Modeling Multi-Agent Systems using UML

Carla Silva[1], João Araújo[2], Ana Moreira[2], [γ]Jaelson Castro[1, 3], Patrícia Tedesco[1], [ξ]Fernanda Alencar[4] and Ricardo Ramos[1]

[1]Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil

{ctlls, jbc, pcart, rar2}@cin.ufpe.br

[2]Departamento de Informática, FCT, Universidade Nova de Lisboa, Portugal

{ja, amm}@di.fct.unl.pt

[3]Istituto Trentino di Cultura, Ist. per la Ricerca Scientifica e Tecnologica, Italy

jaelson@itc.it

[4]Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco

fmra@ufpe.br

Abstract. *Tropos is a framework which offers an approach to guide the development of multi-agent systems (MAS). It relies on the i* notation to describe both requirements and architectural design. However, the use of i* as an architectural description language (ADL) is not suitable, since it presents some limitations to capture all the information required for designing MAS architectures. Recognizing that the Unified Modeling Language 2.0 (UML) supports software architectural description, in this work we present an extension to the UML metamodel to capture the features of agency to support MAS modeling at the architectural level. In doing so, we define a notation to model MAS architectures. Furthermore, we provide a set of heuristics to describe MAS using our UML-based notation derived from an architectural description using i*. We illustrate our approach by modeling a Conference Management System.*

Resumo. *Tropos é um framework que fornece uma abordagem para guiar o desenvolvimento de sistemas multi-agentes (SMA). Ele utiliza a notatção i* para descrever tanto os requisitos como o projeto arquitetural de SMA. Entretanto, o uso da notação i* como uma linguagem de descrição arquitetural (LDA) não é adequado porque esta notação apresenta algumas limitações para capturar toda a informação necessária para projetar a arquitetura de SMA. Reconhecendo que a Unified Modeling Language 2.0 (UML) suporta a descrição arquitetural de software, neste trabalho apresentamos uma extensão do metamodelo da UML para capturar as características de agência de forma a suportar a modelagem de SMA ao nível arquitetural. Com isto, definimos uma notação para modelar a arquitetura de SMA. Além disso, oferecemos um conjunto de heurísticas para descrever SMA usando nossa notação baseada em UML a partir da descrição arquitetural do SMA usando i*. A nossa abordagem é ilustrada usando um Sistema Gerenciador de Conferência.*

[γ] Currently on leave of absence from Universidade Federal de Pernambuco.

[ξ] Currently on leave of absence at FCT / Universidade Nova de Lisboa, Portugal.

1. Introduction

One of the most promising paradigms for developing complex software systems is the agent orientation. However, the benefits promised by the agent paradigm cannot be fully achieved yet because it lacks suitable methodologies to enable designers to clearly specify and structure their applications as agent-oriented systems. To address this issue, we are working on the improvement of Tropos [Castro et al. 2002] - a framework aimed at developing multi-agent systems. Tropos supports the four following phases of software development: Early Requirements, Late Requirements, Architectural Design and Detailed Design.

In this work, our focus is on the architectural design phase. Software architecture defines, at a high level of abstraction, the system in terms of components, the interaction between them as well as the attributes and functionalities of each component [Sommerville 2001]. Tropos relies on the i* notation [Yu 1995] to describe MAS architectural design. However, the use of i* as an architectural description language (ADL) is not suitable, since it presents some limitations to describe the detailed behaviour required for software architectural design, such as protocols, connectors, ports and interfaces. In [Silva et al. 2003] we have proposed an approach to use the UML-RT (UML-Real Time) [Selic and Rumbaugh 1998] as an ADL for Tropos. Part of the UML-RT concepts have been incorporated as architecture description constructs in UML 2.0 [OMG 2005]. Hence, in this paper we present an approach for using UML 2.0 based notation to describe MAS architecture in Tropos which is originally described using the i* notation [Yu 1995]. UML 2.0 is a standard with large tool support, which is tailored for architectural description.

This paper is organised as follows. Section 2 introduces our extension to the UML metamodel to support MAS modelling. Section 3 presents our notation to describe MAS architectural design and a process to use this notation in the context of Tropos. Section 4 illustrates our approach using a case study. Section 5 discusses related work. Finally, section 6 summarises our work and points out directions for future work.

2. Agency Profile

To enable the creation of UML profiles, a profile package has been specifically defined in the UML 2.0 specification [OMG 2005] for providing a lightweight extension mechanism to the UML standard. It contains mechanisms that allow metaclasses from existing metamodels to be extended and adapted for different purposes. This includes the ability to tailor the UML metamodel for different platforms (such as J2EE or .NET) or domains (such as real-time or business process modeling). The profile mechanism is consistent with the OMG Meta Object Facility (MOF) [OMG 2004]. This paper uses this mechanism to adapt an existing UML metamodel with constructs that are specific to the agent paradigm. Such adaptation is grouped in a profile, called *Agency Profile*.

