plan for recovery then it seeks exception handling goal definitions related with *evaluateBarterProposal.* As a result of reasoning *withDrawBarterProposal* agent goal, which is defined as *exceptional* in the goal ontology, is matched to handle occurred exception and this goal is dynamically added to the Customer agent to fulfil exception handling. This agent goal recovers bargaining between customer agents and starts a new objective corresponding withdraw of the barter proposal. Consequently bargaining between agents are recovered by withdraw. Customer roles' withdraw request triggers *cancelProposal* goal of Barter role. This agent goal terminates the bargaining process between customer agents and withdraws the proposal from the barter system.

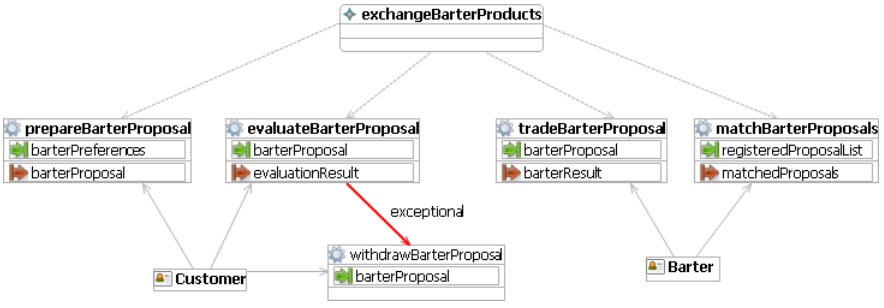


Fig. 8. Agent Goal Level Exception Definition

Lower level exceptions can be handled within the internal life cycle of agents. But all of the exceptions can not be handled via locally. In our case after the bargaining process between Customer agents, Barter agent determines the transport of the barter products. The Barter agent requests the transportation information for the products from the Cargo agent. If an exception occurs during the execution of *organizeProductTransport*, the barter process between Customer agents can not be completed successfully. Transportation of barter products can not be accomplished due to various reasons such as inconvenient weather condition at the date of the transportation. This exception can only be handled cooperatively because the crash of the *organizeProductTransport* agent goal, make other participant agents goals crashed. The reasoning process produces a decision that the deviation can only be handled via system goal named *CancelProductExchange* that is defined as *inverseOf* in the goal ontology related with the crashed one. When Cargo agent percepts this recover objective, it cancels the execution of current goal and starts the execution of proper agent plan related with itself in the definition of *cancelTransport* agent goal. The execution of *cancelTransport* goal within the Cargo agent triggers cooperation with participant agents. Afterwards, the Barter agent starts the execution of *cancelBarterMatching* goal to cooperatively cancel the barter between Customer agents' via *cancelBarterProposalEvaluation* goal.

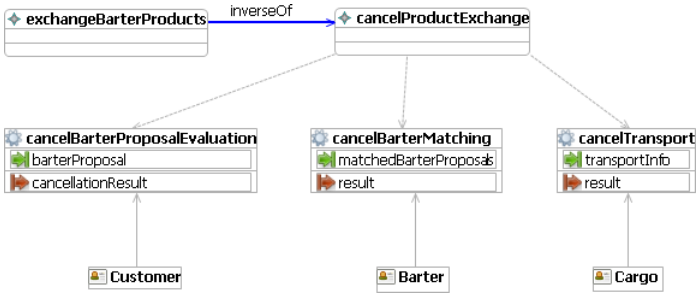


Fig. 9. System Goal Level Exception Definition

5 Related Works

Up to now, agent researchers have worked on different exception handling mechanisms for MASs. These approaches can be group in two categories. First group has an organizational view, which situated the handling mechanism outside the agent [5,7,17]. So, organizational view approaches bring some organizational entities to MAS and these entities monitor agents by listening their internal events and messages to detect exceptional situations. The other main category of handling approaches has an agent view. Researches grouped in this category similarly listen to messages and events of the agent, but they propose a mechanism to handle the exceptions inside the agent [10,15].

5.1 Organizational Views

Approaches subsumed by organizational view can be grouped in two sub-category; centralized and decentralized views. Centralized organizational view, define an exception handling entity in the architecture of MAS for handling exceptions. Conversely, decentralized sub-category of organizational view spread responsibility of handling exceptions through more than one organizational entity.

One of the centralized organizational views is Tripathi and Miller's guardian agent [17]. They propose an agent, named *guardian*, in a multi-agent system that manages exception handling centrally for global exceptions. Guardian agent, dedicated to exception handling, encapsulates defined rules for general exceptions and presents exception handling service to the multi-agent system. When a global exception occurs in MAS, guardian agent performs user defined exception handling rules. For handling exceptions the guardian agents create appropriate exception handlers. At the end of the execution of handler, agent task can continue or terminate its execution as user defined.

Another approach for centralized organizational view is Klein and Dellarocas's Exception Handling Service (EHS) [7]. They propose a shared exception handling service that can be plugged into existing agent system. This service is aware of weak points of multi-agent system. EHS follows all the events in the system for exception detection with specialized agents and performs defined rules for correction. At first glance, this approach seems to be in decentralized organizational view but sentinel agents just responsible for exception detection. These agents intend to prepare a knowledge-base for calling exception handling service. When an exception occurs, exception is compared with the predefined candidate exceptions and exception handling policy is determined. To provide this capability, exception handling service communicates with other agents to define and handle perceived exceptions.

Klein and Dellarocas improve their approach in [8]. In this version, their approach becomes more decentralized by assigning the responsibility of exception handling to sentinel agents. Every agent in the multi-agent system has a sentinel agent that control agent communication and have ability of using a centralized reliability database shared by all sentinels.

We position Haegg's exception handling approach in decentralized organizational view. Haegg [5] proposes special agents, named sentinel, in multi-agent system for exception handling. This approach suggests using a sentinel agent for each agent in the multi-agent system. A sentinel agent controls agent's communication for error detection and recovery. When an exception occurs during the interaction between agents or at the execution cycle, sentinel agent performs error recovery task.

5.2 Agent Views

Souchon et. al suggest using concerted exception handling mechanism defined in SaGE framework [15]. This framework handles exceptions in three vertical category; service, agent and role. They propose a layered handling approach depending on Java call stack structure. And also defined concerted exception handling mechanism resolves errors depending on several agents. When an exception occurs during a shared operation, the operation is terminated to handle exception and participant agents are notified. They have extended the standard Java exception mechanism in order to differentiate higher level exceptions from language levels.

An other approach for agent view is Mallya and Singh's commitment protocols [10]. They propose modelling and handling exceptions via commitment protocols. Exception occurs during the interaction when a protocol is not respected along agent interactions. These exceptions are handled via the definitions of the recovery plan for the exceptional situation. For example alternate protocol or the execution flow of the protocol in an exceptional situation is predefined. They propose an exception handler repository to support handling exceptions dynamically.

5.3 Comparison

Besides all of the suitable features of aforementioned exception handling approaches, these approaches have certain shortcomings. Firstly, we criticize weakness of organizational view approaches. When participant agent count reached huge numbers, it's clear that the centralized organizational exception handling techniques requires great resources for specialized agents to handle exceptions of whole MAS. Although decentralizing of organizational exception handling techniques remove the resource problems of centralization agent, but also don't solve problem of expected exception handling performance. Moreover, organizational exception handling techniques are not efficient since handlers don't have agent's internal knowledge. Without this knowledge, an external handler can not find the proper solution for recovering from the exceptional situations because of lacks of the agent's knowledge.

On the other hand, agent view approaches use resources efficiently by assigning exception handling responsibility to each agent. Most of them propose the usage of agent knowledge to determine exceptional situations. But these handling techniques don't define how autonomy property of agents effects the exception handling process.

We purpose an approach, which can be classified in agent view approaches, by taking care of shortcomings of aforementioned researches. Our approach is based on goal-oriented MAS and defines an architecture how the exception can be handled with goal oriented MAS artifacts**.** Agents are enabling to reason using agent's internal knowledge for recovering exceptional situations. Although our approach is classified in agent view, thanks to the using goal concept, MAS level exception support makes our approach applicable in organizational level like other organizational view approaches. This makes our approach unique in terms of supporting both views using the goal concept. Additionally, proposed handling mechanism respects to autonomy of agents. The autonomy of the agent is provided within its planner's decision making life-cycle, which consists of three main phases: perception, reasoning and action.

6 Conclusion

In this paper we propose an extended exception handling approach for goal-oriented MASs. It distinguishes itself from previous works because our approach is fully integrated to the execution model of agents considering their internal knowledge and autonomy. Agents are enabling to reason using agent's internal knowledge for recovering exceptional situations. We position our work and other approaches in the literature. In section five, we classify other researches in two categories; approaches called as organizational view and agent view. Handling techniques of organizational view propose architectural elements. On the other hand, agent view proposes some mechanism to handle exceptions of agents internally. We specify our exception handling mechanism with the decision making phases and we extend the agency meta-model of MAS by adding exception semantics. Although our exception handling mechanism is specialized for goal-oriented MAS, different exception handling patterns can be implemented by assigning specific exception handling goals to different agents. The variety of applied exception handing patterns offers different handling perspectives to our approach.

Our exception classification and exception handling mechanism is implemented within SEAGENT. To show its ability, we implement a case study on barter domain. It is observed with the case study that our approach and implementation give chance of developing robust applications with respect of agent's autonomy.

References

1. Dikenelli, O.: Seagent mas platform development environment. In: AAMAS (Demos), pp. 1671–1672 (2008)
2. Ekinci, E.E., Tiryaki, A.M., Gürcan, Ö., Dikenelli, O.: A planner infrastructure for semantic web enabled agents. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2007, Part I. LNCS, vol. 4805, pp. 95–104. Springer, Heidelberg (2007)
3. Goodenough, J.B.: Exception handling design issues. SIGPLAN Not. 10(7), 41–45 (1975)

4. Goodenough, J.B.: Exception handling: issues and a proposed notation. Commun. ACM 18(12), 683–696 (1975)
5. Haegg, S.: A sentinel approach to fault handling in multi-agent systems. In: Revised Papers from the Second Australian Workshop on Distributed Artificial Intelligence, pp. 181–195. Springer, London (1997)
6. Kaminka, G.A., Tambe, M., Hopper, C.M.: The role of agent modeling in agent robustness. In: AI meets the real world: Lessons learned, AIMTRW-1998 (1998)
7. Klein, M., Dellarocas, C.: Exception handling in agent systems. In: AGENTS 1999: Proceedings of the third annual conference on Autonomous Agents, pp. 62–68. ACM, New York (1999)
8. Klein, M., Rodriguez-Aguilar, J.-A., Dellarocas, C.: Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. Autonomous Agents and Multi-Agent Systems 7(1-2), 179–189 (2003)
9. Knudsen, J.L.: Better exception-handling in block-structured systems. IEEE Softw 4(3), 40–49 (1987)
10. Mallya, A.U., Singh, M.P.: Modeling exceptions via commitment protocols. In: AAMAS 2005: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 122–129. ACM Press, New York (2005)
11. Maxion, R.A., Olszewski, R.T.: Improving software robustness with dependability cases. In: FTCS 1998: Proceedings of the The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, Washington, DC, USA, p. 346. IEEE Computer Society, Los Alamitos (1998)
12. Miller, R., Tripathi, A.: Issues with exception handling in object-oriented systems. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 85–103. Springer, Heidelberg (1997)
13. Platon, E., Sabouret, N., Honiden, S.: An architecture for exception management in multi-agent systems. International Journal on Agent-Oriented Software Engineering (2008)
14. Russell, S., Norvig, P.: Intelligent Agents. In: Artificial Intelligence: A Modern Approach, 2nd edn., pp. 42–45. Prentice-Hall, Englewood Cliffs (2003)
15. Souchon, F., Dony, C., Urtado, C., Vauttier, S.: Improving exception handling in multi-agent systems. In: Advances in Software Engineering for Multi-Agent Systems, Springer, Heidelberg (2003)
16. Sycara, K., Williamson, M., Decker, K.: Unified information and control flow in hierarchical task networks. In: Working Notes of the AAAI-1996 workshop Theories of Action, Planning, and Control (August 1996)
17. Tripathi, A., Miller, R.: Exception handling in agent-oriented systems. In: Bertino, E. (ed.) ECOOP 2000. LNCS, vol. 1850, pp. 128–146. Springer, Heidelberg (2000)
18. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. Software Engineering 26(10), 978–1005 (2000)
19. Weiss, G. (ed.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge (1999)
20. Yemini, S., Berry, D.M.: A modular verifiable exception handling mechanism. ACM Trans. Program. Lang. Syst. 7(2), 214–243 (1985)

A Reverse Engineering Form for Multi Agent Systems

François Gaillard, Yoann Kubera, Philippe Mathieu, and Sébastien Picault

Equipe Systèmes Multi-Agents et Comportements
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
UMR CNRS USTL 8022
59655 Villeneuve d'Ascq cédex – France
firstname.lastname@lifl.fr
www2.lifl.fr/SMAC/

Abstract. The usual way to design a simulation of a given phenomenon is to first build a model and then to implement it. The study of the simulation and its outcomes tells if the model is adequate and can explain the phenomenon. In this paper, we reverse this process by building a browser in simulations space: we study an automatically built simulation to understand its underlying model and explain the phenomenon we obtain. This paper deals with automated construction of models and their implementations from an ontology, consisting of generic interactions that can be assigned to families of agents. Thanks to the measurement tools that we define, we can automatically qualify characteristics of our simulations and their underlying models. Finally, we offer tools for processing and simplifying found or existing models: these allow an iterative construction of new models by involving the user in their assessment. This simulation space browser is called LEIA for "LEIA lets you Explore Interactions for your Agents".

1 Introduction

Agent-based simulations have taken a preponderant place in life simulation tools, in domains as different as the movie industry, video games, biology, etc. These simulations establish a link between experts of their domain and experts in computer science[12]: this multidisciplinary aspect gave birth to a whole range of frameworks more or less related to the simulated domains.

Many of these platforms like Swarm[14], Madkit[7] or Magic[16] allow considerable freedom to the designer to create agents, the behaviour of those agents and the environment. All design refinements are possible: reusability, genericity, design patterns, components... Other tools like Netlogo[19] are designed to be used by non computer scientists: they rely on a very simple programming. However, openness and genericity are chosen to the detriment of a clear framework for the design of agent behaviour: there is a risk of mixing the framework code with some specific knowledge to the model in the implementation of agents.

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 137–153, 2009.

The IODA[1] methodology [13] is based on a clear separation between agents, their behaviour and the actions selecting process. In this methodology, interactions are independently reified from agents which use them. As a result, we can establish libraries of interactions for a particular area and adaptable to different families of agents, increasing the genericity of modelling work.

An implementation achieved through the JEDI engine [12] offers a generic support of the IODA methodology. This genericity is perfectly illustrated within the generator called JEDI Builder[2]: from a model written according to IODA, we can easily obtain a simulation which is executable with the JEDI engine.

We also offer measurement tools in order to describe the characteristics of each model designed with IODA:

- simulation activity;
- characteristics of the environment and its population;
- development of populations;
- evolution of the use of interactions during the simulation.

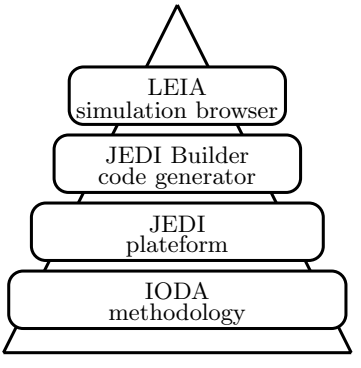


Fig. 1. Hierarchy in the construction of the browser: LEIA is based on automatically generated simulations for the JEDI platform, from a model specified in the methodology IODA

Based on the abstraction provided by IODA and some measurement tools that we set out, we present here a browser in simulations space, i.e. an application which allows the simultaneous display and analysis of several simulations running in parallel. Moreover, to browse the simulations space, we offer tools for processing and simplifying models. These tools enable the iterative design of new models in which the user and measurement tools are the evaluation functions. This browser is called LEIA for "LEIA lets you Explore Interactions for your Agents". LEIA reverses the usual way to design multi agents simulations: we generate a random series of models and their implementation from an already defined ontology, consisting of generic interactions that we can assign to families of agents.

[1] For : *Interaction Oriented Design of Agent simulations.*
[2] www2.lifl.fr/SMAC/projects/ioda/demonstrations/

Then, by successive refinements and use of tools, the user will be able to create a model and study its underlying characteristics. The browser lets the user make a reverse engineering work called "design recovery" by Chikofsky and Cross [2]. The user is able to fully understand the implementations and their underlying model and then study the relationship between interactions. Moreover, by the variations of the models, it acts as a "brain stimulator" as Hofstadter explains in "Metamagical Themas" [8]: "Variations on a thema should be considered as the fuel of creativity".

Section 2 presents the IODA methodology and its relevance in the design of the browser. We describe in section 3 some measurement tools analysing the complexity of systems designed in accordance with the IODA methodology. Then, we develop into section 4 the design of LEIA browser. Finally, section 5 presents the opportunities that the browser opens onto the creation of models by using genetic algorithms.

2 The IODA Methodology

The IODA methodology for simulation design [13] is focused on interactions: they are independently reified from agents. Agents from $\mathbb{A}$, the set of agents in the environment, have basic primitives:

- perception primitives on the overall state of the simulation (noted $\mathbb{E}$), the environment, its internal state and communication with other agents;
- action primitives which can alter the simulation state: its own state, the state of the environment or the states of other agents.

These primitives are used to define the role of the agent in specific interactions. An agent a also has a perception halo $\mathcal{H}_{\mathcal{F}}(a) \subseteq \mathbb{E}$: i.e. the subset of the overall state of the simulation which the agent a discerns through its perception primitives. The neighborhood $\mathcal{N}(a)$ of an agent a is the set of agents in his perception halo.

In the IODA methodology, the agents have a simple specification and are homogeneously represented, allowing the integration of any agent in a simulation model centered on interactions: an agent is an autonomous entity, instantiated from an agent family $\mathcal{F}$, noted $a \prec \mathcal{F}$. An agent family $\mathcal{F}$ in the set of families $\mathbb{F}$ is the abstraction of a set of agents sharing all or part of their perception or action primitives.

An interaction is a sequence of action primitives applied to several agents, which could play the role of source or target (respectively $\mathcal{S}$ or $\mathcal{T}$), and is subject to conditions of activation. The interactions are totally separated from agents who use them, increasing their reusability through simulations and are applicable to different families of agents.

We define the cardinality of an interaction $\mathcal{I}$ as the pair composed with the number of source agents $\text{card}_{\mathcal{S}}(\mathcal{I})$ and the number of target agents $\text{card}_{\mathcal{T}}(\mathcal{I})$ that we need to perform the interaction.

The assignations are the set of interactions that a source agent may perform on target agents. We call assignation $a_{\mathcal{S}/\mathbb{T}}$ of an interaction set $(\mathcal{I}_k)_{k\in[1,n]}$ between

a source agent family $\mathcal{S} \in \mathbb{F}$ and a set of target agent families $\mathbb{T} \subseteq \mathbb{F}$ the set of interactions belonging to agents $\mathcal{S}$ that they may perform as sources together with sets of agents from $\mathbb{T}$ as targets. It is defined as a set of tuples $(\mathcal{I}_k, p_k, d_k)$, $k \in [1, n]$, called assignation elements, where :

- $\mathcal{I}_k$: the interaction which could be performed by sources $\mathcal{S}$ and undergone by targets $\mathbb{T}$;
- $\text{card}_{\mathbb{T}}(\mathcal{I}_k) = q$ the number of targets in $\mathbb{T}$;
- p_k : the priority given to an interaction $\mathcal{I}_k$;
- d_k : the limit distance below which the interaction can occur.

We define the **interaction matrix** as the matrix $\mathcal{M} = (a_{\mathcal{S}/\mathbb{T}})_{\mathcal{S}\in\mathbb{F},\mathbb{T}\subseteq\mathbb{F}^{\mathbb{N}}}$ of each assignations between sources $\mathcal{S}$ and possible targets $\mathbb{T}$ (e.g. Fig.3). Building of a model using the IODA methodology is done through 6 steps:

1. identification of families of agents and interactions, as elements of a matrix of interaction;
2. writing activation conditions and sequences of actions which are primitives of each interaction;
3. identification of action and perception primitives of agents;
4. specification of the priority and the limit distance of each interaction;
5. determining the dynamics, i.e. the evolution of the interaction matrix built at the previous steps;
6. determining the specificities of the model.

During the development of models in the LEIA browser, agent families and interactions are specified a priori. The conditions of action and perception of interactions are also set. Then, the design of the model can be limited to the available choice of interactions and families of agents without the need to generate code in order to use them. We can modify in runtime the model of a simulation without having to stop it.

3 The Measurement Tools

As proposed by Kubera[10], it is possible to characterize part of the complexity of a computer simulation by quantifying the number of computation cycles between the beginning and the end of the simulation. The complexity lies in different aspects of the simulation:

- in the studied phenomenon;
- in the complexity of the cognition of agents;
- in the way to design the simulated phenomenon and the simulation engine;
- in the way to achieve those two models.

We specify in this section some heuristic measures which characterize the complexity of a system designed through the IODA methodology and simulated on the JEDI engine (Fig.2).

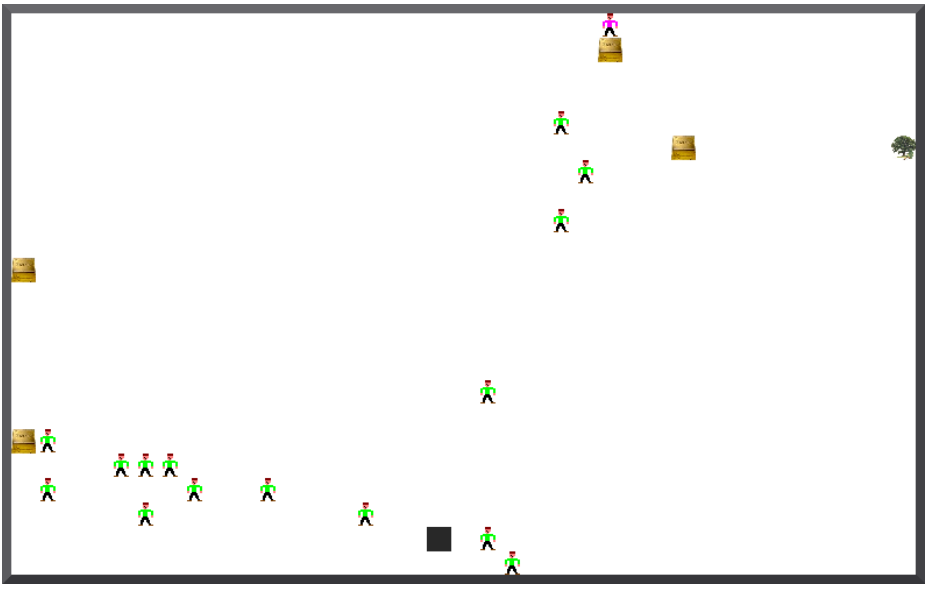


Fig. 2. Screenshot of the environment of an "Age of Empire"-typed simulation using the JEDI engine. Some peasants wander in the environment, find some gold or wood and then bring back them to a forum. They also inform the other peasants where they have found these ressources, creating a chain of workers starting from the forum to the common objective.

These tools are implemented in the JEDI engine [12] which is a sequential engine with a discrete representation of time. An interaction $\mathcal{I}$ has only one source $\mathcal{S}$ ($\text{card}_S(\mathcal{I}) = 1$). At each time step, for each potential source $\mathcal{S}$, it selects a couple (interaction $\mathcal{I}$, target $\mathcal{T}$). The interaction $\mathcal{I}$ is chosen following the assignation of highest priority among all feasible assignations for this source $\mathcal{S}$. The target $\mathcal{T}$ of this interaction $\mathcal{I}$ must also be in the neigbourhood of the source $\mathcal{N}(\mathcal{S})$. For the selected couple, then, we solve the action of the interaction $\mathcal{I}$ between the source $\mathcal{S}$ and target $\mathcal{T}$. This interaction $\mathcal{I}$ can be recorded as a "resolved interaction". At each time step, we can therefore have a specific and quantified return on all the events of the simulation from which our tools of measurement are defined.

These tools are designed to reveal the qualities and defects of simulated models. In what follows, we put ourselves, using the LEIA browser, in a simplified situation where each cell of the environment can only be occupied by one agent. The initial distribution of agents and the primitives are accordingly implemented from this situation.

3.1 Simulation Activity

Agents activity. With the JEDI engine, we can monitor the activity of agents in a simulation, particularly if an agent is able to perform some interactions.

This measure called "Agents activity" characterizes the interactivity of the simulation, i.e. the ability of the simulation to run and therefore evolve.

Definition 1. ***Activity***
With $\mathbb{I}(t)$ *the set of "resolved interactions" during the time step* t *and* $\mathbb{A}(t)$ *the set of agents in a given simulation, the* ***activity*** *of agents is given by:*
$Activity(t) = \text{card}(\mathbb{I}(t)) \ / \ \text{card}(\mathbb{A}(t))$

The provided score is the ratio between the number of agents which are sources of interaction and all agents of the simulation. We can underline that in the JEDI engine, by default, upon being resolved, the source $\mathcal{S}$ and target $\mathcal{T}$ of an interaction are disabled [12]: the target $\mathcal{T}$ cannot participate in another interaction at the same time step, either as target or source. The lower the value of the activity metric, the greater are the number of passive agents: their state will evolve through the few interactions that have been resolved (considering that their internal state does not change by itself through internal lookup by example). In the interaction point of view, this measure can thus reveal some very complex models such as the sale or purchase of items which do not generate changes in the environment and its representation.

3.2 Environment

Environment modifications. The JEDI engine provides an environment similar to a collection of cells that can be occupied by agents (agents have real coordinates [12]). The environment is graphically represented in the simulation as a 2-dimensional grid, made from cells in which agents are represented. It has a set of primitives, such as *Put an agent* or *Remove an agent* of the environment, the use of which requires an update of the associated graphical representation. Therefore, we propose to measure, at time step t, the number of calls for these primitives of the environment, noted $\mathcal{E}(t)$, compared to the number of agents in the simulation.

Definition 2. ***Modifications***
At time step t*, the number of* ***modifications*** *from agents is defined by:*
$Modifications(t) = \mathcal{E}(t) \ / \ \text{card}(\mathbb{A}(t))$

We get an indicator of the visual entropy of the simulation: we mean here the evolution of the representation of the environment between two time steps. This measure is open, typically between 0 and 1 if the resolved interactions call to only one environment primitive requiring an associated graphical update.

Environment stability. This indicator is a measure of stable points of the simulation. This is done through the evolution of the occupation of environment cells in relation to each family of agents. A large deviation in some cells shows that they are occupied repeatedly by the considered family of agents: so we can conclude that this is a stable point in the simulation.

Definition 3. ***Stability***
At time step t, with:

- $N_{\mathcal{F}}(t) = \text{card}(\mathbb{A}_{\mathcal{F}}(t))$ *the number of agents from family* $\mathcal{F}$ *;*
- p *the number of cells in the environment;*
- $\mathcal{O}_{c,\mathcal{F}}(t)$ *the cumulated presence of agents of the family* $\mathcal{F}$ *since the beginning of the simulation in cell c;*
- $\mathcal{M}_{\mathcal{F}}(t) = \frac{N_{\mathcal{F}}(t) \times t}{p}$ *the average occupancy of cells.*

The standard deviation of cells occupancy is:

$$\sigma_{\mathcal{F}}(t) = \sqrt{\frac{1}{p} \times \sum_{c=1}^{p} (\mathcal{O}_{c,\mathcal{F}}(t) - \mathcal{M}_{\mathcal{F}}(t)^2}.$$

Let's imagine a model in which there is absolutly no movement: since the start, every agent has a different cell, cannot move into another cell and the population Nb is absolutly the same in number since the beginning of the simulation. At time step t, this model gives the worst standard deviation:
$\sigma\text{def}_{\mathcal{F}}(t) = \sqrt{\frac{1}{p} \times (\sum_{c=0}^{p-Nb} (0 - \mathcal{M}_{\mathcal{F}}(t))^2 + \sum_{c=p-Nb}^{p} (t - \mathcal{M}_{\mathcal{F}}(t))^2)}$

Then, the ***stability*** *is defined as:*
$Stability_{\mathcal{F}}(t) = \sigma_{\mathcal{F}}(t) / \sigma\text{def}_{\mathcal{F}}(t)$

If at time step t, the population Nb peaks and, subsequently, cell occupancy stagnates, the standard deviation will converge towards the unfavourable metric. A simple measure of Agents stability would have been to determine the proportion of agents with unchanged position after two time steps. Our measure takes into account the cumulative presence of agents since the beginning of the simulation: it enables us to reveal areas of convergence of agents in the environment.

We also provide two indicators on the environment: the mix and the cohesion. The mix is the average percentage of agents from other families in the neighborhood of each agent. Similarly, cohesion is the average percentage of agents from the same family in the neighborhood of each agent. The general idea is the commonly accepted idea of similarity percentage in the Moore neighborhood, defined as the eight cells surrounding a cell centered on the given agent.

Definition 4. ***Mix and Cohesion***
At time step t, with:

- $\mathcal{N}(x)$ *the neighborhood of the agent x;*
- $\mathcal{F}(x)$ *the family from which agent x is instantiated;*
- $Diff(x) = card(\{a \in \mathcal{N}(x) | \mathcal{F}(a) \neq \mathcal{F}(x)\})$ *the number of agents in the neighborhood of the agent x whose family is different;*
- $Same(x) = card(\{a \in \mathcal{N}(x) | \mathcal{F}(a) = \mathcal{F}(x)\})$ *the number of agents in the neighborhood of the agent x whose family is the same;*
- $CellsP(x)$ *the number of cells in the neighborhood of the agent x;*
- $Nb = card(\mathbb{A})$ *the number of agents in the environment;*
- p *the number of cells in the environment.*

*The **mix** is defined as:* $Mix = (1/p) \times \sum_{x=1}^{Nb}(Diff(x)/CellsP(x))$;
*the **cohesion** is defined as:* $Cohesion = (1/p) \times \sum_{x=1}^{Nb}(Same(x)/CellsP(x))$.

We also use a traditional definition of density.

Definition 5. ***Density***
At time step t, with $Nb(t) = \text{card}(\mathbb{A}(t))$ *the number of agents in the environment and p the number of cells in the environment, the **density** of population is defined as:* $Density = \frac{Nb(t)}{p}$.

3.3 Study of the Evolution of Populations and the Resolution of Interactions

Usage of interactions. Thanks to the IODA methodology, the separation of interactions and agents allows us to easily record the "resolved interactions" from each entity. The analysis of those records can reveal the behaviour of agents and the overall usage of interactions, especially some cycles in their usage and order between them, as shown in the next examples.

Let's take two families of agent: *Plant* and *Grasshopper*; and two interactions: *Graze* and *Devour*. In Fig.3, the *Grasshoppers* will *Graze* the *Plants* by priority then, when the *Plants* will have disappeared, the *Grasshoppers* will *Devour* themselves. The interaction *Devour* will only occur when there are no plants.

Let's imagine that in this example, new *Plants* could grow during simulation in sufficient numbers to feed the *Grasshoppers*: the interaction *Devour* can never happen then, which allows a step of simplification in this model. Although we do not have a knowledge of the evolution of the simulation, we can detect as a heuristic measurement the interactions that do nothing to a given model.

Interactions dynamic. In a feedback phenomenon, the result of the phenomenon in question acts back on itself. In the case of simulations, such feedback can occur in solving interactions.

Let's take a simple example (cf. Fig.4): 3 agents are placed around a table. At the beginning of the simulation, an object *Flower* is given to one of the *Char*. During simulation, if an agent has a *Flower*, it performs the interaction *Give* the *Flower* to one of his neighboring agent. Considering one of the agents, the interaction *Give Flower* will be made once every three time steps, waiting for the *Flower* object to go around the table: this phenomenon is periodic. Having a record of each "resolved interaction" for each agent in the JEDI engine allows us to detect cyclic use of interaction that are part of a possible feedback phenomenon.

Source \ Target	Envir.	Plant	Grasshopper
Plant	-	-	-
Grasshopper	-	Graze 1	Devour 0

Fig. 3. Interactions matrix from a simple model of food search: *Graze* has an higher priority than *Devour*

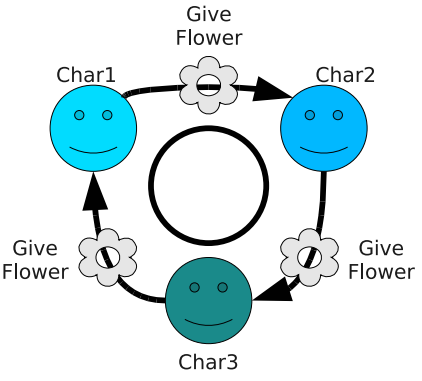


Fig. 4. Simple experience of object transmission between 3 characters

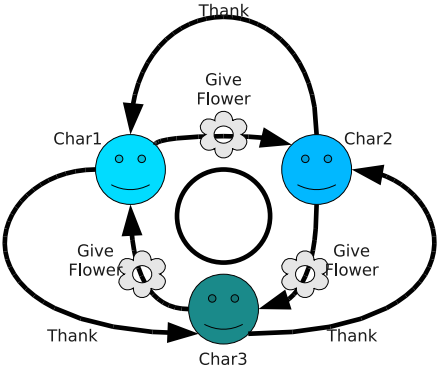


Fig. 5. Simple experience of object transmission with thank between 3 characters

Let's consider the previous example that we modify (cf. Fig.5): after receiving the object *Flower*, the agent first performs the interaction *Thank* with its neighboring donor as target, then performs at next time step the interaction *Give Flower* with his other neighboring agent. If we study the interactions of one agent, we find that the interactions *Thank* and *Give Flower* are performed every 6 time steps (2 time steps for each charecter). *Give Flower* can only be performed after *Thank* during the cycle of 6 time steps: there will be a phase of $\frac{2\pi}{6}$ between the two interactions. If we follow the interactions in general, the interactions *Thank* and *Give Flower* are alternately performed once at each time step. An order between *Thank* and *Give Flower* is viewable, with a periodic phenomenon of 2 time steps. In the case we have a low-level knowledge of the scenario being modelled, we propose the use of frequency analysis tools in order to determine the periodic phenomena in the use (frequency) and order (phase) of interactions.

DFT. By seeing the use of interactions as a discrete signal, we can use the classic definition of the Discrete Fourier Transform (**DFT**) in order to study the interactions in the space of frequencies without constraint in the choice of frequencies and the sampling.

Definition 6. ***DFT***
With $s_{\mathcal{I}}(n)$ *the evolution of usage of an interaction* $\mathcal{I}$ *and* Nts *the number of time steps used by the DFT, the* ***DFT*** *is defined as:*
$\mathcal{S}_{\mathcal{I}}(k) = \sum_0^{\text{Nts}-1} s_{\mathcal{I}}(n) \times e^{-2ikn/\text{Nts}}$

Remarkable frequencies. We propose finding local maxima in the frequency spectrum obtained by Fourier transformation of the evolution of resolved interactions.

Definition 7. ***Remarkable frequencies***
With $\mathcal{S}_{\mathcal{I}}(k)$ *the DFT of* $s_{\mathcal{I}}(n)$*, the set* Freq *of* ***Remarkable frequencies*** *is:*
$\text{Freq} = \{k / \mathcal{S}_{\mathcal{I}}(k-1) < \mathcal{S}_{\mathcal{I}}(k)$ *and* $\mathcal{S}_{\mathcal{I}}(k) > \mathcal{S}_{\mathcal{I}}(k+1)\}$

If the sample is correctly selected, we can detect the periodic usage of an interaction. The sample is limited in time, so we can only find out periodic usage of an interaction which repeats within the sample. As we can't make an infinite run of a simulation, we approximate a repeated usage of an interaction as a cycle.

We can reach 2 levels of analysis in monitoring the interactions:

- A macroscopic monitoring, i.e. taking into account all the interactions that reveal the dynamics of the global system. The discovery of remarkable frequencies may highlight the coupling of some interactions.
- A microscopic monitoring, focusing on one agent, where we can follow the resolution of interactions. This analysis can reveal the dependency between the behaviour of the agent and his assignations. It faces, however, the life expectancy of agents in some models (for example, the model prey / predation). Moreover, some interactions can disable the agent (at the choice of the developer): an agent may only undergo interactions.

The frequency analysis needs to be counterbalanced because some interactions may follow a periodicity intrinsic to them (and therefore independent of the conduct of the simulation itself).

Study of the evolution of populations. Some models, like the prey / predator, will lead to the periodic variation of populations, revealing feedback phenomena. The detection of these periodic phenomena is carried out by Fourier analysis, as described for the analysis of interactions.

4 LEIA: A Browser in Simulations Space

The LEIA browser[3] is an application using n instances of the JEDI engine. It allows the user to instantly make a visual comparison of n simulations by

[3] For *LEIA lets you Explore Interactions for your Agents.* www2.lifl.fr/SMAC/LEIA/applet.html

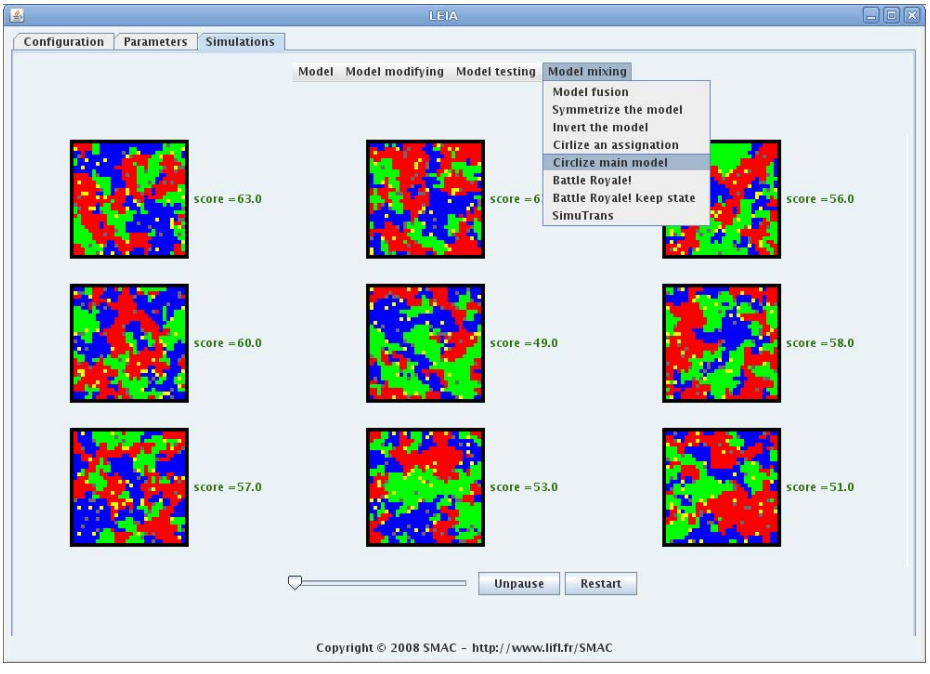


Fig. 6. LEIA, the browser in simulations space

seeing all of them working in parallel (Fig.6). These n instances are created from the same reference model, based on an ontology [6] constituted of already-built family of agents and interactions. We can also define the beginning number of agents and their initial distribution. By giving a domain ontology in input, LEIA is able to build automatically several simulations from the simulations space associated to this domain.

Model manipulation. The LEIA browser provides to the user a set of transformations and generation tools for model, and a set of tests to browse the simulations space. These tools can vary the parameters of the model, either by adding or deleting assignations, changing priorities or limit distance, the initial number of agents, or operations on the interactions matrix as to symmetrize or merge matrices of 2 models. The user can then, by these tools, automatically change the reference model to generate N sub-models. These models are then loaded into the N instances of the JEDI engine: the user can view and compare the N simulations with separate behaviours. LEIA can be run with as many instances as we want. Of course, LEIA will run slower proportionally to the number of instances chosen. Nevertheless, each instance has its own thread, then we can have benefit of a multi-core architecture. At this time, with a quad-core architecture, we are able to run 4 instances at the same time as one.

Simulation analysis. The browser is assisted by a statistics engine to help the user to appreciate the qualities and differences between the displayed models. This statistics engine is based on measurement tools presented in the previous section. We get a quantified return for each simulation in which we consider:

- all interactions resolved by time step compared with envisaged interactions;
- the number of modifications of the environment;
- the repartition of agents: cohesion and mix;
- the evolution of the occupation of the environment;
- remarkable evolution of populations;
- remarkable evolution of interactions.

These heuristic measures allow the user to access informations about each model with complete detail of scores and the associated graphics display is updated in real time.

Model analysis. In addition to the measurement tools that we have already presented, we can study in the specific context of LEIA browser the construction of the model, especially its interactions matrix. The so-called circular assignations are remarkable: i.e. for the same interaction, priority and limit distance, the sources and targets vary cyclically.

Definition 8. ***Cyclic assignations***
With $e = (I, p, d)$ *an assignation element.*
we define: $Assi(e) = \{(\mathcal{S}, \mathcal{T}) \in \mathbb{F}^2 | e \in \text{assi}_{\mathcal{S}/\mathcal{T}}\}$ *the set of assignations from the Assignation Matrix with the same interaction* I*, priority* p *and limit distance* d*. If* $\text{assi} \in \text{assi}_{\mathcal{S}/\mathcal{T}}$*, then:*

- $\text{Src}(\text{assi}) = \mathcal{S}$ *is the agent family of the source of* assi*;*
- $\text{Tgt}(\text{assi}) = \mathcal{T}$ *is the agent family of the target of* assi*.*

the ***cyclic assignations*** *is the set of couples* $(\mathcal{S}, \mathcal{T})$ *taking part in the cycle* e*:* $\text{Assicyliques}(e) = \{(\mathcal{S}, \mathcal{T}) \in Assi(e)/\exists(\text{assi}_i)_{i\in[1,n]} \subseteq Assi(e)/\text{Src}(\text{assi}_1) = \mathcal{S} \wedge \text{Tgt}(\text{assi}_1) = \mathcal{T} \wedge (\forall i \in [1, n-1], \text{Src}(\text{assi}_{i+1}) = \text{Tgt}(\text{assi}_i)) \wedge \text{Src}(\text{assi}_1) = \text{Tgt}(\text{assi}_n)\}$.

Definition 9. ***Cyclic aspect***
The ***cyclic aspect*** *is defined as:*
$\text{Cyclique} = \text{card}(\text{Assicyliques})/\text{card}(\text{Assi})$

The cyclical aspect of a model is the proportion of cyclical assignations among all assignations. The study helps to highlight cycles in the construction of model that can possibly result in feedback loops.

More generally, the study model also opens the prospect of automated simplification of models, for which we are laying the foundations in LEIA by eliminating unreachable interactions, e.g. due to their priorities or limit distance.

Scoring models. We provide to the user a total score to bring together the results of all the tools of measurement.

Definition 10. ***Total score***
With $\mathbb{S} = \{S_1, ..., S_N\}$ *the set of scores provided by the measurement tools (scores between* 0 *and* 100*), the* ***total score*** *is defined as:*
$Totalscore = 1/N \times \sum_{i=1}^{N}(S_i - 50)$

We made the choice to reduce the total score in a note typically between -50 and 50 not to emphasize a score over another. The score is centered on 0 to give a simple reference to the user. We don't use multiplication because there is the risk of hiding interesting data because of a particular score which would be zero.

