Adding Agent-Oriented Concepts Derived from Gaia to Agent OPEN

Brian Henderson-Sellers, John Debenham, and Q.-N.N. Tran

University of Technology, Sydney, NSW 2007 Australia
{brian,debenham}@it.uts.edu.au, numitran@yahoo.com

Abstract. Agent OPEN offers extensions of an object-oriented methodological framework to support agent-oriented software developments. However, to date, it is incomplete. Here, we extend the Agent OPEN repository of process components to include contributions from the Gaia agent-oriented methodology. We have identified one new Task, together with six new subtasks for some pre-existing Tasks. Three extra Techniques and five new Work Products were identified and recommended to be added in order to support the Gaia approach for agent-oriented software development.

1 Introduction

In a distributed computing environment, agents are increasingly being perceived as of high potential applicability. Applying agent technology successfully in an industrial setting requires the application of an appropriate methodology "tailored" to local demands. Ensuring the methodology meets local requirements (both technical and human) can be facilitated by the use of method engineering (ME) (Kumar and Welke, 1992; Brinkkemper, 1996) or, more specifically, situated method engineering (SME) (Ter Hofstede and Verhoef, 1997). A combination of SME and agent technology is the focus of this paper.

ME and SME provide a rational approach to the construction, either fully or partially, of methods (a.k.a. methodologies) from method fragments (often called method chunks (Rolland and Prakash, 1996) or process components[1] (e.g. Firesmith and Henderson-Sellers, 2002)), typically stored in a repository. The method itself is constructed by selection of appropriate method fragments followed by their configuration in such a way as to satisfy the requirements for the method (Ralyté and Rolland, 2001) and create a meaningful overall method (Brinkkemper *et al.,* 1998).

In the object-oriented context, SME has been realized through the OPEN Process Framework (OPF) (Firesmith and Henderson-Sellers, 2002). The OPF is underpinned by a full lifecycle metamodel and contains rules for both creating and using the repository-stored process components. As part of this agent-oriented (AO) methodology-focussed project, the OPF has had some initial enhancements to include support

[1] We take the view that a methodology is a combination of a process and a set of products (e.g. Rolland *et al.*, 1999). Our focus here is on the process portion of a methodology and thus the words "method", "methodology" and "process" can be taken as synonyms in our discussion.

A. Persson and J. Stirna (Eds.): CAiSE 2004, LNCS 3084, pp. 98–111, 2004.

for agent concepts (Debenham and Henderson-Sellers, 2003). Thus an additional number of tasks, techniques, work products and roles has been added to the OPF repository. This paper reports on the next stage of the research: to ensure that the set of process components created in the OPF repository specifically to support agency concepts is as complete as possible. To do this, we analyze the Gaia AO methodology (Wooldridge *et al.*, 1999, 2000; Zambonelli *et al.*, 2003) in order to see what must be added to the OPF repository so that this particular AO methodology (i.e. Gaia) can be (re)created by instantiation from the elements in the OPF.

In Section 2, we outline the OPEN Process Framework as used in the context of SME and, in Section 3, we describe the basics of the Gaia methodology. In Section 4 we describe the elements of Gaia not currently supported in the OPF and which we therefore propose for addition to the OPF repository.

2 An Overview of the OPEN Process Framework

The OPEN (Object-oriented Process, Environment, and Notation) Process Framework (OPF) (Firesmith and Henderson-Sellers, 2002) combines a process metamodel and a repository of process components (Figure 1). Elements from the repository are selected and put together to form a specific process or situational method.

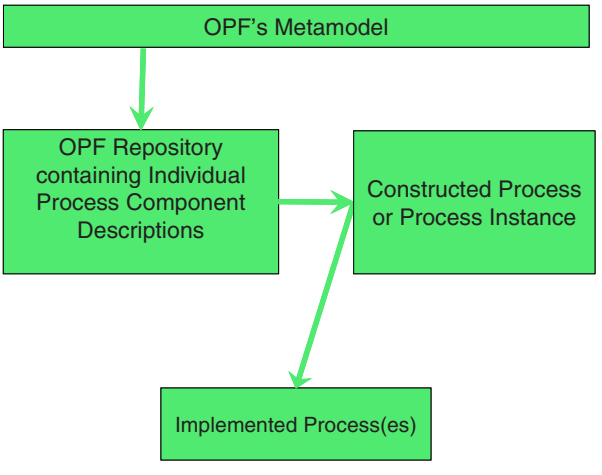

Fig. 1. The OPF defines a framework consisting of a metamodel and a repository of process components.