For simplicity, the metamodel defining the agency features is divided into two categories: intentional and interaction. The intentional category concepts are described in Figure 1 while the interaction category concepts are described in Figure 2. The usage of the concepts and relationships has been motivated by a previous work [Silva et al. 2004] where we have established some agent properties required to specify MAS.

In the intentional category a MAS can be conceived as an *Organization* which is composed of a number of *Agents*. The Agent concept extends the UML metaclass Class

from the StructuredClasses package which extends the metaclass Class (from the Kernel package) with the capability to have an internal structure and ports. *Norms* are required for the *Organization* to operate harmoniously and safely. They define a policy and constraints that all the organizational members must be compliant with [Minsky and Muarata 2004]. The *Organization* is typically immersed in exactly one *Environment* that the agents may need to interact with, in order to access *Resources* according to the agent *Rights* [Zambonelli et al. 2003]. The Organization, Norm, Environment, Right and Resource concepts are extensions (and more exactly, specializations) of the UML metaclass Class.

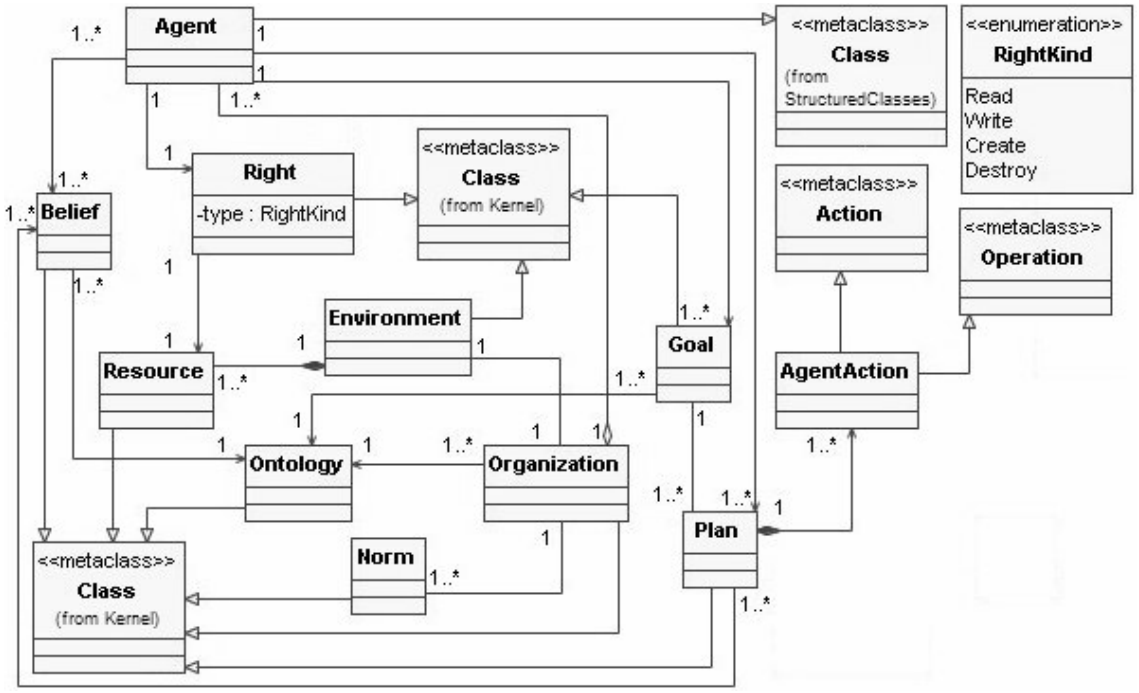

Figure 1. Agency metamodel reflecting intentional concepts

A Goal is a concrete desire of an agent [Braubach et al. 2004]. A Plan encapsulates the recipe for achieving some goal. Beliefs represent the information the agent has about its current environment and itself [Wooldridge 2002] and are conditions for executing plans. These concepts also extend the UML metaclass Class. An AgentAction determines the steps to perform a plan and extends both the Action and Operation UML metaclasses. The description of both Beliefs and Goals must comply with the Ontology used in the Organization, i.e. the vocabulary of concepts which belongs to the system domain. We define the MAS ontology by extending the UML metaclass Class. The rationale is that most of the technology available to define an ontology in MAS is based on object orientation. For example, an ontology in a specific agent platform, called JADE (Java Agent Development Framework) [Bellifemine et al. 2003], is an instance of *jade.content.onto.Ontology* class, in which a set of schemes is added to define the structure of concepts which belongs to the domain being modeled.