By seeing the behaviour of all simulations in parallel, when one of them is considered as interesting by the user with the help of the measurement tools, its model can be defined as the new reference model. Then, the user can repeat the process of replacing the reference model, either manually or by using our tools of transformation. The LEIA browser therefore allows one to explore the simulations space generated from the domain ontology board. It should be noted that the LEIA browser is open to any domain ontology, as the interactions and agent families are designed according to the IODA methodology.

Results. The tools presented here allow the user to perform reverse-engineering on simulations. This simulations can be discovered using the browser among the simulations space.

Like Pachet shows with the "Continuator" [17], which stimulates musical creativity, LEIA tends to be a "brain stimulator" for the discovery of new models, and helps the user to identify interesting models. Even with a simple ontology, with few classical interactions, benefits of the LEIA browser are outstanding, as you can see with the following "infection model".

The observation of an experiment displaying a synchronization phenomena between some agent families pointed out an interesting set of interactions at its origin. This set of interactions contains two interactions: one that clones the source on a neighboring position, and the other that kills the target. Thanks to an analysis of the interaction matrix, this set was simplified to a single interaction. Briefly, the aim of this new interaction is to destroy a targeted agent found in the neighborhood and to replace it with a copy of the source agent.

This model, found by LEIA, cyclically affects several families of agent with this interaction called "Infect" (see Fig.7). At least three families of agent are required to avoid deadlock in this simulation. From initial positions which are random, this model led the agents to form spirals per infection (see left figure in Fig.8). LEIA

Source \ Target	Envir.	Red	Blue	Green
Red	-	-	Infect 0 (1.0)	-
Blue	-	-	-	Infect 0 (1.0)
Green	-	Infect 0 (1.0)	-	-

Fig. 7. Interactions matrix of the infection model. It can be extended to a greater number of agent families than two. "Infect 0 (1.0)" means that the interaction "Infect" only occurs under a maximal distance of 1.0.

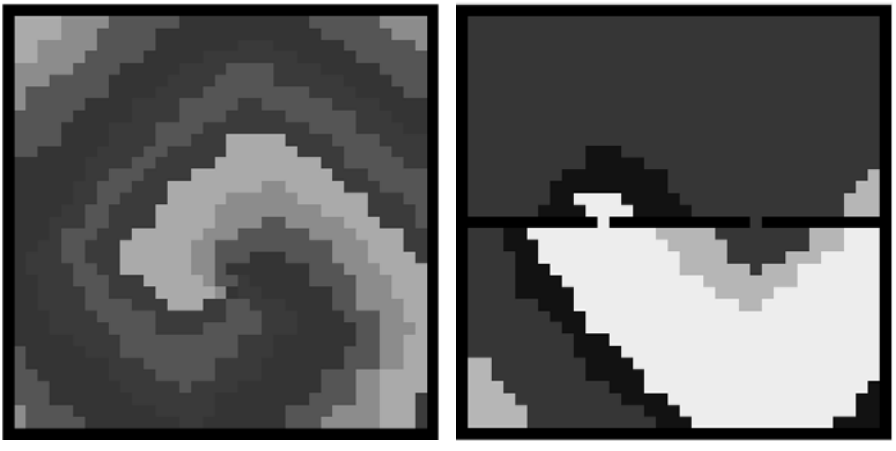


Fig. 8. Two screenshots of the environment of JEDI engine using the "infection model". At left, formation of a spiral in infection model between 7 colors. The figure on the right shows the robustness of this model even if obstacles exists in the picture.

points out that the "infection model" can't work without filling the environment with a huge and equal amount of agents from each family. Indeed, the greater the number of agents, the greater the probability for a source agent to find a target. Moreover, the limit distance is really important: having a higher limit distance for an interaction allows agents to find further a target for this interaction. Thus, the higher the limit distance is for "Infect", the higher is the probablity to fire it. Of course, raising this limit distance helps to increase the number of family agents.

We point out the robustness of this model by adding some obstacles in our experiments. Those obstacles are empty agents that don't interact with any other agents in simulation: they just occupy a place in the environment. Spirals can occur though the presence of obstacles. Moreover, these obstacles can make easier the formation of spiral at their positions, like the right image in Fig.8.

It appears that this dynamic[4] is well known in chemistry, e.g. in the Belousov-Zhabotinsky cyclical reaction [20,1]. Such phenomena are also examined with the help of cellular automata in the Greenberg-Hastings model [5].

5 Towards a Genetic Evolution

In the LEIA browser, designing a model can be automated by drawing random assignations. We can also dynamically implement this model with the view to immediately test it in a simulation. The browser also allows the use of multiple simulations at the same time. We can create and test a model, then help the user to judge its quality by using our measurement tools. Indeed, they facilitate the search for some phenomena, for example to identify:

[4] www2.lifl.fr/SMAC/LEIA/spirale.html

- phenomena of segregation using the measurement tools about cohesion, mixing and stability;
- cyclical evolution of the population from which we can detect remarkable frequencies;
- point out some models which converge towards stable positions by observing the variations of cells occupation;
- models causing a major renewal of the environment by studying its modifications.

The user can identify an interesting model by specifying precise research criteria and, by successive iterations, refine the model in order to obtain a sought phenomenon. Like the user can identify Dawkins biomorphs in "The Blind Watchmaker" [3] whose development meets its desires, he can design models corresponding to his needs by viewing them.

We can link the way to design new models in LEIA to the works about Imagine [15]: designers suggest an original technique for building CSS stylesheets by using a genetic algorithm and successive evaluations through a user interface. Here, each stylesheet parameter is a gene that can be crossed or transferred. The algorithm randomly generates stylesheets, used to the same text. A user can then choose one or more stylesheets with pleasant characteristics. The algorithm then generates new stylesheets by taking into account the previous choices in order to converge, after some iterations, to a stylesheet that the user deems to his liking.

The browser in the simulations space opens the perspective of the generation of models, written in the IODA methodology, through a genetic algorithm. When a problem admits a set of solutions, a genetic algorithm solves it by evaluating a set of solutions parametrized with a fixed number of genes. These genes can evolve by crossing and mutations of the solution in order to maximize an evaluation function [9]. The algorithm converges towards a solution which is considered to be good. The designer also defines a fitness function to fit the sought solution.

In LEIA, the user's choice of specific measurement tools allows the creation of fitness functions. All scores are evaluation functions judging the quality of models. We can see the assignations as genes with which a mutation factor is involved. Then, the mutation can be applied to parameters of an assignation: priority, limit distance, source, target, interaction with different probabilities depending on the supposed impact of the parameter: modifying the distance limit changes the behavior of a model less than changing the interaction.

6 Conclusions and Perspectives

The browser in simulations space allows the iterative design of multi-agent models through the IODA methodology, which provides a clear separation between agents and their interactions allowing thus composition without code generation. We offer a range of tools for processing and simplifying models that can then be viewed in parallel. Then, we propose measurement tools designed from the perspective of IODA. These heuristic measures highlight some features of these

models: system activity, spatial distribution, stability over time, feedback, etc. They make easier the understanding of models built in this way.

We can reverse the usual way of model design, by firstly observing the result (i.e. "what" happens) then the corresponding agent behaviors (i.e. "how" it happens). Through its tools and ease of model design, the LEIA browser acts as a "brain stimulator" whose first result was to find a model similar to the dynamic of Greenberg-Hastings model. Moreover, the browser is open to any ontology as agents and interactions are being designed following the IODA methodology: LEIA aims at exploring simulations space of domains as various as physics or biology. Ultimately, we envisage the construction of models by genetic algorithms, models in which the matrix of assignation can be seen as a set of genes, our measurement tools used for evaluating these models and the search for special features.

References

1. Belousov, B.P.: A periodic reaction and its mechanism. In: Compilation of Abstracts on Radiation Medicine (1959)
2. Chikofsky, E.J., Cross II, J.H.: Reverse engineering and design recovery: A taxonomy. IEEE Software 07(1), 13–17 (1990)
3. Dawkins, R.: The Blind Watchmaker. W.W. Norton & Company, Inc., New York (1986)
4. Desmeulles, G., Querrec, G., Redou, P., Kerdélo, S., Misery, L., Rodin, V., Tisseau, J.: The virtual reality applied to biology understanding: the in virtuo experimentation. Expert Systems with Applications 30(1), 82–92 (2006)
5. Fisch, R., Gravner, J., Griffeath, D.: Metastability in the Greenberg-Hastings Model. In: eprint arXiv:patt-sol/9303005, pp. 3005–+ (March 1993)
6. Gruber, T.R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing in Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers, London (1993)
7. Gutknecht, O., Ferber, J.: The MADKIT agent platform architecture. In: Agents Workshop on Infrastructure for Multi-Agent Systems, pp. 48–55 (2000)
8. Hofstadter, D.: Ma thémagie: En quête de l'essence de l'esprit et du sens. Intereditions, London (1988)
9. Holland, J.: Adaptation in natural and artificial systems. University of Michigan Press (1975)
10. Kubera, Y., Mathieu, P., Picault, S.: La complexité dans les simulations multi-agents. In: Camps, V., Mathieu, P. (eds.) Actes des 15e Journées Francophones sur les Systèmes Multi-Agents (JFSMA 2007), Cépaduès, Carcassone, France, October 17-19, 2007, pp. 139–148 (2007)
11. Kubera, Y., Mathieu, P., Picault, S.: Interaction-oriented agent simulations: From theory to implementation. In: ECAI 2008, July 21-25 (2008)
12. Kubera, Y., Mathieu, P., Picault, S.: Une architecture orientée interactions. Revue d'Ingéniérie des Systèmes d'Information (ISI), Numéro spécial Architectures Logicielles (2008)
13. Mathieu, P., Picault, S., Routier, J.-C.: Donner corps aux interactions. In: Actes des 4e Journées Francophones sur les Modèles Formels de l'Interaction (MFI 2007), Université de Paris Dauphine, Paris, France, May 30-June 1, 2007, pp. 333–340. Papier court (2007)

14. Minar, N., Burkhart, R., Langton, C., Askenazi, M.: The swarm simulation system, a toolkit for building multi-agent simulations (1996)
15. Monmarché, N., Nocent, G., Slimane, M., Venturini, G.: Imagine: a tool for generating HTML style sheets with an interactive genetic algorithm based on genes frequencies. In: IEEE International Conference on Systems, Man, and Cybernetics (SMC 1999), Tokyo, Japan, October 12-15, 1999, vol. 3, pp. 640–645 (1999)
16. Mathieu, P., Bensaid, N.: A framework for cooperation in hierarchical multi-agent systems. Journal of Mathematical Modelling and Scientific Computing 8 (September 1998)
17. Pachet, F.: De la co-construction d'un langage homme-machine: quelques expériences en musique. In: Camps, V., Mathieu, P. (eds.) Actes des 15e Journées Francophones sur les Systèmes Multi-Agents (JFSMA 2007), Cépaduès, Carcassone, France, October 17-19, 2007, vol. 9 (2007)
18. Demazeau, Y., Ricordel, P.-M.: La plate-forme volcano - modularité et réutilisation pour les systèmes multi-agents. Technique et Science Informatiques 21(4), 447–471 (2002)
19. Wilensky, U.: Netlogo (and netlogo user manual) (1999)
20. Zhabotinsky, A.M.: Periodic processes of malonic acid oxidation in a liquid phase. In: Biofizika (1964)

Coping with Exceptions in Agent-Based Workflow Enactments*

Joey Sik-Chun Lam, Frank Guerin, Wamberto Vasconcelos, and Timothy J. Norman

Department of Computing Science
University of Aberdeen, Aberdeen U.K.
AB24 3UE
{j.lam,f.guerin,w.w.vasconcelos,t.j.norman}@abdn.ac.uk

Abstract. A workflow involves the coordinated execution of multiple operations and can be used to capture business processes. Typical workflow management systems are centralised and rigid; they cannot cope with the unexpected flexibly. Multi-agent systems offer the possibility of enacting workflows in a distributed manner, by agents which are intelligent and autonomous. This should bring flexibility and robustness to the process. When unexpected exceptions occur during the enactment of a workflow we would like agents to be able to cope with them intelligently. Agents should be able to autonomously find some alternative sequence of steps which can achieve the tasks of the original workflow as well as possible. This requires that agents have some understanding of the operations of the workflow and possible alternatives. To facilitate this we propose to represent knowledge about agents' capabilities and relationships in an ontology, and to endow agents with the ability to reason about this semantic knowledge. Alternative ways of achieving workflow tasks may well require an adjustment of the original agent organisation. To this end we propose a flexible agent organisation where agents' roles, powers and normative relationships can be changed during workflow enactment if necessary. We use an example to illustrate how this combination allows certain workflow exceptions to be handled.

1 Introduction

The Workflow Management Coalition (WfMC) defines a workflow as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" [34]. Workflows can be formalised and expressed in a machine readable format, and this makes it possible for them to be employed in service-oriented computing scenarios. In such scenarios we may be dealing with open heterogeneous computing systems, where errors and exceptions are likely to occur. We would like the computing systems to cope with these exceptions. Ideally we would like to be able to deal with the unexpected; while we could write specific exception handling routines to deal with some common exceptions which we expect to arise, it will be difficult to anticipate all possible exceptions. Hence it would seem that we need some type of intelligence to deal with the unexpected. Typical workflow management systems (e.g., Taverna [24],

* This work is funded by the European Community (FP7 project ALIVE IST-215890).

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 154–170, 2009.

Kepler [22]) are centralised and rigid; they cannot cope with the unexpected flexibly. Moreover, they have not been designed for dynamic environments requiring adaptive responses [6]. To overcome this we argue that it will be necessary to use agents to control the enactment of a workflow in a distributed manner; agents can be endowed with sufficient intelligence to allow them to manage exceptions autonomously. This should bring flexibility and robustness to the process of enacting workflows.

Different types of exceptions may arise during the enactment of an agent-based workflow. We can identify different levels of adaptivity, and exceptions can occur at any level. The following are the levels of adaptivity [1]:

- Organisation level: exceptions due to changes in the environment may mean that the current organisational structure makes it impossible for a workflow to progress. The organisation must be changed to adapt to the current situation.
- Coordination level: there are exceptions due to changes in the environment, or the agents and their organisational position. For example, some roles may be empty so that the workflow cannot progress. The workflow itself must be altered, possibly to find alternate pathways on which the tasks may be completed.
- Service level: this is the lowest level at which exceptions occur, and the simplest to deal with. A web service is unavailable and an alternative must be found. This may be possible without changing the existing workflows.

It is often the case that exceptions at lower levels can be dealt with by the next higher level; this is indeed one of the main advantages of using an agent based approach rather than a typical workflow execution engine. For example, agents can be used to manage the invocation of Web services, and then they can manage the Web services in an intelligent way [4]; such techniques can also be used to cope with exceptions intelligently. A service-level exception could be one in which a required Web service has gone offline; in this case an agent can use semantic matching [25] or service composition techniques [32] to search for a replacement. For example, if the equipment supplier is not available to give a quote for the required robotics equipment, agents can search for a supplier whose services are described semantically, the replacement supplier can be either an exact match or more general than the one currently specified. Thus coping with service-level exceptions can be done to some extent with existing techniques [35].

Higher-level exceptions are more problematic. For example if the powers or prohibitions in the agent system do not allow the agents to complete the workflow and this leaves the workflow deadlocked at a certain stage. Methods for coping with such exceptions have not been addressed in the literature so far, to our knowledge. If we are to cope with these types of exceptions it would seem that we need organisational flexibility, or the ability to change the social relationships among roles as necessary. Again, higher levels can deal with exceptions at lower levels To facilitate this we make use of an institutional framework with certain speech acts which can modify the roles, powers or obligations of agents in the organisation. For example, suppose that there is a single agent a in the organisation who is empowered to authorise equipment purchases, and that agent is currently unavailable. If an agent b urgently needs an equipment purchase to be authorised, a request can be made to a manager to appoint a suitable stand-in for a, with the appropriate power. This will be further explained in Section 4.

For agents to cope with the unexpected autonomously they must have some understanding of the operations of the workflow and possible alternatives. This means that the tasks specified in the workflow cannot be meaningless labels, but must be associated with some semantic information. To satisfy this requirement we provide an OWL ontology [3] representing the background knowledge for the organisation. This allows agents to reason about the capabilities of agents in the organisation and find alternative ways to deal with workflow tasks when exceptions arise.

In Section 2 we give an overview of our approach. In Section 3 we describe how we model the agent institution, focusing on the powers and normative notions. In Section 4 we describe our example "Equipment Purchase" workflow, and an example of an exception that can happen. In Section 5 we describe our OWL ontology which captures essential aspects of our example. Section 6 looks at related work and Section 7 concludes.

2 Proposed Approach

We focus on adaptation at the organisational and coordination levels. To make our organisation flexible we adopt an institutional framework which allows roles, powers and normative relations to change dynamically, under agents' control. To endow agents with the ability to reason intelligently about how to cope with exceptions, we represent background knowledge about the organisation, and the knowledge and capabilities of its constituent agents, in an OWL ontology. This knowledge enables the agents to recommend appropriate changes when exceptions arise. Our system has two types of organisational knowledge: knowledge about the powers and the norms governing agents are represented as logical rules, as part of the "institutional facts" of the institution; knowledge about the capabilities of agents and the hierarchy of the organisation, and constraints among roles, are represented in an OWL ontology. This separation is appropriate because the former knowledge (norms, powers and roles) may include norms and powers which apply under certain conditions, where conditional rules could not be represented in OWL. This knowledge is also dynamic, being frequently changed by the agents. The latter knowledge (agent capabilities, hierarchy, constraints) is static, and easily represented in a description logic based ontology (i.e., OWL DL).

We use examples from a University institution. In our examples we use a multi-agent system to model the activities in the institution; thus it is not the case that agents are supporting humans in the institution; it is a simulation where the agents are playing the roles of the humans. This modelling is an exercise to test if our framework is able to cope with scenarios arising in human institutions.

3 Modelling the Institution

To model our institution we build on previous work [11] where we described an agent communication framework which allows rules to be defined to describe how events (such as messages being sent) lead to modifications in the institutional facts. This is in line with the "social perspective" [30] on communication; i.e., the institution is not concerned with private mental states of agents. The rules are first-order logic clauses

implemented in Prolog. We describe it here with a Prolog-style notation. In fact our institutional framework is quite similar to [26] which uses event calculus to represent the rules, and implements them also using Prolog. We simply use Prolog directly. We follow the Prolog convention where a string starting with an uppercase case.

We describe the state of an institution by a pair $F = \langle R, A \rangle$ where R describes the current *rules* in force in the institution and A describes the *state of affairs*. Collectively these are called the institutional facts F. *Rules* describe how the speech acts of agents, or other events, lead to changes in the institutional facts. An example of an institutional fact of the *state of affairs* type is having the title "doctor"; examples of institutional rules are the rules of a University describing how the title can be awarded and by whom.

Given an institution described by facts F_0 at some instant, and a subsequent sequence of events $e_1, e_2, e_3 \ldots$, we can use the rules to obtain the description of the institutional facts after each event, obtaining a sequence of facts descriptions: $F_1, F_2, F_3 \ldots$. In our examples events are either speech acts (i.e., messages being sent) or timer events. Note that the above description allows for events which change the rules, although this will not be used in our examples. Thus all events lead to modifications of A; A is simply a list of predicates (facts which hold) or clauses (where they hold conditionally, as in the powers below, for example); events add and remove elements from this list. For convenience, A has been divided into a number of sub-lists for: *roles*, *world_facts*, *powers*, *obligations*, *prohibitions*, *pending_acts*. We will explain the types of facts which populate these sub-lists of A after we introduce our running example below. However, we will firstly explain how speech acts fit within the framework of the institution.

3.1 Speech Acts

The meaning of speech acts is defined by rules in R. These rules define some manipulation of facts in A. We will briefly describe some illustrative examples because there is insufficient space to give the rules for our speech acts in full. The speech acts *assign_role* and *assign_power* define a particularly simple manipulation of A: the new role or power (as specified in the content of the act) is simply added to the appropriate list. The *allocate_task* speech act causes a new obligation to be added for the agent in question. An *inform* speech act simply adds the content of the act to the *world_facts* within the list A; thus an agent can assert some facts and make them publicly available. A *request* speech act adds an obligation for the receiver to reply.

We shall illustrate the notions we introduce below with examples from a University institution (see figure 4). Later we will define a workflow, and illustrate exception handling, using this same institution. The rules R include speech act rules which can change agents' roles. The state of affairs A includes a record of the roles of the institution, and the agents occupying each role. If agent Ag is currently occupying the role of senior lecturer, and the Head of College performs the speech acts "$assign\_role, [Ag, professor]$" and "$remove\_role, [Ag, senior\_lecturer]$", then Ag will be moved into the role of professor. In these speech acts we are just stating the performative and content (a sender and receiver are also needed to complete the speech act). Some speech acts have more elaborate rules; the act *assign_temp_role* (*college_staff*, [Ag, hod, $Duration$]) triggers two acts to be executed: the act *assign_role* (*college_staff*, [Ag, hod]) is executed immediately, and the act *remove_role* (*college_staff*, [Ag, hod]) is placed on the "*pending_acts*"

list, for execution at time $Duration + Current\_time$. This "*pending_acts*" list holds actions that need to be taken at a future time. When the time is reached the action is executed, much like a speech act, however the sender field is blank, meaning that the act is simply executed without any checks for power or prohibition.

3.2 Normative Notions and Institutional Power

Powers and norms (prohibitions and obligations) can be assigned to roles or to agents themselves. An agent inherits powers and norms from the roles it takes on. We will not list all the powers and norms of our example scenario here, but there is a power for each speech act that a role can effectively perform. The following is an example of some of the more interesting powers:

(1) *power*(hod, *allocate_task* (Ag,[$Task$,$Time\_limit$])) if
role (Ag,cs_dept_staff)
(2) *power* (hod, *assign_temp_role* (college_staff,[Ag,hod,$Duration$])) if
role (Ag,cs_dept_staff) and *role* (Ag,professor) and $Duration \leq$21:00:00
(3) *power* (hod, *suspend_role* (college_staff,[Ag,$Role$,$Duration$])) if
role (Ag,cs_dept_staff)
(4) *power* (hoc, *assign_role* (college_staff,[Ag,hod])) if
role (Ag,cs_dept_staff) and *role* (Ag,professor)
(5) *power* (hoc, *assign_power* (college_staff,[Ag,$Power$])) if
role (Ag,college_staff)
(6) *power* (hod, *authorise_purchase* (secretary,Item))

In these powers the speech acts are written as "*performative* (*receiver*,[*content*])", the sender is added when the concrete act is performed. Note that when roles or powers are changed, the speech act is to be sent to the entire college, so that all staff know of the new assignment. Line (1) means that the Head of Department (HoD) can allocate a task to anyone who takes on the role cs_dept_staff. Line (2) means that the HoD can temporarily assign the HoD role to any other professor in the department; the third parameter within the *assign_temp_role* is for the duration after which the temporary assignment will expire – this cannot exceed 21 days. Line (3) allows the HoD to temporarily suspend some agent's membership of a role; it lasts for a time defined by the third parameter. These two powers (2 and 3) are to be invoked when the HoD goes on vacation; the HoD agent will temporarily suspend its own occupancy of the HoD role, and appoint a replacement. Line (4) allows the head of college (HoC) to permanently assign the HoD role to any professor in the CS department. Line (5) allows the HoC to assign a power to an individual agent, or a role.

Obligations are a type of norm. Obligations are always defined with a time limit before which they must be carried out. Some example obligations are

(6) *obliged* (bob, *complete_task* ("upgrade webserver"),"`05-June-12:00`", 103)
(7) *obliged* (fred,
complete_task ("submit paper to conference"),"`09-Sept-18:00`", 103)

After the agent completes a task this is reported as completed in the *world_facts* in A; this means that compliance with the obligation can be checked by looking at the

facts in A. The final parameter above (103) is the "sanction code" which applies if the obligation is broken. Following [26] we associate a 3-figure "sanction code" with each norm violation (similar to the error codes used in the Internet protocol HTTP). The sanction codes gathered by each agent as it commits offences are merely recorded in a list. The use of codes is just a convenient way to record sanctions without yet dealing with them; we would require a separate component to impose some form of punishment.

Obligations do not usually have a condition (as some powers had). If we wish to model the situation where an agent is obliged to reply if it receives a *request*, then we must ensure that the performance of the request creates an obligation to reply; i.e. we make the *request* speech act add an obligation. The condition of any norm cannot be that a speech act is sent because conditions only check facts in A, not events. Thus obligations tend to have a more temporary existence than powers; they are added until they are fulfilled or expire. Obligations cannot have a negative content; i.e., we cannot state that an agent is obliged not to do something. To achieve this effect we use prohibitions. The following is a sample prohibition:

(8) *prohibited* (hoc, *assign_power* (college_staff,[hoc,$Power$]),103)

Again, the final parameter is a sanction code. Lines (5) and (8) model the situation where the head of college is prohibited from assigning new powers to him/herself but is nevertheless empowered to do so (i.e., if the prohibition is violated the assignment of power is still effective).

3.3 Updating the Institutional Facts

We are now ready to present the algorithm which is used to update the instutional facts when an event happens (shown in Fig. 1).

It is in this algorithm that roles are consulted to retrieve the names of the agents occupying the roles; e.g., when checking if an agent who has just sent a message is obliged, the algorithm will consult the facts to see what roles the sending agent occupies.

The main loop of the institution program is executed repeatedly, and invokes the above update algorithm as well as doing some housekeeping:

- For each obligation check if it has timed out. If so, apply the sanction to the agent (or all agents occupying the obliged role) and remove the obligation from A.

algorithm UPDATE-INSTITUTIONAL-FACTS

1. Input: a speech act with *Sender, Receiver, Performative, Content*
2. Check if *Sender* (or one of the roles he occupies) is empowered to send this speech act: If not, discard the act and exit this algorithm.
3. Check if there is a prohibition for *Sender* (or one of the roles he occupies) sending this speech act: If not, go to the next step; If so, apply the specified sanction.
4. Check if there is an obligation which requires that *Sender* (or one of the roles he occupies) send this speech act. If so remove the obligation from A.
5. Process the act as normal (i.e., follow the rules specified for the act).

Fig. 1. Algorithm to Handle Powers and Normative Relations

- For each pending_act check if it is due. If so, execute it and remove it from A.
- If there have been any events, then UPDATE-INSTITUTIONAL-FACTS

Our model of institution is minimal, possessing only those essential features required to illustrate our approach. There are other more complete and sophisticated proposals for representing societies of software agents which could have been used instead. Some of these proposals are, for example, *electronic institutions* [10], *virtual institutions* [7] and organisations for agents [8] – such proposals could have also been used in this paper instead (requiring more space to introduce them, though). Some of the features of our approach have very clear counterparts in those proposals. For instance, in our algorithm to update institutional facts (Fig. 1), the "censoring" of unauthorised utterances which takes place in step 2 corresponds to the *governor agents* of electronic institutions [10]; such agents intermediate all communications between external (foreign) agents and the institution/society. Governor agents check if the messages that the external agents want to send are indeed pertinent to the current state of the interactions; if this is the case, the illocution is forwarded to the appropriate receiver, otherwise the message is discarded.

4 Workflows

We extend the basic framework above with the definition of workflows, allowing us to define structured interaction patterns for agents. A workflow can be enacted by a workflow engine [17] or it can be controlled by individual agents. In our case we are relying on the intelligence of agents to take appropriate actions if the workflow enactment encounters an exception which prevents it from progressing. For this reason we will have the workflow executing in a distributed fashion, controlled at each stage by the agent responsible for that stage.

4.1 Workflow Specification Language

We have developed a simple workflow language. The workflow language assumes that a workflow has a finite number of numbered *places*, with transitions between them (Figure 2 shows a workflow, with places depicted as relationships). A workflow which is currently in progress may occupy one or more places. If the workflow has no branches it will only occupy one place at any time, but if it branches several places may be occupied simultaneously. Figure 3 gives a concrete example of a workflow specification, using a Prolog style notation. A workflow specification has a *place* predicate for each numbered place. The first parameter of a place predicate specifies the number identifying this place in the workflow; this is followed by the role which is responsible for executing the statements in this place, with an identifier in parenthesis for the agent who is taking up that role (e.g. "secretary(D)"). The remainder of a place is completed by a sequence of statements to be executed in order by the agent taking up the role. Statements may be *variable assignments*, or *actions*, or *if... then ... else* constructs. Actions include any action the agent can take such as performing a speech act, querying a Web service, etc.; some actions (e.g. calling a Web service) may return a result. Variable assignments may include (as their right hand side) actions which return results.

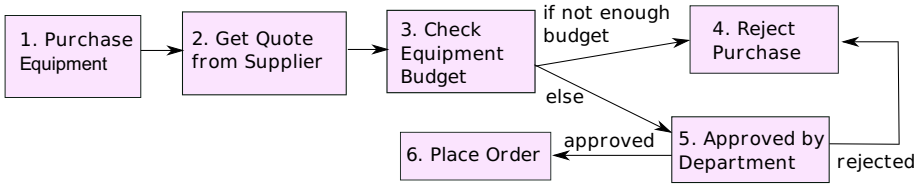


Fig. 2. Diagram of "Equipment Purchase" Workflow

Speech acts within our framework typically have four parameters: sender, receiver, performative, content; however, when a speech act is sent as part of a workflow we add information so that the recipient knows which workflow is being executed and what stage it is at. Thus every message which passes control to another agent includes the number of the next stage to be executed. This ensures that an agent receiving a message can look up the workflow specification and find what is to be done for this stage. It also serves to disambiguate between potentially confusing states: it is possible that a workflow might have two different states where the message being sent to the next agent is the same, thus an agent receiving the message would be unsure about what point had been reached in the workflow. In our example workflow shown in Figure 3 it can be seen that every speech act sent includes, as the final parameter, the name of the workflow ("ep"= equipment purchase) and the state that has been reached within it (e.g. "[ep,2]"). Speech acts are written in the form *speechAct(sender, receiver, performative, content, workflow)*.

For example in the workflow below (Figure 3) `2:technician(T)` means that this is stage 2 and is to be carried out by some agent occupying the role of technician, and that `T` is the variable to be used to hold the name of the actual technician who is carrying out this stage. For example, when the secretary receives the message from the technician, she assigns the variable `T` to be the name of the technician, and can subsequently send messages to `T`. On the other hand, if the secretary performs a speech act with `technician` as the receiver, then it can be sent to any agent occupying the role, and not necessarily the same technician identified as `T` previously.

4.2 Example: The Equipment Purchase Workflow

We consider a simple workflow example – a purchasing equipment workflow in a university – to give an idea of how agents do reasoning to deal with unexpected circumstances during the enactment of workflows. Figure 2 graphically depicts the example. Firstly a research staff issues a request of purchasing a Robotics equipment (1), this request goes to the technician. The technician gets a quote from a supplier via a web service (2), and then finds the cost `C` and passes the request to the department secretary. The secretary checks the equipment budget (3). If the budget is not enough for purchasing, then the purchase request is rejected and the workflow terminates (4). Otherwise, the purchase request is then passed to the HoD for approval (5). The HoD is responsible for deciding to authorise this purchase (6) or reject it (4).

Apart from speech acts, there are three other types of actions that can be required at a stage of the workflow: *web_service*, *local_service*, *query_expertise*. Only *speech_act*

```
place(1,research_staff(R),[
 speech_act(R, technician, request, [purchase, Equipment, Experiment], [ep,2])
                           ]).

place(2,technician(T),[
 web_service(T, supplier_service, request, [quote, Equipment], Result),
 select part(Result, cost, C),
 speech_act(T, secretary, request, [Equipment, C, Experiment], [ep,3])
                      ]).

place(3,secretary(D),[
 local_service(D, dept_data, query, [equipment_budget], Budget),
 (if C > Budget
  then
    speech_act(D, T, inform, [reject, Equipment], [ep,4])),
 (if C <= Budget
  then
    speech_act(D, hod, request, [approve, Equipment, Experiment], [ep,5]))
                     ]).

place(4,secretary(D),[
 speech_act(D, T, inform, [reject, Equipment])
                     ]).

place(5,hod(H),[
 query_expertise(H, Equipment, is_appropriate_equipment(Equipment, Experiment), Useful),
 (if Useful
  then
    speech_act(H, D, authorise_purchase, [Equipment], [ep,6])
  else
    speech_act(H, D, reject_purchase, [Equipment], [ep,4]))
                ]).

place(6,secretary(D),[
 web_service(D, supplier_service, request, [place_order, Equipment], _)
                     ]).
```

Fig. 3. Specification of "Equipment Purchase" Workflow

changes the institutional facts, the other acts give a result which only changes the agent's internal mental state. The *web_service* simply invokes an external web service, while the *local_service* action queries a local computer system, for example the department's finance system. The *query_expertise* action is used when an agent needs to query his own internal knowledge base; in the example below the HoD must execute this action, to query his own internal expertise in robotics equipment and make a decision about whether the robotics equipment proposed is in fact useful for the experiment proposed. This set of actions is known to all agents, although not all agents can carry out all actions; certain actions require certain expertise. For example, to execute *query_expertise* actions to answer queries about a particular topic will require that the agent has expert knowledge in its own knowledge base.

The workflow is a coordination device for agents, in the same way as agent interaction protocols; the powers and prohibitions of agents are not overridden by the workflow – agents are still bound by them. However, the rule which processes a speech act has been extended; when the speech act includes the workflow parameter, then the speech act rule inspects the workflow and adds obligations for the receiving agent to perform its stage of the workflow.

So far agents have not been given any knowledge about each others' capabilities, or about the capabilities required to perform tasks in the workflow. In order for agents to make intelligent decisions about what action to take when a workflow breaks down, agents will need to know about tasks and capabilities, so that tasks may be reassigned to others;

without this knowledge they can only blindly follow a workflow. We encode the required knowledge in an ontology and endow agents with the ability to reason with the knowledge in this ontology, in order to make intelligent decisions when workflows break down.

5 An OWL Ontology

An ontology [33] formally captures a shared understanding of certain aspects of a domain: it provides a common vocabulary, including important concepts, properties and their definitions, and constraints regarding the intended meaning of the vocabulary, sometimes referred to as background assumptions.

More formally, An ontology $\mathcal{O}$ consists of a set of *terminology* axioms $\mathcal{T}$ (TBox) and assertional axioms $\mathcal{A}$ (ABox), that is, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. An axiom in $\mathcal{T}$ is either of the form $C \sqsubseteq D$ or $C \doteq D$, where C and D are arbitrary concepts; an axiom in $\mathcal{A}$ is either of the form $C(a)$ (where C is a concept and a is an individual name; a belongs to C), or of the form $R(a, b)$ (where a, b are individual names and R is a role/property name; b is a filler of the property R for a). The OWL-DL [15] ontology language is a variant of $\mathcal{SHOIN}(\mathbf{D})$ [16] Description Logic, which provides constructs for full negation, disjunction, a restricted form of existential quantification, and reasoning with concrete datatypes. OWL DL benefits from many years of DL research, the benefits include well defined semantics, well-studied reasoning algorithms, highly optimised systems, and well understood formal properties (such as complexity and decidability) [2].

The organisation in the university is represented by the OWL DL ontology, University.owl[1] (part of which is shown in Figure 4). The ontology models persons, the role hierarchy, constraints (such as mutually exclusive roles, cardinality, prerequisite roles) [29], and the range and domain of properties. A person can take more than one role; a role can have many persons. An example of mutually exclusive roles is that the head of college cannot be the head of department simultaneously (see axiom 1 below); a course organiser supervising a student's project cannot mark that student's project (see axiom 2 below). Maximum and minimum cardinality constraints are also used. For example, only one person can fill the role of the head of department; a student has to take at least one course (see axiom 3 below). The concept of prerequisite roles means a person can be assigned to role $r1$ only if the person already is assigned to role $r2$. The Role hierarchy is also modeled in the ontology to reflect authority. More powerful roles are shown toward the top of the hierarchy and less powerful roles toward the bottom. This role hierarchy is consulted by the agents when an exception occurs in a workflow and the agent encountering the problem needs to report to a higher authority. The agent with higher authority may appoint agents to different roles, or change powers to overcome the problem; this is why it is important that the agents have knowledge of the relevant constraints on such appointments. The following shows some of the axioms of the ontology and SWRL rule which are pertinent to our example exception, and its resolution.

1. HoD $\sqsubseteq \neg$ HoC
2. $\exists$ supervises.($\exists$doesProject.Project) $\sqsubseteq \neg \exists$ marksProject.Project
3. Student $\sqsubseteq \geq 1$ takesCourse
4. $\exists$ teaches.Course $\sqsubseteq \exists$ hasExpertise.Course

[1] http://www.csd.abdn.ac.uk/∼jlam/University.owl

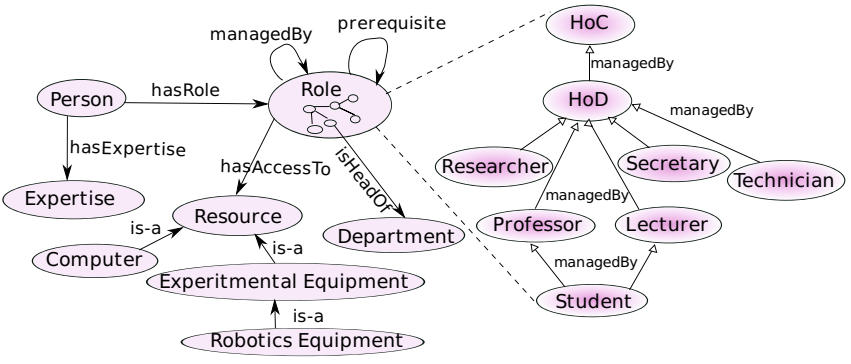


Fig. 4. Parts of the University.owl ontology

5. Professor(dave) // dave is a professor
6. teaches(dave,Robotics) // dave teaches Robotics
7. $\top \sqsubseteq \forall$ teaches.Course // range(teaches) = Course
8. query_expertise $\sqsubseteq \exists$doneBy.(Person $\sqcap \exists$hasExpertise.Expertise)
9. Equipment(?q) $\wedge$ Expertiment(?x) $\wedge$ isNeededBy(?q,?x) $\rightarrow$ is_appropriate_equipment(?q,?x)

5.1 Exception in Equipment Purchase Workflow

In general, upon the failure of a message delivery, the ontology $\mathcal{O}$ must be queried to get the manager of the intended recipient, R (i.e., the next higher agent in the hierarchy). This query in SPARQL [27] can be done as follows:

Prefix uni: ⟨http://www.csd.abdn.ac.uk/∼jlam/University.owl⟩
SELECT ?manager
WHERE { "R" uni:managedBy ?manager }

This manager agent then inspects the tasks in the workflow at this place, in order to find a suitable agent who can perform them. As mentioned above, there are only four types of task: querying expertise, a local service query, a web service, or the sending of a speech act. Local services and Web services could be done by any agent, however querying expertise can only be done by an agent with the required expertise, and performing a speech act can only be done by an agent with the required institutional power. Therefore when the manager agent inspects the tasks which need to be delegated to a new agent, for each "query_expertise" task' the manager must find an agent that has the required expertise. This can be found by the following query, if the expertise is robotics:

Prefix uni: ⟨http://www.csd.abdn.ac.uk/∼jlam/University.owl⟩
SELECT ?person
WHERE { ?person uni:hasExpertise "Robotics" }

For speech acts that need to be delegated to a new agent, the manager may search the institutional facts to find a suitably empowered agent. However, the manager of a group

of agents may typically be empowered to grant new powers to the team he manages, and this is an alternative which can be used to ensure that a nominated agent can fulfill all the tasks required at a place in the workflow.

We now consider a scenario where an exception arises during the enactment of the Equipment Purchase (EP) workflow. As described above, when the HoD goes on vacation he/she is supposed to appoint another member of the department to temporarily act as HoD to cover the vacation period. We consider a scenario where the HoD has failed to do this. The EP workflow then gets stuck at stage 5. The exception is picked up by the secretary who is notified (by the agent platform) of a delivery failure on her message (which should pass control to the HoD and enter stage 5). The exception handling routine is invoked, and this requires that the secretary propagate the problem to the next higher authority of the HoD, the HoC.

The secretary sends the undelivered message to the HoC, and this allows the HoC to know the relevant variable bindings; in this case the HoC will know that the "Equipment" is robotics equipment. By inspecting the protocol an agent can see what needs to be done next, however no agent is empowered to authorise an equipment purchase, except the HoD (see line (6) in Section 3.2). The HoC must rectify this situation by nominating a suitable agent who could authorise the purchase. The *query_expertise* action is one which requires special capabilities; the second parameter of *query_expertise* indicates the type of expertise required (the variable "Equipment" is bound to "robotics"). Thus when the HoC inspects this part of the workflow by querying the ontology (see axiom 8 above), she knows that in order to execute this it is required that the agent must have expertise in robotics. The HoC will perform an ontology query to find an agent with the appropriate expertise. Axioms 4 to 7 implicitly encode the knowledge that "dave" has expertise in robotics. Furthermore, the agent must have power to authorise or reject equipment purchase requests. The HoC can grant the appropriate power to any agent according to power (6) above.

6 Related Work

Many research efforts have been undertaken on distributed workflow enactment mechanisms based on the agent paradigm, their aim is to support flexible and adaptive workflows in open and dynamic environments. In this section, we focus on exception handling in multi-agent systems and workflow management systems.

6.1 Multi-Agent Systems

Exception handling in the agent community has been researched in order to build more reliable multi-agent systems. Klein and Dellarocas [19] proposed the use of specialised agents that handle exceptions. Their approach focuses on observing agent behaviour, diagnosing the possible fault and taking appropriate remedial action. The exception handling service is a centralised approach; the service is characterised as a kind of coordination doctor which actively diagnoses agents' illnesses and prescribes specific treatment procedures. The aim is to simplify agent development and have the agent infrastructure provide the fault-tolerance. Klein and Dellarocas [21] constructed a semi-formal Web-accessible repository of exception handling expertise for learning purposes.

They firstly identified an exception taxonomy which is a hierarchy of exception types, and then described the exception management meta-process. The meta-process specifies which handlers should be used when for what exceptions. Klein et al. [20] describe a domain-independent exception handling services approach to increasing robustness in open agent systems. A directory of agents is used to keep track of the "death" of agents, so that exception which arise due to "agent death" problems can be handled with minimal resource wastage. All of these works use some device which is added into the system to deal with exceptions, for example: specialised agents, an exception repository, or a directory to keep track of agents. In contrast, our approach aims to endow the agents of the system themselves with the ability to deal with exceptions by querying ontologies and changing the organsational structure.

6.2 Workflow Management Systems

The standard approach to representing workflows in business is the Business Process Execution Language (BPEL) [18]. Buhler and Vidal [5] proposed the use of the Business Process Execution Language for web Services (BPEL4WS) as a specification language for expressing the initial social order of a multi-agent system, which can then intelligently adapt to changing environmental conditions. Since BPEL4WS describes the relationship between Web services in the workflow, agents representing the Web service would know their relationships a priori. Buhler and Vidal [4] further proposed to integrate agent services into BPEL4WS-defined workflows. The strategy is to use the Web Service Agent Gateway to slide agents between a workflow engine. The workflow engine calls the target agent instead of the Web service directly; the agent can be configured to respond flexibly. From the above mentioned papers, Buhler and Vidal describe approaches to create adaptive workflow capability through decentralised workflow enactment mechanisms that combine Web services and agent technologies; they claim that that agents representing semantic Web services can organise themselves to enact workflows flexibly.