Process construction is accomplished using a set of OPF guidelines and rules. A major element is the use of deontic matrices, which allocate a possibility value to pairs of process elements such as Activity/Task or Producer/Work Product. Deontic values have one of five values ranging from mandatory through optional to forbidden. This gives a high degree of flexibility to the process engineer, perhaps assisted by an automated tool, who can allocate appropriate deontic values to any specific pair of process components depending upon the context i.e. the specific project, skills set of the development team etc.

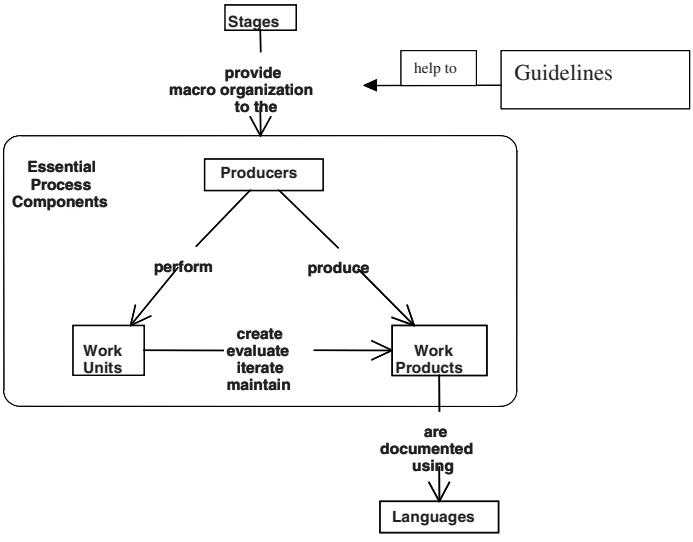

Fig. 2. The five major metaclasses of the OPF's metamodel (after Firesmith and Henderson-Sellers, 2002) © Addison-Wesley.

Underpinning the OPF framework is its full lifecycle metamodel that defines the following five main high level classes of process components (Figure 2):

Work Product: "A Work Product is anything of value that is produced during the development process" (Firesmith and Henderson-Sellers, 2002). Work Products are the result of producers (people) executing Work Units and are used either as input to other Work Units or delivered to a client. Typical examples are use case diagrams and role diagrams.

Producer: "A Producer is responsible for creating, evaluating, iterating and maintaining Work Products" (Firesmith and Henderson-Sellers, 2002). Producers play various roles within the methodology and introduce the human element (although some producers may be other software or indeed hardware). A large number of Producer and Role instances (for example, the requirements engineer and the toolsmith) are described in the OPF repository although it is a volatile set in comparison with, say, the instances of the WorkUnit metaclass.

Work Unit: A Work Unit is defined as a functionally cohesive operation that is performed by a Producer. There are three major classes of Work Unit: Activity, Task and Technique.

- Activities describe, at a high level, what needs to be undertaken. The overall software development process is often configured by the process engineer/methodologist using half a dozen or so of these Activities so they are often (but not always) the first to be identified. A typical example is the Requirements Engineering (RE) Activity.
- Tasks also focus on "what" needs to be done rather than "how" it is do be done. Tasks can be readily tracked and project managed. They are typically allocated to a small team over a period of a few days. A typical example related to the RE Activity is "Elicit requirements".

- Techniques describe the mechanism by which a Task is undertaken. They describe the "how" as compared to the "what" of Activities and Tasks. A typical RE-related Technique is storyboarding.

Language: A Language is defined as a medium for documenting a Work Product; for example; English, UML.

Stage: A Stage is defined as an identified and managed duration within the process or a point in time at which some achievement is recognized. Stages may be Phases or Cycles – examples are the Build Phase and the Development Cycle, respectively.

Each of these metaclasses has many subclasses in the detailed metamodel (see Appendix G of Firesmith and Henderson-Sellers, 2002). From each of these subclasses, one or more process component instances are generated and stored in the OPF repository (Figure 1).