Since agents are going to be used in the system architectural design, we will define them by using the organizational architectural features defined in [Silva et al. 2003]. These features, defining the interaction category, are depicted in Figure 2 and include: OrganizationalPort, AgentConnector, Dependum, Dependee, Depender and

AgentConnectorEnd. They extend respectively the UML metaclasses Port, Connector, Interface, InterfaceRealization, Usage and ConnectorEnd.

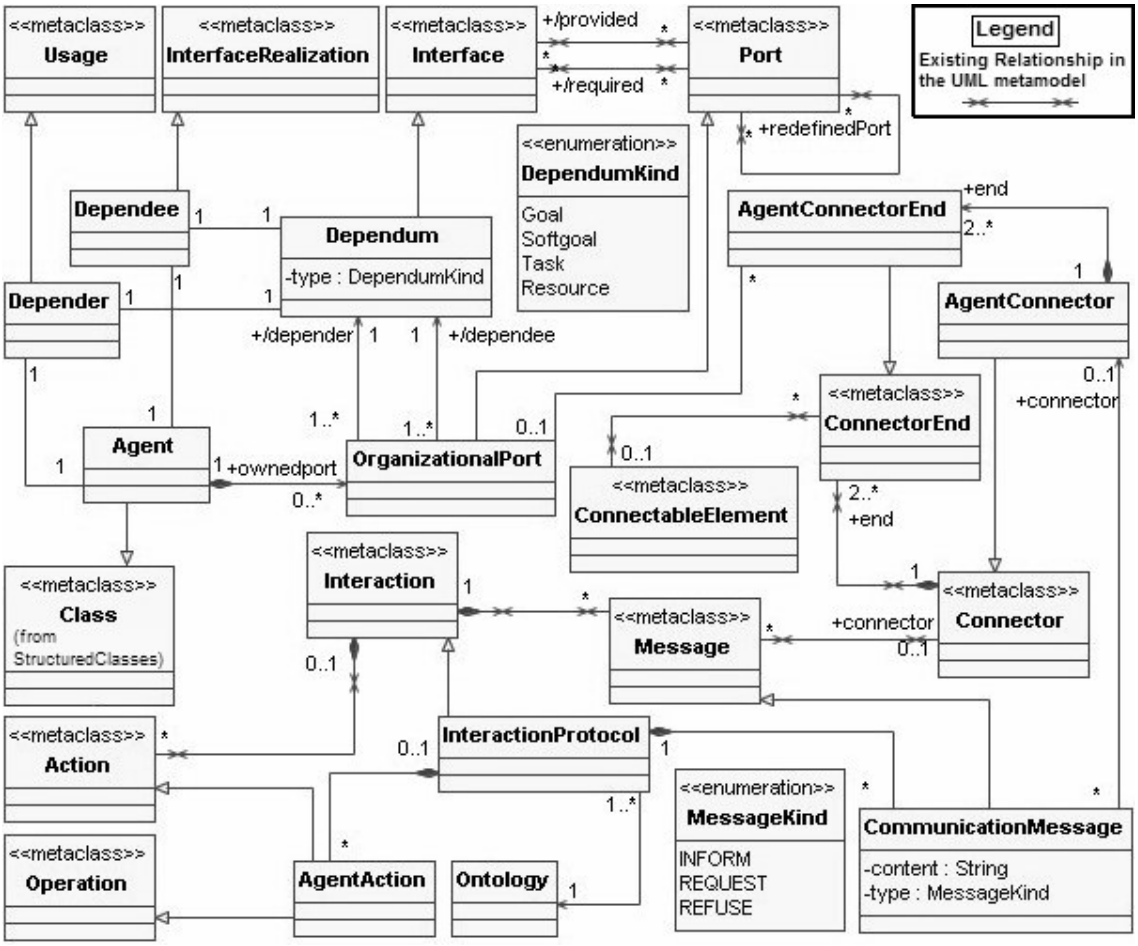

Figure 2. Agency metamodel reflecting interaction concepts

A Dependum defines an "agreement" of service providing between two agents which play the roles of depender and dependee. Thus, the agent responsible for providing the service possesses an OrganizationalPort playing the role of dependee and is related to the Dependum through a Dependee relationship. The agent which request the service possesses an OrganizationalPort playing the role of depender and is related to the Dependum through a Depender relationship. A dependum can be of four types: goals, softgoals, tasks and resources [Yu 1995], defined as the enumeration class DependumKind. Agents need to exchange signals through an AgentConnector to accomplish the contractual agreement of service providing between them. An OrganizationalPort specifies a distinct interaction point between the Agent and its environment. A AgentConnectorEnd is an endpoint of an Agentconnector, which attaches the AgentConnector to an OrganizationalPort.