Similarly, Guo et al. [12, 13, 14] address the downside of current workflow engines which are centralised and suffer from single point-of-failure weakness; they describe the development of a distributed multi-agent workflow enactment mechanism from a BPEL4WS specification. They proposed a syntax-based mapping between some of main BPEL4WS constructs to the Lightweight Coordination Calculus (LCC). The authors claim that with their approach, a BPEL4WS specification can be used directly for constructing a multi-agent system using Web services composition; therefore, its benefit is that existing workflow development methodologies and business process models can be used as much as possible for MAS development. The papers do not cover how their approach deals with unexpected circumstances during the enactment of workflows at runtime.

In [9, 28] the authors address the same problems of workflow management systems which have rigid, centralised architectures that do not offer sufficient flexibility for distributed organisations. In their system, Coloured Petri Nets (CPNs) are used to represent the workflow. A process agent executes a workflow instance by assigning tasks to resource agents which can be seen as representing Web services and can be dynamically discovered. Their aim is to assign suitable resource dynamically to a task. However

the resulting solution is still centralised to some extent, as an agent is managing the enactment of the workflow and calling on resources to do the individual tasks; a truly agent-based enactment should allow each agent to control their part of the workflow (as in the other related works above). In our work, the workflow tasks are dealt by autonomous agents without a central control from a manager agent, resulting in systems that exhibit decentralised flow control.

Similar to our approach, the above mentioned works aim to enable the flexibility of decentralised multi-agent workflow-enactment to deal with dynamic Web services; agents are able to intelligently diverge from prescribed workflows when needed. Although the above works sometimes do not give the details of how they would deal with exceptional circumstances, their techniques can be extended to deal with exceptions. We further include organisational knowledge, such as agents' capabilities represented in OWL ontologies; agents are then able to modify the roles, powers, obligations in the organisation. We believe that the use of ontologies to describe aspects of the organisation and domain can be valuable here, as agents are part of an organisation and will be unable to deal with exceptions entirely on their own.

6.3 The Commitment Approach

Singh and Huhns [31] propose interaction-oriented programming (IOP) as a technique for engineering multi-agent systems to flexibly enact workflows. With the emphasis on facilitating agent autonomy and flexibility in interactions, IOP describes interactions using high-level primitives. The high-level primitive which Singh and Huhns focus on is the "commitment". By making this a first class object agents are able to reason about their commitments to others and vice versa, and can make autonomous decisions about how to act. With commitments capturing the high level meaning of an interaction, agents have the opportunity to intelligently reason about alternative ways of satisfying the high-level goals of an interaction. This approach potentially allows a much greater flexibility than our approach above, however the challenge is to develop an appropriate agent reasoning mechanism to enable such adaptive behaviour. This is an interesting area for future investigation.

Mallya and Singh [23], building on the commitment approach, have proposed novel methods to deal with exceptions in a protocol. They distinguish between expected and unexpected exceptions. Unexpected exceptions are closest to the types of exceptions we tackle here. Mallya and Singh's solution makes use of a library of sets of runs (sequences of states of an interaction) which could be spliced into the workflow at the point where the exception happens. Mallya and Singh do not describe how these sets of runs can be created, but it is likely that one would need access to observed sequences from enactments of similar workflows. The aim of the commitment approach is in line with our work, as it endows the agents with some understanding of the meaning of the workflow they are executing, by giving them knowledge of the commitments at each stage. This would make it possible for agents to find intelligent solutions when exceptions arise. Similarly, in our approach, agents are endowed with semantic knowledge (represented in an ontology) about the capabilities and hierarchy of the other agents so that they can find suitable candidates to execute tasks in the case of exceptions. In the future work, we would like to merge our approach with the commitment protocols

approach to model business processes, in which commitments among roles and business policies can be described.

7 Summary, Discussion and Conclusion

We have shown how workflow exceptions at the coordination or organisational level could be handled by agents. Our solution has two components: (1) a flexible agent institution, so that when an impasse arises in a workflow, the agents can reorganise to find an alternative path to circumvent the problem; by "flexible" we mean that the institution must include speech acts which allow agent roles, powers and normative relations to be altered at run-time; and (2) agents endowed with semantic knowledge about the capabilities and hierarchy of the other agents so that they can find suitable candidates to execute tasks that are preventing the workflow from progressing; we provided this knowledge via an OWL ontology. We illustrated our approach with a University institution example to illustrate how exceptions can be dealt with by agents via ontology reasoning, a decentralised collection of agents in an organisation cooperate to maintaing the workflow's integrity. In general the type of exception we can handle is one where a task which needs to be done cannot be done because an agent is unavailable, or available agents are lacking some attribute. In this case agents perform ontological reasoning or querying to find alternatives. In our example we simply adjusted the powers of an agent so that he could fulfil the task. More generally we may allow agents in an organisation to decide to outsource a task, or to hire a consultant, or send a member of the organisation for training. To tackle more varied and more complex types of exceptions we foresee that agents will need to be given more knowledge about their domain and the tasks and capabilities available. This will require a combination of more ontological knowledge, and also appropriate reasoning mechanisms so that agents can exploit the knowledge.

Our solution has split the knowledge of the system in two parts: the institutional facts (powers, roles, speech act processing rules, etc.) are implemented with Prolog rules, while the knowledge about the tasks the agents will be able to undertake (knowledge of capabilities of the agents, constraints on roles, and the hierarchy of the organisation) are represented with an OWL ontology. In fact some knowledge is represented twice; the knowledge about membership of roles is included in both the ontology and the institutional facts. This information is important both for deciding the institutional updates, and for finding suitable candidates when coping with exceptions; however, this duplication is not optimal.

References

1. Aldewereld, H., Dignum, F., Penserini, L., Dignum, V.: Norm dynamics in adaptive organisations. In: 3rd International Workshop on Normative Multiagent Systems (NorMAS 2008), July 2008 (to appear)
2. Baader, F., Horrocks, I., Sattler, U.: Description logics as ontology languages for the semantic web. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS, vol. 2605, pp. 228–248. Springer, Heidelberg (2005)

3. Bechoffer, S., van Harmlen, F., Hendler, J., Horrocks, I., McGuinnes, D., Patel-Schneider, P., Stein, L.A.: OWL Web Ontology Language Reference (February 2004), http://www.w3.org/TR/owl-ref/
4. Buhler, P., Vidal, J.M.: Integrating agent services into BPEL4WS defined workflows. In: Proceedings of the Fourth International Workshop on Web-Oriented Software Technologies (2004)
5. Buhler, P., Vidal, J.M.: Towards adaptive workflow enactment using multiagent systems. Information Technology and Management Journal 6(1), 61–87 (2005)
6. Buhler, P., Vidal, J.M., Verhagen, H.: Adaptive workflow = web services + agents. In: Proceedings of the International Conference on Web Services, pp. 131–137. CSREA Press (2003)
7. Cliffe, O., De Vos, M., Padget, J.: Answer Set Programming for Representing and Reasoning About Virtual Institutions. In: Inoue, K., Satoh, K., Toni, F. (eds.) CLIMA 2006. LNCS, vol. 4371, pp. 60–79. Springer, Heidelberg (2007)
8. Dignum, V.: A Model for Organizational Interaction: Based on Agents, Founded in Logic. PhD thesis, University of Utrecht, Utrecht, The Netherlands (2004)
9. Ehrler, L., Fleurke, M., Purvis, M., Savarimuthu, B.T.R.: Agent-based workflow management systems(WfMSs): Jbees- a distributed and adaptive wfms with monitoring and controlling capabilities. Information Systems and E-Business Management 4(1), 5–23 (2006)
10. Esteva, M.: Electronic Institutions: from Specification to Development. PhD thesis, Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, IIIA monography, vol. 19 (2003)
11. Guerin, F., Vasconcelos, W.W.: Component-Based Standardisation of Agent Communication. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2007. LNCS, vol. 4897, pp. 227–244. Springer, Heidelberg (2008)
12. Guo, L., Robertson, D., Chen-Burger, Y.: Enacting the distributed business workflows using bpel4ws on the multi-agent platform. In: Eymann, T., Klügl, F., Lamersdorf, W., Klusch, M., Huhns, M.N. (eds.) MATES 2005. LNCS, vol. 3550, pp. 35–46. Springer, Heidelberg (2005)
13. Guo, L., Robertson, D., Chen-Burger, Y.: A generic multi-agent system platform for business workflows using web services composition. In: 2005 IEEE Intelligent Agent Technology, Compiegne University, France, pp. 301–307 (2005)
14. Guo, L., Robertson, D., Chen-Burger, Y.: Using multi-agent platform for pure decentralised business workflows. Journal of Web Intelligence and Agent System 6(3) (2008)
15. Horrocks, I., Patel-Schneider, P.F.: Reducing OWL entailment to description logic satisfiability. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 17–29. Springer, Heidelberg (2003)
16. Horrocks, I., Sattler, U.: A tableaux decision procedure for $\mathcal{SHOIQ}$. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), pp. 448–453 (2005)
17. IBM. BPWS4J (2004), http://www.alphaworks.ibm.com/tech/bpws4j
18. IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business process execution language for web services version 1.1. Technical report (July 2003), http://www.ibm.com/developerworks/library/specification/ws-bpel/
19. Klein, M., Dellarocas, C.: Exception handling in agent systems. In: AGENTS 1999: Proceedings of the third annual conference on Autonomous Agents, pp. 62–68. ACM Press, New York (1999)
20. Klein, M., Rodriguez-Aguilar, J., Dellarocas, C.: Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. Autonomous Agents and Multi-Agent Systems 7(1-2), 179–189 (2003)
21. Klein, M., Dellarocas, C.: Towards a systematic repository of knowledge about managing multi-agent system exceptions. Technical Report ASES Working Report ASES-WP-2000-01, Massachusetts Institute of Technology (2000)

22. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system: Research articles. Concurr. Comput.: Pract. Exper. 18(10), 1039–1065 (2006)
23. Mallya, A.U., Singh, M.P.: Modeling exceptions via commitment protocols. In: AAMAS 2005: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 122–129. ACM Press, New York (2005)
24. Oinn, T., Addis, M.J., Ferris, J., Marvin, D.J., Senger, M., Carver, T., Greenwood, M., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workows. Bioinformatics Journal IEEE Computer 20(17), 3045–3054 (2004)
25. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
26. Pitt, J., Kamara, L., Sergot, M., Artikis, A.: Formalization of a voting protocol for virtual organizations. In: AAMAS 2005: Proceedings of the fourth international joint conference on Autonomous agents and multi-agent systems, pp. 373–380. ACM Press, New York (2005)
27. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (January 15, 2008), `http://www.w3.org/TR/rdf-sparql-query/`
28. Purvis, M., Savarimuthu, B.T.R., Purvis, M.: A multi-agent based workflow system embedded with web services. In: second international workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments (COLA 2004), Beijing, China, pp. 55–62. IEEE/WIC Press (2004)
29. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer 29(2), 38–47 (1996)
30. Singh, M.P.: Agent communication languages: Rethinking the principles. IEEE Computer 31(12), 40–47 (1998)
31. Singh, M.P., Huhns, M.N.: Multiagent systems for workflow. International Journal of Intelligent Systems in Accounting, Finance and Management 8, 105–117 (1999)
32. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics (September 2003)
33. Uschold, M., Gruninger, M.: Ontologies: Principles, Methods and Applications. The Knowledge Engineering Review (1996)
34. WfMC. Workflow management coalition terminology and glosary. Technical Report WFMC-TC-1011, Workflow Managemtn Coalition (1999)
35. Wiesner, K., Vaculín, R., Kollingbaum, M.J., Sycara, K.P.: Recovery Mechanisms for Semantic Web Services. In: Meier, R., Terzis, S. (eds.) DAIS 2008. LNCS, vol. 5053, pp. 100–105. Springer, Heidelberg (2008)

Part IV

Emergence and Self-organisation

Contribution to the Control of a MAS's Global Behaviour: Reinforcement Learning Tools

François Klein, Christine Bourjot, and Vincent Chevrier

LORIA – Nancy University
Campus scientifique BP 239
54506 Vandoeuvre-lès-Nancy Cedex
{Francois.Klein,Christine.Bourjot,Vincent.Chevrier}@loria.fr

Abstract. Reactive multi-agent systems present global behaviours uneasily linked to their local dynamics. When it comes to controlling such a system, usual analytical tools are difficult to use so specific techniques have to be engineered. We propose an experimental dynamical approach to enhance the control of the global behaviour of a reactive multi-agent system. We use reinforcement learning tools to link global information of the system to control actions. We propose to use the behaviour of the system as this global information. The behaviour of the whole system is controlled thanks to actions at different levels instead of building the behaviours of the agents, so that the complexity of the approach does not directly depend on the number of agents. The controllability is evaluated in terms of rate of convergence towards a target behaviour. We compare the results obtained on a toy example with the usual approach of parameter setting.

Keywords: Control, MAS, experimental approach, emergence, global behaviour, reinforcement learning.

1 Introduction

The goal of this study is to control the behaviour of a multi-agent system (MAS) or of a complex system modeled by a MAS. It takes place in the context of reactive MAS [1]. A reactive agent only owns reflex reactions to external stimuli, without a representation of its environment or itself, and has a limited memory. In such a system, interactions are essential and trigger a collective behaviour which is not directly linked to the individual behaviours.

The behaviour of the MAS is unpredictable without simulation, because of its strongly non-linear nature due to the numerous interactions that occur. However, global complex phenomena can be observed. These phenomena emerge from the local interactions between the agents despite of their limited abilities. So there are two levels of description in a MAS: the local level, where the agents evolve and their behavioural rules are set, and the global one, where a particular behaviour of the whole system can be observed.

An emergent structure or property is characterized by a phenomenon involving several agents, whose observation is done at a higher time scale than the local evolution of

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 173–190, 2009.

the system and makes it appear stable. We define the global behaviour of the MAS as a description of its emergent phenomena. Several behaviours can be observed in a given MAS, but it is uneasy to predict which one will occur when the system is running, for any local perturbation can trigger a dramatically different one.

Our purpose is to control the global behaviour of the system, although it is uneasily linked to the local dynamics of the MAS. Namely, we wish to make the system show a target global behaviour thanks to actions correctly chosen and performed at the right time. The actions can be local modifications of the state of the system, or global quantitative changes like the modification of a parameter value. They depend on what the controller is allowed to do, particularly if the MAS models a real-world system. The goal is to give a pragmatic control method, applicable to many reactive MAS. Focusing on reactive MAS allows us to put the stress on its particular properties, without challenging the *intelligence* of the agents to achieve the control of the system.

In this contribution, an original solution of dynamical control of an agent-based model is proposed, but we keep in sight the control of a concrete distributed system modelled by a MAS.

We first expose the issues brought on by the control problem and the specificities a control solution of a MAS should meet. Then we discuss different solutions to solve this problem, especially the one of parameter setting, and their limitations. We differentiate the static and the dynamic approaches. This leads to our proposition of a dynamical method that takes more information on the system into account. We explain how we implemented it on an toy example, and we compare the control performances of our method and the parameter setting solution through different criteria.

2 Control of a Reactive Multi-Agent System

If there is only one possible global behaviour that the system always shows, there is no control problem, otherwise, we consider the different reachable behaviours as global states. So our problem is to lead the system into one particular state. Two kinds of difficulties come up to solve this problem: the ones due to the nature of the MAS, and the ones due to the control problem itself. They are discussed in this section.

Because of the complexity of the interactions in a MAS, the local and global levels are not directly linked. If there are known local rules, and even simple rules in a reactive MAS, they do not give rise to a global view upon the behaviour of the system itself. For this reason, in multi-agent field of research, it is generally admitted (cf. Wegner [2], Edmonds [3,4], DeWolf [5], Amblard [6]) that an analytical model of a MAS is infeasible. These authors recommend then to study MAS through an experimental approach.

This limits the possible ways to solve the control problem: a solution is necessarily experimental, and we cannot use powerful tools as differential equations to predict the global behaviour triggered by an action.

Hence we assume that the studied MAS is totally observable and that we can act on it at will. If it is an already deployed system, like a peer-to-peer network, we can always model it thanks to another MAS and perform our proposition on the model. Before to handle the concrete system, the study of a MAS model is useful since it involves less experimental costs and since it allows to know if the original system can

be controlled. Therefore, in the following, the discussion is restricted to the control of a model.

An experimental study implies a considerable number of simulations. In order to provide realistic tools to control the MAS in a reasonable time, we have to avoid the simulation time soaring by any means. We can consider to reduce the simulation time by reducing either the time of each simulation or the number of simulations. Many solutions have been proposed to handle this issue, we discuss them in §3.1.

Beyond these difficulties linked to the nature of the MAS, since we intent to control the MAS, three questions have to be answered: how to characterise and measure the global behaviour, what are the possible actions, and how to determine the action to perform at any time (see Fig. 1) ?

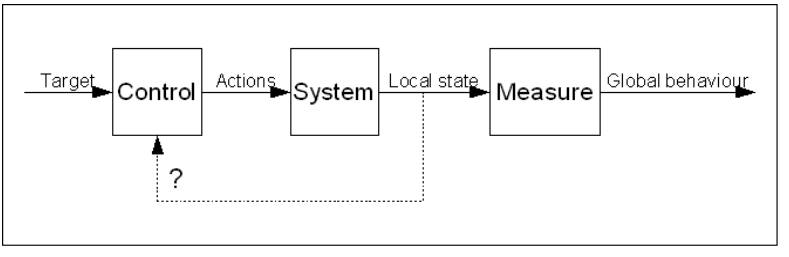


Fig. 1. Principle of control of a MAS. A control method chooses the actions to reach the target, and can take into account features of the system.

Considering the high number of simulations, the study has to be automated – and in particular the evaluation of the influence of each tested action within an experimental study. We do not want a human observer to check each simulation to identify the global behaviour. Thus we have to engineer an automatic measure to detect the global behaviour of the MAS from its local state at any time, at least to verify if the target behaviour is reached.

The actions are instant modifications of the system that have an influence on its behaviour. They can be modifications of the environment – by changing its attributes like its size or by introducing local perturbations like adding obstacles – or modifications of the agents – by modifying their number, adding luring agents or changing their characteristics: size, local behavioural rules, etc. The details of the actions depend on the system itself. Still, the actions that are made available to the control system must be chosen.

Once the behaviour and the actions have been defined, the question of the control itself remains: an action to perform has to be chosen in order to reach the target behaviour. The controller must decide which features of the system to take into account in that choice, and engineer a method to learn the good action to perform given these features.

3 Approaches of Control in Multi-Agent Systems

We expose in this section different approaches found in other works to the problems of controlling a MAS or guaranteeing a global behaviour. We discuss the proposed answers

to the difficulties raised in the former section. We can split these approaches into two classes: static and dynamic ones. Static solutions do not take the evolution of the system into consideration while they control it (no loop in Fig. 1), whereas dynamic solutions perform an on-line control depending on the current state of the system.

3.1 Static Solutions

One way to make the system converge to the target is to set its parameters so as to optimise a convergence criterion. The principle of parameter setting is so to find optimal constant values for the controlled parameters of the system, typically with the use of a metaheuristic performed on the parameter space [7, 8]. We have to emphasise that the only possible actions when using parameter setting are the modification of parameter values, which restricts our definition of controlling a MAS.

A metaheuristic [9] is an algorithm that rules the exploration of a space by testing values of the space. The rules determine the values to choose and try to focus on the relevant areas in spite of a part of random decision, striving to balance the exploration of the space and the exploitation of promising results. So the parameter setting is essentially simulation-based and respects the experimental necessity specified in §2.1.

Parameter setting in the domain of MAS is not necessarily used to control the global behaviour, but any feature of the system (for instance [10]), so the question of characterising the global behaviour is not handled by this approach.

However, much work has been done for the question of reducing the simulation time: [10] proposes to use *equation-free* tools to partially predict the result of a simulation and then speed up each simulation, [11] proposes to reduce the actions space by limiting its dimensionality in a *divide and conquer* way, and many works [12-16] reduce the number of simulations thanks to a principle called *dynamical design of experiments* [6]. The principle consists in exploring the actions space in a non-homogeneous way, focusing on its relevant parts, by contrasting the exploration of the space and the exploitation of the best results found. The distinction can be made with regard to the space areas (this is the point of metaheuristic approaches [12, 13] and of some other works in MAS domain [14, 15]), but also to the quality of the estimation of the results in each point of the space [16].

Eventually, the main limitation of this approach is its static, off-line nature. It gives an optimised solution, but when the parameter values are set, there is no way to decide how to change them, whether the system reaches the target or not. If the MAS undergoes perturbations, that is if the control frame is different from the learning frame, the accurately optimised solution is not relevant anymore.

3.2 Dynamical Approaches

A dynamical control involves the use of information on the system to choose on-line the action to perform.

A dynamical approach is proposed in [17] and [18]. A morphological description of the global behaviour is proposed to characterise it. A control solution is given which takes into account the current state to correct it. Still, the solution of control is purely heuristic: the controller has to explain the system what action to perform in

each state, but no method is proposed to determine the actions in a general case, especially if the link between the actions and the global behaviour is hard to determine.

The same remark can be applied to the AMAS theory [19] and the works based on it [20]: this method asks a human to determine the non-cooperative situations and decides what is to correct in order to make the MAS show a wanted behaviour. There is no notion of optimality and no automatism of the process.

Another dynamical approach, more usual, consists in considering the system as a decentralised Markov decision process (DEC-MDP) [21] and to apply reinforcement learning tools. But in that case, the only possible actions are at the agent level: each agent chooses what local action to perform. Thus actions as adding an obstacle or changing a global parameter are not handled. Furthermore, the complexity of a DEC-MDP problem soars with the number of agents and actions [22].

4 Proposition of a Dynamical Solution

We propose an on-line method to control a MAS using reinforcement learning. First the actions to perform are learned thanks to an experimental approach, then the best actions are chosen during the control phase. It must be noticed that if the MAS models another deployed system, only the learning phase is applied on the first one, and the resulting knowledge can be used to control the "real" system.

The drawback of the dynamicity is that it involves an observation of the controlled system to choose an action. Therefore, the observability of the controlled system must be discussed: if it is observable enough to compute the states described below then the proposition can be used, otherwise an approximation of the states must be computed.

The actions decision is dynamically taken, depending on features of the system: a distinction is made between different states of the MAS, letting us learn what to do in each of them to reach the desired behaviour. In this section, we explain which states to choose and then how to determine the actions to perform, thanks to reinforcement learning tools.

4.1 States Description

In order to choose a states description, the level of these states must be decided first. The local states can easily be defined by describing the exact disposition of the elements of the system – typically the position, speed and internal state of each agent and each object in the environment. The global states represent the behaviour of the whole system.

Since this paper shows a first step in a larger study, we choose the simplest states description to show the utility of the method, that is, global states. Only the behaviour of the MAS is considered in the actions decision, as represented in Fig. 2, which can be compared to Fig. 1. Indeed, there are few possible behaviours compared to the number of sharp local states, so a global states space is easier to explore exhaustively.

The utility of each action in each state will be estimated thanks to many experiments for this couple {state, action}. The decrease of the number of states leads to a lower complexity of an algorithm based on this exploration.

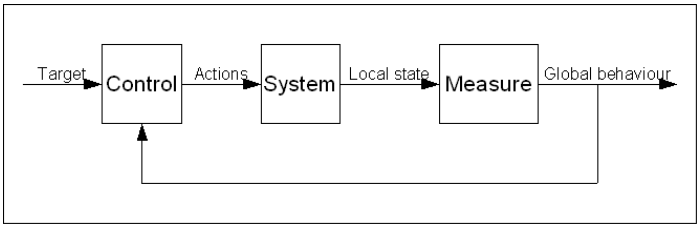


Fig. 2. Dynamical control using global information. We propose to consider the global behaviour in the action decision.

Furthermore, the dynamics of the MAS is observed at the global level, so the choice of states describing the global stable structures of the system is likely to give a good representation of this dynamics. That justifies the use of global states.

4.2 Short Presentation of Reinforcement Learning

All the following refers to [23]. A Markov decision process (MDP) is defined by a quadruple <S, A, T, R> where S is a set of states, A a set of actions, T a transition function that gives the probability to reach a state $s' \in S$ from a state $s \in S$ when $a \in A$ is performed (such as $T(s,a,s') = Pr\{s_{t+1}=s' \mid s_t=s, a_t=a\}$ at any time t), and R a reward function of each transition (such as $R(s,a,s') = E\{r_{t+1} \mid s_t=s, a_t=a, s_{t+1}=s'\}$ where E denotes the expected value).

The transition function T verifies the Markov property that establishes that the probability to reach a given state for a given action only depends on the current state, not on previous states of the system.

The goal of reinforcement learning (RL) is to find a near-optimal policy that defines which action to perform in each state so as to maximize the return (a function that gives the expected reward, parameterised by a value γ, called the discount rate). The policy can be stochastic when it gives probabilities over the actions to perform in each state.

The base of RL is dynamic programming: in order to evaluate the return of an action *a* performed in a state *S*, we try this action, which sends the system in a new state *S'*, and we use the current evaluation of the policy on *S'* to modify the evaluation of *a* in state *S*. Very efficient algorithms exist when T and R are known.

Otherwise, the functions T and R are experimentally estimated, that is the goal of Monte Carlo methods for instance. The combination of dynamic programming and Monte Carlo methods leads to temporal-difference (TD) learning which allows estimating T and R while the optimal policy is learned.

There are two classical algorithms of TD learning: Sarsa and Q-learning [23]. The difference is that Sarsa is an on-policy algorithm which evaluates the policy used to explore the action space in each state, while Q-learning is off-policy and learns an optimal policy different from the one used to explore the action space. Typically, the Q-learning can be used to find a deterministic policy and Sarsa a stochastic one.

4.3 Application to MAS Control

We formulate the hypothesis that our global state model verifies the Markov property, so as to apply these tools on it, even if we know this is just an approximation. Since the evolution of the system is unpredictable, the past global states are insignificant. We presume that in two different situations identified as the same state, the evolution is likely to be the same whatever action is performed.

The probability to reach a state *S'* by doing the action *a* in the state *S* is denoted *P(S, a, S')*. It can be approximated by the proportion of local states in the global behaviour *S* that stabilise in the global behaviour *S'* when an action *a* is performed (cf Fig. 3). This is our transition function. We define the rewards as follow: when the MAS reach the target global behaviour, the MDP receives a positive reward (namely 1), and 0 otherwise. So we have a MDP equivalent to our control problem.

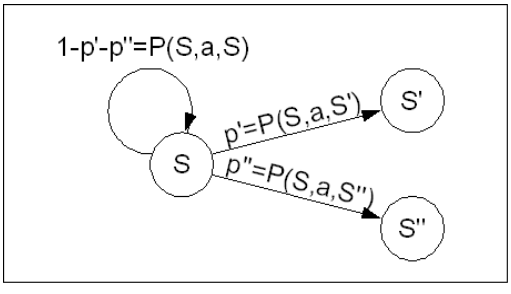


Fig. 3. Transition function. Transitions from a state *S*, for an action *a*, in a 3-states graph.

We would rather learn a stochastic policy to always allow an exit from a global state, since there is a risk that a deemed good action in a global state S keeps the system in S in particular situations. Although the Markov property is a correct approximation, it is likely to be wrong, and the aggregation of local states in a same global behaviour could be a too big approximation.

If we analyse the solution of reinforcement learning with regard to the specificities of MAS control problem, we notice that it is an effective experimental solution and a first step to the reduction of simulation time. RL implements a dynamical design of experiment on the estimation of the results triggered by the actions. In a given state, an action *a* is tested according to the exploration policy which takes into account the estimated reward associated to *a*. Therefore, the better an action is, the higher the probability to test it is. In other words, the actions that seem the most promising have the best estimation, and we do not lose time with the estimation of irrelevant actions.

We wish to underline that in such a representation, an action is not necessarily a quantitative modification of a parameter value like in parameter setting, but can also be any qualitative change of the system, like adding an obstacle.

5 Implementation on a Pedestrians Model

We expose in this section an application of our proposition on a toy example that models pedestrians. The goal of this part is to illustrate how our approach can be applied and to demonstrate its feasibility. The three steps of our method are developed. The first step gives a solution to the global behaviour characterisation issue (for this system only), in the second step we choose the action means, and in the last step we present how to use RL tools to learn a control policy.

5.1 Application Example

Our application example is a MAS that roughly models pedestrians walking in a circular corridor. Realism of the model is not relevant here, we just need a system complete enough to apply the proposition. Agents are leaded by a sum of forces (like boids [24]), which come from their own goals and from the repulsion with other agents and with the walls. This is a reactive multi-agent system, with many parameters to study, already seen in other works [25].

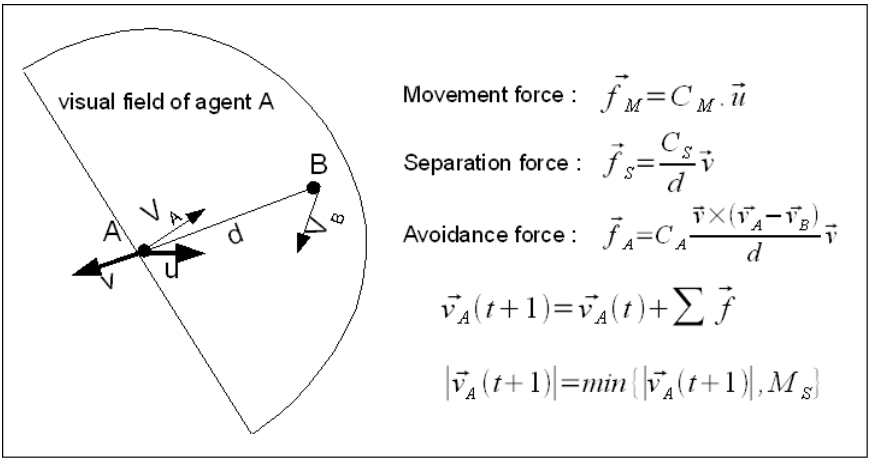


Fig. 4. Forces in the pedestrians system. Computing of the new speed of an agent A, that has a single agent B in its visual field. We assimilate the sum of forces to the acceleration because the mass of the agent can be considered in the coefficients Cm, Cs and Ca without adding a new parameter.

Specifically, agents have a dimension (they cannot overlap) and a visual field directed by their current speed. Each agent has a predilection direction in the corridor and undergoes a force in that direction proportional to a coefficient of movement *Cm*. When an agent has an obstacle (other agent or wall of the corridor) in his visual field, it undergoes an opposite force proportional to a coefficient of separation *Cs*. If the agent in the visual field goes in the opposite direction, a third force occurs, proportional to a coefficient of avoidance *Ca*. The sum of these forces modifies the agent speed according to its inertia. The speed norm is bounded by a maximum speed *Ms*

(Fig 4). Finally, the agent moves as far as possible in the direction of its new speed: until it reaches an obstacle or the speed norm.

When simulated, the system shows up emergent groups of agents: lines of same direction agents and blocks of opposite agents (see Figure 5).

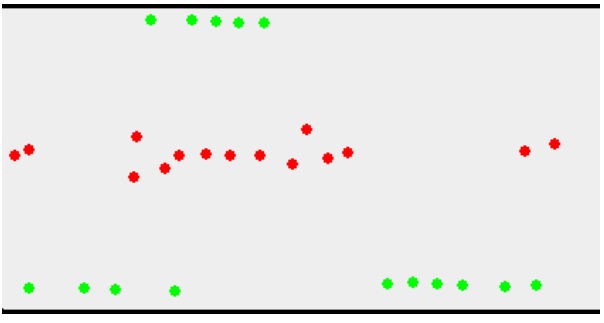


Fig. 5. Emergent structures. Three lines of agents emerging in the pedestrians system. Red (dark) agents go to the right and green (light-colored) agents to the left.

5.2 Global Behaviour Characterisation

The lines and blocks of agents are emergent structures of the system. Their arrangements define a global behaviour. But the description of the arrangements which characterise the global behaviour must be decided.

The global behaviour must be specified with regard to the goal of the controller. Even for such a simple system, several descriptions can be chosen for the global behaviour. If the controller's goal is to avoid blocks in the system, the global behaviour may be the presence or absence of blocks as structures involving agents, or the mean speed of each agent on a given time.

For our study, we consider that the target is to obtain a certain number of lines and a certain number of blocks: the behaviour is the number of each kind of group (lines and blocks), for instance one block and two lines.

Now, we have to find an automatic characterisation to determine the behaviour. Whatever the description, the point is to identify the emergent structures. In our case, this is a clustering problem, with an unknown number of clusters. We tried two solutions, the first using classical clustering tools and the second based on a decentralised clustering.

Many algorithms exist [26], which give a repartition of entities into clusters for a given number of clusters to find and a distance defined between these entities. For the pedestrians, the distance can be a combination of the difference between the positions and between the speeds of two agents. The difficulty is to determine the number of clusters, so we implemented a solution proposed in [27]: we used a hierarchical clustering algorithm to create clustering solutions for any number of clusters, and we compared them with regard to the dispersion inside each cluster and between the clusters to find the optimal number of clusters. The solution gives good results but lacks generalisation if the system is slightly modified: the distance could have to be redefined to match the new emergent structures.

Thus we propose a simple and partially decentralised method which can be generalised to some other systems. Here is the principle: we ask each agent A to remember which other agents are *often* in its visual field (*often* is to be defined, we took 70% of the 100 last steps of simulation), and we consider that these agents belong to the same cluster as A does. In a graphic representation, if the agents are vertices connected by an edge when they belong to the same cluster, the clusters are the strongly connected components of the graph. This allows us to determine the clusters in the system and gave empirically as good results as the former solution: when a global behaviour is unambiguously observed, the measure identifies the same behaviour.

5.3 Control Actions

Control actions can be distinguished between environmental and behavioural actions. The first ones consist in modifying the environment, like its size or its structure (obstacles), in order to make the agents that interact with these changes react. Even if they are centralised, these actions do not counteract the distribution of the system. The second kind of actions are modifications of the behaviour of an agent or a group of agents. In our study, we restricted the possible actions to these modifications of individual behaviours.

Since we wish to compare in a first time our proposition to parameter setting solution, we have to define a problem solvable by the latter: the only actions we take into consideration are modifications of parameter values (an action is the decision of the value of each controllable parameter). The parameters we choose to use are the coefficients Cm, Cs and Ca, and the maximum speed Ms (instead of, for instance, the number of agents or the environment size).

The modification of the value of an individual parameter is equivalent to change the behaviour of the agent. In that way, the proposition is close to the notion of adjustable autonomy [28], except that each agent has only one level of autonomy: it applies a policy chosen by the control system. The agent is not aware of the behaviour of the whole system, especially the target behaviour.

5.4 Model of the Dynamics of the MAS

Given the states and the control actions discussed before, the dynamics of the MAS could be represented by the Figure 6. But this is just a formal model, which shows the hypothesis that the Markov property stands at this level. The only useful information to control the MAS is to know which action to perform in each state. This is the goal of the RL algorithm presented in the next section.

5.5 Implementation and Algorithmic Choices

We summarise here the decisions we made in the implementation of our control solution. In a MDP view, the transition function of our system is unknown, so we chose to use a TD-learning algorithm to evaluate the influence of the actions and achieve the control of the system.

We saw in §4.3 why to choose a stochastic policy that could avoid the MAS to be in an absorbing state different from the target by repeating a wrong action. Many

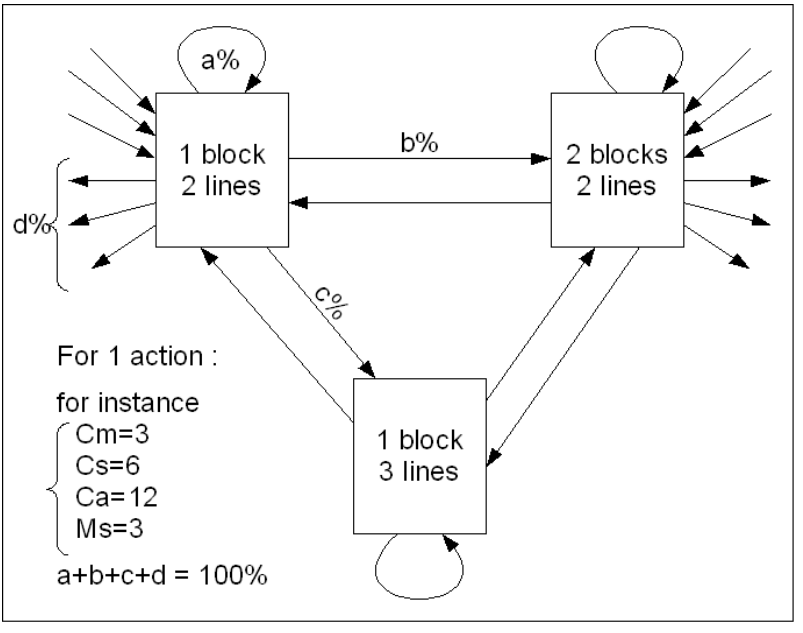


Fig. 6. Representation of the model of the global dynamics. The states are the global behaviours. There is a different graph for each possible action. The probability to escape from a state to another by performing an action is equal to the proportion of observed transitions between the two same states when this action is performed.

stochastic policies can be imagined, from the simple ε-greedy to Boltzmann policy for the most classical ones [23]. They differ on the choice of under-optimal actions, depending or not on how these actions are close to the optimal one. As our intent is just to achieve a better control than parameter setting allows (cf §6.2), not to create an optimal control model, we limit this study to ε-greedy policies.

Since the chosen policy is a stochastic one, we use an on-policy algorithm, namely the Sarsa algorithm, to learn this policy. Finally, to compute the return of an action in a given state, we use the classical value γ=0,9 for the discount rate.

The sarsa algorithm only stores the expected reward value for each action in each states Q(s,a), without explicitly computing the model. The complexity of this algorithm mainly depends on the number of values Q(s,a) to estimate. Since it gives a better estimation of the best actions, the number of actions is less relevant than the number of states for the complexity.

We give below the algorithms used to learn the policy and to control the MAS, without the details of the sarsa algorithm. Two limits are used in order to avoid to indefinitely wait for a simulation to stop: a maximum number of actions denoted k, and a maximum number of steps of simulation before a new state is identified.

A particular state s0 is added, where no behaviour is observable. In a simulation, if the number of steps is too high before a stabilised behaviour is identified, we consider that s0 is reached. The initial situation of a simulation is a random distribution of the agents in the environment, and the initial state is s0.

Learning algorithm

```
∀ state, ∀ action
  Q(state, action) ← 0
Repeat (nbSimulations)
  MAS.initialise() // random positions for instance
  S1 ← MAS.getCurrentState()
  // try to reach the target in less than k actions
  nbActions ← 0
  While (nbActions<k & S1 • target)
    // choice of an action
    action ←  -greedy(Q(S1, a))
    MAS.apply(action)
    nbActions++
    nbSteps ← 0
    // let the MAS stabilise in less than a maximum number of steps
    repeat
      MAS.simulate()
      nbSteps++
    until (MAS.isCurrentStateIdentified()
           OR nbSteps = limit) // the current state is then s0
    // update Q-values with sarsa
    S2 ← MAS.getCurrentState()
    sarsa(Q(S1,action), S2)
    S1 ← S2
```

Control algorithm

```
MAS.initialise()
// try to reach the target
Repeat forever
  currentState ← MAS.getCurrentState()
  // choice of an action
  action ←  -greedy(Q(currentState, a))
  MAS.apply(action)
  nbSteps ← 0
  // search for a stable state
  repeat
    MAS.simulate()
    nbSteps++
  until (MAS.isCurrentStateIdentified()
         OR nbSteps = limit)
```

6 Experimental Comparison of Control Solutions

The goal of this section is to evaluate and compare the performances of three different control solutions: a parameter setting based method, our reinforcement learning proposition and a reference control method based on random actions. We first present two scenarios corresponding to the control problems on which the methods are applied.

Then we present the performance measures used to evaluate them. The control methods and their implementation are clarified in a third sub-section. Finally we expose the experimental results and compare them.

6.1 Scenarios

Two control problems – or scenarios – have been tested, each one involving a target behaviour and possible actions. In the first problem, the target behaviour to reach is one block and two lines of agents. We control the agents thanks to two parameters: the coefficients of movement Cm and separation Cs (§5.1). These two parameters can take five values each, so that we have only 25 possible actions.

In the second problem, the target behaviour presents two lines and no block, and the controlled parameters are Cm, Cs, and the maximum speed of the agents Ms. The last parameter can take 5 values too, bringing the number of different actions to 125.

The parameters are discrete because it is simpler to solve in a first study, and it is enough to prove the utility of reinforcement learning tools. RL algorithms exist in continuous spaces (of states and especially of actions [29]) and could be used if necessary for continuous parameters.

6.2 Performance Measures

A control method is evaluated with regard to the adequacy between the target and the behaviour of the system obtained using it. The evaluation is statistical and involves two aspects: the ability to reach the reference and the difficulty or the time necessary to reach it. The first aspect can be approximated by the proportion π of simulations that reach the reference when the method is used to control the system, and the second by the average number ν of actions needed before to reach the reference.

A simulation begins by setting the MAS in a random local state. The controller then decides what action to realise, and the MAS is let run with the specified parameters until a global behaviour is identified by the measure. This step is repeated until a stop criterion: either the target behaviour is reached or a maximum number of steps k has been performed and we consider that the MAS will never reach the target.

We took k=50, what can be experimentally justified by plotting the repartition of the simulations by their number of steps. For instance, the figure 7 represents the plotting of 300 simulations following the policy computed with the RL method on the first problem. We can see that a great majority of simulations that reach the reference do it after the first few actions. Hence there is no use to consider too many simulation steps after the 30^{th}: k=50 seems a good balance between a good approximation and the limitation of the simulation time.

A total of n simulations are realised, so as the proportion of convergence π is approximated by the number m of simulations that reached the reference divided by n. The greater n, the smaller the approximation and we give the radius of the statistical confidence interval at 90%: π=80%±5 means that there are 90% chances that π is in [75, 85].

In order to estimate the average number of actions ν we only consider the number of steps of the m simulations that actually reached the reference, otherwise the simulations artificially stopped after k steps would bias the estimation.

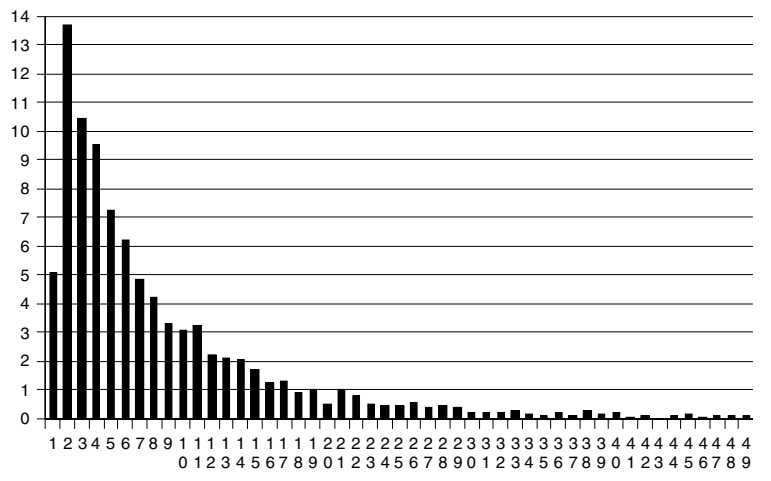


Fig. 7. Number of steps. Repartition in % of the simulations among the number of steps necessary before to reach the target: for instance, 3% of the simulations reached the reference after 10 steps.