Initially, the OPF repository contained about 30 predefined instances of Activity, 160 instances of Task and 200 instances of Techniques (the three main kinds of Work Unit) as well as multiple instances of Role, Stage, Language etc. Some of these are orthogonal to all others in their group and some overlap. Consequently, during process construction both association and integration strategies (Ralyté and Rolland, 2001) are needed. For example, there are several Techniques in the repository for finding objects e.g. textual analysis, use cases simulations, CRC card techniques.

Finally, when used on a specific project in real time, this is known as a process instance or "implemented process" (Figure 1). A company-customized OPEN version is then "owned" by the organization, becoming their own internal standard, while retaining compatibility with the global OPEN user community.

Since its first publication in 1997, several additions have been made to the OPF repository to enhance its support for various new technologies including additions of relevance to agent technology (Debenham and Henderson-Sellers, 2003; Henderson-Sellers *et al.*, 2004a,b) . Here, we extend the OPF repository even further to offer additional support for agent orientation (AO) by extracting new process components from the Gaia AO methodology (Wooldridge *et al*., 2000; Zambonelli *et al.*, 2003).

3 Major Elements of Gaia

Gaia views the process of multi-agent system (MAS) development as a process of *organizational design*, where the MAS is modelled as an organized society with agents playing different roles. The methodology allows a developer to move systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly. It supports both macro (societal) and micro (agent) aspects of MAS design, and is also neutral to both application domain and agent architecture. The newest version of Gaia (Zambonelli *et al*., 2003) extends the original version (Wooldridge et al., 1999; Wooldridge et al., 2000) with various organizational abstractions, enabling it to be used for the design of open MAS (which was not achievable previously). The discussion in this paper accounts for the new tasks and models presented in the newest version as well as those in the original (1999/2000) publications.

3.1 Tasks Characterizing Gaia

There are a number of tasks described in the publications on Gaia which, together, permit its use for AO systems development. These are:

- *'**Identifying sub-organizations in the system**':* To promote modularity, an analyst should determine whether the target system contains multiple sub-organizations that co-exist as autonomous interacting MASs.
- *'**Modeling the environment**':* The MAS environment is modeled as a collection of abstract computational resources, each characterized by the types of actions the agents can perform on it.
- '***Identifying roles in the system***': Gaia models each role by its responsibilities and permissions. Responsibilities represent the role's functionality, and are divided into two types: : *liveness* and *safety*. Liveness responsibilities specify the states of affairs that an agent must bring about, while safety responsibilities are typically predicates, specifying the acceptable states of affairs that should be maintained across all states of execution.
- ***'Identifying inter-role interactions'***: Gaia defines inter-role interaction protocols in terms of the essential nature and purpose of the interactions, rather than the precise ordering of particular message exchanges. Specifically, each protocol definition consists of an interaction's *purpose*, *initiator*, *responder*, *inputs*, *outputs* and *processing* (which is simply a brief textual description of the initiator's processing during interaction).
- *'**Defining organizational rules**':* Organizational rules (liveness or safety) are responsibilities of the agent organization as a whole.
- *'**Choosing the organizational structure**':* The designer needs to choose an organizational structure that provides the most appropriate topology and control regime.
- ***'Identifying agent types and agent instances'**:* Gaia identifies agent types from roles, and agent instances from these types. Agent types are arranged in an Agent Type Hierarchy, which includes only aggregation relationships (if any) but no inheritance.
- *'**Specifying services of each agent**'*: A service is a single, coherent block of activity in which an agent will engage.
- *'**Specifying agent acquaintances**'*: This task involves identifying the communication links/pathways between agent types. It does not include defining what messages are sent or when messages are sent. The purpose of this task is simply to identify any potential communication bottlenecks and to ensure that the system is internally loosely-coupled.

3.2 Techniques Recommended by Gaia

As well as tasks, a number of specific techniques are recommended:

- ***For 'Identifying sub-organizations in the system'***: Gaia suggests considering multiple sub-organizations when there are portions of the target system that exhibit behaviour specifically oriented towards the achievement of a given sub-goal, that interact loosely with other portions of the system, or that require competencies not needed in other parts of the system.