Each *Agent* can interact with other agents according to an *InteractionProtocol* determined by the *AgentAction* performed by the *Agent*. An *InteractionProtocol* describes a sequence of *CommunicationMessages* that can be sent or received by agents. In addition, the *InteractionProtocol* must comply with an *Ontology,* i.e. the vocabulary of the terms used in the message contents and their meaning (both the sender and the receiver must ascribe the same meaning to symbols for the communication to be

effective). The *InteractionProtocol* concept extends the UML metaclass Interaction. The CommunicationMessage concept extends the UML metaclass Message and can be of several types including REQUEST, INFORM and REFUSE, among others (defined as the enumeration class MessageKind). These are the defined by the Foundation for Intelligent Physical Agents [FIPA 2004] which indicate what the sender intends to achieve by sending the message.

A Profile has been defined in the UML 2.0 specification as a specific meta-modeling technique in which a stereotype defines how an existing metaclass may be extended. The intention of profiles is to give a straightforward mechanism for adapting an existing metamodel with constructs that are specific to a particular domain, platform, or method. For example, in our approach we have extended some UML metaclasses to address agent-oriented concepts. An extension (a kind of association) is used to indicate that the properties of a metaclass are extended through a stereotype. In Figure 3, we present some extensions we have made, such as the stereotype Agent extending the UML metaclass Class (from StructuredClasses package), the stereotype OrganizationalPort extending the UML metaclass Port, the stereotype Dependum extending the UML metaclass Interface, the stereotype Depender extending the UML metaclass Usage, the stereotype Dependee extending the UML metaclass InterfaceRealization and the stereotype AgentConnector extending the UML metaclass Connector.

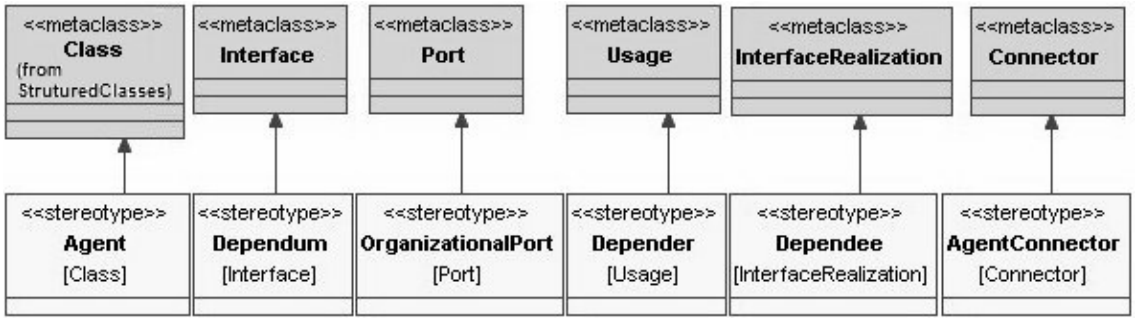

Figure 3. Agency stereotypes

3. Agent-Oriented Modeling

In this section, we present the MAS modeling diagrams specified according to our agency metamodel. These diagrams were conceived to model four views of MAS design: Architectural, Communication, Environmental and Intentional.

3.1. Architectural diagram

The architectural diagram reflects the client-server pattern [Shaw and Garlan 1996] tailored for MAS. It is defined in terms of agents which possess goals achievable by plans. Since an agent is not omnipotent, it needs to interact with other agents in order accomplish its responsibilities. An Agent possess OrganizationalPorts which enable the exchange of messages with other agents through AgentConnectors in order to accomplish some Dependum (i.e., service contract). For example, in Figure 4 we have the Provider agent which is responsible for performing the service defined in the Dependum. This agent aims at achieving the ServicePerformed goal by executing the PerformPlan plan, which, in turn, consists of performing the service() AgentAction. The Client agent aims at achieving the ServiceRequest goal by executing the RequestPlan

plan, which, in turn, consists of performing the request() AgentAction. Therefore, the Client agent is responsible for requesting the service defined in the Dependum. Both the message for requesting the service execution and the message for confirming whether the service was successfully concluded are sent through the AgentConnector.