6.3 Evaluated Control Methods

We compare three different methods to choose the actions to perform: a reference random choice, a parameter setting based method and our RL proposition.

The last one involves a learning phase where the Sarsa algorithm is used, and a control phase where the resulting policy is evaluated. In the learning algorithm, simulations are performed as in the control phase. We limited these simulations to k steps in order to learn the policy in the same conditions than its evaluation. The following results are given for 3000 learning simulations and n=300 evaluation simulations.

The parameter setting solution presents the two same phases of learning and control, but with its static nature the simulations are of only one step long. We learn the best parameter setting action by computing 500 simulations for each action. In order to get performance measures comparable to those of our proposition we slightly improve the evaluation simulations. Instead of using a static policy we alternate the optimal found action and a random action: first we apply the optimal parameter values recommended by the parameter setting, then we set random values, and so on, until the target is reached or k actions have been performed. Hence we can compute estimations of π and ν comparable to those found with the proposed method.

Finally, we evaluate a reference random method that needs no learning phase since it chooses a random action at each step of the evaluation simulations. It is used to verify that both the other methods get better performances than if there was no learning at all.

6.4 Experimental Results and Discussion

In the proposed method, we use a stochastic, ε-greedy policy, but the value of ε is to determine so as to optimise the performances. In this study, the optimisation is

secondary, our goal is essentially to make sure that a dynamical control is applicable. Thus we do not try to find the best value of ε, but we give the results found for different values of ε.

Table 1. Evaluation of the control performances of the three methods (with different values of ε for our proposition) on two problems. The performance measures are the proportion of target reaching simulations π, the radius of its confidence interval at 90%, and the average number of actions ν needed before reaching the target.

Method		**First problem**			**Second problem**		
		π (%)	radius	ν	**π (%)**	radius	ν
Random method		**68,7**	4,4	15	**23**	3,1	15,5
Parameter setting		**89**	3	11,8	**48**	3,7	7,1
Proposed method (RL)	ε = 10%	**89,6**	2,9	10,6	**66,6**	3,5	11,4
	ε = 20%	**91,3**	2,7	7,4			
	ε = 30%	**93,2**	2,4	7,8	**60,8**	3,6	12,8
	ε = 40%	**94,2**	2,2	8,2			
	ε = 50%	**93,3**	2,4	9,7			

All the control performances of the three methods of each scenario are summarised in Table 1. In the first problem, our proposition improves the reference performances, with a raise of 25 points for π (69% to 94%), and twice less necessary actions (8 instead of 15). We observe an optimum close to ε = 40% whereas the classical value used for ε is 10%. Our solution is just slightly better than the parameter setting based method.

In the second problem, the improvement of the proportion π is even greater, from 23% to more than 60%, and the parameter setting method only reaches the target in 48% of the situations. Still, this method gives the best improvement for the number of actions needed. We see that the parameter setting makes it possible to reach quickly the target in simple situations, but our method has the target reached in more situations. Here, a less stochastic policy (with the classical value of ε = 10%) gives better performances.

The second problem has more possible actions, hence possibly more actions which lead the system away from the target. We can see here that a random policy gives performances dramatically lower than in the first problem. Thus a policy that chooses "good" actions triggers an improvement even better than it would do on a simpler problem. The difference between parameter setting and our proposition is explained by the fact that no action is "good" in every situations (states), but different best actions can be found for each state.

Finally, the proposed method proves to give a pragmatic and usable way to control MAS. An optimisation of the method can be done depending on the studied system and scenario, for instance by setting an optimal value for the parameter ε.

A difficulty has been ignored in the paper: the states of the MDP and the actions have been chosen *a priori*, but the main improvement of the proposition remains in

this choice and its experimental evaluation. An experimental approach imposes to repeat this step with different choices and to compare the results. For this reason, in particular, the chosen discretisation of the actions is to question just as the parameters and the use of the global behaviour itself.

7 Conclusion

In this paper we propose a first step in an approach to create a pragmatic and experimental control method for a multi-agent system, especially a reactive one, and to evaluate the control performances. The method involves three main steps: the characterisation of the global behaviour and its automatic measurement, the selection of the possible control actions, and the determination of a policy that indicates the actions to perform in order to reach a target.

The originality of our proposition is to control the system on-line, by considering its current state, so that if it undergoes perturbations we can counteract them. Furthermore, the complexity of the approach does not directly depend on the number of agents since the states description is global and involves all the agents, and since the actions can be described at a global level – instead of building the behaviours of each agent. So the complexity does not suffer the scalability of the MAS.

The proposition can be pragmatically applied thanks to the limitation of the number of necessary simulations, by focusing on the relevant actions with RL tools and by reducing the size of the exploration space with the use of global information instead of local one. We show that the use of global information and actions improve the controllability – in terms of rate of convergence to the target – compared to classical parameter setting solution, and allows to build a dynamic control method.

As it is presented, the proposition can be applied to a simulated MAS. The use of the computed model to control of a concrete, distributed system is feasible under two conditions: the control actions can be applied to this system, and the observation is sufficient to measure the global states.

The actions chosen in our application system are just an example of what can be done. Tuning identical parameters of all the agents can seem quite centralised, but the actions space can be changed without challenging the proposition. For instance, environmental actions can be considered, or the model computed in this paper can be applied only to a part of the agents. In this last case, a further study is needed to check the robustness of the model if a part of the agents is not controlled.

Another interesting further development would be to estimate global states thanks to local information when the controlled system is not fully observable. Anyway the automatic measure is already an estimation of the behaviour, and uses few global knowledge. But a study is necessary to know how the control behaves when the states used during the control are less reliable than the states used to learn the policy.

We also think that the states description could be optimised and that local or semi-local information available in a distributed system could be enough to control the system in a decentralised way.

Eventually, as a future technical improvement of the approach we could optimise the learning phase and the tools used. For instance other policies (Boltzmann) and algorithms (Sarsa(λ), [23]) could improve the control performances and the learning

speed. A constraint of the RL algorithms used is that they learn a policy for a single target and they do not allow to change the target without learning a new policy from the beginning. Other learning methods could be researched and applied to avoid this.

References

1. Ferber, J.: Multi-Agent System: An Introduction to Distributed Artificial Intelligence. Addison Wesley Longman, Harlow (1999)
2. Wegner, P.: Why interaction is more powerful than algorithms. Communications of the ACM 40, 80–91 (1997)
3. Edmonds, B.: Using the Experimental Method to Produce Reliable Self-Organised Systems. In: Engineering Self Organising Sytems: Methodologies and Applications, Springer, Heidelberg (2004)
4. Edmonds, B., Bryson, J.: The Insufficiency of Formal Design Methods - the necessity of an experimental approach for the understanding and control of complex MAS. In: Proceedings of the 3rd International Joint AAMAS 2004, pp. 938–945. ACM Press, New York (2004)
5. De Wolf, T., Holvoet, T.: Towards a Methodology for Engineering Self-Organising Emergent Systems. In: Proceedings of SOAS 2005, Glasgow, Scotland (2005)
6. Amblard, F.: Comprendre le fonctionnement de simulations sociales individus-centrées. Thèse de doctorat en Informatique, Université Clermont II (2003)
7. Sauter, J.A., Parunak, H.V.D., Brueckner, S., Matthews, R.: Tuning Synthetic Pheromones Withe Evolutionary Computing. In: Genetic and Evolutionary Computation Conference Workshop Program (GECCO 2001), San Fransisco, CA (2001)
8. Sierra, C., Sabater, J., Agusti, J., Garcia, P.: Evolutionary Computation in MAS Design. In: Proceedings ECAI, pp. 188–192 (2002)
9. Dréo, J., Petrowski, A., Taillard, E., Siarry, P.: Metaheuristics for Hard Optimization Methods and Case Studies. Springer, Heidelberg (2006)
10. De Wolf, T., Samaey, G., Holvoet, T.: Engineering Self-Organising Emergent Systems with Simulation-based Scientific Analysis. In: Brueckner, S., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.) Proceedings of the Third International Workshop on Engineering Self-Organising Applications, Utrecht, The Netherlands, pp. 146–160 (2005)
11. Fehler, M., Klügl, F., Puppe, F.: Approaches for resolving the dilemma between model structure refinement and parameter calibration in agent-based simulations. In: AAMAS 2006, pp. 120–122 (2006)
12. Calvez, B., Hutzler, G.: Automatic tuning of agent-based models using genetic algorithms. In: Sichman, J.S., Antunes, L. (eds.) MABS 2005. LNCS(LNAI), vol. 3891, pp. 41–57. Springer, Heidelberg (2006)
13. Narzisi, G., Mysore, V., Bud Mishra, B.: Multi-objective evolutionary optimization of agent-based models: An application to emergency response planning. In: Kovalerchuk, B. (ed.) The IASTED International Conference on Computational Intelligence, CI 2006 (2006)
14. Klein, F., Bourjot, C., Chevrier, V.: Approche expérimentale pour la compréhension des systèmes multi-agents réactifs. In: JFSMA 2006, Annecy (2006)
15. Calvez, B., Hutzler, G.: Ant Colony Systems and the Calibration of Multi-Agent Simulations: a New Approach. In: MA4CS 2007 Satellite Workshop of ECCS 2007 (2007)
16. Brueckner, S., Van Dyke Parunak, H.: Resource-aware exploration of the emergent dynamics of simulated systems. In: AAMAS 2003, pp. 781–788 (2003)

17. Campagne, J.C., Cardon, A., Collomb, E., Nishida, T.: Using morphology to analyse and control a Multi-Agent system, an example. In: STAIRS ECAI 2004 (August 2004)
18. Campagne, J.-C., Cardon, A., Collomb, E., Nishida, T.: Massive multi-agent systems control. In: Hinchey, M.G., Rash, J.L., Truszkowski, W.F., Rouff, C.A. (eds.) FAABS 2004. LNCS, vol. 3228, pp. 275–280. Springer, Heidelberg (2004)
19. Bernon, C., Camps, V., Gleizes, M.-P., Picard, G.: Engineering Adaptive Multi-Agent Systems: the ADELFE Methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, June 2005, pp. 172–202. Idea Group Pub. (2005)
20. Bernon, C., Gleizes, M.-P., Picard, G.: Enhancing Self-Organising Emergent Systems Design with Simulation. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS, vol. 4457, pp. 284–299. Springer, Heidelberg (2007)
21. Thomas, V., Bourjot, C., Chevrier, V.: Interac-DEC-MDP: Towards the use of interactions in DEC-MDP. In: Third International Joint Conference on Autonomous Agents and Multi-Agent Systems - AAMAS 2004, New York, USA, pp. 1450–1451 (2004)
22. Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of markov decision processes. Mathematics of Operations Research 27(4), 819–840 (2002)
23. Sutton, R., Barto, A.: Reinforcement Learning: an introduction. MIT Press, Cambridge (1998)
24. Craig Reynolds' boids; `http://www.red3d.com/cwr/index.html`
25. Lacroix, B., Mathieu, P., Picault, S.: Time and Space Management in Crowd Simulations. In: Proceedings of the European Simulation and Modelling Conference (ESM 2006), Toulouse, France, pp. 315–320 (2006)
26. Jain, A.K., Murty, M.N., Flynn, P.J.: Data Clustering: A Review. ACM Computer Survey 31(3), 264–323 (1999)
27. Handl, J., Knowles, J.: Multiobjective clustering with automatic determination of the number of clusters. In: Technical Report TR-COMPSYSBIO-2004-02. UMIST, Manchester (2004)
28. Scerri, P., Pynadath, D.V., Tambe, M.: Towards Adjustable Autonomy for the Real World. J. Artif. Intell. Res. (JAIR) 17, 171–228 (2002)
29. Van Hasselt, H., Wiering, M.: Reinforcement Learning in Continuous Action Spaces. In: Proceedings of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), Honolulu, HI, USA, pp. 272–279 (2007)

Peer Pressure as a Driver of Adaptation in Agent Societies

Hugo Carr[1], Jeremy Pitt[1], and Alexander Artikis[2,1]

[1] Electrical & Electronic Engineering Department,
Imperial College London, SW72BT
[2] Institute of Informatics and Telecommunications,
National Centre for Scientific Research "Demokritos", Athens 15310

Abstract. We consider a resource access control scenario in an open multi-agent system. We specify a mutable set of rules to determine how resource allocation is decided, and minimally assume agent behaviour with respect to these rules is either selfish or responsible. We then study how a combination of learning, reputation, and voting can be used, in the absence of any centralised enforcement mechanism, to ensure that it is more preferable to conform to a system norm than defect against it. This result indicates how it is possible to leverage local adaptation with respect to the *Rules of Social-Exchange, Choice, and Order* to promote a 'global' system property.

1 Introduction

We are interested in engineering multi-agent systems for applications which require that the system be:

- open: in the sense of Artikis et al. [1] where agents are opaque, heterogeneous, may be competing, and may have conflicting goals;
- fault-tolerant: agents may not conform to the system specification, but the system should maintain operation, and demonstrate autonomic recovery;
- volatile-tolerant: agents may join and leave the system, but the 'system' itself remains recognisably the same even if all the components change;
- accountable: who performed which action, and to what effect, is significant, so social relations like trust, reputation, responsibility, liability and sanction are all significant;
- decentralised: there is no central mechanism for either knowledge or control, no agent is guaranteed to have full knowledge of the entire system or control over the behaviour of all other components;
- ruled by law: there is a theoretical limit on those making decisions affecting the constraints and/or requirements of behaviour of other components;
- mutable: there is a mechanism by which the specification itself can be changed by the expressed consent of the participants.

Our approach to satisfying these requirements is based on organised adaptation of agent societies. By an agent society we mean a formal specification of:

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 191–207, 2009.

- A set of social constraints (physical capabilities, institutional powers, norms (permissions, obligations, and prohibitions), sanctions, and enforcement policies)
- A communication language
- A social structure (roles and the relationships between roles)
- Other socio-cognitive relations between agents (e.g., in particular, trust).

By organised adaptation we mean the intentional modification of such a specification to achieve a commonly-understood goal. This requires understanding what can be adapted (for example, the set of social constraints, or individual behaviour wrt. to that set); when to adapt; how to adapt (e.g. by voting); and evaluating the outcomes of adaptation.

This is a wide-ranging programme of research, but within this paper we focus attention primarily on the adaptation of social constraints and relations with respect to the agent population to address the issue of fault tolerance (as here understood). We define these constraints in terms of the *Rules of Social-**:

- *Rules of Social Exchange* - The rules pertaining to the communication and interaction of individual agents with one another in the society (eg. gossiping) [2,3]
- *Rules of Social Choice* - The rules defining how agents' preferences and beliefs can be aggregated (eg. elections and voting protocols) [4,5]
- *Rules of Social Order* - Where the system characterises the permissions, obligations and (institutional) powers of each agent (eg. rights to system resources) [1,6,5]

We start from a scenario with multiple agents providing/consuming resources to/from a central repository. However, the set of resources requested is more than those available for distribution, so we define a set of social constraints which determine which agent is allocated resources. Depending on how 'sociably' the agents act during this negotiation, the system may be destroyed, either by agents becoming dissatisfied and leaving the system, or by the over-consumption of resources.

In this scenario, the allocation of resources is decided by a vote. However, Arrow's Impossibility Theorem [7] states that any non-dictatorial voting method can be manipulated by agents expressing a false set of preferences. Voters are therefore capable of either responsible or selfish behaviour, and have the option to choose between the two. But in a system with no social constraints, it is likely that they will objectively find that selfish behaviour yields a higher short-term return.

This suppression of collaboration has been widely studied in game theory as the Prisoner's Dilemma, but can be avoided if agents' reputations affect their global social standing [8]. As such, we present in this paper an agent endowed with a reputation monitor based on voting histories, a learning algorithm, and gossiping protocol. With these tools, we show that by adapting the *Rules of Social-**, it is possible to ensure that, in the absence of any central enforcement mechanism, it is more preferable (in the long run) to comply with a set of norms

than it is to violate them. For an open system, which any agent should be able to participate in, this is an important requirement for long term stability.

In the next section, we describe the basic scenario and multi-agent system in more detail. In Section 3 we describe the three primary mechanisms used in this paper: the exploration/learning method, the reputation mechanism, and election protocol. Section 4 describes experimental results from a 'society' of fifty agents implemented in the PreSAGE platform [9]. We discuss some related work and draw some conclusions in Section 5. In particular, we note that just by making the assumption of responsible or selfish behaviour (i.e. without compromising the assumption of heterogeneity), individual learning algorithms can be employed - and improved with gossiping - to facilitate the successful induction of a newcomer into an open agent society.

2 Background

2.1 Scenario and Multi-Agent System

The scenario is based on a 'tragedy of the commons' situation based on the scenario presented in [10]. We also present the animation/simulation platform which we have used to implement the system: further details of the platform can be found in [9].

There is a set of agents U, interacting during an infinite sequence of time slices $t_0, t_1, \ldots, t_n, \ldots$; with each agent requiring, at each time slice, access to resources stored in a bank B.

At each time slice, an agent may be present or absent: the set of agents present at any t is denoted by A_t, $A_t \subseteq U$. To satisfy each of their individual goals, each agent $a \in A_t$ offers, at each time slice, an allocation of resources O_t^a for B, and requests, at each time, an allocation of resources R_t^a from B. We stipulate that, for all $a \in A_t$, $R_t^a > O_t^a$: in other words, the agents can only satisfy each of their individual goals by mutual sharing their collective resources.

Clearly, not all of the requests can be satisfied without 'bankrupting' the system. Therefore, at each t, the set of present agents A_t take a vote on who should have their resource request satisfied. If an agent a receives a number of votes greater than or equal to a threshold τ_t then its request is granted. The problem then is that:

- If τ is too low, too many resources will be distributed, which this will result in the "Tragedy of the Commons" as the system is bankrupted;
- If τ is too high, too few resources will be distributed, which this will result in "voting with their feet" as dissatisfied agents leave the system.

The challenge then is for the agents to agree – again by a vote – a new value for τ in time slice $t + 1$ based on their prediction of how many agents will be present, available resources, and so on. In other words, they are adapting the *Rules of Social Choice*, specifically:

the resource controller is obliged to grant access to the resource to a requester, if the number of votes for the requester is greater than or equal to τ

by manipulating the value of τ. We define responsible behaviour to be recognised as voting for an a value of next-τ which will not bankrupt the system or under distribute resources.

Formally, the external state of the multi-agent system $\mathcal{M}$ is specified, at a time-slice t, by:

$$\mathcal{M}_t = \langle U, \langle A, \rho, B, \mathbf{f}, \tau \rangle_t \rangle$$

where:

$$\begin{aligned}
U &= \text{the set of agents} \\
A_t &\subseteq U, \text{ the set of } \textit{present} \text{ agents at } t \\
\rho_t &: U \rightarrow \{0,1\}, \text{the presence function s.t. } \rho_t(a) = 1 \leftrightarrow a \in A_t \\
B_t &: \mathbb{Z}, \text{the 'bank', indicating the overall system resources available} \\
\tau_t &: \mathbb{N}, \text{the threshold number of votes to be allocated resources} \\
\mathbf{f}_t &: A_t \rightarrow \mathbb{N}_0
\end{aligned}$$

The resource allocation function $\mathbf{f}_t$ is constructed by:

$$\begin{aligned}
\mathbf{f}_t(a) &= R_t^a, \mathbf{card}(\{b | b \in A_t \wedge \mathbf{v}_t^b(\ldots) = a\}) \geq \tau_t \\
&= 0, \text{otherwise}
\end{aligned}$$

where $\mathbf{v}_t^b : (\ldots) \rightarrow A_t$ is the expressed preference (vote) of agent b in time-slice t, whose inputs are local parameters (in particular agents' reputations) and whose output is a preference array of agents in A_t. This second vote is the mechanism which agents use to decide on who is empowered and permitted to receive and use resources this round, and is therefore an adaptation of the *Rules of Social Order*.

2.2 Simulation/Animation Platform

To animate this system and experiment with different agent behaviours, we have used the agent society animation/simulation platform PreSAGE [9]. PreSAGE is a rapid prototyping tool whose emphasis is on the simulation of agent societies and the social relationships between agents, intended to facilitate the study of the social behaviour of components, the evolution of network structures, and the adaptation of conventional rules. To develop a prototype, it is necessary to define agent participant types: this can be done by extending the abstract class supplied with PreSAGE (to guarantee compatibility with the simulation calls and provide core functionality like message handling etc.) or by defining a new class.

To define the participant class for our purposes, we extend the PreSAGE abstract participant with the following data and functions (we drop the superscript a since it is implicit from context):

⟨ Name	a,
Presence	$p : t \rightarrow \{0, 1\}$,
Resources offered	$O : t \rightarrow \mathbb{N}$,
Resources required	$R : t \rightarrow \mathbb{N}$,
π	a set of predictor functions which compute $\lvert A_{t+1} \rvert$
A set of Norms	ν
Norm Utility	$N : \nu \rightarrow [0, 1]$
Reputation Monitor	$r : U \rightarrow \{0, 1\}$
Satisfaction	$\sigma \in [0..1]$,
Satisfaction Increase Rate	$\alpha \in [0..1]$,
Satisfaction Decrease Rate	$\beta \in [0..1]$,
$\mathbf{v}$	voting function which maps a list of agents' historical actions, to an ordered list of agents A_p representing a preference array ⟩

The combination of the reputation tracking, norm evaluation and voting are defined in section 3 as the tools we use to show how we can harness peer pressure in the system.

Each time slice sees each active agent follow the system cycle:

Phase 1: set threshold
  $A_t = \{a | a \in U \wedge \rho_t(a) = 1\}$
  each agent $a \in A_t$ uses π^a to propose and **vote** on a value for τ_t
Phase 2: resource request
  each agent $a \in A_t$ offers resources O_t^a, and requests resources R_t^a
  each agent $a \in A_t$ computes **reputation** values for each agent $b \in A_t$
Phase 3: resource assignment
  each agent $a \in A_t$ uses $\mathbf{v}^a$ to **vote** for a vector of agents comprised of $a \in A_t$
  $\mathbf{f}_t$ is computed from the votes cast and τ_t
Phase 4: update
  B_t is updated according to the resources allocated
  each agent updates its satisfaction rating (see below)
  each agent updates its utility histories for each Norm for **reinforcement learning**

Phase 4 is where an agent evaluates its personal success and potentially changes its behaviour to try to improve its standing. In section 3, we elaborate in more detail how our participants use the norm evaluation and reputation monitors towards this goal.

2.3 Related Research

Although the use of learning techniques to change system parameters is addressed in [12, 13], the scenario described here defines an institution to be the sum of its participants rather than a separate entity. If we define this system as a set of agents forming an institution, it should be emphasised that the only 'universal'[1] truth in

[1] i.e. A brute fact not constructed relative to the agents' beliefs (like for example A, ρ) or a matter of conventional significance (B,$\mathbf{f}$, τ).

$\mathcal{M}_t$ is the existence of an agent. All other social constructions exist as projections of a composite of the beliefs of the population.

Normally we would begin by defining the existence of an agent in propositional logic, and construct the institutional facts from this basis.

$$\begin{array}{rcl} \text{First Order Logic} & & \text{Set Theory} \\ agent(a) & \equiv & a \\ member(a) & \equiv & a \in U \\ present(a,t) & \equiv & a \in A_t \end{array}$$

In this formalisation however, we have considered the concept of membership of the institution ($a \in U$) and the propositional fact $agent(a)$ to be equivalent. This should not be confused with membership of the temporal sub group of U, i.e. $a \in A_t$ which is an institutional fact rather than a brute fact. This can be illustrated if we assume all participants believe a to be active in time slice t. If a did not share this belief, relative to the institution it would still be true, including all the implicit sanctions regarding unfulfilled obligations.

Therefore if an agent is empowered by an institution norm to perform a speech act in the sense of Jones et al. [11] we should not infer the institution is tangible, the institution exists only as a marker to supply context. If agents believe these norms to be true, then relative to their perception they are. In another example, although we have referred to 'the bank' as an external agency, a physical exchange of resources never takes place, the fact that agents believe the resource has changed hands is sufficient. Whether the resource itself takes physical form and is centralised is a separate problem, as it is only entities that hold beliefs about the institution which make up a society.

The research most closely aligned with the current work is the foundational work of Axelrod [14] on the evolution of norms[2]. In this work, he posited a *norms game*, in which an individual has an opportunity to defect against a norm, as determined by its propensity to *boldness*. If it defects, then it gets a positive payoff, and all the others get a negative one. A defecting agent may or may not be seen (to defect); if it is seen by another agent then the agent may choose to punish or not, as determined by its propensity to *vengefulness*. If it chooses to punish then the punished agent (the defector) gets a large negative outcome, and the punishing agent a small negative outcome (i.e. there is a cost associated with enforcing a norm).

Axelrod ran computer simulations where the population changes over a sequence of generations, whereby those agents with more successful boldness/vengefulness strategies produce more descendants than less successful ones (keeping a fixed population size). The outcome was, starting from an average level of boldness and

[2] N.B. An Axelrod norm refers to a general social norm, which can arguably be deconstructed in terms of the norms defined by Pitt et al [5]. As an example, a guest at a dinner party is generally obliged to request permission to smoke from an empowered entity; which by convention is the host. We do not make a distinction between defecting against an Axelrod norm, or a norm as defined here.

vengefulness: first boldness fell, because it was costly to be bold when vengeance was (relatively high); then vengefulness fell, as was is costly to be vengeful without direct benefit; then boldness rose sharply, destroying the restraint originally shown – as Axelrod notes: "a sad but stable state" ([14]:p1100).

To redress this situation, Axelrod introduced a variant of the game with a *metanorm*, in this case the punishment of defection may or may not be seen, and not punishing itself may be punished. So there is some incentive to be vengeful. Axelrod simulations now showed that if a population started with 'sufficient' vengefulness the restraint could be maintained, but if not, then the metanorms game collapsed just like the norms game.

To some extent, the scenario in this paper is a partial reconstruction of Axelrod's norms game, with some important variations. The four phases of each time slice are comparable to one path in the norms game, where (Phase 1) the agents vote either selfishly or responsibly (defect or do not defect); (Phase 2) every agent sees what each other agent has done; (Phase 3) agents punish defectors through the reputation mechanism; and (Phase 4) agents update behaviour based on the reinforcement learning (notionally equivalent to the production of the next generation).

However, in our scenario, there are pre-established conventional rules, with a meta-norm enforcing a 'sociable' adaptation of these rules. Therefore we do not deal with generations of agents and the evolution of norms, but with one generation (whose numbers may change, e.g. when a new selfish agent is introduced) and the robustness of its norms wrt. maintaining a stable state in face of potentially disruptive components (i.e. when a new selfish agent is introduced). Our agents do not perform game-theoretic decision making along boldness vs. vengefulness dimensions, but instead express a preference (a vote) based on a larger number of local parameters, thus the internal complexity of our agents, while hidden from general view, is greater than the individual players of Axelrod's game.

Furthermore, the two votes required, one for the value of τ and one for the candidate order, require each agent to express preferences. This signalling introduces an element of communication which side-effects the game, and in combination with the reputation system and learning (individual adaptation) provides the robustness to resist the disruption of selfish agents.

3 Mechanisms for Peer Pressure

3.1 Overview of Mechanisms

Figure 1 illustrates the cycle through which we can maintain a stable system with self-enforcing behaviour using peer pressure. The voting functions and patterns of each agent are public, and will feed into the reputation monitors of each participant. The reputation monitor then generates a list of preferred agents derived from how socially each agent is perceived to be acting. This vote generates a result which depending on a win or loss of resources, drives the learning

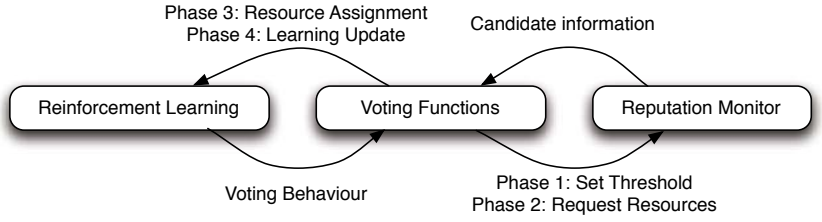


Fig. 1. The mechanisms and dataflow for each timecyle

algorithm of an agent. An agent will then choose a voting behaviour (social or antisocial) which the learning algorithm calculates to give the highest return.

3.2 Voting Functions

As defined in section 2.1, τ will symbolise the threshold number of votes an agent requires from the population to receive resources, and is the metric the agents use to adapt the *Rules of Social Choice*. In order to select an appropriate value we must ensure that the electorate is not split. To avoid this we select τ using a two round election system where round one consists of suggestions for τ, and round two a vote between the two most popular suggestions. This way even if the responsible voters are split on their suggestions, it is probable that at least one of the popular choices is closer to theirs than a selfish one.

Once τ has been decided we can move on to resource requests, offers, and the main vote for who will receive resources this time slice. For simplicity we have fixed the resource requests and offers as we believe the significance of the introduction of a normal distribution would not justify the increase in complexity[3]. We have found a plurality vote in this round to be ineffective for discouraging antisocial behaviour, as the ideal value of τ for a system cycle, tends to be less than or equal to the number of votes that an agent is granted to use. For example if each agent is allowed to vote for two candidates to receive resources, the value of τ which will ensure a stable resource stockpile for a responsible population, tends to oscillate around two. Therefore if we allow agents to use these votes for themselves, selfish behaviour will almost always be rewarded. We need to force agents to be less introspective, as the solution to this problem lies in the opinion an agent has of its neighbours.

To this end we have changed the system's main voting protocol to Borda which requires agents to vote in the form of an ordered preference list[4]. Repetitions in the list will be ignored and incomplete lists will be penalised. Agents behaving

[3] It is the deficit which is important rather than how it is distributed. Although reputations could be influenced by greed ie. disparate offers and request, this would just reinforce existing mechanisms and not demonstrate anything extra.

[4] τ in this case, represents the minimum number of Borda points required to receive resources.

Table 1. Borda preference array interpretation used when aggregating the votes

Preference Array	Vote weight
p_1	x
p_2	x-1
.	.
.	.
p_{x+1}	2
p_x	1
p_{x-1}	0
.	.
.	.
p_m	0

selfishly are loyal to no one and will rank themselves before anyone else. Responsible agents however, will conscientiously rank the voters, leaving selfish agents with no extra Borda points. It will therefore be the agents behaving responsibly who will receive on average a larger number of Borda points. The interpretation of a Borda preference array can be viewed in table 1 in which x = the number of agents receiving points; we define this to be approximately half of the total population (ie. $|A_t|$).

3.3 Reputations and Behaviours

In order to implement a Borda election, agents must diligently rank their neighbours in a preference relation. This requires the use of a reputation monitor which can distinguish between selfish and social behaviour. Then depending on individual reputation values the preference relation can be constructed. Due to their activity profiles agents do not have perfect knowledge of the system. They may rejoin with no reputation information on a number of new agents. It is important that we carefully define how to recognise antisocial behaviour as quickly as possible.

The system has therefore been specified in terms of two poles of behaviour, responsible and selfish. Agents behaving responsibly are defined to be altruistic capitalists, putting the needs of the system before their own, but expecting some sort of return for their efforts. The priority lies in avoiding the tragedy of the commons and bankrupting the system. For agents which are attributed the same reputation we make sure to randomise their positions between one another. This can be achieved by basing a preference relation on a random variable which takes the reputation as an input rather than a strict ordering based on historical behaviour.

Agents behaving selfishly are simpler than their responsible counterparts as their duty is only to themselves. They will always vote so that they are first on their preference list and vote for a low value of τ; increasing the likelihood of receiving resources. These actions cannot be hidden from their neighbours so the gamble is that the 'pro' of voting for oneself will offset the 'con' of a poor social standing. Needless to say, a system comprised solely of selfish agents

would bankrupt, so our aim is to create a set of norms that rewards responsible behaviour while punishing selfish.

We define a set of prediction functions randomly initialised and attribute a set to each agent. These predictors are used by the responsible population to calculate the expected optimal τ for the coming time cycle. As the simulation progresses, agents gather historical information on what a good value of τ would have been in the previous timecycles. Agents can evaluate how successful their predictors are and rank functions returning a good estimate of τ_{opt} over less accurate ones.

We employ a set of predictors, each constructed using a randomly weighted average of historical values. We take x_i to be a random value between zero and one adhering to a uniform distribution.

$$w_i = \frac{x_i}{\sum_{\forall j} x_j}$$
$$pred = \sum_{\forall i} w_i.a_i$$

where a_i refers to the historical values.

The historical information is selected by collating all the votes and ranking the most popular agents. We then hypothetically give each agent resources until all the offered resources have been exhausted. The last agent to receive resources then becomes the benchmark for τ, and we enter the number of votes that it received as the historical τ threshold.

An important duty of the responsible population is to exclude agents which behave selfishly. This meta-norm is key to the peer pressure mechanism, and the sole sanction which we need in order to make it clear to newcomers which behaviours are acceptable. If a sanction for selfish behaviour didn't exist, the responsible agents would observe a higher return for selfish behaviour and consider defecting.

The way in which agents can recognise selfish behaviour lies in the voting for τ. An agent's reputation monitor will consider any neighbour to be selfish if their vote differs by the ratio threshold defined below. For these experiments, we have defined the assumption of selfish behaviour to be when $\gamma > 0.4$, but this will vary for different implementations.

$$\gamma = \begin{cases} \frac{|vote_{neigbour} - vote_{my}|}{vote_{neigbour}} & \text{if } vote_{neigbour} > vote_{my} \\ \frac{|vote_{neigbour} - vote_{my}|}{vote_{my}} & \text{if } vote_{neigbour} < vote_{my} \end{cases}$$

Note that an agent will immediately be relabelled as 'responsible' if it changes it's voting patterns accordingly, and vice versa. This is in keeping with Axelrod's work on promoting co-operation by immediately punishing and rewarding undesirable and desirable behaviour. Using this dynamic, agents can intelligently adapt the *Rules of Social Order* to maintain their a 'moral' context of acceptable and non-sanctionable behaviour.

3.4 Reinforcement Learning

Reinforcement learning demonstrates how an agent can test and evaluate system norms. With limited prior knowledge of the system an agent can ascertain through trial and error what are the globally acceptable norms. We use a basic exploration learning algorithm, which treats the dialogic framework as a finite state machine. An agent will maintain a history of previous actions and their respective payoffs. Using this information, depending on a random variable linked to an exploration metric, agents will either try to maximise their utility by choosing either by what it believes will been the most lucrative action, or 'explore' by choosing an arbitrary action.

We link the system norms to a set actions $x \in X$ in states $s \in S$ using the function $g(s)$, with buffers of size m saving reward information r_k at time k. The Norm evaluation function can be defined using:

$$N_{t+1}(g(s,x)) = \frac{1}{m}\sum_{i=1}^{m}(r_{k_i})$$

where

$$r_k \in [0,1]$$

Learning takes place during the declaration of the election result. The behaviour currently being used will interpret the list of winners as $r \in \{0,1\}$, and will update the history for this action accordingly. In our system we have only two histories to update as we have only one state and two 'actions' representing responsible and selfish behaviour.

Behaviour for the next timecycle depends on the exploration metric ε which we have fixed in these experiments to 0.95^t

$$y \sim U(0,1)$$

$$x' = \begin{cases} argmax_{x \in X} N_t\left(g(s,x)\right), & \text{if } y < \varepsilon \\ \text{Random Action}, & \text{if } y > \varepsilon \end{cases}$$

This allows a participant to try all possibilities, before eventually settling into what it considers to be a locally optimal behaviour.

4 Experimental Results

In this section we describe how an agent is animated in the system, followed by the experiments which have been run. We include several illustrative examples to underline the effects which peer pressure has on the system, and talk through the social forces that brought it about.

4.1 Agent Animation

Agents begin their life cycle with a role assignment we assume to have been established in advance. This can be done through a role assignment protocol as outlined in [5]. The chair of the session then calls for participation in the system, and the voters send confirmation messages. A voter has an activity profile which is linked to a Markov chain, resulting in a stable population, but as mentioned in previous sections they may refuse to participate if they no longer consider the system to be viable. A confirmation of participation is tantamount to a commitment to provide resources in this time slice, regardless of the result.

During the τ selection vote, agents' votes are kept public so they can update their reputation monitors accordingly. The main vote follows from the τ selection by opening a ballot for the agents who voted in the first round. It is at this point that agents need to form their preference arrays and send them to the chair for collation. After a defined time-out the chair will accept no more votes and calculate how many resources are left in the system post distribution. For brevity the optimal value of τ, for use in the prediction functions, is calculated by the chair and circulated with the election results. This reduces the complexity of the system significantly.

Once agents know whether they have received resources this timeslice, they are able to update how successful their state-action pair was this time slice, and adjust their satisfaction rating. Satisfaction is representative of an agent's overall success in the system and is maintained parallel to individual action histories. We define satisfaction to lie between zero and one, and to be governed by the equations:

$$\sigma_{t+1} = \sigma_t + (1 - \sigma_t)\,\alpha$$
$$\sigma_{t+1} = \sigma_t - \sigma_t \beta$$

where the former is used to improve an agent's satisfaction in a society, and the latter to regress it. α and β represent the satisfaction increase and decrease rates respectively.

We have included a gossiping routine, for some preliminary work on adapting the *Rules of Social Exchange*. This takes the form of sharing election results with one another. Agents can evaluate norms based on personal experience, and the experience of others, resulting in faster learning. These results are however at a preliminary stage, and agents to do not yet vote to change these rules.

4.2 Experiments

We have shown in [10] that this experiment is stable amongst a group of these agents who have a pre-established a responsible moral context. We plan to introduce several new populations of agents in an attempt to destabilise the system. These populations will take the form of a control group of responsible agents, a set of purely selfish agents, and finally two sets of learning agents as described in section 3, one which incorporates gossiping and another which doesn't. We will examine how and whether the learning/exploration algorithm works in conjunction with the reputation monitor.

4.3 Results

We include here an example of a simulation of 55 agents. We start with a community of 50 responsible and stable agents, and introduce a set of 5 new ones. We have chosen to illustrate the three possible convergences of the system. Either an agent is accepted, excluded permanently, or excluded until they conform to the system norms. The following figures are typical examples of simulation test runs as convergence analyses will take several weeks to compile. Figure 2 shows a control group of responsible agents being added to the system at timecycle 3000. This is to establish a benchmark on how quickly a set of agents will be accepted into the institution when then know in advance which norms to adhere to. It is clear from the graph that for any new population the exclusion mechanisms initially work against them, and that joining this society is a matter of 'speculate to accumulate'.

Figure 3 shows another control group of selfish agents being added to the system at timecycle 3000. This demonstrates the efficiency of the exclusion sanctions and the agents' ability to recognise selfishbehaviour. We note that around time cycle 4500 the satisfaction for the selfish agents greatly increases. This is due to a surplus of resources, and a low value of τ_{opt}. Selfish agents who always vote for a low τ become difficult to distinguish from responsible agents. An interesting side note we see that in Figure 3 the average satisfaction goes slightly up for the responsible agents when the selfish agents enter the system. This is because the selfish agents are being excluded from distribution of resources, and are effectively forfeiting their contribution.

Figures 4 and 5 are typical examples of a group of learning agents converging towards responsible behaviour. Figure 4 demonstrates how an agent which only learns based on personal experience of the system will eventually join the responsible population. We see from this typical run that the newcomers become indistinguishable from the original population around timecycle 5000. This is in contrast to Figure 5, where the convergence is much faster. This is because agents in Figure 5 use gossiping to share information on past successes and failures. This greatly improves the accuracy of the action histories in each agent, and appears to yield a similar convergence rate to the control group.

4.4 Summary of Results

The experiments reported here offer additional supporting evidence for Axelrod's original claims, make their own contribution, and serve as a basis for a successively richer set of experiments in further work. The experiments confirm, as stated by Axelrod, that norms and conventions are a powerful mechanism for resolving conflicts of interest in disputes between multiple parties even in the absence of a central authority. In addition, social norms (e.g. the 'norm' is to vote for a 'reasonable' value of τ) and social constraints (i.e. the reputation mechanism) work well in preventing minor defections given that the cost of enforcement is low. In their own right, the experiments show how effective it is to give control over the adaptation of rules to those whose outcomes are most

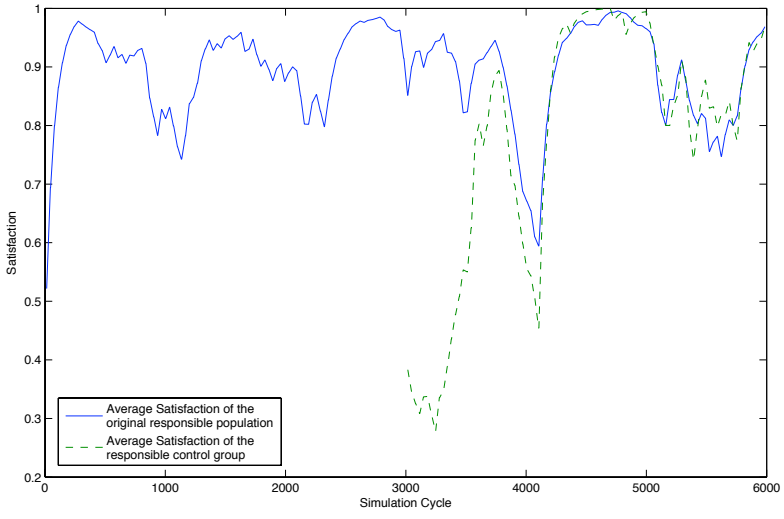


Fig. 2. Graph of the average satisfaction of an initially responsible population admitting a group of new responsible members at time cycle 3000

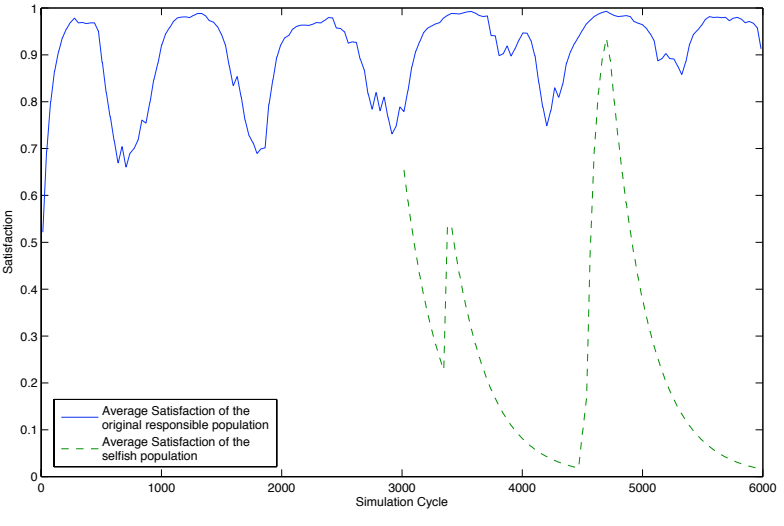


Fig. 3. Graph of the average satisfaction of an initially responsible population admitting a group of new selfish members at time cycle 3000

directly affected by the adaptation (cf. [15]), and how it is possible to leverage local adaptation with respect to a set of rules to achieve an intended 'global' system property.