- ***For 'Modeling the environment'***: GAIA does not commit to any specific modeling techniques for the specification of environmental resources.
- ***For 'Identifying roles for the system'***: The process of role identification roughly goes through two phases. Firstly, the key roles in the system are identified from the "basic skills" required by the organization to achieve its goals. However, Gaia does not provide techniques for identifying these skills, or for identifying roles from these skills. The output of this phase is a Preliminary Role Model which provides informal, unelaborated descriptions of the key roles in the system. This is then refined into a fully elaborated Role Model on the basis of the organizational structure. With regard to each role's responsibilities, Gaia does not offer any explicit techniques for the identification of responsibilities, except to suggest that liveness responsibilities may follow certain patterns, being modelled in terms of *activities* and *protocols*. With regard to a role's permissions, the developer is recommended to investigate the *types* and *limits* of the information resources that an agent accesses to carry out its roles.
- ***For 'Identifying inter-role interactions'*:** Gaia offers no specific technique to identify interactions between roles, except for stating that each protocol description should focus on the nature/purpose of the interaction and the involved parties. No precise ordering of message exchanges needs to be defined.
- ***For 'Defining organizational rules'***: Liveness organizational rules can be derived from liveness responsibilities of different roles, i.e. from the way different roles can play specific activities. Meanwhile, safety organizational rules can be related to safety responsibilities of different roles or to expressions of the environmental resources in different roles.
- ***For 'Choosing the organizational structure'***: Gaia suggests selecting an organizational structure that optimizes the organizational efficiency and simplicity (e.g. balanced workload, low coordination costs) that respects the organizational rules and that reflects the structure of the real-world organization..
- ***For 'Identifying agent types and instances'*:** Gaia suggests a general rule of one-to-one mapping between roles and agent types. However, it also recognizes the need for grouping closely related roles to the same agent type for the purpose of convenience and efficiency. Nevertheless, Gaia recommends considering a trade-off between the coherence of an agent type and the efficiency considerations. Gaia offers no techniques for the instantiation of agent types.
- ***For 'Specifying services of each agent'***: An agent's services can be derived from the list of *responsibilities* (both liveness and safety), *protocols* and *activities* of the roles that the agent encapsulates. In general, there will be at least one service associated with each protocol. Every activity identified in the agent role's responsibilities will correspond to a service, though not every service will correspond to an activity. The safety responsibility may also imply a service, defined in terms of inputs, outputs, pre- and post-conditions. The former two elements can be derived from protocol definitions in the Interaction Model, while the latter two can be revealed from the safety responsibilities of the role.
- ***For 'Specifying agent acquaintances'***: The communication pathways between agent types can be directly derived from Role, Interaction and Agent Models.

3.3 Work Products Advocated by Gaia

Gaia suggests the creation of seven specific work products during AO software development:

- ***Environment Model***: containing a list of resources, each associated with a symbolic name, types of actions that can be performed on it and possibly textual comments and descriptions. No specific notation is mandated.
- ***Organizational Structure Model***: The designer can either adopt a formal representation scheme or a more intuitive graphical notation for the organization structure model depending on the application. Graphically, roles can simply be represented as blocks, connected by arrows to represent organizational relationships (e.g. control, peer, dependency) as shown in Figure 3.

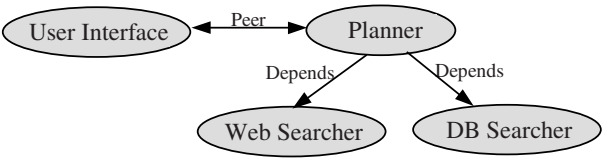

Fig. 3. Example of Gaia Organizational Structure Model.

- ***Role Model***: containing a textual Role Schema for each role (Figure 4). Role responsibilities are modelled in Gaia using Fusion notation.

Role Schema:	*name of role*
Description	*short English description of the role*
Protocols and Activities	*protocols and activities in which the role plays a part*
Permissions	*"rights" associated with the role*
Responsibilities	
Liveness	*liveness responsibilities*
Safety	*safety responsibilities*

Fig. 4. Gaia's Role Model (figure 3 from Wooldridge *et al.,* 2000. With kind permission of Kluwer Academic Publishers).