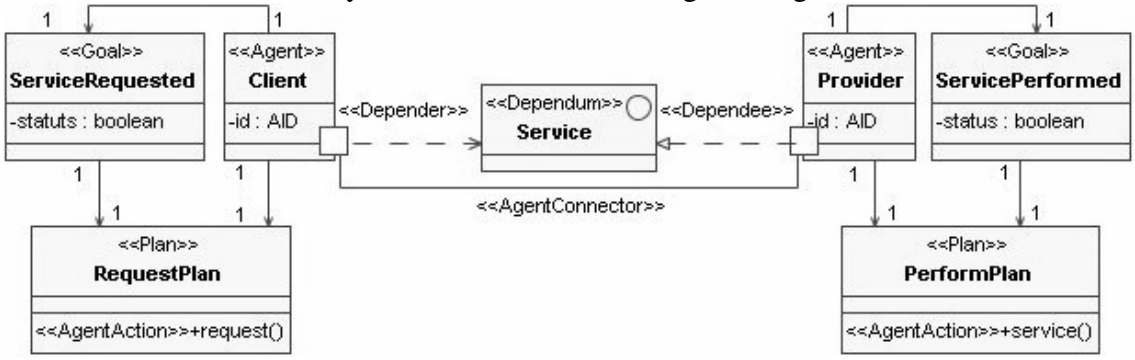

Figure 4. MAS Architectural Diagram

3.2. Communication diagram

The communication diagram is defined in terms of instances of agents and the messages exchanged between them to achieve a service providing. For example, the Figure 5 shows an interaction involving the Client and Provider agents. The Client sends a message requesting the execution of some service while the Provider sends a message informing the requested service has been performed. The interaction specified using the communication diagram is asynchronous.

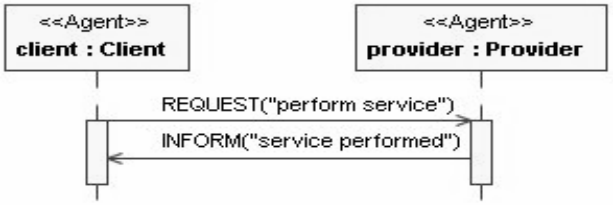

Figure 5. MAS Communication Diagram

3.3. Environmental diagram

The environmental diagram is defined in terms of agents composing an organization which is situated in an environment. This environment is composed of resources which are accessed by the agents according to their rights in order to accomplish their responsibilities. For example, in Figure 6 we have the *Provider* agent composing the *Org* organisation which is situated in the *Env* environment.

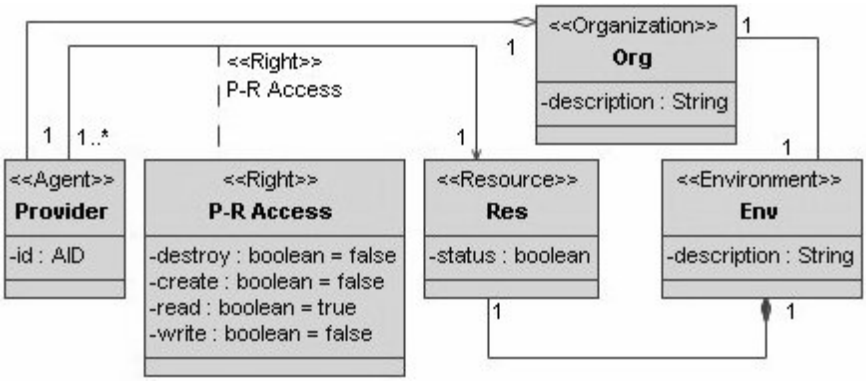

Figure 6. MAS Environmental Diagram

The Provider agent needs to access a *Res* resource available in the *Env* environment to fullfil its responsibilities. The *Provider* agent can only read the *Res* resource, according to its *P-R Access* right (read *Provider-Res Access* right)).

3.4. Intentional diagram

The intentional diagram is defined in terms of agents, their beliefs, goals, plans, as well as the norms and the ontology used in the organization. For example, in Figure 7 we have the *Provider* agent composing the *Org* organisation which must comply with the OrganizationalNorm norms. The Provider agent has a belief about if some request message has been received. Hence, the Request Received is a belief the Provider agent has.