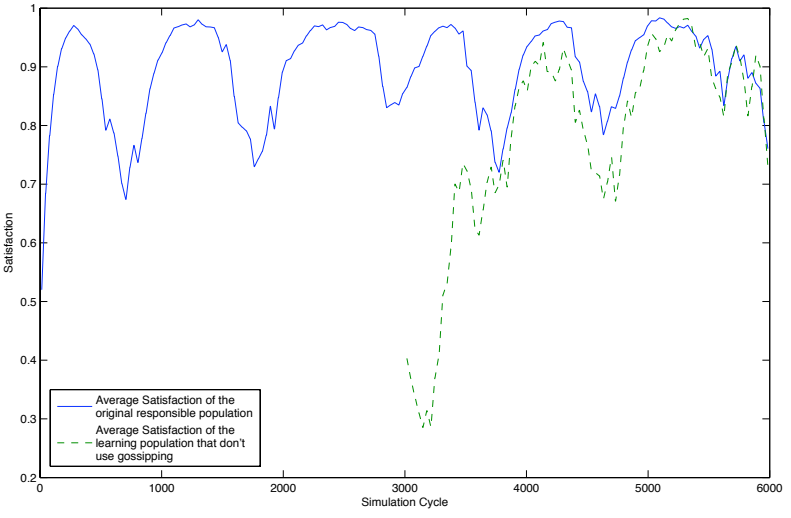


Fig. 4. Graph of the average satisfaction of an initially responsible population admitting a group of new learning members at time cycle 3000

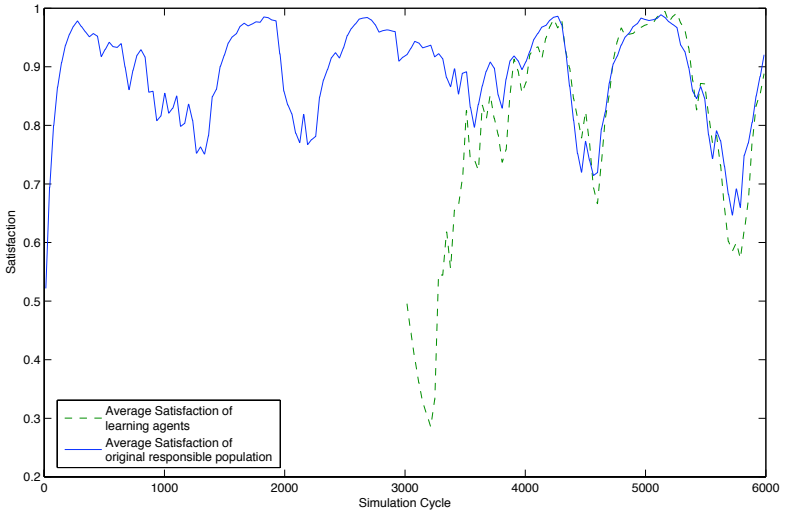


Fig. 5. Graph of the average satisfaction of an initially responsible population admitting a group of new learning members at time cycle 3000

There are several lines of further investigation opened up by this work. One is a more fine-grained behaviour rather than responsible or selfish. Rather, we would have a propensity to selfish behaviour, and correspondingly allow a propensity

to punish. This would necessitate a more subtle implementation of *forgiveness* which is an important element of autonomic systems for self-repair [16]. A second line of investigation concerns a peer to peer system allowing a more advanced form of 'gossiping' between agents, allowing groups to converge their opinions. For this, we could use the models of opinion formation formalised by [3].

5 Summary and Conclusion

In this paper we have outlined an agent society which maintains fault tolerance through peer pressure. We chose three mechanisms to create this dynamic: The reputation monitor, the reinforcement learning, and the voting functions. The agents used these tools to adapt the *Rules of Social-**, out of which emerged emerged a system comprised of responsible agents which when pitted against an unsure population effectively pressurised the latter into their preferred way of behaving (i.e conforming to a norm, in the sense of Axelrod).

Through the platform PreSage, we have shown that a social norm can be enforced in a system with a strong moral pretext without the use of a centralised enforcement agency, given that the cost of enforcement is low or non-existent. However, it would be interesting to investigate the effects of scale (size of population) and the corresponding increased cost of a more centralised enforcement mechanism. For example, in a relatively small society, enforcement could be based on peer-pressure, word-of-mouth or other reputation mechanism (as here) with low or no cost. In a relatively large society where neighbours are governed by a geolocation function, central reputation registers could be provided, with punishment provided by the equivalent of a 'police force', but at a much higher cost.

On a closing note, we also agree with Axelrod [14] when he observes that the probabilistic effects and complexities of population diversity make it difficult (if not impossible) to determine the consequences of a given behavioural model. However, computer simulation techniques offer a viable alternative which can reveal the system dynamics and stable states, as well as specific influence of identified agent behaviour profiles.

Acknowledgements

The first author is supported by a UK EPSRC studentship. The second author is supported by the EU FP6 Project ALIS 027958. Thanks for support to Lloyd Kamara, Brendan Neville, and Daniel Ramirez-Cano.

References

1. Artikis, A., Pitt, J., Sergot, M.: Animated specifications of computational societies. In: Castelfranchi, C., Johnson, L. (eds.) Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 1053–1062. ACM Press, New York (2002)

2. Pigozzi, G., Hartmann, S.: Aggregation in multiagent systems and the problem of truth-tracking. In: AAMAS 2007: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, pp. 1–3. ACM Press, New York (2007)
3. Ramirez-Cano, D., Pitt, J.: Follow the leader: Profiling agents in an opinion formation model of dynamic confidence and individual mind-sets. In: IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2006, December 2006, pp. 660–667 (2006)
4. Chevaleyre, Y., Endriss, U., Lang, J., Maudet, N.: A short introduction to computational social choice. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 51–69. Springer, Heidelberg (2007) (paper to go with Lang's, J. invited talk, to appear)
5. Pitt, J., Kamara, L., Sergot, M., Artikis, A.: Voting in multi-agent systems. Comput. J. 49(2), 156–170 (2006)
6. Artikis, A., Sergot, M., Pitt, J.: An executable specification of an argumentation protocol. In: ICAIL 2003: Proceedings of the 9th international conference on Artificial intelligence and law (June 2003)
7. Arrow, K.J.: A difficulty in the concept of social welfare. Journal of Political Economy 58, 328 (1950)
8. Axelrod, R.: The Evolution of Cooperation (2006)
9. Neville, B., Pitt, J.: Presage: A programming environment for the simulation of agent societies. In: Proceedings AAMAS Workshop on Programming Multi-Agent Systems, PROMAS 2008 (2008)
10. Carr, H., Pitt, J.: Adaptation of voting rules in agent societies. In: Proceedings AAMAS Workshop on Organised Adaptation in Multi-Agent Systems, OAMAS (2008)
11. Jones, A.J., Sergot, M.: A formal characterisation of institutionalised power. Logic Jnl IGPL 4(3), 427–443 (1996)
12. Bou, E., López-Sánchez, M., Rodríguez-Aguilar, J.-A.: Towards self-configuration in autonomic electronic institutions. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006. LNCS, vol. 4386, pp. 229–244. Springer, Heidelberg (2007)
13. Bou, E., López-Sánchez, M., Rodríguez-Aguilar, J.-A.: Using case-based reasoning in autonomic electronic institutions. In: Sichman, J.S., Padget, J., Ossowski, S., Noriega, P. (eds.) COIN 2007. LNCS, vol. 4870, pp. 125–138. Springer, Heidelberg (2008)
14. Axelrod, R.: An evolutionary approach to norms. The American Political Science Review 80(4), 1095–1111 (1986)
15. Ostrom, E.: Governing The Commons: The Evolution of Institutions for Collective Action. Cambridge University Press, Cambridge (1990)
16. Vasalou, A., Pitt, J.: Reinventing forgiveness: A formal investigation of moral facilitation. In: Herrmann, P., Issarny, V., Shiu, S.C.K. (eds.) iTrust 2005. LNCS, vol. 3477, pp. 146–160. Springer, Heidelberg (2005)

A Multi-Agent Resource Negotiation for the Utilitarian Welfare

Antoine Nongaillard[1,2], Philippe Mathieu[2], and Brigitte Jaumard[3]

[1] Dept. Computer Science and Software Engineering
Concordia University, Canada
[2] Laboratoire d'Informatique Fondamentale de Lille
Université de Lille 1, France
[3] Concordia Institute for Information Systems Engineering
Concordia University, Canada

Abstract. The multi-agent resource allocation problem is the negotiation of m resources among the n agents of a population, in order to maximize a social welfare function. Contrary to some former studies, the purpose is here neither to simply determine a socially optimal resource allocation nor to prove the existence of a transaction sequence leading to this optimum. The objective is to define the individual behavior of the agents which leads to an optimal resource allocation as an emergent phenomenon, based on any kind of contact network and on any utility value. With this intention, we study various agent behaviors in order to identify which one leads to an optimal resource allocation.

After a study of different transaction types, we show that, among the set of studied transactions, the so called "social gift" transaction, is the most efficient one for solving the utilitarian resource allocation problem.

1 Introduction

The resource allocation problem can be defined as the allocation of a set of resources among a set of entities, that have preferences over this resource set, in order to maximize an objective function. Centralized approaches tackle the multi-agent resource allocation problem as an optimization problem. Such centralized approaches require perfect information to determine an optimal allocation, whereas nowadays, people accept less and less to reveal their information. The agents have to report their preferences on the resources to an auctioneer, which then determines the final resource allocation. Different transaction models have been suggested for given types of auction [4,15]. These approaches make strong assumptions on the communication possibilities of each agent and the provided resource allocation may not be reachable in practice. The lack of adaptability of these centralized approaches constitute another important drawback. Indeed, a small variation in the data leads most of the time to a restart of the whole solving process. Thus, centralized approaches are not well-adapted to agent communities, especially when dynamic systems are considered.

Alternative approaches have been then studied in order to fulfil these requirements. These approaches are based on a population of autonomous agents

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 208–226, 2009.

who negotiate among them according to their own preferences. Then, the initial resource allocation evolves by means of local negotiations among the agents. Mathematical studies have been done along theorems on the existence or not of a transaction sequence leading to an optimal allocation [14]. Classes of utility and payment functions have also been studied [8] in order to design negotiation processes which end after a finite number of iterations. Some authors focus the acceptability criterion and the transaction properties [9]. They describe the impact of classes of deals on the efficiency of the resource allocation process. Other authors have defined agent behaviors, and identify conditions favoring equitable deals [10] or the envy-freeness in the resource allocation process [5,7]. However, these studies does not consider the agent network: they assume that an agent is able to talk with all other agents inside the community.

Most of these studies cannot be directly applied in real world application, due to assumptions made. In contrast, our multi-agent approach can be used in practice to solve problems like the selfish routing with incomplete information [11], some issues in peer-to-peer networks [16], or to provide an electronic commerce system in a social network. In this paper, we introduce the notion of contact network. Such a graph represents the relationship among agents. We consider that the contact network can be any connected graph, ranging from complete graphs, to small worlds [1]. We seek to find out the simplest and most efficient transactions, which lead negotiation processes to an optimal resource allocation as an emergent phenomenon, or when the need arises, to a socially close resource allocation.

Section 2 describes the resource allocation problem and the assumptions of this study. Section 3 details the centralized solution, whereas Section 4 describes the negotiation process and discusses its convergence issues. Section 5 details the experiment protocol and the evaluation criteria of the negotiation processes. Finally, Section 6 investigates further the social gift transaction, and the impact of the agent behavior in a negotiation process based on such transactions.

2 Multi-agent Resource Allocation Problem

2.1 Definitions and Notations

The multi-agent resource allocation problem is based on a population $\mathcal{P}$, $\mathcal{P} = \{a_1, \ldots, a_n\}$, and on a set $\mathcal{R}$ of available resources, $\mathcal{R} = \{r_1, \ldots, r_m\}$.

This set of resources $\mathcal{R}$ is initially distributed over the population of agents $\mathcal{P}$. Each agent a owns a bundle of resources, $\mathcal{R}_a$ which contains m_a resources. A resource allocation A is a distribution of all the resources over the agents, which can be expressed using the resource bundle of each agent: $A = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$. Let $\mathcal{A} = \{A_1, \ldots, A_p\}$ be the set of all the possible allocations.

The preferences of the agents are here expressed by means of a modular utility function [18], also called 1-additive function in [6,12]. They are defined using the following functions: $u_a : \mathcal{R}^{m_a} \rightarrow \mathbb{R}$ and $u'_a : \mathcal{R} \rightarrow \mathbb{R}$ with the following relationship:

$$u_a(\mathcal{R}_a) = \sum_{r_\in \mathcal{R}_a} u'_a(r).$$

Even if their mathematical definitions are different, since they are used in the same purpose, u_a will be used equally in order to simplify the notations.

The usual definition of a transaction is the following one: A transaction, which is initiated by the agent a, $\delta_a(A, A')$, is a pair of resource allocations, where A and A' define the state of the multi-agent system respectively before and after a given negotiation involving a given subset of agents. In practice, an agent does not have a global view of the system. This is the reason why, in our study, we consider that initially, agents only know their preferences and their neighbor list. This implies that transactions are based on local information only. Let $\mathcal{R}_{a \leftrightarrow a'}$ be the set of involved resources during a transaction between agents a and a'.

Definition 1 (Transaction). *A transaction, initiated by an agent a and in which agents $a', a'', \ldots$ are involved, is a list of resource sets that are exchanged between the agent-initiator and the involved agents.*

$$\delta_a = [\mathcal{R}_{a \leftrightarrow a'}, \mathcal{R}_{a \leftrightarrow a''}, \ldots]. \tag{1}$$

In our study, we focus on a homogeneous agent society, in which resources are assumed discrete, not sharable, not divisible, not consumable and unique. Hence, the resources cannot be modified by the agents, but only transacted during the negotiation process.

2.2 Contact Network

In real life, nobody knows everybody ! Thus, the relationships among the agents can be represented by means of a graph: the contact network. Each agent has a list of neighbors with whom he is able to talk. Most of the studies rely on the hypothesis of a complete and symmetric contact network. Symmetric means that if agent a knows agent a', then a' knows a. Complete implies that any agent is able to talk with any other agent in the multi-agent system: This has a strong impact on the resource traffic, and then on the negotiation process.

However, this hypothesis is not realistic as soon as real world applications are considered. For instance, in the case of social networks, a person only knows a subset of the overall set of actors in the network. In this study, we consider that the contact network can be any connected graph, ranging from a complete graph to a small-world [17].

According to the allowed transaction types, a negotiation process which converges towards an optimal resource allocation in the case of a complete contact network, may only converge toward a sub-optimal resource allocation in the case of a restricted contact network. The mean connectivity degree of a contact network is defined in this study as the average number of neighbors of an agent.

2.3 Social Welfare

Social welfare functions [2,3,13] are used in order to evaluate a multi-agent system like a whole, through a welfare evaluation of each agent in the system.

Definition 2 (Utilitarian welfare). *The utilitarian social welfare, denoted by sw_u, is defined as the summation of all agent welfare. For a given resource allocation A:*

$$sw_u(A) = \sum_{a \in \mathcal{P}} u_a(\mathcal{R}_a) = \sum_{a \in \mathcal{P}} \sum_{r \in \mathcal{R}_a} u_a(r). \quad (2)$$

The purpose of our study is to design the individual agent behavior which leads to an optimal resource allocation, as an emergent phenomenon, after a finite number of negotiations, based on any kind of contact network and on any utility value. This emergent resource allocation should be as close as possible to a socially optimal resource allocation, for any arbitrary connected contact network.

3 Centralized Approach

Of course, this resource allocation problem can be solved using a centralized framework. It can be simply solved by allocating each resource to the agent who estimates it the most, as described in Algorithm 1. This algorithm is quite

Algorithm 1. Centralized Solving

```
begin
    forall r ∈ R do
        bestAgent ← a ;
        bestValue ← u_a(r) ;
        forall a' ∈ P do
            if u_a'(r) > bestValue then
                bestValue ← u_a'(r) ;
                bestAgent ← a' ;
            end
        end
    end
    allocate r to bestAgent
end
```

simple, and gives an optimal resource allocation, for any utility values. However, such an algorithm does not consider the contact network and thus makes the implicit assumption that an agent is able to talk with any other agent. It may lead to a solution which is not reachable in practice. Moreover, there is no notion of private information, and a complete knowledge is then required, which is not realistic.

The social value, which is obtained with such a method, corresponds to a *global optimum.*

Definition 3 (Global optimum). *A resource allocation is a global optimum if there does not exist any other resource allocation with a better social value. A global optimum is independent of the kinds of transactions that are allowed among the agents. Moreover, the social value is unique but several resource allocations can correspond to it.*

Table 1. Agent preferences

Agents	Resources					
	r_1	r_2	r_3	r_4	r_5	r_6
a_1	10	7	10	9	2	1
a_2	6	10	3	4	8	3
a_3	1	2	1	2	1	6

Example 1. Let an example illustrate this centralized approach. Let us consider a multi-agent system which is constituted by a population of 3 agents, $\mathcal{P} = \{a_1, a_2, a_3\}$, and a set of 6 available resources, $\mathcal{R} = \{r_1, r_2, r_3, r_4, r_5, r_6\}$. Their preferences are expressed by means of a utility function, as described in Table 1. For instance, for the agent a_1, resources r_1 and r_3 are more valuable than r_4, which is itself more valuable than r_2, then r_5 and finally r_6. The centralized algorithm then returns an optimal allocation $A = [\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3] = [\{r_1, r_3, r_4\}\{r_2, r_5\}\{r_6\}]$ which is associated with the social welfare $sw_u(A) = 29 + 18 + 6 = 53$.

4 A Distributed Approach

The purpose of our distributed approach is to define the individual behavior of the agents which leads to an optimal resource allocation as an emergent phenomenon, at the end of the negotiation process. At the opposite of centralized approaches, the distributed approach presented in this section can be based on any type of contact network, as discussed in Section 2.2. The social value, which is obtained with the emergent allocation, corresponds to a *T-global optimum.*

Definition 4 (T-global optimum). *A resource allocation is a T-global optimum if there does not exist any sequence of transactions, belonging to the set of allowed transactions T, that leads to a resource allocation with a greater social welfare value. Such an allocation is most of the time suboptimal. If a T-optimum has a social value as great as a global optimum, then the distributed algorithm is efficient. However, depending on the initial allocation or on the allowed transaction types, this global optimum may not be reachable.*

In a multi-agent resource allocation problem, compensatory payments are usually allowed during a negotiation process. Allowing the compensatory payments, from an agent's point of view, corresponds to an extension of the acceptable transaction set. However, even if the use of money is constrained (no money creation during a transaction), there is often no limit on agent budgets in order to perform the transactions in most published studies. Questions related to compensatory payments are beyond the scope of our study and, hence, are not studied in the sequel. Our study is restricted to a specific transaction family: Bilateral transactions in which only two agents at a time can be involved. We next discuss the cases where transactions are rational or more generally subject to some acceptability criteria.

4.1 Acceptability Criteria

Such criteria have a strong impact on the negotiation process. In [14], it has been proved that there always exists a sequence of non rational *original contracts* leading, from any initial resource allocation, to an optimal resource allocation. An *original contracts* corresponds to the purchase of a resource. The existence of such a sequence does not mean the end of the negotiation process with the optimum. Indeed, if an agent can accept any kind of deal, then the negotiation process will not be able to stop. Since the agents are not rational, they are not able to distinguish a profitable deal from another one. Thus, without a proper design, no resource allocation will emerge from the negotiation process. It is really important to be sure that the process stops in a finite number of negotiations, in order to obtain an optimal resource allocation, without centralized management. Otherwise, even if the resource allocation process reaches an optimal state, the agents will continue to negotiate among them. Acceptability criteria help the agent to determine the profitability of a transaction, with respect to his behavior. Such criteria restrict a lot the set of possible transactions among the agents. A negotiation process ends when no agent in the population is able to find an acceptable deal.

Let two agents, a and a', illustrate the considered criteria. The agent a initiates a transaction $\delta_a(A, A')$ with an agent a': the initial resource allocation $A = [\ldots, \mathcal{R}_a, \ldots, \mathcal{R}_{a'}, \ldots]$ evolves towards a new one $A' = [\ldots, \mathcal{R}'_a, \ldots, \mathcal{R}'_{a'}, \ldots]$.

Definition 5 (Rational agent). *A rational agent is an agent who only accepts transactions that increase his utility. If the agent a is rational, he accepts a transaction only if:*

$$u_a(\mathcal{R}'_a) > u_a(\mathcal{R}_a).$$

The rationality criterion is the most widely used in the literature, especially in the case of non cooperative selfish agents.

Definition 6 (Rational transaction). *A rational transaction is a transaction in which all involved agents are rational. If a transaction is rational, involved agents accept it if:*

$$u_a(\mathcal{R}'_a) > u_a(\mathcal{R}_a) \quad \textit{and} \quad u_{a'}(\mathcal{R}'_{a'}) > u_{a'}(\mathcal{R}_{a'}).$$

Proposition 1. *A multi-agent resource allocation process that uses rational transactions ends after a finite number of transactions.*

However, this criterion restricts a lot the set of possible transactions, and may lead the negotiation process to a sub-optimal resource allocation.

Another criterion that ensures the end of the negotiation process after a finite number of transactions is the sociality. This criterion is based on a local evaluation of the social welfare evolution.

Definition 7 (Social agent). *A social agent is an agent who can only accept transactions that increase the social welfare function of the multi-agent system.*

Definition 8 (Social transaction). *A transaction is social if the value of the social welfare function considered increases. Such a transaction can only be accepted by the involved agents if:*

$$sw(A') > sw(A), \quad A, A' \in \mathcal{A} \quad \text{such that } A \xrightarrow{\delta} A'.$$

In order to determine the value associated with the social welfare function, it is essential to have a global knowledge of the multi-agent system state: The utility of each agent is used to compute the social value. However, it is possible to determine the variation of this value based on local information only: It is then not necessary to determine its value:

$$\begin{aligned} & sw_u(A') > sw_u(A) \\ \Rightarrow \quad & \sum_{a\in\mathcal{P}} u_a(\mathcal{R}'_a) > \sum_{a\in\mathcal{P}} u_a(\mathcal{R}_a) \\ \Rightarrow \quad & u_a(\mathcal{R}'_a) + u_{a'}(\mathcal{R}'_{a'}) > u_a(\mathcal{R}_a) + u_{a'}(\mathcal{R}_{a'}). \end{aligned}$$

Indeed, since only two agents are involved in the current transaction, only their resource bundle changes. Then, the utility of the agents that are not involved in the transaction can be considered as a constant value. Let us note that a rational transaction is always social, whereas the opposite is not true.

4.2 Transaction Kinds

Our study is restricted to bilateral transactions, i.e., transactions involving simultaneously two agents. Indeed, our aim is to define the simplest agent behavior which lead the negotiation process to an optimal resource allocation, in order to favor the scalability of the algorithm. Three types of bilateral transactions can be distinguished. Others are combinations of these basic transaction kinds. In each case, the agent a initiates the deal and negotiates with a neighbor a'. They respectively own m_a and $m_{a'}$.

First, *the gift* transaction. During such a transaction, the initiator gives one of his resources to the selected neighbor.

Then, *the swap* transaction. Each agent provides a unique resource. This transaction is symmetric, and the number of resources per agents cannot vary: an agent that initially owns m_a resources, will have the same number of resources at the end of a swap sequence. Hence, an optimal resource allocation can be reached only if the initial resource allocation has the same resource distribution as one of the optimal resource allocations. The total number of swaps between a and a' is $m_a \times m_{a'}$.

Finally, *the cluster-swap* (CS) is a transaction during which the agents can involve a subset of their resources. This transaction can be asymmetric. The swap is a particular case where both agents involve only one resource each. The number of possible cluster-swaps between a and a' is $2^{m_a+m_{a'}}$. The cluster-swap is not only a generalization of the previous transactions. Indeed, depending on the considered acceptability criterion, a cluster-swap transaction can not be

decomposed into a sequence of swaps and gifts where each transaction satisfies the same criterion.

When combining these three kinds of transactions with the acceptability criteria which are defined in Section 4.1, an exhaustive set of negotiation process can be defined.

1. the social gift,
2. the social swap,
3. the rational swap,
4. the social cluster-swap,
5. the rational cluster-swap.

A gift can be rational if the associated utility is negative. However, this transaction is not considered since it only allows agents to deal resources with a negative utility, which is too restrictive to obtain a significant welfare. Indeed, with such an allowed transaction kind, the negotiation process ends as soon as every agents have in their bundle only resources with a non-negative utility.

4.3 Negotiation Policies

The agent behavior is rooted and flexible, as described in Algorithm 2. An agent, who initiates a negotiation, randomly selects one of his neighbor. During this negotiation, he cannot change the selected neighbor. However, he can offer different resources. First, the agent-initiator sorts his resource bundle according to his preferences. Even if agents are not rational, they always try to give their cheapest resources first. Then, he offers his cheapest resource, and if the deal is not acceptable, he changes the offered resource by a gradual increase of its utility. If no acceptable deal has been found and the agent-initiator has no more resource to offer, the negotiation thus ends.

Algorithm 2. Behavior of the agent-initiator a

```
begin
    Sorts the resource bundle R_a ;
    Selects randomly a neighbor a' ;
    for r ∈ R_a do
        if the transaction δ_a is acceptable then
            Performs the transaction δ_a ;
            Ends the negotiation ;
        end
    end
end
```

Example 2. Let us consider the same multi-agent system as defined in Example 1, with 3 agents, 6 available resources and the preferences described in Table 1. The negotiation process starts from a random initial resource allocation $A = [\{r_3, r_5, r_6\} \{r_1\}\{r_2, r_4\}]$, associated with the utilitarian value $sw_u(A) = 13 + 6 + 4 = 23$.

Table 2. Example of a negotiation process: a gift sequence

Initiator a	Participant a'	Given Resource r	Current Allocation A	Welfare Value $sw_u(A)$
-	-	-	$[\{r_3, r_5, r_6\}\{r_1\}\{r_2, r_4\}]$	13+6+4 = 23
a_3	a_1	r_2	$[\{r_2, r_3, r_5, r_6\}\{r_1\}\{r_4\}]$	20+6+2 = 28
a_2	a_1	r_1	$[\{r_1, r_2, r_3, r_5, r_6\}\{\}\{r_4\}]$	30+0+2 = 32
a_1	a_3	r_6	$[\{r_1, r_2, r_3, r_5\}\{\}\{r_4, r_6\}]$	29+0+8 = 37
a_1	a_2	r_5	$[\{r_1, r_2, r_3\}\{r_5\}\{r_4, r_6\}]$	27+8+8 = 43
a_1	a_2	r_2	$[\{r_1, r_3\}\{r_2, r_5\}\{r_4, r_6\}]$	20+18+8 = 46
a_3	a_1	r_4	$[\{r_1, r_3, r_4\}\{r_2, r_5\}\{r_6\}]$	29+18+6 = 53

Table 2 described a sequence of social gifts, initiated by the agent a who involves the agent a' in a negotiation. Each line of the table describes the resource allocation and its utilitarian value for each step of the negotiation process. During the first gift, the agent-initiator a_3 gives the resource r_2 to the agent-participant a_1. The example can also be done with negative utility values.

4.4 Communication Protocol

In order to compare and evaluate the different types of transactions, we develop a multi-agent system with sequential negotiations: Only one agent at a time is able to negotiate. Note that if parallel transactions were performed, except maybe for very specific synchronization rules, it would only affect the convergence speed but not the quality of the final allocation.

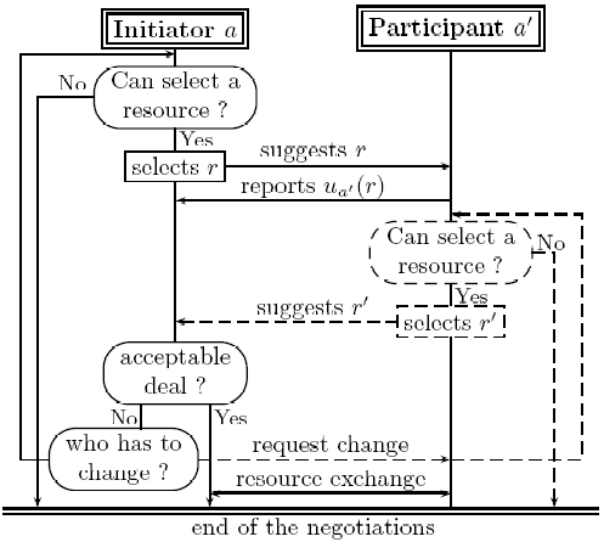


Fig. 1. Communication protocol

The agent-initiator is randomly chosen in the multi-agent system. Agents accept or refuse transactions according to their own criterion. The negotiation process ends when no agent is able to find an acceptable transaction in his neighborhood.

The communication protocol is described in Fig.1. In the specific case of gift transactions, during which only the agent-initiator gives a resource without counterpart, the dashed part of the figure should be omitted. When the agent-initiator a selects and offers a resource r, the involved agent a' has to report the utility that he associates with a resource r and offers a resource r'. Then, the agent-initiator determines whether or not the transaction is acceptable. He decides to perform the transaction if the acceptability criterion are satisfied for both agents, or he has to determine who has to change his offer and then suggests another resource. For instance, with a utilitarian social welfare function, the test can be done on the comparison of what agents give and what they receive: $u_a(r') - u_a(r) >^? u_{a'}(r) - u_{a'}(r')$. If no agent is able to suggest a different resource, the negotiation then ends.

5 Experiments

The experiments have been done on multi-agent systems of various sizes. For each of them, different types of contact network have been created, either with a complete connectivity or with a random one with a mean connectivity of 25% (i.e., $\frac{n}{4}$). For each setting, a large number of multi-agent systems has been generated, and in each case, 100 instances have been run using different initial resource allocations. The agent-initiator is randomly chosen, and the speech turn is uniformly distributed: no agent is able to talk twice if others did not talk at least once. When no one is able to find an acceptable deal, the negotiation process ends. Each resource can take an integer utility value between 0 and 100: The utility value range has to be great enough in order to avoid too many optimal resource allocations.

5.1 Evaluation Criteria

The evaluation of a negotiation process is not an obvious issue: Indeed, depending on the considered negotiation process, it is always possible to find a metric that makes it better than others. In order to avoid such biases, several metrics have been considered:

Number of performed transactions. It is the overall number of transactions that are performed during the negotiation process. Negotiations using restrictive transactions, such as rational transactions, stop faster than negotiations using more permissive transactions, such as social transactions.

Number of exchanged resources. Some transactions, such as the cluster-swap, tolerate that an agent involves more than one resource at a time whereas others prohibit this, such as the gift. One cluster-swap is equivalent to a sequence of, at least, two gifts.

Number of speech turns. It corresponds to the number of negotiation that are initialized. A rooted behavior, in which the initiator cannot change the selected neighbor, means a higher number of speech turns during the negotiation process. If associated with the number of performed transactions, the rate of aborted negotiations can be deducted.

Number of attempted transactions. Depending on the agent behavior, it could be more or less difficult to find an acceptable deal: a flexible agent behavior, in which the agent can change the involved resource, means a higher number of attempted transaction during a single negotiation. This measure gives an estimation of the negotiation length.

In addition to these criteria, we evaluate the gap between the optimal social value that is obtained by means of the centralized approach, and the social value associated with the emergent resource allocation. We also evaluate the relative standard deviation, which corresponds to the deviation among the social value associated with the emergent allocation. A large value means that the considered negotiation process is very sensitive to the initial resource allocation, and thus the quality of the emergent allocation quite vary.

5.2 Utilitarian Efficiency of the Transactions

A summary of all the experiments are presented in this section. First, the results related to a complete contact network are presented, then the results related to a random contact network with a mean connectivity degree of $\frac{n}{4}$. The size of the instances are characterized by n, the number of agents and m, the total number of resources that are distributed randomly at the outset.

The results obtained with a complete contact network are summarized in Table 3. For instance, the process involving a population of 50 agents who negotiate by means of social gifts leads to a resource allocation which corresponds to 95.27% of the optimal social value. The quality of the emergent allocation can vary of 0.74%.

The social gift and the social cluster-swap obtain almost similar results, which are better than the ones obtained with other transactions. Even if, in these experiments, a global optimum is seldom reached, the social efficiency of these two transaction kinds is high: more than 94% of the optimal value. It is thus possible to reach a resource allocation that is socially close to the optimal social value. The swap transaction, either social or rational, leads to the emergence of an allocation which is far from the optimal. This is due to the property of the swap, which preserves the initial resource distribution. Negotiation processes that use rational transactions stop further of the optimal social value than the ones that use social transactions, as a consequence of a more restrictive criterion. The size of the instances does not seem to have not a strong impact on the quality of the final allocation.

Figure 2 (a) shows the number of performed transactions, depending on the kind of contact network and on the kind of allowed transactions. The social criterion is more flexible, thus more transactions can be performed by social agents. The number of exchanged resources is greater in the case of social agents

Table 3. Social Efficiency(%) and relative standard deviation of the behaviors on a complete contact network

n	m	Social			Rational	
		Gift	Swap	CS	Swap	CS
10	60	94.25	79.01	95.01	64.08	90.44
		1.52	8.54	1.51	9.35	2.27
20	120	94.50	77.13	94.60	59.48	90.44
		1.24	7.78	1.12	8.33	1.79
50	300	95.27	73.58	95.82	54.79	90.31
		0.74	11.41	0.67	6.10	1.10

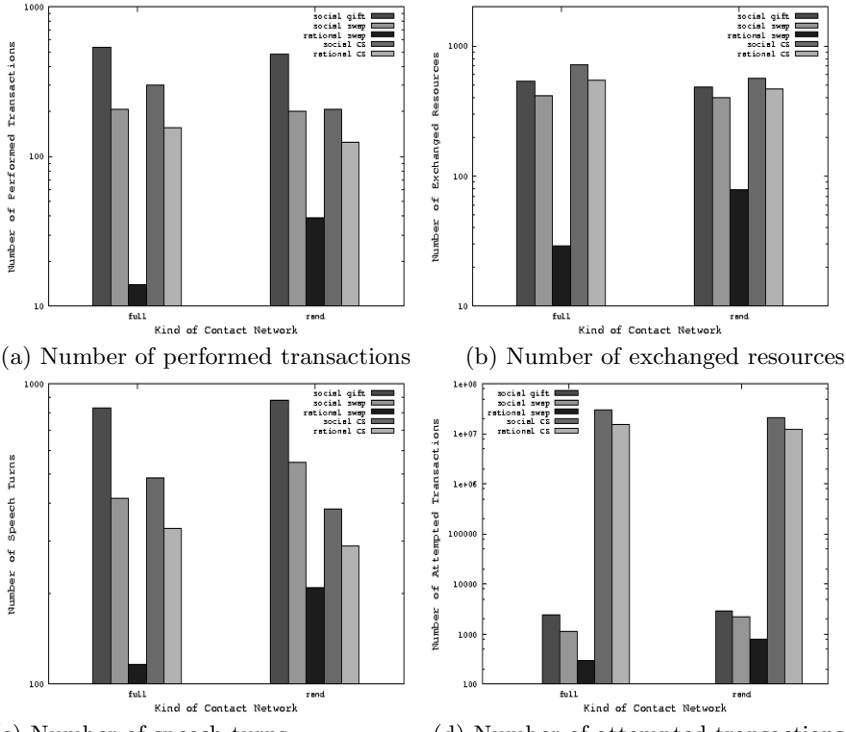
Table 4. Social Efficiency(%) and relative standard deviation of the behaviors on a random contact network

n	m	Social			Rational	
		Gift	Swap	CS	Swap	CS
10	60	86.54	78.21	84.94	65.70	83.03
		2.50	4.80	2.84	6.66	2.85
20	120	88.51	76.37	87.90	63.85	84.04
		2.01	3.99	1.46	4.95	1.91
50	300	90.10	75.57	91.83	61.20	86.69
		1.5	3.24	0.93	3.64	1.18

than with rational agents, however the difference is not significant as shown in Fig.2 (b). Figure 2 (c) describes the number of speech turns: the social gift requires a higher number of speech turns, due to the fact that only one resource can be exchanged at a time. Finally, Fig.2 (d) shows the number of attempted transactions. One can notice that using cluster-swap transactions leads to a very large number of attempted transactions, which means a greater communicational cost and a more time-consuming negotiation process.

Results based on a random contact network are shown in Table 4. The contact network itself has a large impact on the quality of the final allocation. Depending on the used transactions, the network limits more or less the resource traffic. During the experiments, the global optimum is never reached. The smallest gap is always obtained by the social gift. The negotiation process ends on socially weaker allocations if restrictive transactions are used. However, the weaker the connectivity of the contact network is, the larger the gap is.

Even if the social cluster-swap provides flexibility and good results, since its cost is exponential, its usage it not so interesting. Thus, the social gift is the best transaction and leads to resource allocations that are close to the optimal social value, as an emergent phenomenon. Since we identified the most efficient transaction, we next study different behaviors using the social gift in order to determine whether it is possible to improve the results or not.



(a) Number of performed transactions (b) Number of exchanged resources

(c) Number of speech turns (d) Number of attempted transactions

Fig. 2. Parameter comparisons for various transaction types

6 Social Gift

6.1 Behavior Variants

The behavior of the agents has an important impact on the quality of the resource allocation that is finally reached. In order to study further the influence of the agent behavior, the social gift is used on a complete contact network. If the agent-initiator and the selected neighbor find an acceptable transaction, they perform this transaction. In a case of a refusal, three different options are possible for the agent-initiator:

1. Aborts the negotiation,
2. Chooses another resource with the same neighbor,
3. Chooses another neighbor with the same resource.

Based on this option set, four different behaviors can be defined. For each behavior, the initiator a gives a unique resource according to the definition of the

gift in Section 4.2. After the identification of an acceptable deal or the end of the negotiation, a new initiator is randomly chosen.

First, behavior A is described in Algorithm 3. The agent-initiator a selects randomly a neighbor and tries to give his cheapest resource. If this is not an acceptable transaction, then the agent-initiator aborts the negotiation.

Algorithm 3. Behavior A - rooted and stubborn agent-initiator a

```
begin
    Sorts the resource bundle R_a ;
    Selects the cheapest resource r ;
    Selects randomly a neighbor a' ;
    if the transaction δ_a is acceptable then
        Performs the transaction δ_a ;
        Ends the negotiation ;
    end
end
```

If the agent-initiator a adopts behavior B described in Algorithm 4, then he selects randomly a neighbor and negotiates, starting with his cheapest resource and increasing its value gradually. If, no resource can be use to define an acceptable transaction, then the negotiation stops.

Algorithm 4. Behavior B - rooted and flexible agent-initiator a

```
begin
    Sorts the resource bundle R_a ;
    Selects randomly a neighbor a' ;
    for r ∈ R_a do
        if the transaction δ_a is acceptable then
            Performs the transaction δ_a ;
            Ends the negotiation ;
        end
    end
end
```

Behavior C in Algorithm 5 is defined as follows. The agent initiator a negotiates only his cheapest resource with all the agents of his neighborhood. In order to avoid a bias due to the sequential selection of the selected neighbor, a random permutation is applied on the neighbor list of the initiator. If no agent assigns a greater utility to the resource than the initiator, then the negotiation aborts.

Last, behavior D in Algorithm 6 is such that the agent initiator a negotiates every resource as for agent behavior C one after the other, with all his neighbors for each of them. The same technique is used in order to avoid the bias due to a sequential selection of the neighbor. After the negotiation of all his resources with all his neighbors, the agent initiator aborts the negotiation.

Algorithm 5. Behavior C - volatile and stubborn agent-initiator a

```
begin
    Sorts the resource bundle R_a ;
    Selects the cheapest resource r ;
    for a' ∈ P do
        if the transaction δ_a is acceptable then
            Performs the transaction δ_a ;
            Ends the negotiation ;
        end
    end
end
```

Algorithm 6. Behavior D - volatile and flexible agent-initiator a

```
begin
    Sorts the resource bundle R_a ;
    for r ∈ R_a do
        for a' ∈ P do
            if the transaction δ_a is acceptable then
                Performs the transaction δ_a ;
                Ends the negotiation ;
            end
        end
    end
end
```

6.2 Behavior Efficiency

These four behaviors have been evaluated using the criterion defined in Section 5.1. Experiments have been conducted on both kind of contact network: complete and random with a connectivity degree of 25%. The experiment protocol is similar as the one described in Section 5: The initial resource allocation is done randomly, and the speech turn distribution is uniform.

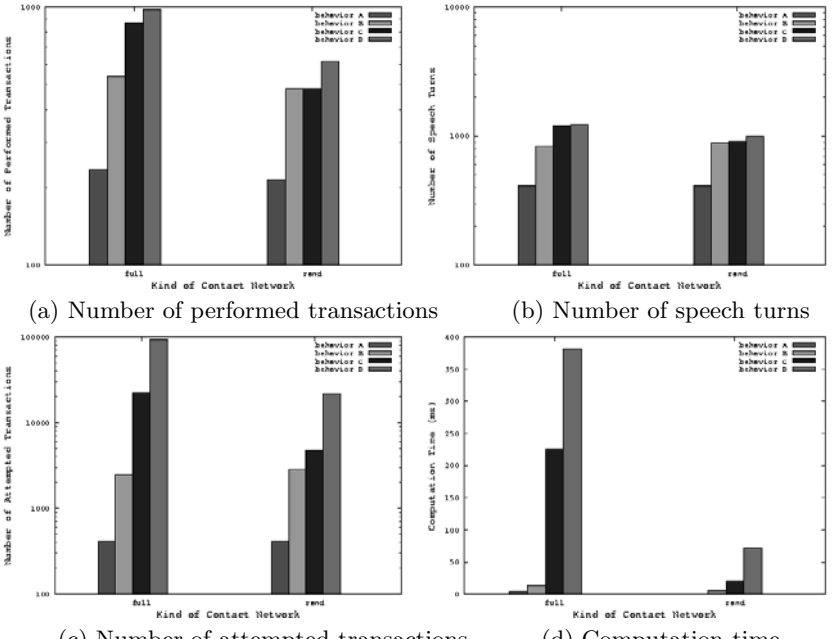
On a complete contact network, behavior D is the only one who always leads to an optimal resource allocation. The proof is described in the next section. Behaviors B and C are not able to reach an optimal resource allocation, but they still lead to socially close resource allocation with a social gap less than 10%. Behavior A obtains the worst results by leading to resource allocations which correspond to 77% of the optimal value.

On a random contact network, none of these behaviors can reach an optimal allocation. However, behavior D still obtains the best result by leading to the closest resource allocation, and with the smallest relative standard. Generally, negotiation processes based on gifts obtain small relative standard deviation, which means that the social value of the emergent resource allocation does not almost vary.

Table 5. Social Efficiency(%) and relative standard deviation of the behaviors on a complete network(left) and on random network(right)

n	m	A	B	C	D
10	60	77.92 5.40	92.25 2.43	95.55 0.82	100 0
20	120	77.77 3.78	92.50 1.84	98.66 0.23	100 0
50	300	77.89 3.14	92.27 1.34	99.19 0.08	100 0

n	m	A	B	C	D
10	60	74.13 5.13	84.56 2.90	80.87 3.77	96.94 2.18
20	120	75.07 3.83	86.51 2.01	84.54 2.24	89.33 1.28
50	300	76.13 2.85	90.10 1.35	90.43 0.92	93.63 0.48



(a) Number of performed transactions (b) Number of speech turns

(c) Number of attempted transactions (d) Computation time

Fig. 3. Comparison of behaviors based on social gift transaction

The best results that are obtained by the behavior D have a cost in terms of performed transactions (Fig.3) (a), speech turns (Fig.3) (b), attempted transactions (Fig.3) (c), and computation time (Fig.3) (d): The greater is the number of neighbor per agent, the greater will be the difference.