- ***Inter-role Interaction Model***: containing a list of definitions of inter-role interaction protocols. Each protocol definition consists of a purpose, initiator, responder, inputs, outputs and processing description.

  In the exemplar protocol definition (Figure 5), the *SearchForAnswer* protocol is initiated by the role *Planner* and involves the role *WebSearcher*. Input to the protocol is the sub-query derived from the user query.The protocol involves the *Planner* forwarding the sub-query to the *WebSearcher* to process, and results in the answer being returned by the *WebSearcher*.

SearchForAnswer		
Planner	WebSearcher	sub-query
Description: Planner forwards a sub-query to WebSearcher to process and return a reply		answer

Fig. 5. Example of Gaia Interaction Model.

- ***Agent Model***: Gaia suggests a simple diagram for the Agent Model which shows, for each agent type, the roles that map to it (Figure 6). The bottom leaf nodes correspond to roles, while other nodes represent agent types. The arrows denote mappings from roles to agent types. Agent instantiation is documented by Fusion-based annotations below the agent types (e.g. an annotation '+' means that there will be one or more agents of the type at run-time)..

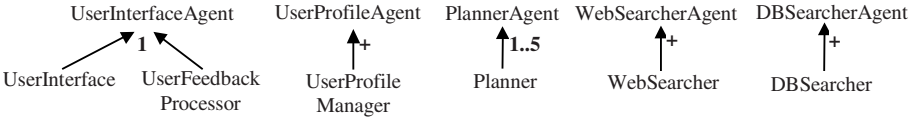

Fig. 6. Example of Gaia Agent Model.

UML Class Diagrams can be adapted for the modelling of agent types and their relationships. Major considerations are whether the association relationships between agents have a different meaning from those between objects (e.g. inter-agent associations represent communication pathways). UML also offers instanceOf relationships that can be used to model agent class instantation.

- ***Service Model***: Gaia suggests a tabular template for modelling services (Figure 6).

Service	*Inputs*	*Outputs*	*Pre-condition*	*Post-condition*
Accept user query	userQuery	awaitMessage	true	true
Create user profile	userDetails	custID, custPassword	userStatus = nil	userStatus = member

Fig. 7. Example of Gaia Service Model (for User Agent).

Currently UML does not offer a separate diagram for service modelling. However, in the AO version of the UML Class Diagram, services can be included as an internal component of an agent class (just as methods are a component of an object class).

- ***Agent Acquaintance Model***: Eliminated in Version 2 of Gaia (Zambonelli *et al.*, 2003), this information can probably best be depicted with a UML or AUML Interaction Diagram.

3.4 Stages Used in Gaia

In order to support development across the whole system lifecycle, we identify the stages (cycles and phases) advocated in Gaia:

Cycle: Gaia is iterative within each phase as well as across all phases.. This description fits the '*Iterative, Incremental, Parallel Life Cycle*' model of OPEN.

Phases: Gaia covers Analysis and Design (particularly from the statement of requirements to a design that is sufficiently detailed that it can be implemented directly). In the context of OPEN, Gaia supports '*Initiation*' and '*Construction*'.

3.5 Languages

Gaia's notation mainly comes from Fusion (Coleman *et al.,* 1994). Considering the contents of Gaia's models, UML (with necessary adaptations/extensions) can be employed as an efficient modelling language. In contrast, Gaia's design models can be implemented in any programming language.

4 Adding Support to the OPF Derived from Gaia

In this section, we outline the various Tasks, Techniques and Work Products that are proposed here as additions and modifications to the OPEN repository in order to incorporate agency concerns as identified in Gaia.

In total, only one new Task is identified, six new subtasks, three Techniques and five new Work Products are identified. These are all described in the following subsections.

4.1 Existing Support and Mapping between OPF and Gaia

4.1.1 Tasks

Roles are of high importance in Gaia (and several other AO methodologies). However, the concept of "role" has not been well supported in the object-oriented literature. Although OPEN supports role modelling more than many OO methods via its use of the "CIRT" (standing for Class, Instance, Role or Type), it does not consider "role" as a first-class concept in system analysis and design. It also does not promote the organisation-driven approach in the development of systems. Roles are covered in OPEN through the 'Construct the Object Model' Task and 'Identify CIRTs' Task. Therefore a new Task: Model agents' roles was introduced by Debenham and Henderson-Sellers (2003). Using this and the existing OPF Task: 'Map roles on to classes' adequate support is offered for both the identification of agents and their responsibilities and permissions. However, two new subtasks need to be made explicit for Task: 'Model agents' roles' (Section 4.2).