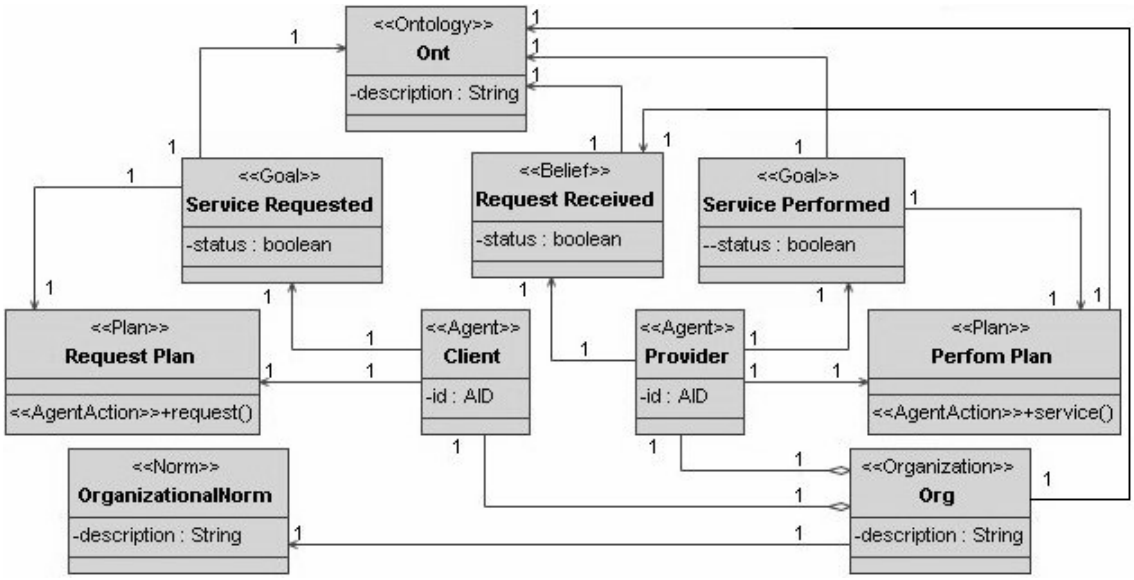

Figure 7. MAS Intentional Diagram

3.5. Mapping i* to UML at the architectural level

To support modeling and analysis during the architectural design phase, Tropos adopts the concepts and models offered by the *i** framework [Yu 1995]. However, the use of i* as an architectural description language is not suitable, since it does not support MAS architectural features, such as ports, connectors, protocols and interfaces. In section 3, we have provided a UML-based notation which supports MAS specification at the architectural level. In order to identify the information related to this notation, it is necessary to perform a means-end analysis [Yu 1995] of each actor which belongs to the MAS architecture represented in i*. This analysis helps identify the reasons/motivations associated with each dependency that an actor possesses. However, the process for performing the means-end analysis of the actors is a specialization of the original process, since we begin by establishing a main goal and from this goal we refine the actor responsibilities. The main goal is operationalized by one or more tasks through the means-end link. Each task corresponding to a means element in some means-end relationships must be further decomposed. The task can be subdivided into other tasks, (soft) goals and resources. Subtasks cannot be further decomposed. The end element in the means-end relationship can only be a (soft)goal.

In [Silva et al. 2006], we have presented some heuristics to map the i* concepts to both agency and UML-RT concepts [Selic and Rumbaugh 1998]. However, at that

stage we did not take into account the architectural concepts supported by UML 2.0. Hence, in this work, we redefine these heuristics to consider the MAS architectural concepts (Figure 4) extended from UML 2.0. The following heuristics guide the mapping of i* description of MAS architecture to our UML-based notation:

1. Each actor in the i* model becomes an «Agent» class in the architectural diagram.
2. Each dependum in the i* model becomes a «Dependum» interface in the architectural diagram. Observe that a «Dependum» can be of four types (goals, softgoals, tasks and resources) according to the agency metamodel Figure 2. This type is not provided explicitly in the model since it is a property of the model element.
3. Each depender in the i* model becomes a «Depender» dependency in the architectural diagram.
4. Each dependee in the i* model becomes a «Dependee» realization in the architectural diagram.
5. Each dependency (depender -> dependum -> dependee) in the i* model becomes a «Connector» association in the architectural diagram. Ports are added to the agents to enable the link through the connector.
6. Each resource related to the actor in the i* model becomes a «Resource» in the architectural diagram. It represents an environmental resource which the agent needs to access to perform its responsibilities. In this work we do not provide guidelines to define the agent rights to access the resources.
7. Each goal (or softgoal) in the i* models becomes a «Goal» in the architectural diagram. It represents the objectives the agent intends to accomplish.
8. Each task in the i* models becomes a «Plan» in the architectural diagram. It represents the means through which a goal is going to be achieved.
9. Each leaf task in the i* models becomes an «AgentAction» in the architectural diagram. It represents each step which composes a plan.
10. A «Belief» is some condition for performing a task (i.e, a «Plan» or an «AgentAction»). It represents the knowledge the agent has about both the environment and itself.
11. The «Organization» is the MAS the agent belongs to.