As shown on Fig.3 (a), the number of performed transactions is higher with the behavior D, however, the difference with behaviors B and C is not so large. Fig.3 (b) shows that the number of speech turns does not vary appreciably from one behavior to the next. This is due to the uniform distribution of that speech turn. Once every agents fail one time to find an acceptable deal, the negotiation process stops. However, rooted behaviors like behaviors A and B might perform

a new transaction by selecting another neighbor. A random distribution of the speech turn should increase a lot the corresponding metric for these behaviors. Since behaviors C and D can change the involved neighbor and therefore benefit from the neighbor list, a random distribution of the speech turn should not impact the results. Let us note, on Fig.3 (c), that behavior D is more expensive in terms of attempted transactions. Negotiation processes among agents that use behavior D take more time than others as shown on Fig.3 (d).

6.3 Proof of Convergence

We now focus on behavior D, the lone behavior for which it is possible to guarantee the end of the negotiation process on a resource allocation that corresponds to a global optimum. Recall that, in the context of the study of this paper, we only consider set of resources which are discrete, not sharable, static and unique.

Let us introduce the allocation graph $\mathcal{G}$: Each node of $\mathcal{G}$ represents a resource allocation, and a directed link $\delta(o, o')$ between two nodes o and o' exists if an acceptable transaction δ which changes o in o' exists. Assume that $\mathcal{G}$ is a connected graph (i.e., no isolated node).

An outgoing link δ of a node o corresponds to an acceptable transaction that changes the resource allocation o into another one, say o'. An incoming link δ to a node o corresponds to an acceptable transaction that changes a given initial resource allocation o' into o.

Proposition 2 (Global Optimum). *Assume $\mathcal{G}$ is connected and all possible allowed transactions have been translated throughout the arcs of $\mathcal{G}$. Assume further that the contact network is complete. Any resource allocation corresponding to a node with only incoming links is a global optimum.*

Proof. A resource allocation, which corresponds to a utilitarian global optimum is such that each resource is distributed to an agent who assigns the greatest utility to it. Indeed, if the current resource allocation is a local or global social optimum, no single acceptable transaction allows the improvement of the social welfare value, meaning no outgoing link exists. In addition, since the contact graph is complete, all optimum have the same value, hence any local optimum is global. □

Theorem 1. *The negotiation process of a multi-agent resource allocation instance based on such resources and on a complete contact network converge toward the global optimum using social gifts.*

Proof. Since the contact network is complete, an agent can always initiate a social gift with any other agent, which associates a greater utility to the involved resource, unless the resource allocation is already a global optimum. Moreover, the allocation graph $\mathcal{G}$ is connected: It is always possible from any initial node to find a sequence of social gifts leading to an optimum. Hence, the current resource allocation is a global optimum. □

7 Conclusion

In this study, we have designed an individual behavior for all agents of a complete contact network, which always leads to the emergence of a socially optimal resource allocation. Then we have shown that on a partial contact network, this behavior leads to the emergence of a socially close resource allocation. This agent-based approach can be used efficiently on any kind of contact network, and with any utility value range (positive and negative). Moreover, it is an adaptive process: the addition of new agents is possible during the negotiation process without decreasing the quality of the emergent resource allocation. It is also an "anytime algorithm": the quality of the solution increases gradually and the negotiation process can be interrupted anytime.

This negotiation process is not well-adapted to all kind of social welfare, however it improves efficiently the solution of the multi-agent resource allocation problem with an utilitarian social welfare. It provides a transaction sequence leading to an optimal allocation, in a limited amount of time, even for large instances.

References

1. Albert, R., Barabási, A.L.: Statistical Mechanics of Complex Networks. Reviews of Modern Physics 74(1), 47–97 (2002)
2. Arrow, K.J.: Social Choice and Individual Values. Yale University Press (1963)
3. Arrow, K.J., Sen, A.K., Suzumura, K.: Handbook of Social Choice and Welfare. Elsevier, Amsterdam (2002)
4. Bellosta, M.-J., Kornman, S., Vanderpooten, D.: An agent-based mechanism for autonomous multiple criteria auctions. In: IAT 2006, China, Hong-Kong, December 2006, pp. 587–594 (2006)
5. Bouveret, S., Lang, J.: Efficiency and envy-freeness in fair division of indivisible goods: logical representation and complexity. In: IJCAI 2005, UK, Scotland, Edinburgh, July 2005, pp. 935–940 (2005)
6. Chevaleyre, Y., Endriss, U., Estivie, S., Maudet, N.: Multiagent resource allocation with k-additive utility functions. In: Proc. DIMACS-LAMSADE Workshop on Computer Science and Decision Theory, Annales du LAMSADE, vol. 3, pp. 83–100 (2004)
7. Chevaleyre, Y., Endriss, U., Estivie, S., Maudet, N.: Reaching envy-free states in distributed negotiation settings. In: IJCAI 2007, India, Hyderabad, January 6-12, 2007, pp. 1239–1244. AAAI Press, Menlo Park (2007)
8. Chevaleyre, Y., Endriss, U., Lang, J., Maudet, N.: Negotiating over small bundles of resources. In: AAMAS 2005, EU, The Netherlands, Utrecht, July 25-29, 2005, pp. 296–302. ACM Press, New York (2005)
9. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Negotiating socially optimal allocations of resources. Journal of Artificial Intelligence Research 25, 315–348 (2006)
10. Estivie, S., Chevaleyre, Y., Endriss, U., Maudet, N.: How equitable is rational negotiation? In: AAMAS 2006, Japan, Hakodate, May 8-12, 2006, pp. 866–873. ACM Press, New York (2006)
11. Gairing, M., Monien, B., Tiemann, K.: Selfish Routing with Incomplete Information. Theory of Computing Systems 42(1), 91–130 (2008)

12. Miranda, P., Grabisch, M., Gil, P.: Axiomatic structure of k-additive capacities. Mathematical Social Sciences 49, 153–178 (2005)
13. Moulin, H.: Axioms of cooperative decision making. Cambridge University Press, Cambridge (1988)
14. Sandholm, T.W.: Contract types for satisficing task allocation: I theoretical results. In: AAAI Spring Symposium: Satisficing Models, USA, California, Stanford University, March 23-25 (1998)
15. Sandholm, T.W.: eMediator: A Next Generation Electronic Commerce Server. Computational Intelligence 18(4), 656–676 (2002)
16. Shneidman, J., Parkes, D.C.: Rationality and Self-Interest in Peer to Peer Networks. LNCS pp. 139–148 (2003)
17. Travers, J., Milgram, S.: An experimental study of the small world problem. Sociometry 32(4), 425–443 (1969)
18. Wellman, M.P., Doyle, J.: Modular Utility Representation for Decision-Theoretic Planning. In: AIPS 1992 - Artificial Intelligence Planning Systems, USA, Maryland, College Park, June 15-17, 1992, pp. 236–242. Morgan Kaufmann, San Francisco (1992)

Part V
Simulation

Interaction Biases in Multi-Agent Simulations: An Experimental Study

Yoann Kubera, Philippe Mathieu, and Sébastien Picault

LIFL UMR USTL-CNRS 8022
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
Cité Scientifique - 59655 Villeneuve d'Ascq Cedex – France
{yoann.kubera,philippe.mathieu,sebastien.picault}@lifl.fr

Abstract. How to ensure that two different implementations of a simulation will produce the same results ? In order to assure simulation reproducibility, some domain-independent functional unit must be precisely described. We show in this paper that the management unit that rules the participation of an agent in simultaneous interactions is one of them. Usually, many choices concerning this unit are made implicitly, even if they might lead to many simulation biases. We illustrate this issue through a study of biases that appear even in simple cases, due to a specification lack, and we propose as a solution a classification of interactions that makes those choices explicit.

1 Introduction

Multi-Agent Based Simulations (MABS) – and more generally computer simulations – are a tool used to reinforce, invalidate or compare hypothesis on the origin of a particular emergent phenomenon. The model of a simulation needs these hypothesis become concrete, and has to provide all the information required to perform experiments. Consequently, implementations of this model made by different persons have to produce results with similar nature – *i.e.* results of such a model have to be reproducible.

Building a simulation is a process leading from a domain-specific model to an operational model – through knowledge representation formalisms – and then from the operational model to its implementation in a given programming language, on a given simulation platform [1]. Sadly, there is no consensus about what information each model should contain, since the separation between domain-specific model, operational model and implementation – *i.e.* computer science-specific model – is ambiguous itself [2,3]. Thence, each step of the simulation design process involves choices – both explicit or implicit – regarding ambiguous parts of the previous step model. Those choices have a more or less dramatic influence on the execution and outcomes of the simulation. Since simulations have to be reproducible, the biases these choices may introduce must be studied, and issues about how and to what extent each choice may change simulations outcomes have to be handled.

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 229–247, 2009.

We uphold that the modeler has to be aware of – and to understand – each possible choice, and has to specify the ones he chooses for its model. Without this specification, the ambiguity of the models leads to implementations that do not behave as it was initially expected and thus produce unexploitable results.

The spectrum of implicit choices is wide, and concerns very different parts of agent's and simulation's architecture. To make their study easier, the architecture of a multi-agent simulation is considered here through three almost independent functional units that underlie any kind of simulation. These units are the Activation Unit that manages time related elements of the simulation like *when agents trigger their behavior*, or *in which interactions an agent may participate simultaneously*, the Definition Unit that specifies all the interactions the agents are able to initiate, and the Selection Unit that corresponds to interaction selection. In this paper the Activation Unit is studied.

Interactions between agents – *i.e.* actions involving simultaneously two or more agents – are the source of simulation's emergent properties. Thus, they have a major role in MABS. But, because current MABS design methodologies focus only on the behavior of independent agents, many design choices concerning interactions are not explicit. In particular, the participation of an agent in interactions occurring at the same time is almost never tackled, because of not adapted knowledge representation.

This paper aims at studying implementation choices concerning the Activation Unit, and more precisely on how simultaneous interactions are handled. In order to make the study of this issue possible, we use the knowledge representation provided by the IODA methodology [4], which is fit to model such problems. Thanks to a study of some experiments, we present the two main patterns – called *interaction classes* – used to handle simultaneous interactions in any kind of simulation. This study also illustrates the consequences of wrong implementation choices – *i.e.* the misuse of interaction classes or wrong time representations. We uphold that defining what simultaneously means in the model, and providing a class for every interaction in the model determines precisely how the Activation Unit is supposed to manage simultaneous interactions without ambiguities. Thus, it makes sure that the model is implemented without biases.

This paper is organized as following. First, related work concerning the studied functional unit, the Activation Unit, are presented in section 2. Then, the functional decomposition that underlies any multi-agent based simulation, on which this paper's studies are based, and the knowledge representation of the IODA methodology are presented in section 3. Section 4 describes the protocol followed by the experiments of the study. Section 5 to 6 describe two experiments , which results are interpreted to identify interaction classes, and to illustrate the consequences of erroneous implementation choices. Then, section 7 summarizes the results of experiments, and proposes a classification of interactions in order to avoid implementation biases of the Activation Unit. Eventually, section 8 discusses about these results, and emphasizes the need to understand what *"simultaneous interactions"* means in the Activation Unit.

2 Related Works

In many simulation platforms, design is centered on agents and the actions they initiate, rather than on the interactions that may occur between them. Consequently, the definition of which interaction an agent may initiate at a particular time does :

- neither take into account in which interaction it already participates as a target (see section 3.1);
- nor take into account in which interaction the other agents already participate in.

This leads to many biases in the simulation and its outcomes.

For instance, Epstein and Axtell [5] presented an ecosystem simulation where an agent may reproduce more than twice at the same time (once as a the initiator of the interaction, and one or more times as a target). This bias was identified by Lawson [6], and corrected with **a modification of the model** through the addition of a gestation period.

Yet, this issue is not restricted to ecosystem simulation. Indeed, it applies to every simulation where agents have to perform particular interactions at most once at a time (for instance agents that trade goods). Thus, it has to be dealt with **in the domain-independent architecture of the simulation** rather than in the models.

This problem leads Michel [3] to manage interactions depending on their *Strong* or *Weak* nature. This solution is adequate if agents are the only participants in interactions. Thus, it does not solve the problem for interactions between an agent and an object. Indeed, interactions like *Withdraw cash from an automated teller machine* may be performed simultaneously by two different agents with the same machine.

Weyns [7] proposes a more refined solution through the qualification of the relationship between two actions[1]. Two actions may be *Independent*, *Joint*, *Concurrent* or *Influencing*. This solution manages interactions by getting from each agent *"intend to perform the* I *interaction with the* A *agent as target"* like messages. A mastering unit then gathers these messages, finds out the relationship between them and executes compatible ones.

In spite of its undeniable advantage of concurrent and influencing interaction handling, this solution has a major issue. Indeed, because interactions not compatible with already occurring interactions are not considered during decision making, an agent may try to perform an impossible interaction. Thus, the agent performs nothing at that time, even if another interaction is possible.

To fill this gap, simultaneous interactions have to be considered at decision making. This requires an interaction-oriented design of decision making, like the one shortly presented in the next section.

[1] Although the author uses the term "action", it keeps the same meaning than our "interaction" (see section 3.1).

3 Multi-Agent Simulations

Even if the application domains of multi-agent simulations are heterogeneous, they can be split into different and weakly dependent functional units [8,9], like agents scheduling, communications, modification conflicts solving, *etc.*

We consider here a particular decomposition of a simulation (see Fig. 2) in three main units, called Activation Unit, Definition Unit and Selection Unit, which respectively drive time related elements in the simulation, declaration of what agents are able to perform, and finally how interaction selection is made. This decomposition underlies any kind of simulation.

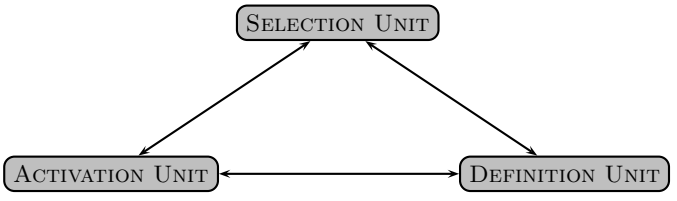


Fig. 1. The three main functional units of a multi-agent simulation

This kind of separation in different software units is usual in cognitive agent architectures with plans like the *Act-R* [10] or *Soar* [11] language, where knowledge representation is at the center of the simulation, but does not exist in reactive simulation platforms. Moreover the notion of interaction – *i.e.* semantic block of actions involving simultaneously a fixed number of agents (see Sect. 3.1) – is generally hard-coded in the behavior of agents. Because the design of simulations implies crucial choices about those three units, we claim that it is important to make this separation clear, even in reactive simulations, in order to make modeling choices explicit.

3.1 An Interaction-Oriented Design of Simulations

The definition of interactions, and how they are integrated in the knowledge of agents, are based on *IODA* concepts [4]. Please note that *IODA* provides advanced methodological and software engineering tools to design interactions in MABS. Since we do not need all refinements it provides, we use a simplified version of its definitions.

To make the difference between the abstract concept of agent (for instance Wolves), and agent instances (a particular Wolf), we use the notion of ***agent families*** as abstract concept of agent. Thus, the word ***agent*** refers to an agent instance.

Definition 1. *An* **agent family** *is an abstract set of agent instances, which share all or part of their properties and behavior.*

Definition 2. *An* **interaction** *is a structured set of actions involving simultaneously a fixed number of agents instances that can occur only if some conditions are met.*

An interaction is represented as a couple (*conditions*, *actions*), where *condition* is a boolean function and *action* is a procedure. Both have agent instances as parameters. Agents that are involved in an interaction play different roles. We make a difference between ***Source*** agents that may initiate the interaction (in general the one selected by the ACTIVATION UNIT) and ***Target*** agents that may undergo it.

Definition 3. *Let $\mathbb{F}$ be the set of all agent families. Let $\mathcal{S} \in \mathbb{F}$ and $\mathcal{T} \in \mathbb{F}$ be agent families.*

We note $a_{\mathcal{S}/\mathcal{T}}$ the **set of all interactions** *that an instance of the $\mathcal{S}$ agent family is able to initiate with an instance of the $\mathcal{T}$ agent family as a target.*

Thanks to these definitions, we can specify the knowledge of an agent family $\mathcal{S} \in \mathbb{F}$ as the set $\bigcup_{\mathcal{T} \in \mathbb{F}} a_{\mathcal{S}/\mathcal{T}}$, which contains every interactions it is able to initiate as source with any agent family as target.

To unify knowledge, actions (for instance WANDER or DIE) are considered as interactions, which target is implicit (either the environment, or the agent itself). This kind of interactions is called **degenerate interaction**. We do not add this to our notations, please see [4] for more information.

The definition of perceived affordances uses the notion of realizable interaction, in order to determine if two agents can participate in an interaction.

Definition 4. *Let I be an interaction, and $x \prec \mathcal{S}$, $y \prec \mathcal{T}$ two agents. The tuple (I, x, y) is* **realizable** *(written $r(I, x, y)$) if and only if :*

- *$I \in a_{\mathcal{S}/\mathcal{T}}$, i.e. agents of $\mathcal{S}$ family are able to perform I with agents of $\mathcal{T}$ family;*
- *the conditions of I hold true with x as source and y as target.*

A realizable tuple represents one interaction that an agent can initiate with a particular target agent. Moreover, the agent's perceived affordances are the set of all interactions it can initiate in a given context. Thus, at a time t, the perceived affordances of the x agent are the set of all realizable tuples that x may perform.

Definition 5. *Let $\mathbb{A}_t$ be the set of all agents in the simulation at a time t, and $x \in \mathbb{A}_t$.*
Then, the **perceived affordances** *$\mathbb{R}_t(x)$ that x may perform at time t is the set :*

$$\mathbb{R}_t(x) = \bigcup_{y \in \mathbb{A}_t} \bigcup_{I \in a_{x/y}} \{(I, x, y) | r(I, x, y)\}$$

3.2 Discussion about This Knowledge Representation

In some cases, the distinction between source and target agents might appear as a restriction. This implies that the knowledge representation of IODA cannot be used to model any kind of simulation.

For instance the HANDSHAKING interaction does not seem to make the difference between the source and the target agent – *i.e.* source and target roles are symmetric.

The IODA methodology upholds that, even if the roles in the interaction are symmetric, the two agents do not spontaneously choose to perform the HANDSHAKING together. One agent is at the origin of the HANDSHAKE, and requests the other agent if it wants to perform that interaction.

Consequently, interactions have always an initiator – *i.e.* a source. Thus, the knowledge representation provided by IODA can be used to model any kind of simulation.

3.3 Time Representation and Simultaneous Interactions

The model of a simulation has to define how time is represented in the simulation. Indeed, this representation has a deep influence on how the ACTIVATION UNIT is supposed to work. Moreover, the representation of time defines directly the meaning of *simultaneous interactions*.

Mainly, two different time representation are used[2] in MABS. Both are based on the discrete event paradigm, where time is considered as a number that increases during simulation, given a particular evolution process. Time can :

- either evolve by steps of fixed length. Usually, these kind of simulation are called *Discrete*, because time can be considered as a finite set of integers called *time steps*. At each time step, the ACTIVATION UNIT will ask one or more agents to perform their SELECTION UNIT. The choice of asked agents is up to the policy used in the ACTIVATION UNIT. For instance, ask all agents sequentially in random order, or ask one agent chosen randomly, *etc*;
- or evolve event by event. Usually, these kind of simulation are called *Continuous*, or at least *Pseudo-Continuous*, because the time elapsed between two event is not fixed. At each event, an agent performs its SELECTION UNIT, and schedules an event for its next activation, depending on the interaction it initiated.

For each time representation, interactions have a duration – *i.e.* a time interval during which the interaction is considered as being performed. The side-effects of the interaction are considered consistent only at the end of this interval.

In the case of continuous time, the duration has to be defined by the modeler, because the scheduling of events depends on the date agents finish their current interaction. Thus, duration is explicit in the model.

For discrete time, this notion of duration is not as obvious. Implicitly, the duration of actions and interactions is the length of a time step. Sadly, the lack of specification concerning this point causes critical issues in simulation (see section 8).

Thanks to this definition, simultaneous interactions can be defined :

[2] these two are used for illustration purposes, in order to elicit the notion of interaction duration. Other time representation might exist.

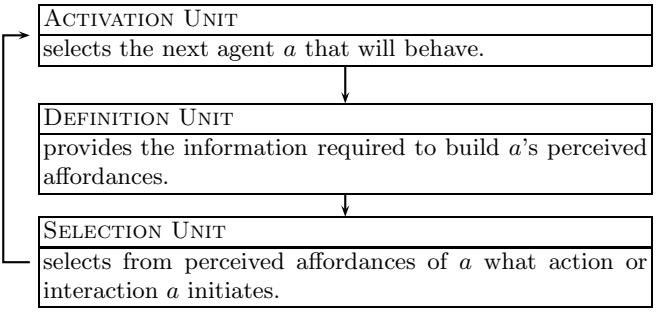


Fig. 2. How the three main functional units of a multi-agent simulation are used to run a simulation

Definition 6. *Two interactions are considered as* **simultaneous** *iff their duration time interval are intersecting.*

We uphold that the meaning of simultaneous interactions has to be explicitly defined in the model, in order to assure simulation reproducibility.

3.4 Functional Decomposition of a Multi-Agent Simulation

Each functional unit is in charge of a specific feature of a multi-agent simulation.

The Activation Unit tells when agents may act, the time elapsed between their actions/interactions, what to do if an agent tries to interact with an already acting agent, *etc.* It describes all time-related elements in the simulation.

The Definition Unit lists all interactions in the simulation, under what conditions and between what kind of agents they are possible, and what actions they launch. It is the set of all possible behaviors, defined independently from agents specificities. This unit provides information required to build the space of all possible interactions the selected agent may initiate as a source – *i.e.* all realizable tuples *(Interaction, Selected Agent, Target agents)*, also corresponding to the perceived affordances of the source, as defined in [12].

The Selection Unit describes the cognitive or reactive process an agent uses to select which interaction it initiates, and, if many are possible, decides among them the one to initiate.

A simulation is a repetition of 3-steps sequences, where each step exploits a different functional unit (see Fig. 2).

We already argued in [4] for the advantage of agent-independent defined interactions and proposed a formal definition for it, thus creating a software separation between the Definition Unit – which is domain dependent – and both Selection Unit and Activation Unit.

4 Experimental Frame

The goal of this paper is to measure to what extent modifications of the Activation Unit may change simulation outcomes, and how an adequate one may avoid simulation biases. This point is illustrated through two experiments, each confronting two different implementations of the Activation Unit. Thus, the only variating parameter in an experiment is the Activation Unit : its Definition Unit and Selection Unit remain the same.

This section presents the Definition Unit, Selection Unit, the protocol used in our experiments, and preliminary discussions concerning the Activation Unit.

4.1 The Definition Unit

The Definition Unit, which defines all domain-dependent information, will change from one experiment to the other. In order to make the comprehension of our examples easier, every experiment deals with the same overall simulation problem : the evolution of an ecosystem containing predators and preys. Please note that experiments provide only an illustration of the general issue we deal with. The solutions presented in this paper are obviously not restricted to that particular simulation, and do not avoid only the biases emphasized in this paper.

4.2 The Selection Unit

The Selection Unit, which corresponds to agent's decision making process, will keep the same architecture in all our experiments.

The architecture we use is the most used one for reactive agents : a subsumption-like architecture [13] that tries every interaction sequentially until a realizable one is found. Every source agent gives to every interaction it can initiate a priority value, which denotes the order it tries the interactions (see Fig. 1.).

4.3 The Activation Unit

The model of a simulation has to define how time is represented in the simulation.

In the experiments of this paper, we consider that :

- time is *Discrete* (see Sect. 3.3);
- during every time step, the agents trigger their Selection Unit in sequence. The order of this sequence is defined at random for each time step, in order to keep equity between agents.
- the duration of interactions is the length of a time step. Thus, two interactions occurring at the same time step are considered simultaneous.

These choices are the most usual ones in classical reactive simulations. Moreover, we make the assumption that an agent may initiate at most one interaction at a time – *i.e.* it cannot be simultaneously the source of two or more interactions. This issue will be discussed in section 8.

Algorithm 1. SELECTION UNIT used in the experiments of this paper. It defines how an agent a chooses what interaction it initiates at a time z.

$\mathbb{R}_z(a) \Leftarrow$ the set of all realizable interactions a may initiate
$\mathbb{P} \Leftarrow$ the decreasing set of all priorities a gives to the interactions it can initiate
$\mathbb{L} \Leftarrow \emptyset$
for p **in** $\mathbb{P}$ **do**
  for (I, a, t) **in** $\mathbb{R}_z(a)$ **do**
    if I has p priority for a **then**
      $\mathbb{L} \Leftarrow \mathbb{L} \cup \{(I, a, t)\}$
    end if
  end for
  if $\mathbb{L} \neq \emptyset$ **then**
    a initiates the interaction of a tuple of $\mathbb{L}$ chosen at random
    stop
  end if
end for
a initiates nothing

4.4 Experimental Protocol

The experimental protocol used in this paper is :

1. first, the aim of the experiment is outlined;
2. then, the DEFINITION UNIT and SELECTION UNIT used by both implementations of this experiment are defined;
3. next, the two ACTIVATION UNIT used in the experiment, and experiment's initial conditions, are described;
4. eventually, the results of the execution of both implementation of the experiment are presented and discussed. From this discussion, an interaction class is emphasized to avoid a possible simulation bias.

Please note that all experiments presented below are voluntary basic to stress out where the problems lie : they obviously do exist in more complex situations as well.

5 First Experiment: Multiple Participation to Interactions Bias

This experiment studies the limits of the usual naive algorithm and introduces as a solution a first interaction class called *exclusive* interaction.

Model used. This simulation studies an ecosystem composed by grass and sheep. Because sheep can move, classical analytical models cannot be used to model the population of species : this simulation requires multi-agent systems.

The environment is a two dimensions toroidal continuous space split into unitary square parcels. Every parcel $\mathcal{P}$ has an attribute $q(\mathcal{P})$ that increases of one unit at every simulation step. $\mathcal{P}$ is said containing grass when $q(\mathcal{P}) > 0$. If $\mathcal{P}$ is emptied by an agent, then $q(\mathcal{P}) = 1 - r_{grass}$ (*i.e.* r_{grass} is the time grass needs

to grow) . A sheep $\mathcal{S}$ is an agent with an energy attribute $e(\mathcal{S})$ representing its health, which can initiate the interactions :

1. To **Die** if $e(\mathcal{S}) \leq 0$. Then :
   - $\mathcal{S}$ is removed from the environment.
2. To **Reproduce** with another sheep $\mathcal{S}'$ at a maximal distance of 1 from $\mathcal{S}$ if $e(\mathcal{S}) > 0$ and $e(\mathcal{S}') > 0$. Then :
   - A new sheep $\mathcal{S}''$ is created at $\mathcal{S}$'s location, and $e(\mathcal{S}'') = Min(e(\mathcal{S}), e_{repr}) + Min(e(\mathcal{S}'), e_{repr})$, where e_{repr} stands for the energy consumed by reproduction.
   - $e(\mathcal{S})$ and $e(\mathcal{S}')$ are decreased by e_{repr}.
   - $\mathcal{S}$, $\mathcal{S}'$ and $\mathcal{S}''$ execute the WANDER interaction (see below).
3. To **Eat** grass on $\mathcal{S}$'s parcel if it contains some. Then :
   - $e(\mathcal{S})$ increases from e_{eat}, where e_{eat} is the energy gained by eating.
   - $\mathcal{S}$ empties the parcel he is onto.
4. To **Wander** with no conditions. Then :
   - $\mathcal{S}$ turns itself from an angle in $[-\pi, \pi[$ and moves forward from 1 unit.
   - $e(\mathcal{S})$ decreases from e_{wan}, where e_{wan} is the energy consumed by moving.

Sheep behave by using the order $1 > 2 > 3 > 4$ in their SELECTION UNIT, thus they first have to DIE, if they don't, they try to REPRODUCE, if they don't, they try to EAT, ...

Experimental design. We used this model in a 33×33 environment containing 1089 parcels, where 30% have $q(\mathcal{P}) = 0$ and 70% $q(\mathcal{P}) \in]-r_{grass}, -1]$, and 70 sheep such that $e(\mathcal{S}) = 2 \times e_{repr}$. We also set $r_{grass} = 10$, $e_{repr} = 15$, $e_{wan} = 2$ and $e_{eat} = 7$.

Algorithm 2. *"Naive* ACTIVATION UNIT*"*. In this implementation, an agent does not take into account simultaneous interactions. MAX is the duration of the simulation, in time steps.

```
for i = 1 to MAX do
  Update the environment
  for a in agents in the environment at time z do
    ℝ_z(a) ⇐ all realizable tuples that a may initiate
    (I, a, t) ⇐ a tuple of ℝ_z(a) selected with a particular SELECTION UNIT
    if (I, a, t) ≠ null then
      Execute I with a as source and t as target.
    end if
  end for
end for
```

Figure 3 compares the evolution of the sheep population of this model implemented with respectively the *naive* (*single interaction*) ACTIVATION UNIT, from Algorithm. 2. (from Algorithm. 3.).

Algorithm 3. *"Single Interaction* ACTIVATION UNIT*"* In this implementation, an agent participates only in one interaction at a time, either as the source or as the target. MAX is the duration of the simulation, in time steps.

for $i = 1$ **to** MAX **do**
  Update the environment
  for a in agents **in** the environment at time z **do**
    Tag a as *operative*
  end for
  for a in agents **in** the environment at time z **do**
    $\mathbb{R}_z(a) \Leftarrow$ all realizable tuples that a may initiate
    for (I, a, t) **in** $\mathbb{R}_z(a)$ **do**
      if a is not operative **or** t is not operative **then**
        Remove (I, a, t) from $\mathbb{R}_z(a)$
      end if
    end for
    $(I, a, t) \Leftarrow$ a tuple of $\mathbb{R}_z(a)$ selected with a particular SELECTION UNIT
    if $(I, a, t) \neq$ null **then**
      Execute I with a as source and t as target.
      Tag a and t as *not operative*
    end if
  end for
end for

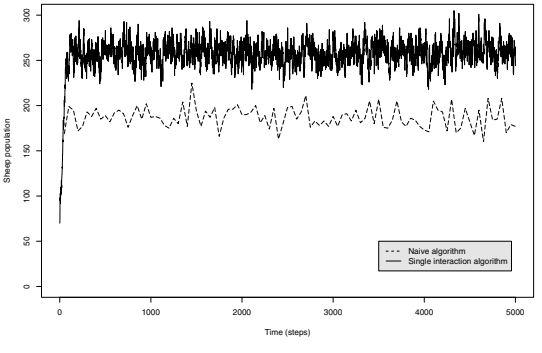


Fig. 3. Sheep population evolution against time (in simulation steps) respectively displayed in dots (in line), for the model presented in Sect. 5 implemented with the *naive* (*single interaction*) algorithm of Algorithm. 2. (Algorithm. 3.)

Results and Discussion. With exactly the same initial environment, the experiment using the *naive* algorithm (see Algorithm 2.) produces in overall 68 more sheep than the one using the *single interaction* algorithm (see Algorithm 3.).

This difference lies in the number of interactions an agent may participate in during a simulation step. In the *naive* algorithm, a sheep targeted by a REPRODUCE interaction can be the source of another interaction. On the opposite, in the *single interaction* algorithm an agent participates at maximum in one interaction, either as a source or as a target. This difference has a great impact on

the sheep energy dynamics, their reproduction and death rate, and their density, and, as a consequence, on the sheep population dynamics.

In this simple experience, this difference may be considered as the result of different interpretations of the model, thus it cannot be considered as a bias. Nevertheless, it corresponds to a bias in other experiences, as shown in [3] : if a sheep participates in more than one reproduction per time step, then the sheep reproduction probability is different from the one designed in the model.

We outlined with this experience that the number of interactions an agent can simultaneously participate in has to be restricted. To cope with this problem, we identified a first interaction class called *exclusive* interactions. An agent can participate in such an interaction only once at a time, either as source or as target.

6 Second Experiment: Single Participation to Interactions as Target Bias

This section illustrates a simulation problem concerning the target of an interaction that *exclusive* interactions alone are too restrictive to solve.

Model used. This model adds to the one of experiment 5 a new agent named wolf, and a new interaction to a sheep $\mathcal{S}$:

5. To **Flee** from a wolf $\mathcal{W}$ at a maximal distance of 10 from $\mathcal{S}$. Then :
   - $\mathcal{S}$ turns its back towards where $\mathcal{W}$ is and moves forward from 1 unit.
   - $e(\mathcal{S})$ decreases from e_{wan}.

Sheep behave using the order $1 > 5 > 2 > 3 > 4$ in their Selection Unit, and wolves only Wander in the environment.

In this simulation, sheep Flee systematically wolves. As a consequence, wolves are supposed to be at the center of an empty area.

Experimental design. They are the same as in the experiment of Sect. 5, except that there is a wolf at a random position, and that the simulation is implemented with :

- firstly with the *single interaction* Activation Unit (from Algorithm. 3.);
- then with the *parallel interactions* Activation Unit (from Algorithm. 4.) where
  - Flee is from $\mathcal{I}_2$ interaction class;
  - other interactions are from $\mathcal{I}_1$ class.

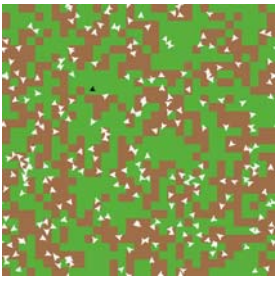
The outcomes of such implementations of this model are displayed in Fig. 4.

Results and Discussion. The *parallel interactions* Activation Unit produces the expected result (right on Fig. 4), and the *single interaction* Activation Unit (left on Fig. 4) is obviously biased : there is no empty halo around the wolf.

The difference lies in the number of interactions a wolf can undergo simultaneously. When a wolf is the target of a Flee interaction, it is set not operative,

Algorithm 4. *"Parallel Interaction* Activation Unit*"*. In this implementation, an agent participates only in one interaction of $\mathcal{I}_1$ at a time, either as the source or as the target. An agent can be the target of as many interaction of $\mathcal{I}_2$ as necessary. MAX is the duration of the simulation, in time steps.

for $i = 1$ **to** MAX **do**
  Update the environment
  for a in agents **in** the environment at time z **do**
    Tag a as *operative*
  end for
  for a in agents **in** the environment at time z **do**
    $\mathbb{R}_z(a) \Leftarrow$ all realizable tuples that a may initiate
    for (I, a, t) **in** $\mathbb{R}_z(a)$ **do**
      if I is from $\mathcal{I}_1$ class **and** (a is not operative **or** t is not operative) **then**
        Remove (I, a, t) from $\mathbb{R}_z(a)$
      else if I is from $\mathcal{I}_2$ class **and** a is not operative **then**
        Remove (I, a, t) from $\mathbb{R}_z(a)$
      end if
    end for
    $(I, a, t) \Leftarrow$ a tuple of $\mathbb{R}_z(a)$ selected with a particular Selection Unit
    if $(I, a, t) \neq$ null **then**
      Execute I with a as source and t as target.
      if I is from $\mathcal{I}_1$ class **then**
        Tag a and t as *not operative*
      else if I is from $\mathcal{I}_2$ class **then**
        Tag a as *not operative*
      end if
    end if
  end for
end for



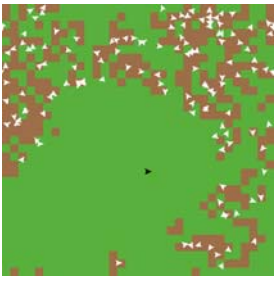


Fig. 4. Outcomes screenshot of the experiment presented in Sect. 6. This model was respectively implemented with the *single interaction* (*parallel interactions*) Activation Unit from Algorithm. 3. (from Algorithm. 4.), displayed to the left (right), where respectively light (dark) arrows are sheep (wolves), and light (dark) squares are empty (full) parcels.

thus it cannot be the target of another Flee interaction. Consequently, a wolf is fled once per simulation step, and other sheep behave as if there was no wolf.

We outlined with this experiment that all interactions do not put the same restrictions onto their target agent. Interaction classes have to reflect this difference, thus, in addition to *exclusive* interactions, we introduce *parallel* interactions, where agents may simultaneously be targeted as many times as needed.

7 Experiments Summary: A Classification of Interactions

Through two experiments, we have shown that the model has to answer the question *"is the source (or target) agent of an interaction allowed to be simultaneously the source (or target) of another interaction ?"*. Otherwise the lack of specifications leads to ambiguities from which many biases may result.

7.1 Interaction Classes

As a solution, we identified two interaction classes, each answering differently to the question above. Considering our modeling experience, these classes correspond to the two main recurrent patterns used to handle simultaneous interactions. The association of a class to each interaction in a model describes explicitly how they are managed, and thus avoids many biases at implementation.

The two interaction classes are :

exclusive interaction. An agent is allowed to participate only to one exclusive interaction at a time, whether as source or as target. In the experiments and in the Algorithm. 3, and Algorithm. 4, it corresponds to $\mathcal{I}_1$ interaction class. It is the case of the REPRODUCE interaction.

parallel interaction. An agent is allowed to be simultaneously the target of as many parallel interaction as needed. In the experiments and in the Algorithm. 4, it corresponds to $\mathcal{I}_2$ interaction class. It is the case of the FLEE interaction.

Moreover, we consider that an agent can initiate only one interaction at a time. Thus an agent cannot simultaneously be the source of a parallel interaction, and participate to an exclusive interaction (either as source or target).

Table. 1 provides a summary of what interaction classes allow and forbid.

Table 1. Summary of what interactions an agent may participate in simultaneously, depending on its role in them. The cross at the intersection of the line (Exclusive, S) and column (Parallel, T) is read *"An agent can simultaneously be the source (S) of an exclusive interaction and the target (T) of a parallel interaction"*. The empty cell at the intersection of the line (Exclusive, T) and column (Exclusive, T) is read *"An agent cannot simultaneously be the target (T) of an exclusive interaction and the target (T) of another exclusive interaction"*.

		Exclusive		Parallel	
		S	T	S	T
Exclusive	S				×
	T				×
Parallel	S				×
	T	×	×	×	×

7.2 Use of Interaction Classes

In practice, interaction classes are exploited before the SELECTION UNIT executes. The agent that performs the SELECTION UNIT cannot initiate all the affordances – *i.e.* all realizable tuples – it listed. Indeed, this agent might already be involved in some interactions. Thus, it has to remove from its affordances all interactions that cannot be initiated simultaneously with the interactions already occurring. This removal is based on the table 1 that summarizes what interactions are allowed simultaneously. For instance, an agent cannot initiate an exclusive interaction with an agent that is already the target of an exclusive interaction (see the intersection of the line $(Exclusive, S)$ and the column $(Exclusive, T)$ in Fig. 1).

Thus, the 3-steps sequence of Fig. 2 in section 3.4 becomes as displayed in Fig. 5. The algorithm 4. provides an implementation of such an ACTIVATION UNIT, where :

- time is *Discrete* (see Sect. 3.3);
- during every time step, the agents trigger their SELECTION UNIT in sequence. The order of this sequence is defined at random for each time step, in order to keep equity between agents.
- the duration of interactions is the length of a time step. Thus, two interactions occurring at the same time step are considered simultaneous.
- an agent may initiate at most one interaction at a time, *i.e.* it cannot be simultaneously the source of two or more interactions.

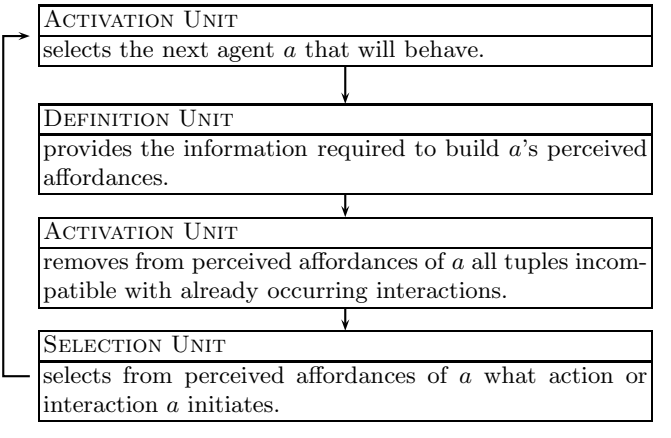


Fig. 5. How the three main functional units of a multi-agent simulation are used to run a simulation, This takes into account simultaneous interactions

7.3 Why Defining Interaction Classes in the Model ?

Interaction classes answer the question *"is the source (or target) agent of an interaction allowed to be simultaneously the source (or target) of another interaction ?"*.

These interaction classes are deeply bound with the algorithms used to process interactions at implementation. Thus, knowledge on which interaction classes are present in an operational model :

- removes ambiguities found during implementation.
- determines if a simulation platform is fit to implement the model. For instance, a simulation platform like *Netlogo*[14] is no fit by default to implement models containing exclusive interactions : the user has to develop his own Activation Unit;

The implementation of such specifications is made easier by a software separation between Activation Unit, Definition Unit and Selection Unit. Indeed, it forces the user to choose interaction classes explicitly, and thus forces to understand the underlying algorithms. It is the case of the *IODA* methodology and the *JEDI* simulation platform [4] where this separation is made by the reification of interactions through the whole simulation process.

Note that even if these classes are defined in the context of discrete simulations – *i.e.* with simulation steps of fixed length – they remain valid for other simulations.

8 Discussion about Our Solution

Our solution makes the assumption that an agent can simultaneously be the source of at most one interaction. This seems to be an hindrance for some simulations. We illustrate this point on an example.

8.1 Issues about Simultaneous Initiation of Many Interactions

Many reactive simulations, such as the *Wolf Sheep Predation* of *Netlogo* (see Algorithm. 5.1) seem to make the assumption that the Selection Unit can select and initiate more than one interaction during a time step.

For instance, in the *Wolf Sheep Predation* :

- the Activation Unit is defined by the *"go"* block. This block ends with a *"tick"* command, which tells that the Activation Unit uses discrete time. The code *"ask sheep"* tells that for each simulation time step, sheep agents are asked in a random sequence to perform once the content of the *"ask sheep"* block.
- the Definition Unit and Selection Unit are mixed, and defined in the content of the *"ask sheep"* block. In this block sheep may initiate a move interaction, then a eat-grass interaction, then a reproduce-sheep interaction and finally a death interaction.

Sheep seem to be able to initiate up to four interactions during a time step.

Algorithm 5. Part of the implementation in *Netlogo* of a Wolf Sheep predation simulation

```
to go
  (...)
  ask sheep [
    move
    if grass? [
      set energy energy - 1
      eat-grass
    ]
    reproduce-sheep
    death
  ]
  (...)
  tick
  (...)
end

to move  ;; turtle procedure
  rt random 50
  lt random 50
  fd 1
end
```

```
to eat-grass
  if pcolor = green [
    set pcolor brown
    set energy energy + sheep-gain-from-food
  ]
end

to reproduce-sheep
  if random-float 100 < sheep-reproduce [
    set energy (energy / 2)
    hatch 1 [ rt random-float 360 fd 1 ]
  ]
end

to death
  if energy < 0 [ die ]
end
```

8.2 Discussion about This Simulation

Time steps are the most easiest way to build an Activation Unit. In simulation using this kind of Activation Unit, the most atomic representation of time is the time step. Thus, interactions are considered as simultaneous if they occur during the same time step.

In the simulation presented above, this means that an agent is able to initiate up to four interactions simultaneously. This seems to invalidate our hypothesis that an agent is able to initiate at most one interaction at a time, and consequently seems to invalidate our solution.