The specification of other organizational abstractions, including the MAS environment, sub-organizations, organizational rules and organizational structure, is addressed to some extent by OPF requirement engineering Tasks: 'Context modeling' and 'Analyze customer organization'. Debenham and Henderson-Sellers (2003) also introduced Tasks 'Model the agent's environment' and 'Identify system organization'

that deal with MAS environmental and organizational design issues. However, due to their significance, four new subtasks need to be made explicit: subtask 'Model environmental resources' for Task 'Model the agent's environment', and subtasks 'Identify sub-organizations', 'Defining organizational rules', and 'Defining organizational structures' for Task 'Identify system organization'.

Agent interactions can be described by a variety of existing Tasks in the OPF, particularly augmented by the Agent OPEN Task: 'Determine agent interaction protocol' and Task: 'Determine agent communication protocol'. These offer similar and adequate support for the Gaian Tasks of 'Identifying inter-role interactions' and 'Specifying agent acquaintances'. Services are identified and specified using tasks (and techniques) similar to those for classes in OO developments but extended to include agents as part of the OPF CIRT.

Each of the tasks above is related to the new Task: 'Construct the agent model' (see Section 4.2) which also offers support for the Gaia task of 'Identifying agent types and agent instances'.

4.1.2 Techniques

For Gaia's 'Identifying roles for the system' techniques, OPEN offers the Technique: 'Role Modelling', which covers various aspects of role modelling, although is weak on guidance for the identification of roles.

Support for responsibility and permissions identification and modelling is found in OPEN's original Technique: 'Responsibility identification' and supplemented by the use of various user requirements Techniques such as 'CRC card modelling' and 'Scenario development'.

Techniques for inter-role interactions are found in the 'Collaboration Analysis' Technique of OPEN (Henderson-Sellers *et al.*, 1998) and the 'Reactive reasoning (ECA) rules' of Agent OPEN (Debenham and Henderson-Sellers, 2003).

OPEN offers various techniques for OO class identification/modelling (such as 'Abstract Class Identification' and 'Class Naming'). These techniques are useful but need to be oriented more towards agent classes, for example, taking into account the major differences between OO classes and agent classes - agent classes are generally more coarse-grained than OO classes (Wooldridge *et al.*, 2000) and thus the OPF Technique: 'Granularity' should be extended to account for this difference. The OPF Technique: 'Relationship modelling' may also be useful although Gaia appears to eschew the generalization and association relationships common in OO modelling languages, such as the UML (OMG, 2001).

To support Gaia's 'Specifying services of each agent', OPEN offers Technique: 'Service Identification' that can be applied (with necessary adaptations) to the specification of agents' services. Furthermore, the OPF Technique: 'Collaborations analysis' serves well to support the Gaian task of 'Specifying agent acquaintances'. Although Gaia's scope does not include the specification of communication messages between agent classes, Gaia does state that its design models should be further realized by traditional design techniques. In this case, the OPF Technique: 'Interaction modelling' provides a useful basis for modelling the communications between agent classes.

No techniques exist in the OPF to directly support the definition and modeling of system environment's resources, organizational rules and organizational structures. Therefore, these need to be added.

4.1.3 Work Products

While role models are supported within the OPF, say using UML, there is no document to capture the requirements for an individual role in terms of responsibilities and protocols. Thus the Gaia Role Schema needs to be made available through the OPF Repository (Section 4.3).

Interaction protocols in OPEN are modelled via UML Sequence Diagrams and Collaboration Diagrams. These diagrams can be readily adapted for the modelling of AO interactions by means of an Agent Protocol diagram as found in Prometheus (Padgham and Winikoff, 2002). Consequently, the Gaian template for an Interaction Model as shown in Figure 4 is probably unnecessary; however, it is added to the OPF repository for completeness. There it is renamed Protocol Schema in order to avoid confusion with UML interaction diagrams.

While the Gaia agent model is readily subsumed by UML diagrams, it is worth adding its service model as a new OPF Work Product component: the Service table.