In fact, a preliminary mapping of i* concepts into agent, goal, belief, plan and resource concepts were originally proposed in [Castro et al. 2002]. However, here we did not provide a process or a notation to capture the mapped information. This new work, introduces a notation to be used in the MAS specification at the architectural level. Finally, we define a specialization of the original means-end analysis process which is appropriate for specifying the rationale of each agent before the mapping of the i* concepts to the agency concepts.

4. Case Study

To illustrate the usage of our approach, we consider the domain of conference management introduced in [Zambonelli et al. 2003] and modeled using the Tropos framework in [Silva et al. 2006]. A conference involves several individuals. During the

submission phase, *Authors* submit papers, and are informed that their papers have been received and have been assigned a submission number. In the review phase, the *Chair* has to handle the review of the papers by contacting potential *Reviewers* and asking them to review a number of papers according to their expertise. Eventually, reviews come in and are used to decide about the acceptance or rejection of the submissions. In the final phase, *Authors* need to be notified of these decisions and, in case of acceptance, will be asked to produce and submit a revised version of their papers. The *Publisher* has to collect these final versions and print the proceedings. Figure 8 presents the Conference Management System, a solution developed as an example of MAS for the conference management domain. The structure-in-5 architectural style [Kolp et al. 2002] has been chosen and applied to the MAS architectural design, but due to lack of space we do not show how we made the choice. Our focus is on the design of the MAS architecture according to agent modeling diagrams (Section 3).

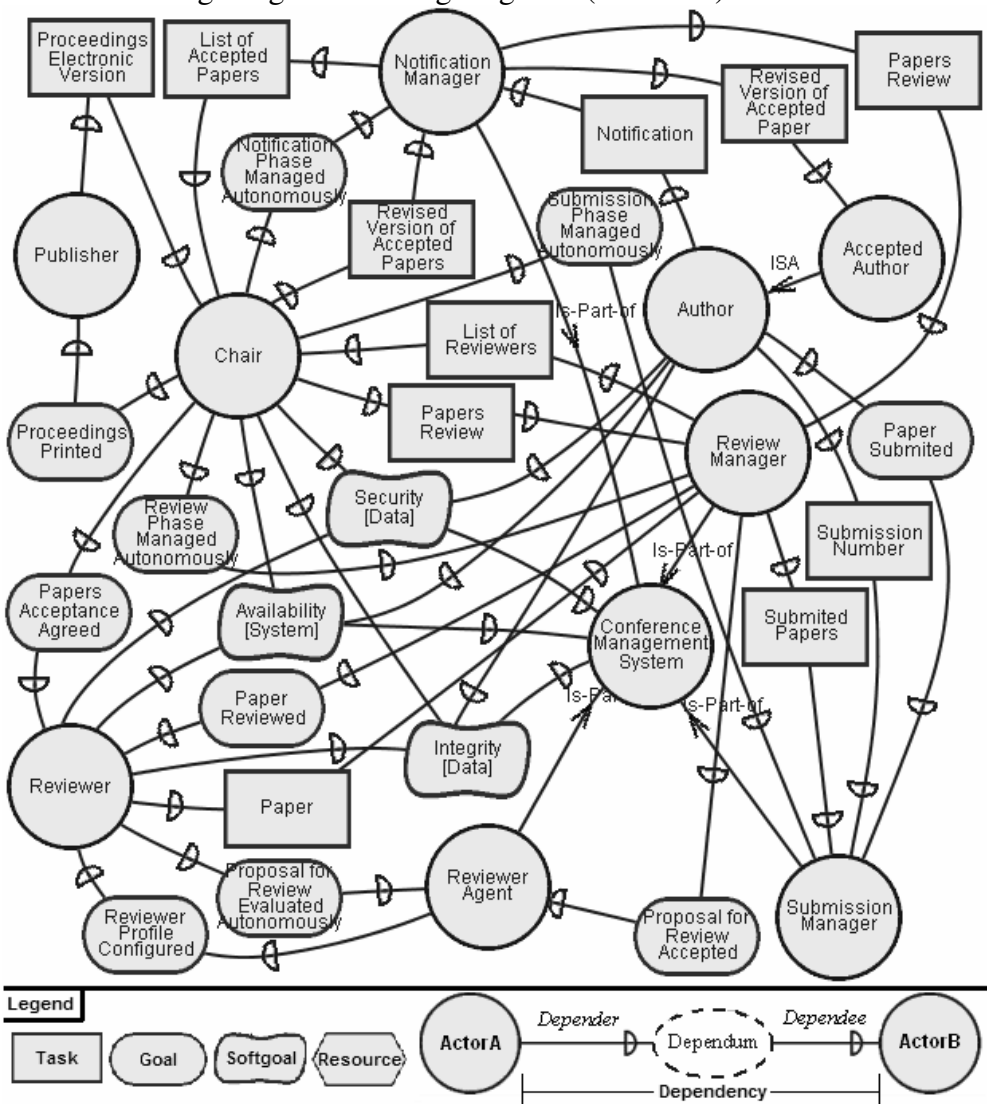

Figure 8. Conference Management System Architecture

The initial version of the Conference Management System supports the submission, review and notification phases of the conference process. The Conference Management System architecture is decomposed into four actors: Submission Manager, Review Manager, Notification Manager and Reviewer. Each of these actors is linked to