In fact, there is no such thing like initiating simultaneously more than one interaction. The issue is rather related to the wrong use of discrete time, and its underlying definition of simultaneous interactions. Indeed, it makes no sense to uphold that a sheep can initiate Die and Reproduce-Sheep simultaneously. The same holds for any other combination containing two interactions among Die, Reproduce-Sheep, Eat-Grass and Move. These interactions are meant to be executed separately, in sequence. Thus, they are not simultaneous, and have to occur during different time steps. In this case, the implementation does not reflect what the model means.

We uphold that simulations that let agents initiate more than one interaction during the same time step are misusing discrete time Activation Unit, and do not implement the model as it was meant. Thus, initiating at most one interaction at a time is sufficient to model all kinds of simulations.

9 Conclusion

Most simulations assume and compare hypothesis on a given phenomenon. The model is the mirror of such hypothesis. Thus, it has to contain enough information to assure simulations reproducibility – *i.e.* two implementations of the same model have to produce outcomes with similar natures. Sadly, many modeling and implementation choices are left implicit. This can lead to biases, and thus to non-reproducible simulations. Thus the biases that may result from modeling and implementation choices must be identified and quantified.

In this paper we have shown that the reproducibility of a simulation is not possible without specifying a particular domain-independent functional unit that underlies any simulation, called Activation Unit.

This unit specifies all time related elements in the simulation. In particular, it indicates to what interactions an agent can participate simultaneously. Experiments showed that the lack of specifications concerning the particularities of these interactions may introduce biases in simulation outcomes. Indeed, as an example, the target of a reproduction behavior cannot initiate simultaneously another interaction, otherwise an agent may reproduce twice at the same time.

To solve this kind of problem, we uphold that the model must specify :

- what *simultaneous interaction* means. For discrete simulations, where time is divided in time steps of the same length, interactions are simultaneous if they occur during the same time step;
- the interaction class of each interaction of the simulation, among *exclusive* and *parallel*. Thanks to these classes, an agent can determine which interactions it can initiate – *i.e.* be the source – at a particular time, according to all the interactions already occurring at that time.

We illustrated on an experiment that the lack of knowledge on what simultaneous interactions means may lead to implementations that do not correspond what the model meant. Thus, the specification of what interactions agents may participate in simultaneously have meaning only if the notion of *simultaneous interactions* is known and understood.

Taking into consideration time representation and these classes while conceiving the model removes ambiguities that would have led to biases. Without the specification of these two points, two different developers will likely obtain very different outcomes for the same model.

References

1. Fishwick, P.A.: Computer simulation: growth through extension. Trans. Soc. Comput. Simul. Int. 14(1), 3–20 (1997)
2. David, N., Sichman, J.S., Coelho, H.: Towards an Emergence-Driven Software Process for Agent-Based Simulation. In: Sichman, J.S., Bousquet, F., Davidsson, P. (eds.) MABS 2002. LNCS, vol. 2581, pp. 89–104. Springer, Heidelberg (2003)
3. Michel, F., Gouach, A., Ferber, J.: Weak interaction and strong interaction in agent based simulations. In: MABS 2003. LNCS, vol. 2927, pp. 43–56. Springer, Heidelberg (2003)

4. Kubera, Y., Mathieu, P., Picault, S.: Interaction-oriented agent simulations: From theory to implementation. In: Proceedings of ECAI 2008, Patras Greece, July 2008, pp. 383–387 (2008)
5. Epstein, J., Axtell, R.: Growing Artificial Societies. Brookings Institution Press, Washington, DC (1996)
6. Lawson, B., Park, S.: Asynchronous time evolution in an artificial society mode. Journal of Artificial Societies and Social Simulation (2000)
7. Weyns, D., Holvoet, T.: Model for simultaneous actions in situated multi-agent systems. In: Schillo, M., Klusch, M., Müller, J., Tianfield, H. (eds.) MATES 2003. LNCS, vol. 2831, pp. 105–118. Springer, Heidelberg (2003)
8. Demazeau, Y.: From interactions to collective behaviour in agent-based systems. In: Proceedings of ECCS 1995, Saint-Malo, France, pp. 117–132 (1995)
9. Weyns, D., Parunak, H., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems: State-of-the-art and research challenges. In: Environments for Multiagent Systems, New York, NY, USA, pp. 1–47 (2004)
10. Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., Qin, Y.: An integrated theory of the mind. Psychological Review 111(4), 1036–1060 (2004)
11. Newell, A.: Unified theories of cognition. Harvard University Press, Cambridge (1994)
12. Norman, D.A.: The Psychology of Everyday Things. Basic Books (1988)
13. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE Journal of robotics and automation 2(1), 14–23 (1986)
14. Wilenski, U.: Netlogo. Center for connected learning and computer-based modeling (1999), http://ccl.northwestern.edu/netlogo/

Engineering Self-modeling Systems: Application to Biology

Carole Bernon[1], Davy Capera[2], and Jean-Pierre Mano[1]

[1] Institut de Recherche en Informatique de Toulouse, University of Toulouse III, 118 route de Narbonne, 31062 Toulouse cedex 9, France
[2] UPETEC, 10 avenue de l'Europe, 31520 Ramonville, France
{bernon,mano}@irit.fr, davy.capera@upetec.fr

Abstract. Complexity of today's systems prevents designers from knowing everything about them and makes engineering them a difficult task for which classical engineering approaches are no longer valid. Such a challenge is especially encountered in actual complex systems simulation in which underlying computational model is very tough to design. A prospective solution is to unburden designers as much as possible by letting this computational model self-build. Adaptive multi-agent systems are the foundation of the four-layer agent model proposed here for endowing systems with the ability to self-tune, self-organize and self-assemble. This agent model has been applied to an application (MicroMega) related to computational biology which aim is to model the functional behavior of unicellular yeast Saccharomyces Cerevisiae.

Keywords: complex system, self-organization, cooperation, biological modeling.

1 Introduction

Nowadays systems are becoming more and more complex due to, on the one hand, the huge number of heterogeneous, autonomous and evolving components and, on the other hand, their required features of openness and scalability. A few years ago, referring to information technology systems, IBM underlined that "Even if we could somehow come up with enough skilled people, the complexity is growing beyond human ability to manage it…" and brought out the need of new approaches for dealing with complexity. Namely building "autonomic systems" capable of "running themselves and adjusting to various circumstances…" [21]. This vision may be enforced to every complex system surrounding us, especially biological ones on which this paper focuses, since natural complexity prevents designers from knowing everything about such systems, let alone controlling them.

For example, in computational biology, complexity comes from the huge amount of constantly increasing heterogeneous data that have to be gathered, visualized, exploited or processed. In systems biology, complexity arises from the need of modeling large-scale biological interaction networks for which interactions are not always known; moreover experimental data are not always available or homogeneous [23]. Furthermore integrative biology adds a level of complexity by aiming at reconstructing "the whole by putting the parts together (once enough parts have been collected

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 248–263, 2009.

and understood)" [29]. In other words, the aim is to assemble several different yet coupled models in order to obtain an upper level one that explains a higher level of functioning. For instance, at the level of a unicellular organism, integrative biology is expected to determine how all genes and their products interact to produce the functioning organism [1].

Due to the lack of satisfactory theories for explaining biological systems, biologists usually rely on modeling and simulations to understand their behavior and different approaches exist to build these models, from mathematical ones to neural networks-based ones. The latter ones offer some interesting results in finding correlations and extracting some explaining variables but like black boxes they prevent any knowledge on the real structure and functioning of the system [14]. Moreover they hardly take into account contingency of phenomena that are a bit delayed.

Many approaches of computational modeling are concerned with biological networks [7]: Boolean networks emphase causal and temporal relationships between activation of different molecules; Petri nets extend Boolean networks with stochastic and non deterministic properties, but both approaches are hard to compose in larger models [24]. Interacting State Machines focus on state and produce models of transition of states. Process Calculi focus on events and enable modeling causality relationships between events. Both offer composition and may work in parallel or be given with hierarchical structures (with microlevels of functioning combined in a more abstract view of the system) [11][14].

To combine any of those approaches in a single (rather not simple) model some hybridization is possible through discrete event techniques. But all those approaches are not able to help scientists to construct models especially when formalism adds constraints in discrepancy with biological reality.

Ideally, biologists would like to understand underlying mechanisms of biological systems without requiring very costly *in vivo* experimentations, or at least would like to have means for focusing on really interesting ones. Most of the time the models they are provided with are static ones; influencing or modifying them in a dynamic way for trying to understand or discover new virtual experiments is usually impossible. Models have then to constantly reflect experimental data and user's desiderata in order to be useful and thus need adaptation capabilities to stay functional in evolving environments.

This feature adds some more complexity when designing this kind of systems and classical top-down approaches are no longer helpful. New ways of engineering them are therefore mandatory for enabling them to self-build, self-tune and self-assemble.

Adopting a bottom-up approach to enable a model to build itself by giving the basic components and letting them interact in the right way is an imaginable solution. Multi-agent systems (MAS) are suited for this.

For some years now, agent technology is considered as a possible answer to biological domain problems [27][2]. Agent-based models are primarily used to deal with huge quantities of data [27] or for simulating *in virtuo* experiments: protein docking or folding [3][4], modeling signaling pathways [15][22], modeling unicellular organisms alone [25][33] or within a population [12][16]. For example, in CellAK [33] complexity is dealt with software engineering principles: a UML class diagram expresses inheritance relationships between the different components of a eukaryotic cell (membrane, cytoplasm and nucleus) that are decomposed in turn until a certain

level of complexity is reached. If in this case, agents are a way to obtain more easily understandable models compared to models based on differential equations like Gepasi [26], such a top-down approach does not make the model flexible and able to self-build.

Simulating biological systems using actual multi-agent systems is still in an early stage of study. MAS are very promising for helping understanding underlying self-organization mechanisms in populations of biological organisms. [31] considers interactions between agents representing cells for modeling tissues and focuses on self-organization phenomena within these tissues. [30] studies a self-organization phenomenon at the molecular level. [10] models the behavior of stem cells within a niche to study the emergent global organization of this population. [5] and [20] are more focusing on modeling intracellular phenomena. An AGR [13] approach is extended in [5] for describing the internal organization of a unicellular organism (E. Coli) and proposing a software environment that enables formulation of dynamic properties within this organization. In [20] internal states of agents are described using representations and motivations on a BDI basis. The aim is to model dynamics at the agent level and at the level of its externally observable behavior with the intent to study whether internal dynamics generates the external behavior. Although this approach may help to understand relations between the two levels of a cell, it does not provide biologists with a tool that enables them to discover new phenomena within a cell. Actually, to our knowledge, most of MAS aiming at modeling biological systems consider that laws governing components of these systems are known, or can be inferred. The organization between agents is therefore predefined and static in these models which are unable to evolve and dynamically respond to disruptions. Furthermore these laws are generally not completely known for all the levels that need to be modeled in a unicellular organism (genome, proteome, metabolome) and letting the model learn these rules in order to reflect experimental data would alleviate complexity and then modelers' workload.

Outlining how complex systems modeling can be engineered in order to make the obtained model self-build is the objective of this paper. This is illustrated with an application, MicroMega, related to systems biology which aims at modeling the functional behavior of the unicellular yeast Saccharomyces Cerevisiae.

This article is organized as follows. Concepts adopted for making a model build itself are described in Section 2. These concepts are applied in section 3 for setting up the architecture implementing the MAS related to MicroMega. Some preliminary results obtained by simulating the glycolysis metabolic pathway in Saccharomyces Cerevisiae are discussed in Section 4 before concluding.

2 Towards Self-building Systems

Few approaches exist for engineering systems with self-organizing or emergent properties. For instance, [8] merges an analysis algorithm with simulation runs in order to tune variables reflecting chosen macroscopic properties. This approach does not completely unburden engineers because the analyzed macroscopic variables have still to be identified and the feedback has to be used in the engineering process. Engineers' role has to be reduced by initially providing the system with existing expertise and letting it

build itself while giving it, if possible and required, some minimal feedback from time to time. Under these conditions, making a complex system build itself is done by letting it autonomously change the organization between its components but also by enabling these latter parts change as well their behavior in an autonomous way. Principles of self-organization are an answer to the former point [18][9]. The second point can be fulfilled by endowing components with abilities of learning what is unknown or incompletely known from experts designing the system. This learning concerns their features (for instance, chemical or physical laws they apply) as well as they ability to appear into or disappear from the system depending on whether they are useful or not. Self-building requires then properties of self-organization at the system level, and self-tuning, self-reorganization and evolution at the component level.

A four-layer model for engineering systems having those properties is detailed in this section and applied in the following sections on an example coming from the biological domain.

2.1 Self-organization by Cooperation

The approach proposed here rests on the Adaptive Multi-Agent Systems (AMAS) theory in which self-organization is led by cooperation which embodies the local criterion that makes agents self-reorganize [6]. When an agent locally detects, at any time during its lifecycle, a situation that may be harmful for its cooperative state, it changes its relationships with others to stay or come back to a cooperative state. Situations that are against the cooperative social attitude of an agent are called Non Cooperative Situations (NCS). Furthermore processing these NCS enables an agent to constantly adapt to changes coming from its environment and therefore provides it with learning abilities.

Since the objective is to simulate the functional activity of a given system, agents of the self-building model represent either elementary domain-related objects or functions which manipulate these objects. Usually, elementary objects are easy to identify by answering the simple question « What elements are making up my targeted system? ». Such a naive approach is usually (and historically) set aside because of the tremendous implied computation load of the simulation and the huge complexity of the required model design and control. In order to deal with these two aspects, we chose to totally rely on the emergence property of Adaptive MAS:

- Simulation computation load is reduced by the fact that MAS are only composed of agents with very light computational capabilities. There is no need of any non local control to ensure consistency of the whole system activity. Moreover, as agents behave according to purely local and limited information, they are more readily to be computed in a distributed way.
- Model design complexity is greatly reduced by locally cooperation-driven self-adaptation. The adaptation process is not based on any global feedback from the system environment toward the whole MAS; no fitness function neither performance measures of the whole MAS are used. On the contrary, this adaptation process fully relies on emergence to ensure consistency and keep advantage of low computation load, readily distribution etc.

2.2 A Four-Layer Model of Agent

Within the model, agents are all designed alike and consist of three main modules:

- *RepresentationModule*: this module contains all information an agent deals with: internal parameters, characteristics, knowledge about other agents etc. All other modules have to use the representation module to obtain and store all relevant data;
- *InteractionModule*: it manages all kinds of interaction between an agent and its environment (including other agents). Since communication between agents is often based on message exchange, the default interaction module manages a peer-to-peer communication system;
- *BehaviorModule*: this module defines the agent behavior which consists of two parts: the nominal behavior and the cooperative one which is subdivided into tuning, reorganization and evolution. Figure 1 shows how these behaviors interact with one another and the agent environment (other agents, user, external data).

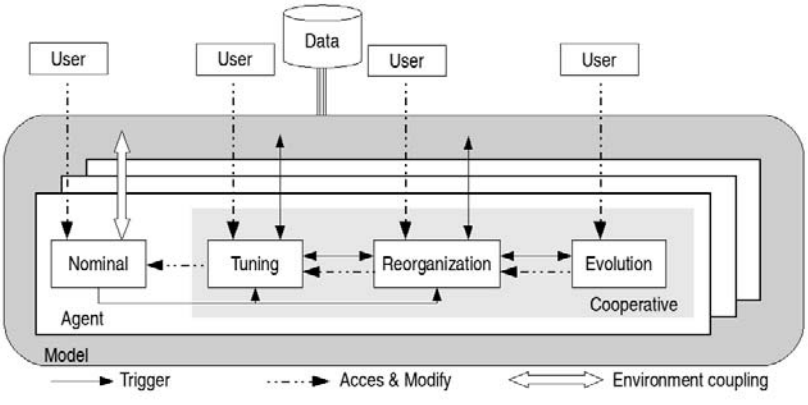


Fig. 1. Four-layer model of a self-adaptive agent

Nominal Behavior. The nominal behavior of an agent is based on the perceive-decide-act lifecycle. Basically, this behavior corresponds to whatever this agent does unless the action performed is a learning action. Learning actions modify the agent itself or its relation to its environment in such a way that, from now on, it will act in a different way than before for at least one situation. In stochastic-driven behaviors (like Monte-Carlo), the action of randomly selecting an action according to a set of parameters is not a learning action. In that context, the randomized selection IS the nominal behavior while the parameters do not change: the fact that the agent acts randomly is neither unpredictable nor random... even if the resulting actions are.

However, nominal behavior can sometimes be so complex (specification of a wide range of situations associated with smart actions) that an agent may appear to be adaptive whereas it only behaves contextually. The limit between nominal adaptation and actual adaptation is often problematic.

In Fig. 1, one can notice nominal behavior obviously interacts with the agent environment (two-sided empty arrow) or a user (dotted arrows) and sometimes it may

directly trigger tuning or reorganization behaviors (solid arrows) if it is unable to execute because of a bad parameter value or a missing link with other agents.

Tuning Behavior. This behavior is the first cooperative layer of adaptive agents: it manages parameters values tweaking for an agent. Basically, agent tuning consists in analyzing the nominal behavior computation to find cooperation failures. These Non Cooperative Situations (NCS) can be either endogenous (exception or error while executing the nominal behavior) or exogenous (messages from other agents, conflict or concurrence of actions etc.). Tuning behavior tries to solve these NCS by modifying the parameters that take an active part in nominal behavior.

Figure 1 shows tuning behavior may modify the nominal one (dotted arrow on the left), trigger reorganization if it fails to solve a problem and send messages into the environment (toward other agents) in order to delegate/propagate non cooperative situations. Tuning activation usually comes from messages sent by tuning behavior of other agents, nominal behavior failure or reorganization behavior modifications. User is free to suspend or resume tuning activity as well as modifying the parameters of the tuning algorithm.

Reorganization Behavior. It consists in modifying the way in which an agent interacts with its environment and other agents. Cooperation failures that require reorganization can occur either during nominal or tuning behavior. Usual NCS that lead to agents reorganization are:

- partial or total uselessness: an agent needs to establish a link with a new partner because it is currently not able to execute its nominal behavior;
- incompetence: an agent is unable to perform a satisfying behavior (other agents always send requests) and the tuning behavior seems unable to cope with the problem.

As illustrated in Fig. 1, reorganization behavior is caused by tuning failure, new agent appearance (evolution) or messages reception (other agents are looking for some help). Sometimes, a nominal critical failure can directly lead to reorganization-related actions: for example an agent whose main behavior consists in adding two numbers provided by two other agents cannot execute this if it is only connected with zero or one agent.

Evolution Behavior. It is the last kind of modification an agent can perform to solve any problem coming from the previous layers and, more precisely, coming from reorganization failures (see Fig. 1). Evolution[1] actions concerns system openness and consist in creating new agents or removing itself. New agents are generally created when the reorganization process is unable to find new agents to solve uselessness. Agent self-removal can only be performed when agents are in total uselessness.

As a matter of fact, the model is continuously evolving according to the data it is perceiving while the cooperative behavior is enabled and non cooperative situations are detected. Nevertheless this adaptation is not transient (purely instantaneous) because cooperative behaviors must remain consistent with past learnt states. So, the

[1] Evolution here is not related to Darwin's theory of evolution (species evolution) but only refers to the dynamics of the system from an openness point of view (add/remove agents).

model is supposed, according to the AMAS theory[2], to converge towards a stationary state. Once this state is reached cooperative behaviors can be disabled or even removed, the only remaining behavior being the nominal one.

The next section details how this framework has been applied for modeling a biological complex system.

3 System Architecture in MicroMega

MicroMega[3] application aims at simulating the functional behavior of a yeast cell. The computational model has to integrate phenomena from genomic level (e.g. genes activity) to macroscopic data (e.g. quantity of consumed/produced O_2, CO_2, glucose etc.). Since chemical elements involved into cellular activity are hugely numerous as well as their transformations/interactions, MicroMega has to be designed taking into account the need of autonomous building and tuning of the yeast model according to experimental data and user wishes as well as adaptive capabilities toward user interaction to help or drive this highly complex process.

MicroMega is based on a unified model with a multi-agent system that simulates chemical reactions from RNA production at genomic level to exchanges of substrates with extracellular environment. The system is made up of two main classes of agents: functional agents (elements or reactions) or viewer agents. These different types of agent and the way the simulation is handled to control their computations are presented hereafter.

3.1 Functional Agents

In MicroMega, functional agents represent either physical elements or chemical reactions. Elements and reactions interact: reactions produce or consume elements and elements may act as regulators for reactions.

Element Agents. These agents are representative of physical items that constitute the cell: RNA, substrates, proteins, protons (H+), water etc. Their core function is to manage the quantity of the element they represent during the simulation. As this quantity is generally not uniformly distributed in the cell, each physical element is actually represented by a group of element agents in which each agent manages the quantity into a given compartment (extracellular, cytoplasm etc.).

However, element agents are quite passive from a functional point of view: they do not actually modify or compute their quantities by themselves because these modifications are computed by reaction agents (see below). Some information like unitary mass or internal energy of an element can also be managed by element agents.

Element agent tuning behavior is able to handle incompetence: the agent receives a message from either a viewer agent or a reaction agent it regulates. This message requests a different quantity value. The agent modifies its current quantity value according to the requested one and sends requests to the reaction agents it is linked with

[2] If our implementation actually fits the AMAS theory i.e. if all NCS are properly identified and processed.

[3] Project funded by ANR (National Agency for Research) under the number BLAN-05-0202.

to increase or decrease their reaction speed (and therefore be more or less consumed or produced for tuning their quantity).

Reorganization behavior handles:

- uselessness: an element agent with no reaction agent is useless: it has to broadcast messages to inform reaction agents that it is available;
- incompetence: an element agent receives messages related to its quantity value from either viewers (value not matching experimental data) or reactions whose this element is a regulator (the reaction agent wants to change its context – see below). This element agent has tried to change its quantity during its tuning behavior and this is not sufficient. Therefore the confidence it may have toward its current partners is going to decrease and it may search for new partners in order to regulate other reactions.

$$1\ \text{Fructose1,6DP} + 2\ \text{ADP} + 2\ \text{NAD+} \rightarrow 2\ \text{Pyruvates} + 2\ \text{ATP} + 2\ \text{NADH,H+} \quad (1)$$

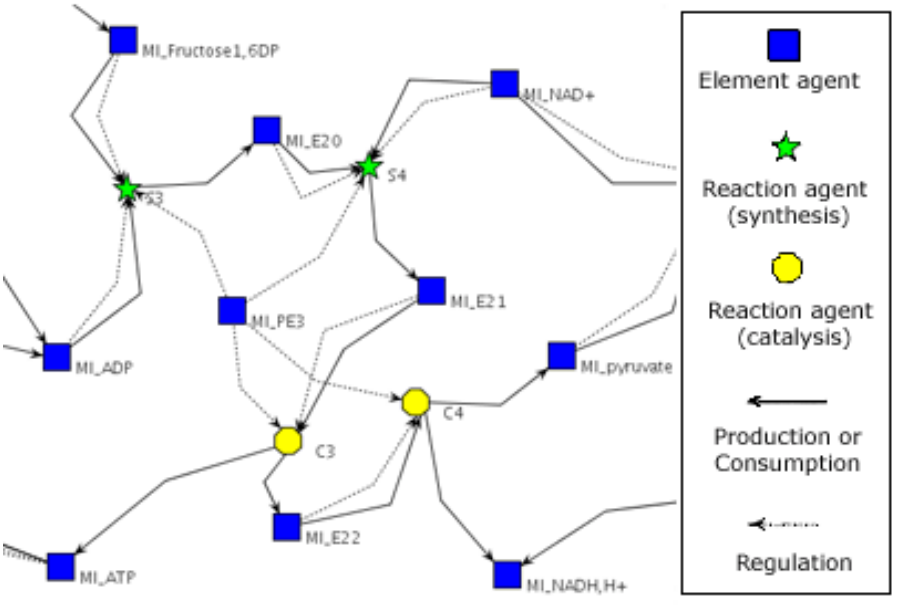


Fig. 2. Agentification of the complex reaction (1) catalyzed by one protein. Names of elements are prefixed by MI_ because this reaction takes place in the mitochondrial compartment.

Reaction Agents. They manage all transformation and transportation of elements within the cell. Chemical reactions are reduced to two elementary subtypes of reaction:

- Synthesis: a given element is broken down into two parts. An example of basic synthesis reaction is the electrolysis of water to give hydrogen and oxygen: $2\ H_2O \rightarrow 2\ H_2 + O_2$.

– Catalysis: two elements are docked together to build a new one. Hydrogen combustion ($2\ H_2 + O_2 \rightarrow 2\ H_2O$) is a well-known example of such a reaction.

Each synthesis or catalysis agent is also characterized by its stoichiometry, that is to say the quantitative relationship between the reactants and products of the chemical reaction, or -from a practical point of view- the weights of inputs and outputs of reaction agents. By combining these two subtypes of reaction and using intermediate elements, more complex reactions can be built. Figure 2 shows a combination of 2 synthesis (stars), 2 catalysis (octagons) and 3 intermediate elements (E20, E21, E22 – squares) to achieve transformation of 1,6 DP to pyruvate with one protein (PE3) as the reaction speed regulator (dotted lines represent regulation influence).

Two types of reaction agents manage transport:

– Passive carriers which produce/consume ambient energy (temperature, pressure). This energy is usually dissipated or absorbed around the reactive site;
– Active carriers which need to consume high energy (from ATP for instance) to achieve transportation.

Carrier agents are only used for intercompartment transport because we assume elements are uniformly distributed in each compartment.

Moreover genes are modeled as reaction agents: they consume several metabolites and produce one RNA and one protein. Genes design is peculiar because they must be paired with an RNA element. This RNA is produced by a gene but can also be consumed by this gene and is involved as a regulator of its activity.

During the simulation, each reaction agent consumes and produces elements this agent is linked with according to its stoichiometry and a contextual reaction speed. This speed determines how fast the reaction is running and as a consequence how much of the quantities of elements the reaction consumes and produces. Speed is contextual because reactions are regulated by their environment: some elements can speed up the reaction rate or slow it down. These elements define the reaction context as the vector of their current quantities. Thus one speed is associated with each possible vector in order to define the discrete function which enables speed computation.

Reaction agent tuning behavior handles the following NCS:

– Unproductiveness: the current context is unknown so the agent has to use a default context (with a speed value of 0). The agent tries to solve this NCS by adding a new context which corresponds to the current situation.
– Incompetence: the agent receives requests from neighbor element agents to change a previous consumption or production. The agent tries to adjust either its stoichiometry values or the speed. Speed adjustment can be performed by modifying the speed associated with the context used or by influencing the selection of contexts. The selection can be changed either by adjusting the ranges of regulators of the existing contexts or by requesting different values from the regulators themselves to switch situation for one that will allow to select a better context.

Reorganization behavior handles uselessness (missing consumption/production link) and incompetence. For instance, if a reaction agent is unable to tweak the speed of a

given context (because two different speed values are alternately requested in the same context); it may find a new regulator to help distinguishing the two situations.

3.2 Viewer Agents

Viewer agents enable interactions between functional agents and the system user by extracting comprehensible data (for analysis and display purposes) and injecting experimental data and constraints specified by this user. This is a way to inject experts' knowledge into the model. Two types of viewers are currently used in MicroMega:

- *ElementViewerAgent*: this basic type of viewer agent is dedicated to gather quantities of a given list of element agents. It is possible to specify experimental values associated with these quantities to dynamically compare them with simulation results;
- *ElementSetterAgent*: this is a viewer agent which remotely controls the quantity value of a given list of element agents. Whatever the computed quantity value of the element an *ElementSetterAgent* will nominally set the quantity value according to its database. These viewer agents are used to control the activity of specific elements during simulation (some regulators, genes etc.).
  However, other kinds of viewer agents could be defined:
- *BioMassCheckerAgent*: this viewer agent evaluates the whole mass of the cell by summing the mass of all elements of the system;
- *CompartmentsAgent*: this viewer agent analyses the system focusing on carrier agents to identify different compartments within the cell. This information could be useful if the system is able to self-reconfigure and the user might wish to control such a critical parameter in order to limit changes at the global cell structure level.

The nominal behavior of viewer agents consists in accessing data of functional agents and storing the gathered values. Viewer agents which compare the gathered data to experimental ones can also compute errors. These errors will be used during the tuning part which usually deals with conflicts. If the error computed during nominal behavior equals 0, the viewer agent sends a positive reinforcement message to the involved element agents; otherwise it sends them quantity error messages.

3.3 Simulation Control and Computation

MicroMega simulation process is handled by a single thread and decomposed into elementary time steps. Each step represents an arbitrary discrete item of time and is decomposed into two phases (each one is handled by a scheduler): the nominal computation and the cooperative computation which is itself broken down into tuning, reorganization and evolution (according to the model given in section 2.2).

Nominal computation corresponds to simulation computation. Within each step, the nominal scheduler notifies each agent of the system to execute its nominal behavior. These behaviors actually manage yeast simulation through chemical elements quantity updates and trigger data gathering and injection in the case of viewer agents. As a matter of fact, nominal scheduling is decomposed as follows for each step t:

1. Activation of each reaction agent to compute quantity variations of elements;

2. Activation of each element agent to update its quantity according to the quantity variations computed by reaction agents;
3. Activation of all viewer agents to update data from agents and/or check whether MAS organization and parameters fit with the user data (experimental data, for example).

Cooperative computation corresponds to yeast model adaptation according to both internal problems (for example, if model state becomes inconsistent by computing negative element quantities) and external data (for example, if collected data no longer fit experimentally probed quantities or well-known properties of the cell). Each agent sequentially executes parameters tuning (adjustments of quantities, of reaction speed etc.), reorganization (addition/removal of reaction regulations or producer/consumer link between reactions and elements etc.) and evolution (addition/removal of new elements or reaction agents). During each step t, the cooperative scheduler activates agents following the reverse order of the nominal phase:

1. Activation of each viewer agent to notify functional agents if they detect any problem for the parameters computed during step t of nominal computation;
2. Activation of each reaction agent to check if they have received any message (under- or overestimated speed) or have detected any computation problem (like an unknown contextual value or speed, missing element agents) while performing their production/consumption during step t-1;
3. Activation of each element agent to check inconsistent parameters (negative values) or bad/good quantity values (from step t-1) notified by quantity-related messages from either viewer agents or reaction ones.

As one can acknowledge, all the agents of each group (element, reaction, and viewer) can be readily triggered simultaneously because no direct interaction occurs between agents that belong to the same type.

4 Simulation Results

To test MicroMega we have first addressed glycolysis modeling as an example of biochemical pathway. Glycolysis allows cells to transform absorbed glucose into energy or smaller metabolites like pyruvate that roughly are used in biomass production. Mathematical models in cell biology are overcome by the complexity of described systems and are limited to linear dynamic systems in steady state. Model reduction is of major use to produce more simple and stable models that balance between approximation accuracy and numerical efficiency [19].

Biochemical network models transpose metabolic systems into differential equation systems like: $s(t) = Nv(s(t); p)$ where s is the vector of metabolite concentrations and v is the vector of reaction velocities. The vector p contains the kinetic parameters, and the stoichiometric matrix N contains coefficients for corresponding reactions [17].

When MicroMega is provided with a set of reactions (see left part of Fig. 3) under the shape of a stoichiometric matrix, it will produce a multi-agent system (see right part of Fig. 3), containing quantitative elements (including reaction intermediates) and functional elements. A second matrix containing data of regulation can be loaded to help the system during the learning of interactions.

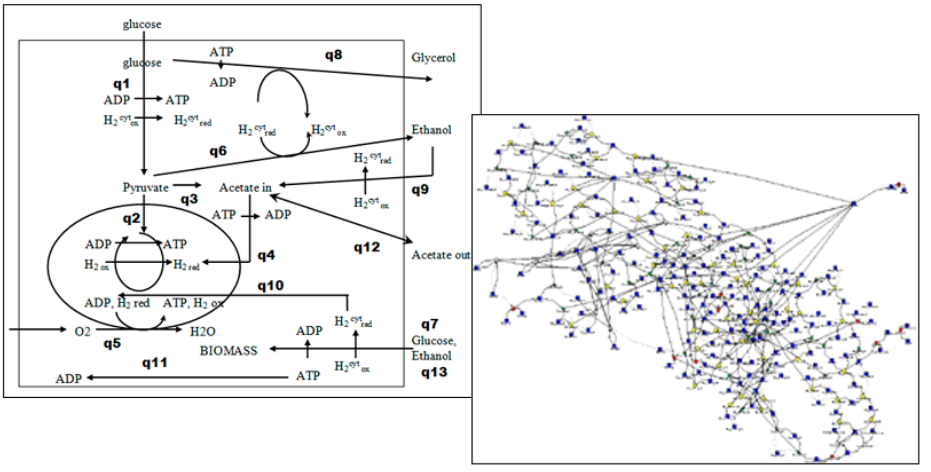


Fig. 3. Simplified yeast glycolysis metabolic pathway (on left) and MicroMega graph visualization associated (on right, legend is the same as in Fig. 2)

For a case study, a simpler model of glycolysis has been produced and implemented in MicroMega. This toy model has no biological validity, since it has been given with arbitrary kinetic values, but shows MicroMega platform about context selection functioning. Figure 4 presents some curves of this virtual experiment with 2 pulses of glucose.

It is of interest to notice that curve profiles are different after first and second pulse. Differences of kinetics are based on different intracellular concentrations of some metabolites. Therefore previous states of the system influence its current behavior.

Another interesting consequence of MicroMega simulation process is visible on Fig. 4 (zoomed area) and concerns emergence of combined kinetics based on oscillation between different states. As it can be seen in A and B, a mean speed results of quick oscillation between two or more very different speeds.

Behavior of the system is the result of given or learned kinetic parameters and of previous states of the system; under these conditions MicroMega goes beyond classical steady state description of metabolic reactions. Each reaction agent locally observes its context of functioning and consequently determines a kinetic. By this way any reaction agent can become aware of transient states triggered by some fluctuating values of element agents. A consequence is visible in Fig. 4 (zoomed box), the appearance of high frequency oscillations in A and B enlightens us of at least two underlying concomitant states. Steady state models could only express such a phenomenon with a mean kinetic.

As it can be seen on the curves related to products of glycolysis CO_2 and acetate (red and yellow ones, Fig. 4) after the second pulse of glucose, the glycolytic activity of the model is more important than after the first glucose pulse. This difference is due to specific repartition of intracellular metabolites that sets up during the first part of the simulation.

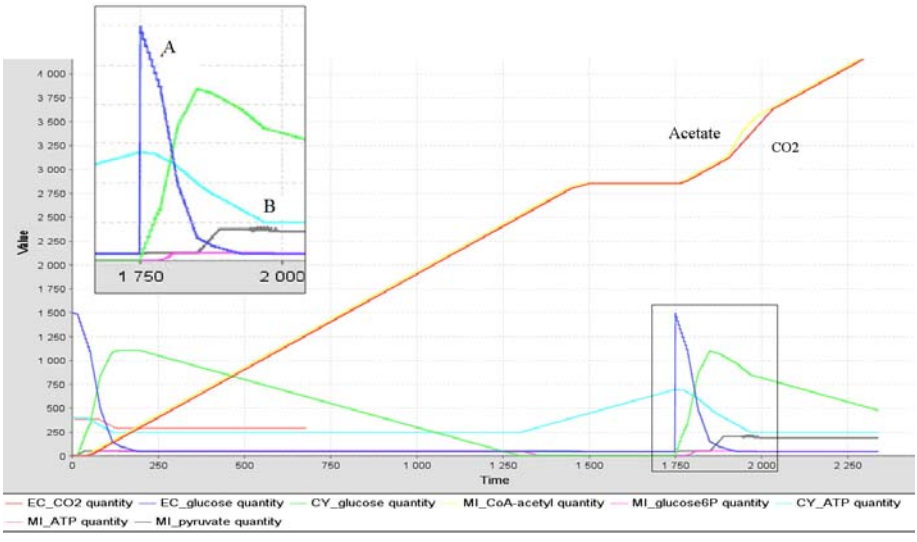


Fig. 4. Simulation results of a simplified model of glycolysis with two glucose pulses (1500 units), first at t=0 and second at t=1750. Extra-cellular glucose is consumed, in a first stage turned into intra-cellular glucose, then metabolized into acetate and CO_2 that are stored up in this closed model. Emergence of some quick kinetic oscillations in A (glucose consumption) and B (mi_pyruvate) after a second pulse of substrate can be seen in the zoomed area.

These results show that the simulated system even though in strict nominal functioning, has a contextual nominal adaptive behavior. We could qualified this observed adaptation as "weak" in the sense that neither the system nor one of its part does actually learn something but just acts contextually and according to its changing internal states. Concentrations of many intracellular metabolites and enzyme cofactors provide the model with a kind of memory of previous states that will modify future behavior related to environmental changes.

5 Conclusion

Due to the complexity of nowadays systems, engineers are no longer able to know everything in order to implement them or to fully control them. Although multi-agent systems are a recognized paradigm for implementing complex systems, engineering them is also a complex task. New ways of engineering complexity are thus required and the aim of this paper was to present a model for making such nonlinear complex systems self-build in order to lighten engineers' workload. Self-organization driven by cooperation was chosen to enable not only an autonomous evolution of the system organization but also an autonomous adjustment of the behavior of its agents. The four-layer agent model proposed separates the "basic" behavior of agents composing an adaptive MAS from the behavior that enables them to self-tune, self-reorganize and evolve. Application of this agent model to an application related to biology is then detailed and some preliminary results are given.

Focusing on a specific domain such as biology, biologists rely on models to understand natural phenomena or to discover new laws through less expensive *in silico* experiments. Openness and adaptation are two required features in the modeling process and with their inherent potential MAS are becoming a promising answer toward automatic modeling. In this sense, we applied the proposed framework for modeling the intracellular functioning of S. Cerevisiae yeast. Although the model obtained is still incomplete, preliminary results show that it exhibits adaptation abilities when disrupted. Among the few multi-agent approaches that exist for modeling biological problems, fewer try to deal with dynamical phenomena and disruptions. For example, [28] uses features of oRis, a dynamic language, for disturbing the model of the MAPK while the simulation is running. Adding, removing of modifying agents can be done without restarting the simulation and this brings the user closer to *in vivo* or *in vitro* experiments. Dynamic aspects are then handled however the model does not self-build. In [32], a combination of top-down and bottom-up approaches for modeling biological networks rests on holonic MAS. Reactions (transformation, transport or binding process) are viewed as interactions between two holons and several rules. Each holon can manage its rate and stoichiometry by inferring from the rule engine. However dynamic configuration is possible at run-time to get closer to real situations: user can add or remove substrates or products, change rules or regulate the best rate for a reaction and this latter can also be learnt using neural networks or genetic algorithms. So, dynamic changes in the model are endured, however concluding that the model self-builds is difficult due to the lack of details about how learning is done at the level of holons.

From a pure modeling point of view, MicroMega approach of pathway modeling is generic and not restricted to glycolysis: any cellular process that can be described as regulated production/consumption mechanisms is potentially transposable into a set of element and reaction agents. Nevertheless, offering multi-state and transient possibilities using contextual functioning has an important drawback: the list of contexts a reaction agent deals with is rather more difficult to visualize and analyze than global differential equations.

Finally this article wants to claim that this self-building ability is the only answer for dealing with complexity, notably in biological modeling, and MicroMega aims at sustaining this argument. It has been designed in such a way that building and refining complex models will be greatly facilitated by a cooperative behavior. Even if some phases of the model are still under development (such as the evolution behaviors of agents), this approach supports the claim that only an emergent approach has chances to permit complex systems to self-build.