Both the Environment Model and Organizational Structure Model can be represented as UML class diagrams. However, they should be explicitly specified as new products in the OPF repository.

4.2 New Tasks

One new Task and six subtasks are identified from Gaia tasks in Section 3.1 for inclusion in the OPF repository.

TASK NAME: Construct the agent model

Focus: Static architecture

Typical supportive techniques: Intelligent agent identification, Control architecture

Explanation: An analogue of the "object model" as the main description of the static architecture needs to be constructed. This model will show the agents, their interfaces and how they are connected both with other agents and other objects within the system being designed.

New subtasks for Task: Model agents' roles

Subtask: Model responsibilities: these include the accepted OO responsibilities (knowing, doing, enforcing) but classified in terms of liveness and safety properties.

Subtask: Model permissions: these are associated with the responsibilities allocated to each agent role.

New subtask for Task: Model the agent's environment

Subtask: Model environmental resources: these are abstract computational resources that are available to agents for sensing, effecting or consuming.

New subtasks for Task: Identifying system organization

Subtask: Identify sub-organizations: this task analyzes the target system organization to identify sub-organizations that co-exist as autonomous interacting MASs.

Subtask: Define organizational rules: focussed on modelling the responsibilities of the organization as a whole in terms of liveness and safety organizational rules.

Subtask: Define organizational structures: focussed on selecting an organizational structure that offers the most appropriate topology and control regime.

4.3 New Techniques

Three new OPF Techniques, derived from Gaia's techniques described in Section 3.2, are to be added to the OPF repository.

TECHNIQUE NAME: Environmental resources modelling
Focus: System environment
Typical tasks for which this is needed: Model the agent's environment
Technique description: Each resource should be described in terms of its symbolic name, types of actions that agents can perform on it, and if appropriate or necessary, its detailed data structure. Graphical representation of the logical/physical relationships between resources, and the specification of how and from where a resource can be accessed may be useful.
Technique usage: Identify and describe each resource in the MAS environment. The resources' details and the model's representation notation can be decided by the designer depending on the application at hand.
Deliverables: Environment model

TECHNIQUE NAME: Organizational rules specification
Focus: System organization
Typical tasks for which this is needed: Identify system organization
Technique description: Roles that affect the system organization as a whole should be described in terms of liveness and safety organizational rules. Liveness rules define how the dynamics of the organization should evolve over time, while safety rules define time-independent global invariants that the organization should respect.
Technique usage: Liveness organizational rules can be derived from liveness responsibilities of different roles, while saftety rules can relate to safety responsibilities of different roles or to expressions of the environmental resources in different roles. The definition of organizational rules is particularly necessary when the system is open.
Deliverables: Organizational rules specification

TECHNIQUE NAME: Organizational structure specification
Focus: Static architecture
Typical tasks for which this is needed: Identify system organization
Technique description: The organizational structure selected for the target system should offer the most appropriate topology and control regime. Forces affecting this choice may include: the need to achieve organizational efficiency; the need to respect organizational rules; and the need to minimize the distance from the real-world organization.
Technique usage: Together with the analysis of the above factors, the designer should exploit the existing libraries of organizational patterns. Organizational structures can be modelled either by a formal notation or a more intuitive graphical representation.
Deliverables: Organizational structure model

4.4 New Work Products

From Section 3.3, five new work products are recommended for inclusion in the OPF repository in order to support AO development using a Gaia-based approach/ philosophy.

Role schema: Textual description of each role. One schema per identified agent role. Information is displayed about the protocols, permissions and responsibilities for each agent role (Figure 3).

Protocol schema: The Gaia-based interaction model is used here to define and describe the agent protocol schema. It shows the purpose, the initiator, the responder, the inputs and outputs and the processing.

Service table: A tabular representation of each service, indicating the inputs, outputs, preconditions and postconditions for each service.

Environment Model: Specification of each resource. Typical information include the resource's name, types of actions to be performed on it, internal data structure, and textual comments.

Organizational structure model: Organizational structure is modelled using either a formal notation or a graphical representation. The model should show the topology and the control regime of the structure. Typical control relationships are control, peer and dependency.