References

[1] Alon, U.: An Introduction to Systems Biology: Design Principles of Biological Circuits. Chapman & Hall, Boca Raton (2006)

[2] Amigoni, F., Schiaffonati, V.: Multiagent-Based Simulation in Biology: A Critical Analysis. In: Magnani, L., Li, P. (eds.) Model-Based Reasoning in Science, Technology, and Medicine. Studies in Computational Biology, vol. 64, pp. 179–191. Springer, Heidelberg (2007)

[3] Armano, G., Mancosu, G., Orro, A., Vargiu, E.: A Multi-agent System for Protein Secondary Structure Prediction. In: Priami, C., Merelli, E., Gonzalez, P., Omicini, A. (eds.) Transactions on Computational Systems Biology III. LNCS (LNBI), vol. 3737, pp. 14–32. Springer, Heidelberg (2005)
[4] Bortolussi, L., Dovier, A., Fogolari, F.: Multi-Agent Simulation of Protein Folding. In: Proc. of the First Workshop on Multi-Agent Systems for Medecine, Computational Biology, and Bioinformatics (MAS*BIOMED 2005@AAMAS 2005), pp. 91–106 (2005)
[5] Bosse, T., Jonker, C., Treur, J.: Simulation and Analysis of Complex Biological Processes: an Organisation Modelling Perspective. In: 39th Annual Simulation Symposium (2006)
[6] Capera, D., Georgé, J.-P., Gleizes, M.-P., Glize, P.: The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In: 12th IEEE International Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises (WETICE), Linz, June 9-11, 2003, pp. 383–388. IEEE Computer Society, Los Alamitos (2003)
[7] De Jong, H.: Modelling and Regulation of Genetic Regulatory Systems: a Literature Review. Journal of Computational Biology 9, 67–103 (2002)
[8] De Wolf, T., Samaey, G., Holvoet, T.: Engineering Self-Organising Emergent Systems With Simulation-Based Scientific Analysis. In: Brueckner, S., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.) Proceedings of the Third International Workshop on Engineering Self-Organising Applications, Utrecht, pp. 146–160 (2005)
[9] Di Marzo Serugendo, G., Gleizes, M.-P., Karageorgos, A.: Self-Organization in Multi-Agent Systems. The Knowledge Engineering Review 20(2), 165–189 (2005)
[10] d'Inverno, M., Saunders, R.: Agent-Based Modelling of Stem Cell Self-organisation in a Niche. In: Brueckner, S.A., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (eds.) ESOA 2005. LNCS, vol. 3464, pp. 52–68. Springer, Heidelberg (2005)
[11] Efroni, S., Harel, D., Cohen, I.R.: Toward Rigorous Comprehension of Biological Complexity: Modeling, Execution, and Visualization of Thymic T-Cell Maturation. Genome Research 13, 2485–2497 (2003)
[12] Emonet, T., Macal, C., North, M., Wickersham, C., Cluzel, P.: AgentCell: a Digital Single-cell Assay for Bacterial Chemotaxis, Bioinformatics Advance Access. Bioinformatics 21, 2714–2721 (2005)
[13] Ferber, F., Gutknecht, O.: A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. In: Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS 1998), Paris, France, pp. 128–135 (1998)
[14] Fisher, J., Henziger, T.: Executable Cell Biology. Nature Biotechnology 25(11), 1239–1249 (2007)
[15] González, P., Cárdenas, M., Camacho, D., Franyuti, A., Rosas, O., Lagúnez-Otero, J.: Cellulat: an Agent-based Intracellular Signalling Model. Biosystems 68(2-3), 171–185 (2003)
[16] Guo, D., Santos, E., Singhal, A., Santos, E., Zhao, Q.: Adaptivity Modeling for Complex Adaptive Systems with Application to Biology. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 272–277 (2007)
[17] Heinrich, R., Schuster, S.: The Regulation of Cellular Systems. Chapman & Hall, Boca Raton (1996)
[18] Heylighen, F.: The Science of Self-organization and Adaptivity. In: The Encyclopedia of Life Support Systems. EOLSS Publishers Co. Ltd. (2001)
[19] Hynne, F., Dano, S., Sorensen, P.G.: Full-scale Model of Glycolysis in Saccharomyces Cerevisiae. Biophys. Chem. 94, 121–163 (2001)

[20] Jonker, C.M., Snoep, J.L., Treur, J., Westerhoff, H.V., Wijngaards, W.C.A.: BDI-modelling of Complex Intracellular Dynamics. Journal of Theoretical Biology 251, 1–23 (2008)
[21] Kephart, J., Chess, D.: The Vision of Autonomic Computing. Computer 36(1), 41–50 (2003)
[22] Khan, S., Makkena, R., Gillis, W., Schmidt, C.: A Multi-agent System for the Quantitative Simulation of Biological Networks. In: Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003), Melbourne, pp. 385–392. ACM, New York (2003)
[23] Kitano, H.: Systems Biology: A Brief Overview. Science 295, 1662–1664 (2002)
[24] Li, C., Ge, Q.W., Nakata, M., Matsuno, H., Miyano, S.: Modelling and Simulation of Signal Transductions in an Apoptosis Pathway by using Timed Petri Nets. J. Biosci. 32, 113–127 (2007)
[25] Lints, T.: Multiagent Modelling of a Bacterial Cell, a DnaA Titration Model Based Agent Model as an Example. In: Vene, V., Meriste, M. (eds.) Proceedings of the Ninth Symposium on Programming Languages and Software Tools, Tartu, Estonia, pp. 82–96 (2005)
[26] Mendes, P.: GEPASI: A Software Package for Modelling the Dynamics, Steady States and Control of Biochemical and other Systems. Computer Applications in the Biosciences 9(5), 563–571 (1993)
[27] Merelli, E., Armano, G., Cannata, N., Corradini, F., d'Inverno, M., Doms, A., Lord, P., Martin, A., Milanesi, L., Moller, S., Schroeder, M., Luck, M.: Agents in Bioinformatics, Computational and Systems Biology. Briefings in Bioinformatics 8(1), 45–59 (2006)
[28] Querrec, G., Rodin, V., Abgrall, J.F., Kerdelo, S., Tisseau, J.: Uses of Multiagents Systems for Simulation of MAPK Pathway. In: Proceedings of the Third IEEE Symposium on Bioinformatics and Bioengineering (BIBE 2003), pp. 421–425 (2003)
[29] Ripoll, C., Guespin-Michel, J., Norris, V., Thellier, M.: Defining Integrative Biology. Complexity 4(2), 19–20 (1998)
[30] Troisi, A., Wong, V., Ratner, M.: An Agent-based Approach for Modeling Molecular Self-organization. Proceedings of the National Academy of Sciences of the USA (PNAS) 102(2), 255–260 (2005)
[31] Santos, E., Guo, D., Santos Jr., E., Onesty, W.: A Multi-Agent System Environment for Modelling Cell and Tissue Biology. In: Arabnia, H.R. (ed.) Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 3–9. CSREA Press, Las Vegas, USA (2004)
[32] Shafaei, S., Aghaee, N.: Biological Network Simulation Using Holonic Multiagent Systems. In: Tenth International Conference on Computer Modeling and Simulation (UKSIM 2008), April 1-3, pp. 617–622 (2008)
[33] Webb, K., White, T.: Cell Modeling with Reusable Agent-based Formalisms. Applied Intelligence 24(2), 169–181 (2006)

From Individuals to Social and *Vice-versa*

André Campos[1,2], Frank Dignum[2], and Virginia Dignum[2]

[1] Computer Science and Applied Mathematics Department
UFRN Lagoa Nova, 59072-970 Natal-RN, Brazil
andre@dimap.ufrn.br
[2] Information and Computing Sciences Department
Padualaan 14 3584CH Utrecht, The Netherlands
{dignum,virginia}@cs.uu.nl

Abstract. The concept of cyclical influence between individuals and society is widely accepted, but hard to understand in all details. This paper proposes the use of three processes of social influence as a way to study the link between social and individual levels of abstraction. These processes are used to design an agent architecture which tries to provide explicit links to its social context. In order to detail the impact of the social influence, the architecture also includes personality and emotional aspects.

Keywords: Social influence, BDI agents, personality, emotional agents.

1 Introduction

The study of social dynamics based on individual behaviors is not recent ([1] *apud* [2]). However, in the last years, this approach has been deeply boosted by the capacity of current models to represent in a more precise way the structures, norms, culture, and resources available in a society, as well as by the capacity to model in more detail the complexity of the human behavior. This includes the results from the interactions between emotions, personality traits, cognitive reasoning and, certainly, how the individuals relate to their social context.

These two levels of complexity (society and human behavior) are extremely interdependent to each other. Several works have already being conducted to study their relationships. However, those works normally address one of the two research directions, either: 1) by studying the emergence of collective patterns and structures from the individuals, or 2) by studying the influence of social elements (norms, policies etc.) over the individuals. Some researchers have also conducted analyses to evaluate the two pathways, composing the cycle macro-micro-macro. Nevertheless, for the sake of simplicity, these studies normally address only specific relations (e.g. norms and emotions [3]) and do not involves many aspects of the human being.

In order to facilitate the model translation between micro and macro levels of abstraction some authors (e.g. Dignum et. al [4]) propose the use of a mediation layer, named meso-layer. The latter connects the two other levels of abstraction by gathering information from the macro level elements (like global behavioral patterns) and including new ones (like norms and organizational structures) that "influence" individual

A. Artikis, G. Picard, and L. Vercouter (Eds.): ESAW 2008, LNAI 5485, pp. 264–279, 2009.

behavior. The meso-layer presented in Dignum et. al [4] aims to study the influence of policies over the individuals. These policies are defined by policy makers at the macro and meso levels in an agent-based simulation, and the resulting behavior is analyzed. Briefly, it is used in a top-down approach, which means that there is no change in the meso level arising from the bottom-up (the agents).

Although this is enough for studying policy making, it is not adequate to study the emergence of elements present in the meso level. An example is the dynamics of existing groups or the creation of new ones. Groups, as elementary social structures, are part of the meso layer. However, they should not necessarily be formed or imposed as part of a social or organizational policy. Individuals can spontaneously leave or take part of a group according to their interests, needs, preferences and objectives, following a bottom-up approach.

In order to make changes in the meso level that reflects human societies, it is necessary to use coherent theories linking psychological foundations with social behavior. This paper proposes an agent architecture addressing this need, based on the model of social influence proposed by Kelman [5]. The architecture also incorporates other theories in order to detail several human characteristics, more precisely: emotions, personality, personal and social values.

The following two sections introduce the elements of social influence proposed by Kelman and how they related to different levels of abstraction, respectively. The Section 4 presents other elements of human behavior that we consider important to configure the link between micro and meso, as well as the reasons for choosing specific models. After, the components and the cognitive process of the architecture of the social agent are described. Then, some final remarks are made, pointing out potential applications and the next steps of the work.

2 Social Influences

The behavior of a society can be seen as a consequence of the choices and actions performed by their members, but their choices are also influenced by the whole system. This recurrence generates a highly dynamic behavior cycle that may explain the resistance and/or desire for changes within a society.

In order to examine this cycle, it is necessary to study the mechanism that influences individual behavior from a social point of view. Kelman investigation on social influence, proposes a linking model between the individual and social systems. Although his studies were initially used as a mean to understand the mechanisms which allow a person to influence a target audience [6], the original model proved to be also useful in different contexts: from group psychotherapy to large social systems, involving organizations and very large social contexts, involving nations and its links to individual values (e.g. national identity) in the context of peacemaking (these works are summarized in [5]).

For Kelman, individual beliefs are not necessarily (fully) integrated into the person's own value system. The latter is highly dependent on external influence. His investigation on this dependency led him to distinguish three processes of social influence: compliance, identification and internalization [7].

These three processes address the issue of "when" an individual accepts the influence of other person. The first one, *Compliance*, occurs when the individual wants to attain a favorable reaction from the other (like a child who adopts a behavior to be rewarded or to not be punished). The second one, *Identification*, occurs when the individual wants to establish or maintain a satisfying relationship to the other (like a husband who changes his attitudes to satisfy his wife expectations). The last one, *Internalization*, occurs in order to maintain the equivalent correspondence of actions and beliefs with his or her own value system (like a teenager who imitates other's attitudes for maximizing or confirming his or her own values).

Along with these processes, there is also a dimension orthogonal to the aspects of social influence, which is the distinction between two types of personal concerns. These concerns drive how an individual will react to social influence: either by instrumental concerns (e.g. assuring rewards or avoiding punishments) or by self-maintenance concerns (e.g. managing one's public image).

In large social systems, these elements of social influence are related to three distinct concepts: interests, relationships and identities [5]. These concepts are captured from tasks that all social elements (individuals, groups, organizations, societies) must perform as they negotiate their social environment:

- Protecting and promoting their interests: This task is related to the compliance process, where individuals and groups may influence each other to attain their own interests (or goals);
- Establishing and maintaining their relationships: This tasks is related to the identification process, where individuals and groups may establish the set of roles for their expectations;
- Affirming and expressing their identities: This tasks is related to the internalization, where individuals and groups share and exchange their values (or identities).

According to Kelman, "*In managing their interests, relationships and identities, individuals and groups must attend to the requirements of both social order and self-maintenance, and of ensuring the proper balance between them*" [5]. In the following section, we present how this is applied to the meso and individual layers by introducing the appropriate links between these two levels of abstraction.

3 Elements of the Meso Layer in the Social Influence

The meso level of abstraction, as described in Dignum et. al [4], refers to a intermediate level connecting and translating the elements found on the macro level to the micro level. It is composed of three types of components. The first one comes from the descriptive elements that was empirically validated in the domain, but are not in the focus of the simulation. They are regarded in the meso level as "law of nature", which the agents abide. The second component comes also from the macro level, but is in the focus of the simulation. This component is treated as a benchmark to which the agent behavior is compared. The third type of component tries to influence individual behaviors, through mechanisms that regulate their joint activities [8]. They include elements like norms, organizational structures and cultural backgrounds.

The next paragraphs present a correspondence between these elements (norms, structures, and cultural backgrounds) and the elements discussed in the previous section (interests, relationships, and identities). It establishes a means to express the social influence in a bi-directional way through a link between the micro and meso levels of abstraction.

3.1 From Interests, Relationships, and Identities to Rules, Roles, and Values

It is easy to identify that interests, relationships, and identities are inherent properties of both social and individual entities, as long as they refer to the link between them. As argued by Kelman, "*Individuals have interests, relationship and identities, which they pursue and express through the various groups and organizations with which they are affiliated. The groups and organizations – formed, essentially, to serve their members – in turn develop their own interests, relationships, and identities, which become personally important to the members and which the members are expected to support*" [9].

Interests reflect the goals that both individuals and groups have. In this aspect, groups establish a set of rules necessary to assure their member to attain the group's goals. Members are then influenced by these norms and rules through a mechanism of rewarding or punishment. For instance, the existence of a rule like "One should not drive faster than 100 km/h" reflects a specific goal of a social system, which tries to assure the behavior of their members in order to achieve it. On the other hand, individuals, when they cannot (or do not want to) leave a social system, may also influence it through their own internal rules and their respective rewarding or punishment strategies. Strikes are good examples of such strategies.

Rules, also described as norms in the meso level, are then a key concept which is present in both levels of abstraction. The social influence through norms is related to a compliance process, which represents adherence to them. In accepting influence via this process, members (and sometimes groups) assure themselves to have continuously rewards and approval (or also to avoid punishments).

Relationships reflect the roles assigned to members of a group and their respective expectations from them, as well as the role of the group itself to their members. Members are then influenced by the expectation from the others according to the role they are playing in the group. A teacher would, for instance, try to behave according to the expectation of the students (e.g. to give a good lecture). Groups have also their roles for their members. This means that individuals belong to a group only when there is a gain for them, i.e. when the group is playing its role as expected. Following the same example, a student may leave a college if the lectures are not given as expected by him or her.

Roles are then another key concept from the meso level that it is present at the individual level, and reflect the structure of the group, organization or society. The social influence through roles is related to an identification process. In accepting the influence via this process members and groups are meeting the expectations of their roles, thus maintaining their desired relationship to the group or to the members, as well as their wish to fully accomplish their roles. It is important to notice that the influence here is not about changing the role of a member or a group, but to change their settings in order to better accomplish that role.

Finally, identities reflects the system of values that everyone possesses, from their past experiences or cultural backgrounds. Members are then influenced by values shared in their social contexts as a way to maximize or maintain their own value system. A foreigner may, for instance, incorporate a behavior present in a host country according to what he or she considers important (its own values), and the feeling of belonging to such a society would be manifested by the set of values that s/he share with the society. Group identity is also a product from the values shared by their members, even if it can be considered as being almost independent from the individuals. This notion of collective identity is intrinsically related to the culture of the social system.

Value system is another key concept present in both meso and individual levels. The social influence through value system is related to an internalization process. In accepting the influence via this process members assure the maintenance or maximization of the equivalence of actions and beliefs with his or her own value system.

3.2 Linking the Meso and Micro Layers

The three processes of social influence proposed by Kelman suggest three different ways in which individuals and social systems are integrated: by adherence to its rules (or norms), by the involvement in its roles, and by the sharing of its values. The way in which rules and roles are exchanged between groups and individuals are, however, different. But in both cases, values serve as guides for the other elements.

For the groups, the rules for the individuals are a way to constraint individual behavior, while for the individuals, their rules are a way to assure that their roles (established by the group) does not deviate them from their values. For instance, let us consider a working environment where a rule forbidding couples to work together exists, and that there is couple whose group roles imply in working together. What should the couple do? Their personal values (to lie or to not lie, how much they are attached to each other, how much they are committed to the work etc.) will drive their standard action patterns (i.e. personal rules) toward the group, saying if they obey the rules (breaking the relationship), do not follow the rules (lying about the relationship), leave the group, or decide to try to change the rules of the group. Indeed, one can see the individual rules as his/her principles, ethic code or moral, which set what one should or should not do. They are based on the personal values.

For groups, the roles of the individuals are a way to structure and coordinate actions of their members, while for the individuals, the role of a group is to help them to maximize their personal values. Thus, the same kind of decision may happen when a role is assigned to an individual who does not see the role related to anything s/he considers important. Even worst, the role may be against some personal values. Let us consider, for instance, the obligation of an individual against a specific war to serve in the Army (it is an obligatory role in some societies). Although the process of Identification in a higher level of abstraction (National identity), if the role in such a group (Army) does not contribute to the personal values of the individual, why s/he would want to belong to the group (Army)? Some enforcement rules (punishments) may try to change this picture, but it is the values which guide the link between group and individual.

The relationship between the goal of rules, roles, and values for groups or individuals is them enriched through the existence of different types of social attitudes. For instance, Dastani et. al [10] distinguish three basic types of role enactment: selfish enactment (the individual gives priority to its own goals), social enactment (the individual gives priority to its role's goals), and maximally social enactment (the individual ignores its own goals for the duration of the role enactment). Following this line, V. Dignum introduces another type: maximally selfish enactment (the individual ignores the role's goals) [11].

Figure 1 illustrates the links previously described showing an individual belonging to two different groups and the three processes of social influence between them. Rules, roles, and values connect the individual to the groups s/he belongs to.

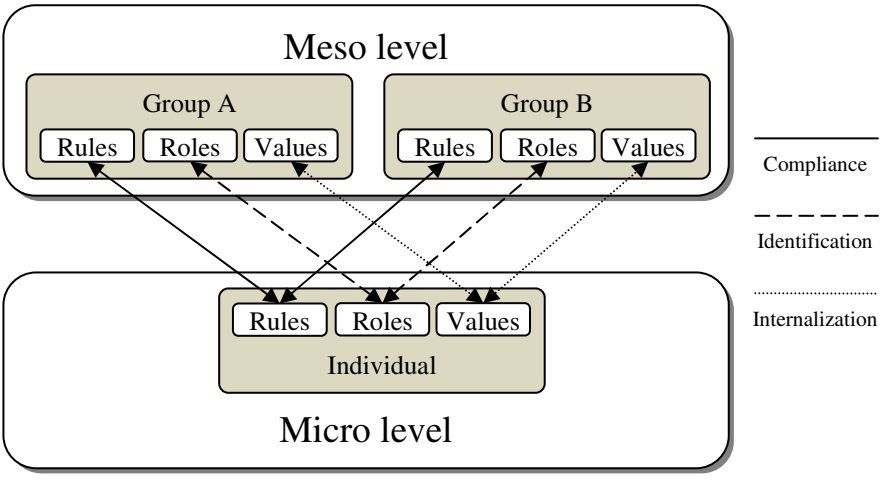


Fig. 1. Rules, roles, and values are the elements which interface the meso and micro levels of abstraction. Rules enforce the achievements of goals, roles define expected behaviors, and values constitute their identities. They reflect the three processes of social influence: compliance, identification and internalization.

Figure 2 illustrates (next page) how such a dynamics occurs explaining the vision of rules, roles and values from the perspectives of individuals and groups. This dynamics points out the social influence from the meso to the micro level, and *vice-versa*.

It is easy to observe that, although presented as three different concepts, rules, roles, and values are highly interdependent (as expressed in the legend of Figure 2). In order to structure how those dependencies should be treated internally by the individuals it is necessary to go deep on other aspects of human behavior. The following section explores some concepts of the individual behavior that we consider essential for configuring the whole picture of social influence: rationality, emotions, and personality.

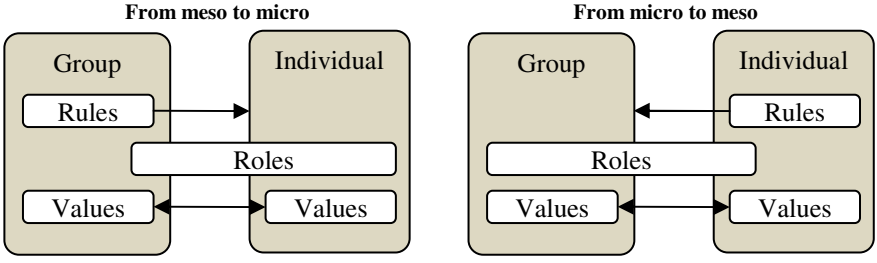


Fig. 2. The direction of social influence according to the elements involved. One can read the left-side block of the diagram as the group saying "*These are my rules that you must follow when playing your roles to me in order to bring or maximize our shared values*". The right-side block can be read as the individual saying "*These are my rules that I will follow as a member of group playing these roles to me in order to bring or maximize some shared values*".

4 The Behavior of Individuals

In the last years, several studies have been conducted in order to model human behavior..This includes physiological aspects (stress, fatigue,…), emotions (happiness, disappointment, …), behavioral preferences (personality) and socials (reputation, trust,…). Each of these domains regards human being from a particular perspective. Taken independently, they help to understand some specific aspects of human behavior. Taken in an integrative way, the aforementioned studies help to construct the whole panorama of the human being.

However, to integrate many different theories and/or models into a coherent architecture is a hard task with several potential validation issues (that is, to join the pieces of a puzzle does not mean to solve it). As any complex system, the resulting behavior from the interaction of its elements is very sensible to the way in which each element and the respective interactions are modeled. The complexity, already present in the individuals, still increases when they interact together. Indeed, the increasing number of agents in a simulation makes difficult to validate the model, assuring the expected macro behavior from an adequate internal representation.

Although such a difficulty as well as the practical issue of integrating different theories, we understand that to simulate real human social behavior it is necessary to incorporate different aspects of the human being, i.e. it is important to integrate the concepts from different areas and also to study the relationship between them. Following this line, prior to addressing the potential validation issues outlined here, it is firstly necessary to address how to integrate the several facets of human behavior, as well as to conceive the required links between them and the social framework presented in the previous sections. The elements that can be clearly identified as playing part in the social dimension are: the decision making process, the emotional component and the individual personality. The following paragraphs present rationales behind the choices made for modeling each of those dimensions.

4.1 Decision Making

Agents representing individuals within a social simulation should be able to reason about their surrounding environment, which encloses the social context where they

are embedded. Several cognition architectures have already been proposed for such a purpose. Each one has its own strengths and weaknesses, according to where they are applied.

Cognitive models, such as ACT-R [12] and SOAR [13] aim at understanding how people organize knowledge and produce intelligent behavior based on numerous facts derived from psychology experiments, employing quantitative measures. However, these models lack realism since they do not incorporate demographics, personality differences, cognitive style, situational and emotive variables, group dynamics and culture. On the other hand, neurological oriented models that mimic the brain, such as neural networks, lack transparency to link observed behavior to the implementation. Realistic agent models should combine the characteristics of the different types.

The model of the human mind CLARION [14] aims to explore the interaction of implicit and explicit cognition, emphasizing bottom-up learning (i.e., learning that involves acquiring first implicit knowledge and then acquiring explicit knowledge on its basis). CLARION's goal is to form a (generic) cognitive architecture that captures a variety of cognitive processes in a unified way and thus to provide unified explanations of a wide range of data. The BDI model is also a generic cognitive architecture [15]. It has formal logic-grounded semantics and introduces well established concepts from the theory of intentionality [16].

The CLARION model and the BDI model are both excellent candidates for the extension as aimed for in this paper. However, although the fact that BDI requires extensive computational resources, it provides clear clues where the concepts presented in the previous sections can be applied. The intentional paradigm behind the BDI model fits well for modeling the agent interests, represented by its desires, and their matching with the group interests, represented as a commitment to the goals established by the agent's roles.

4.2 Emotions

Emotions can indeed deeply influence the interactions among the members of a group, by stimulating or inhibiting behavior and, as consequence, influencing in the behavior of the whole system. The inclusion of the emotional component in the agent architecture helps to set up the internal consequences of individual choices regarding the rules, roles and values. For instance, the emotional reaction when people find themselves deviating from standards in the domain of responsibility may be: 1) social fear, when they deviate from external social rules or norms; 2) guilt, when they deviate from role expectations; or 3) regret, when they deviated from social values [7].

Emotion is, however, a subjective concept, which means that several authors have already presented their own vision. It is not our purpose to explore all existing work in the area, but to focus on those approaches that integrate emotions into a cognitive cycle of perception – action – perception. The model of emotions proposed in [17], also known as OCC model, addresses such a need. They support the idea that emotions are a product resulting exclusively from our cognition, generated from our perception and our expectations. As a cognitive outcome, emotions have their origins in our perception and involve some sort of positive or negative reaction to what was perceived. Following this line, they consider emotions as valenced reactions to events

(e.g. pleased, displeased,…), agents (e.g. approving, disapproving,…), and objects (e.g. liking, disliking,…).

Some authors criticize such a theory for not including physiological aspects of emotions, like other theories (e.g [18]). However, the exclusively-cognitive approach proposed in the OCC model is enough for our purposes. Although we recognize the influence of physiological aspects to the emotions, the level of granularity for the model we want to address in this paper does not deal with issues ranging from social system to hormonal specificities at the individual level.

4.3 Personality

The use of personality models in agents in order to study the micro-meso relationship becomes essential as it facilitates the creation of realistic complex scenarios. It improves and reproduces in a more realistic way the autonomy of the agents. Indeed, autonomy is related to how the individuals behave and what make them to behave differently from each other, even when they face the same situation. Psychologists have tackled this issue for several decades through what they named Personality [19].

It is personality which enables individuals with the same roles, following the same constraints (rules), sharing the same values, and having (virtually) the same beliefs, to behave differently. It provides a clear mechanism of preferences for specific behavioral patterns which are independent from any other aspect (beliefs, values, etc.). More precisely, personality represents the structured and dynamic set of characteristics of an individual, normally acquired from the environment and personal experiences [20]. These theories can be divided into two categories: personality types and personality traits [21]. Personality traits theories express the human characteristics through quantification values (for instance, 0.9 represents a strong characteristic). Examples of this category are the Big Five model (also known as OCEAN or Five Factor model) [22] and the model proposed by Theodore Millon [23]. Personality type theories do not express characteristics through values, but rather through a set of category (or types) in which the individual belongs. An example is the MBTI model [24], proposed as an extension of the theory of personality preferences developed by Jung.

For our concerns, we are interested in representing personality in social simulations independently from the context where the simulation is running. Following this line, we need a model that can be more easily focused on the process rather than the contents. A personality type model fits well this requirement since it is not expressed as a continuous (like the personality traits). Thus, according to an individual's type, a different cognitive process, involving his/her emotions and decision making, can be performed. The MBTI model fits well the needs of our approach. Besides that, it has a solid background through the several decades of use in organizational behavior studies [21].

5 The Architecture of the Social Agent

The approach used to present the agent architecture is to show the overall picture of the agent cognition taking into account the different aspects mentioned in the previous sections. The architecture is then presented in a high-level of abstraction rather than concentrating in a particular and focused problem.

Our approach to support the micro-meso interaction as a process of social influence uses: 1) a decision making process, based on the BDI model; 2) an emotional component, based on the OCC model; and 3) a personality-based mechanism, based on the MBTI model. These elements must connect to a social component which set up the individual standards (rules), the roles that the agent plays (as well as the groups where it plays), and its personal values.

The Figure 3 provides a general overview about the different components in the proposed architecture. The personality component is introduced over the others, since it does not save or process information, but rather it establishes the way in which the other components do that. The following subsections explain how the dynamics of the architecture.

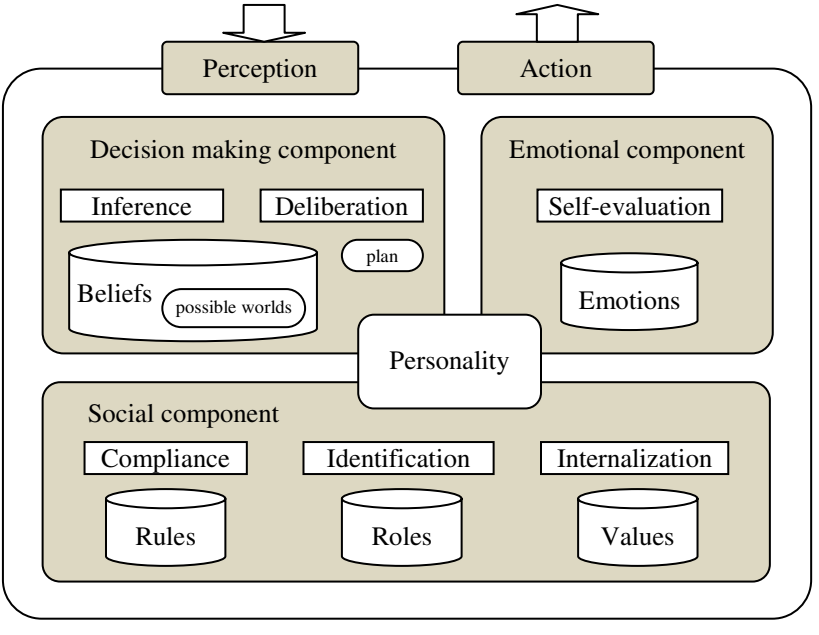


Fig. 3. The architecture of the proposed social agent. It incorporates four main components: The Decision Making, the Emotional, the Social and the Personality.

5.1 Components of the Architecture

The decision making component is responsible for defining the social action plan which the agent will be committed to, as well as for deciding when the agent should reevaluate its goals and, consequently, its plan. The desires (BDI element) in this model come from different components in the architecture: from its rules (not the group rules, since they represent constraints and not desires), from the objectives of the roles the agent in playing, from its personal values, from the plan it is committed to, and from the emotions. Those set of desires can be inconsistent and/or unachievable. Following a BDI mode of reasoning, the Deliberation element is responsible for filtering them into consistent and achievable goals, according to the agent beliefs. It

will then choose the goal (or goals) to commit, defining the agent intention, and a plan for it is setup.

An approach used in this model that differs from existing BDI architectures is the separation of the inference mechanism from the deliberation process. In other words, in the Deliberation element, there is no reasoning about the possible worlds. No conclusion about which world state can be achieved is drawn. The inference about possible worlds are made in advance by the Inference element and put available into the agent beliefs. The reason for such a separation is to make the possible worlds structure not only dependent on the rational deliberation (goals), but also dependent and based on the agent emotional focus, the processes of social influence and its personality type (they are explained further on). The structure representing the possible worlds (for instance, a tree) is then not based only on rational choices. Instead, the rational choice will search for the best option according to only what it possible to foresee (what is present in the tree). An agent representing an individual in a panic state (emotional state), for instance, will not develop further its possible worlds' analysis. Its behavior would be almost reactive. This approach fits several studies involving cognition and emotions pointing up that emotions set boundaries for our decision-making process (otherwise, we could think about the best option of an issue forever). It also fits to the Damasio concept of "somatic markers" [25].

The Figure 4 (next page) illustrates the inference process taking into account not only a rational choice but the emotions which can be triggered by possible world states. The figure shows the current world state (as the agent believes) as W_0 and a set of possible worlds deriving from it. In the example, the agent believes that a desired state can be achieved ($W_{1.2}$), but an undesired state can also occurs (W_2). It is worth to explain here that desired and undesired states are derived from the agent values and

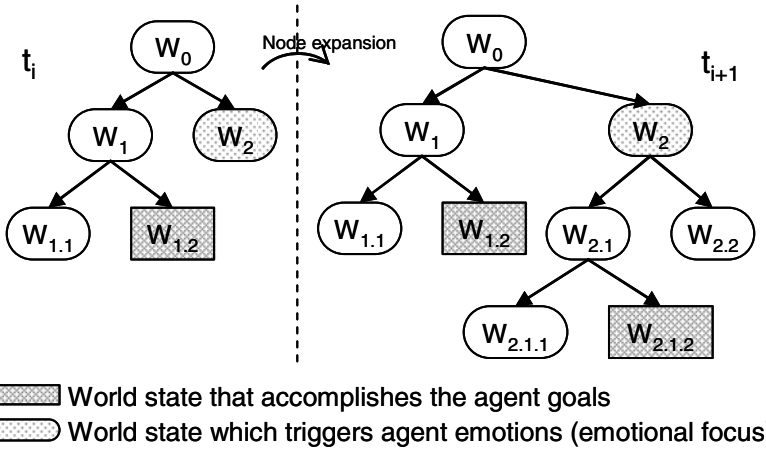


Fig. 4. An example of how the agent infers the possible worlds, taking into account the desired and undesired world states. States can trigger agent emotions which also drive the inference mechanism.

its social influence (rules and roles). As an undesired state, W_2 can produce emotions (explained in the next paragraph) and define the agent emotional focus. The emotional focus is then explored in a next iteration in order to develop new possible worlds deriving from it. In other words, the agent will not infer in any possible direction, but in virtue of its goals and affective states.

The emotional component is responsible for managing how the represented emotions in the agent affect the other components. As mentioned in the previous paragraph, emotions can influence the decision making by restraining or expanding the vision of possible worlds. The latter can also influence the current emotional state of the agent. For instance, if a not desired world can be foresee in the possible worlds (W_2), the emotion "fear" is produced by the emotional component.

Following the OCC cognitive structure of emotions [17], agent emotions are represented as a tree of valences. In our model, each tree node has an intensity value, which represents how deeply the agent is affected by the perceived current world or by the possible future worlds. The emotion intensity of a world state is based on the elements from the social component, i.e. the agent values related to such a state, the roles the agent is playing, and its personal rules. The self-evaluation box presented in Figure 3 represents this process of updating the intensity values of the agent's emotions. The element which has produced the most intensive emotion in the agent becomes its emotional focus, which will in return influence the possible worlds' vision (as mentioned in the previous paragraph).

The social component is responsible for interacting with the outer social world through the three processes of social influence proposed by Kelman. Its interaction with the other components has already been briefly outlined. Although the three processes of social influence can be defined into the decision making component (they are part of a decision mechanism), we choose to define them as separated processes in order to explicitly identify them, being easier to adapt to different social contexts without having to redefine the deliberation mechanism. This separation also helps the management of the possible worlds' structure, since the compliance, the identification, and the internalization would alter it only when there are perceived changes in the rules, roles and values of the groups, respectively.

Finally, the role of the personality in this architecture is to define how some processes are performed. The configuration should reflect the four dichotomies presented in the MBTI model. For instance, according to the type of the agent in the Sensing-Intuition dimension, the way in which the possible world's structure is expanded should be different. A sensing agent would expand the tree in depth, trying to figure out the real and concrete consequences of its actions (they tend to be pragmatic), while an intuition agent would expand the tree in breadth, trying to figure out all the possible outcomes (they tend to want a whole overview of possibilities). Another example is how the deliberation valuates a world state in the tree in order to choose the best option. Thinking agents would give more importance to world states that achieve the goals of its roles without deviating from its principles (rules), while Feeling agents would valorize the world states which satisfy its personal and its groups' values.

5.2 The Cognitive Process

This subsection provides an overview about how information is transformed from a perception stimulus to an action to be performed. Although there is no flux of information in Figure 3, there are several data dependencies between the components of the architecture. The choice of not explicitly represent them in the figure is to avoid surcharge of information, which could make it harder to understand. The following paragraphs present the process in a descriptive way, and after as an algorithm.

In short, the perception component receives stimuli from the outer world and updates the beliefs according to the agent personality. Indeed, the personality provides different ways to interpret a stimulus, which means that different agents facing the same situation may generate different beliefs. For instance, individuals with a *Thinking* personality type "*are direct to the extent of seeming insensitive to others*" (with *Feeling* type) [21].

After the beliefs have been update, the social component, through the processes of compliance, identification, and internalization, evaluates how this perceived world affects the agent's rules, roles, and values, respectively. This is, in fact, an evaluation about how the changes in the outer world impacts on the inner world, and it can be made by comparing the rules, roles, and values from both. Let us consider, for instance, the previously mentioned example of the individual against a war who has just invited to serve the Army. It is in this cognitive step that an agent representing this individual will contrast its values, roles, and rules to the society ones, and elaborate the impact of this new perceived outer world in its inner world. This impact can be considered as the individual level of deviation from social standards.

If the inner world is affected, for instance some personal values are infringed, emotions are raised from the current world state (W_0). The self-evaluation process updates then the emotions, following the OCC-based type of reactions (to events, agents, or objects), and an emotional focus is defined. In the previous example, *dissatisfaction* and *disappointment* emotions might arise from the impact caused by the invitation to serve the Army. If such emotions are those which contribute the most for the current emotional state of the agent (the sum of all its emotions), this event becomes then the agent's emotional focus.

The next step is performed by the decision making component, which expands or restraints the possible worlds' structure base on the plan the agent is currently committed (intention), the emotional focus, the level of deviation from social standards, as well as its personality type (as previously described) and its beliefs. Following the same example, the agent will expand the possible world's reasoning about the consequences and possibilities related to the Army invitation, which is its emotional focus. After a fixed number of expansions in the structure, the emotions are reevaluated. Emotions like *fear* or *anxiety* may then appear as consequence of new possible worlds, and new inferences are made based on the new emotional focus.

The cycle may continue according to the personality type (*Sensing* persons will expand the possible worlds searching for concrete actions, while *Intuition* persons will expand to get an overview of all possibilities involved) and level of emotions raised (e.g. *panic*). After this, the deliberation process will filter the desires (spread in the other components) into consistent and achievable goals, reevaluate the current plan, and may trace a new plan to commit with. This plan can express a way of accepting

the social influence (through compliance, identification, or internalization), as well as a way to not accept the social influence, trying so to change the social environment. In both cases, if there is a new plan, it is send to the action component, which will replace the last one. The following code specifies in a high-level of abstraction the description above.

Pseudo-code of the reasoning process of the social agent (in a high-level of abstraction), where: T is the personality type, Ru, Ro, and Va are the rules, roles, and values, P is the plan the agent is currently committed and A is the set of possible actions.

```
01: while alive
02:   S = get stimuli (outer world)
03:   B = update beliefs (S, T)
04:   Q = evaluate impact (B, Ru, Ro, Va)
05:   E = update emotions (B, Q, T)
06:   repeat
07:     W = review possible worlds (B, P, E, Q, T, A)
08:     E = update emotions (B, Q, T)
09:   until not(panic(E)) or has to react (B, T)
10:   if empty(P) or not(achievable(P,W)) or reconsider(P)
10:     D = gather desires (Ru, Ro, Va, E, P, T)
11:     I = define intention (B, D, I, T)
12:     P = generate plan (W, I, A, T)
13:   execute action (P, A)
```

6 Final Remarks

In this paper we propose the use of three processes of social influence, namely *Compliance*, *Identification*, and *Internalization*, into an agent-based social simulation. Those processes are presented into an organizational approach (rules, roles, and values) where explicit links between individuals and the society are provided. In order to compose the general framework where rules, roles, and values influence and are influenced by individuals, other concepts related to human behavior were introduced and an agent architecture was conceived. We presented the main components of such an architecture and described its cognitive reasoning process.

The work described here is still in its early phase. The general concepts and abstractions were proposed and an architecture in a high-level of abstraction was designed. Connections from meso layer toward the micro were stressed and some hooks for the opposite direction were also stated. The latter is currently being developed based on the idea that, for instance, norms emerge through their *immergence* in the agents' minds [26]. The present work includes norms in the agent cognition process, which can facilitate such a process.

Since the presented work employs an unusual approach (it embraces several aspects of social and individual at once), few works could be related in depth. Some agent architectures addressing the social component within the agent cognition do not address emotions and personality (e.g. B-DOING [27] and EMIL-A [26]). Others, which link social aspects of the emotions, do not integrate an organizational or normative approach (e.g. [28]). The PMFServ architecture [29] uses however the same approach, embracing a large number of concepts. PMFServ exploits however a quantitative way of modeling,

where the agent decision-making is based on weighted sum of values and do not look further then the next world state. The architecture presented here is target to qualitative models and is the agent is able to reason about plans.

Although the lack of simulation results showing the impact of social influence over the individuals and *vice-versa*, the architecture was grounded in well established concepts. We foresee that the proposed model can be applied in several social contexts, ranging from the study of group formation to the study of emergence of insurgent movements, including its respective causes and consequences from and to the society.

The next steps are to provide formal specifications regarding the dependencies between meso and micro elements, as well as the dependencies inside the agent architecture, and to implement them in a simulation platform. We envisage implementing the agents in an extension of the 2APL language [30], since both are founded on the BDI model of reasoning.

Acknowledgments. This work is partially supported by the Brazilian National Research Council (CNPq).

References

1. Halpin, B.: Simulation in Sociology: A review of the literature. In: Proceedings of the Workshop on Potential of the computer simulation for the social sciences, University of Surrey (1998)
2. Conte, R., Gilbert, N., Sichman, J.S.: MAS and Social Simulation: A Suitable Commitment. In: Sichman, J.S., Conte, R., Gilbert, N. (eds.) MABS 1998. LNCS, vol. 1534, pp. 1–9. Springer, Heidelberg (1998)
3. Staller, A., Petta, P.: Introducing Emotions into the Computational Study of Social Norms: A First Evaluation. Journal of Artificial Societies and Social Simulation 4 (2001), http://jasss.soc.surrey.ac.uk/4/1/2.html
4. Dignum, F., Dignum, V., Catholijn, J.: Towards Agents for Policy Making. In: Proceedings of the Multiagent-based Simulation, MABS 2008, Estoril, Portugal (2008)
5. Kelman, H.C.: Interests, Relationships, Identities: Three Central Issues for Individuals and Groups in Negotiating Their Social Environment. Annual Review of Psychology 57, 1–26 (2006)
6. Kelman, H.C.: Attitude change as a function of response restriction. Human Relations 6, 185–214 (1953)
7. Kelman, H.C.: Social influence and linkages between the individual and the social system: Further thoughts on the processes of compliance, identification, and internalization. In: Tedeschi, J. (ed.) Perspectives on social power, pp. 125–171 (1974)
8. Feltovich, P.J., Bradshaw, J.M., Clancey, W.J., Johnson, M.: Toward an ontology of regulation: Socially-based support for coordination in human and machine joint activity. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS, vol. 4457, pp. 175–192. Springer, Heidelberg (2007)
9. Kelman, H.C.: Continuity and change: My life as a social psychologist. In: Eagly, A.H., Baron, R.M., Hamilton, V.L. (eds.) The social psychology of group identity and social conflict: Theory, application, and practice, pp. 233–275. American Psychological Association Press (2004)

10. Dastani, M., Dignum, V., Dignum, F.: Role-assignment in open agent societies. In: Proceedings of the 2nd international joint conference on Autonomous agents and multiagent systems, AAMAS 2003, Melbourne, Australia, pp. 489–496. ACM Press, New York (2003)
11. Dignum, V.: A model for organizational interaction: based on agents, founded in Logic. PhD Thesis. Utrecht University, The Netherlands (2003)
12. Anderson, J.R.: Rules of the Mind. Lawrence Erlbaum Associates, Mahwah (1993)
13. Laird, J.E., Newell, A., Rosenbloom, P.S.: SOAR: an architecture for general intelligence. Artificial Intelligence 33(1), 1–64 (1987)
14. Sun, R.: The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In: Sun, R. (ed.) Cognition and Multi-Agent Interaction, pp. 79–99. Cambridge University Press, New York (2006)
15. Rao, A., Georgeff, M.: BDI-agents: from theory to practice. In: Proceedings of the 1st International Conference on Multiagent Systems, pp. 312–319 (1995)
16. Bratman, M.: Intention, Plans, and Practical Reason. Harvard University Press (1987)
17. Ortony, A., Clore, G.L., Collins, A.: The Cognitive Structure of Emotions. Cambridge University Press, Cambridge (1990)
18. McNaughton, N.: Biology and Emotion. Cambridge University Press, Cambridge (1989)
19. Revelle, W.: Personality Processes. Annual Review of Psychology 1, 295–328 (1995)
20. Funder, D.C.: The Personality Puzzle, 4th edn. W. W. Norton Press (2006)
21. Zeisset, C.: The Art of Dialogue: Exploring Personality Differences for More Effective Communication. Center for Applications of Psychological Type Press (2006)
22. Digman, J.: Personality Structure: Emergence of the Five-Factor Model. Annual Review of Psychology 41, 417–440 (1990)
23. Millon, T.: Reflections on Psychosynergy: A Model for Integrating Science, Theory, Classification, Assessment, and Therapy. Journal of Personality Assessment 72, 437–456 (1999)
24. Myers, I.B., McCaulley, M.H., Quenk, N.L., Hammer, A.L.: MBTI Manual: A guide to the development and use of the Myers Briggs type indicator. Consulting Psych. (1998)
25. Damasio, A.: Descartes Error Emotion, Reason, and the Human Brain. G P Putnam (1994)
26. Andrighetto, G., Campenni, M., Conte, R., Paolucci, M.: On the Immergence of Norms: A Normative Agent Architecture. In: AAAI Fall Symposium. Emergent Agents and Socialities: Social and Organizational Aspects of Intelligence, pp. 11–18 (2007)
27. Dignum, F., Kinny, D., Sonenberg, E.: From Desires, Obligations and norms to Goals. Cognitive Science Quarterly Journal 2, 407–430 (2002)
28. Jiang, H., Vidal, J.M., Huhns, M.N.: EBDI: an architecture for emotional agents. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, AAMAS 2007, Honolulu, Hawaii, pp. 38–40. ACM Press, New York (2007)
29. Silverman, B., Johns, M., Cornwell, J., O'Brien, K.: Human Behavior Models for Agents in Simulators and Games. Presence: Teleoperators and Virtual Environment 15, 139–162 (2006)
30. Dastani, M.: 2APL: a practical agent programming language. Autonomous Agents and Multi-Agent Systems 16, 214–248 (2008)

Author Index