5 Summary and Conclusions

As part of an extendive research programme to combine the benefits of method engineering and existing object-oriented frameworks (notably the OPF) to create a highly supportive methodological environment for the construction of agent-oriented information systems, we have analysed here contributions from the Gaia AO methodology. We have identified one new Task, six new subtasks (to pre-existing tasks), three new Techniques and five new Work Products, although no additional Roles or Stages were identified and no changes to the OPF metamodel were necessary.

Acknowledgements

We wish to acknowledge financial support from the University of Technology, Sydney under their Research Excellence Grants Scheme. This is Contribution number 03/27 of the Centre for Object Technology Applications and Research.

References

Brinkkemper, S., 1996, Method engineering: engineering of information systems development methods and tools, Inf. Software Technol., 38(4), 275-280.

Brinkkemper, S., Saeki, M. and Harmsen, F., 1998, Assembly techniques for method engineering. Proceedings of CAISE 1998, Springer Verlag, 381-400.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C. and Gilchrist, H., 1994, Object-Oriented Development. The Fusion Method, Prentice Hall, Englewood Cliffs, NJ, USA, 313pp

Debenham, J. and Henderson-Sellers, B., 2003, Designing agent-based process systems - extending the OPEN Process Framework, Chapter VIII in Intelligent Agent Software Engineering (ed. V. Plekhanova), Idea Group Publishing, 160-190.

Firesmith, D.G. and Henderson-Sellers, B., 2002, The OPEN Process Framework. AN Introduction, Addison-Wesley, Harlow, Herts, UK

Henderson-Sellers, B., Simons, A.J.H. and Younessi, H., 1998, The OPEN Toolbox of Techniques, Addison-Wesley, UK, 426pp + CD

Henderson-Sellers, B., Giorgini, P. and Bresciani, P., 2004a, Enhancing Agent OPEN with concepts used in the Tropos methodology, Procs. ESAW'03 (Engineering Societies in the Agents World), LNCS, Springer-Verlag, Berlin (in press)

Henderson-Sellers, B., Debenham, J. and Tran, N., 2004, Incorporating the elements of the MASE methodology into Agent OPEN, Procs. ICEIS2004 (eds. I. Seruca, J. Cordeiro, S. Hammoudi and J. Filipe), INSTICC Press, Portugal (in press)

Kumar, K. and Welke, R.J., 1992, Methodology engineering: a proposal for situation-specific methodology construction, in *Challenges and Strategies for Research in Systems Development* (eds. W.W. Cotterman and J.A. Senn), J. Wiley, Chichester, 257-269

OMG, 2001, OMG: OMG Unified Modeling Language Specification, Version 1.4, September 2001, OMG document formal/01-09-68 through 80 (13 documents) [Online]. Available http://www.omg.org (2001)

Padgham, L. and Winikoff, M., 2002, Prometheus: A Methodology for Developing Intelligent Agents. In proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS'02.

Ralyté, J. and Rolland, C., 2001, An assembly process model for method engineering, in K.R. Dittrich, A. Geppert and M.C. Norrie (Eds.) Advanced Information Systems Engineering), LNCS2068, Springer, Berlin, 267-283.

Rolland, C. and Prakash, N., 1996, A proposal for context-specific method engineering, IFIP WG8.1 Conf. on Method Engineering, 191-208, Atlanta, GA, USA

Rolland, C., Prakash, N. and Benjamen, A., 1999, A multi-model view of process modelling, *Requirements Eng. J.*, **4(4),** 169-187

Ter Hofstede, A.H.M. and Verhoef, T.F., 1997, On the feasibility of situational method engineering, Information Systems, 22, 401-422

Wooldridge, M., Jennings, N.R. and Kinny, D., 1999. A Methodology for Agent-Oriented Analysis and Design. Proceedings of the 3rd International Conference on Autonomous Agents (AA'99), 69-76.

Wooldridge, M., Jennings, N.R. and Kinny, D., 2000, The Gaia methodology for agent-oriented analysis and design, J. Autonomous Agents and Multi-Agent Systems, 3, 285-312

Zambonelli, F., Jennings, N. and Wooldridge, M., 2003, Developing multiagent systems: the Gaia methodology, ACM Transaction on Software Engineering and Methodology, **12(3),** 317-370