the system through an *is-part-of* relationship (see Figure 8). The Submission Manager is responsible for handling the submission phase of the process. The Review Manager is responsible for distributing the set of submitted papers to at least *n* reviewers according to their research area. The Notification Manager is in charge of handling the notification phase process. The Reviewer actor is responsible for evaluating a paper proposal according to the reviewer preferences and skills.

4.1. From i* to UML

The process presented in Section 3.5 is going to be used to produce MAS UML-based models at the architectural level in the context of Tropos. We begin by performing the means-end analysis for each actor which belongs to the MAS architecture described using the i* notation. Then, we rely on the mapping heuristics to specify each diagram presented in Section 3. For example, in our case study we perform the specialized means-end analysis of the Reviewer, Review Manager, Notification Manager and Submission Manager actors in order to capture their rationale when pursuing their goals and dependencies. The Review Manager expects to have *Papers Reviewed*. One alternative to satisfy this goal is to perform the *Manage Review Phase* task. This task is decomposed into four sub-tasks (see the refined model in Figure 9): *Collect Papers Review, Select "n" Reviewers of Paper Research Area, Propose Paper Review* and *Assign Paper Reviewer*.

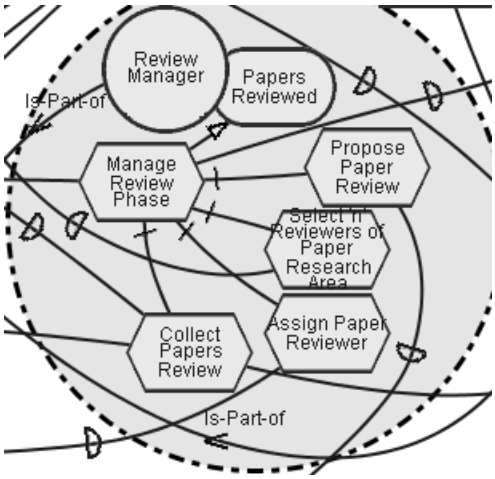

Figure 9. Means-end analysis for Review Manager

Analogously, the Reviewer actor expects to get hold of *Proposal for Review Evaluated* and, to achieve this goal, it has the alternative of performing the *Evaluate Proposal for Review* task. To carry out this task it is necessary to perform several subtasks: *Evaluate Relevance of Conference*, *Evaluate Time Availability, Evaluate Interest in Paper Subject* and *Set Personal Profile*. The Submission Manager actor is in charge of having the *Papers Submission Managed* and to accomplish this goal it has one alternative which is performing the *Manager Submission Phase* task. This task is decomposed into the *Assign Submission Number* and *Collect Paper Submission* sub-tasks. The Notification Manager actor expects to obtain the *Papers Notification Managed* and to reach this goal it has one alternative which is performing the *Manager Notification Phase* task. This task is decomposed into the *Notify Authors* and *Collect Revised Version of Accepted Papers* sub-tasks. The means-end analysis models of the Reviewer, Submission Manager and Notification Manager actors have been omitted here due to the lack of space

4.2. Architectural diagram

Having concluded the means-end analysis of Reviewer, Submission Manager, Notification Manager and Review Manager actors, we can now move on to identifying the properties that characterize that agent according to the MAS modeling diagrams (Section 3). The heuristics presented at Section 3.5 can be of some assistance to describe the Reviewer, Submission Manager, Notification Manager and Review Manager actors according to the architectural diagram (Figure 4). For example, the *Papers Reviewed* goal present in the means-end analysis of the Review Manager actor becomes a «Goal» associated to the Review Manager «Agent» class (colored area of Figure 10). The *Manage Review Phase* task becomes a «Plan» associated to both the Review Manager «Agent» class and Papers Reviewed «Goal» class. Each of the *Collect Papers Review, Select "n" Reviewers of Paper Research Area, Propose Paper Review* and *Assign Paper Reviewer* tasks becomes an «AgentAction» in the Manage Review Phase «Plan» class.

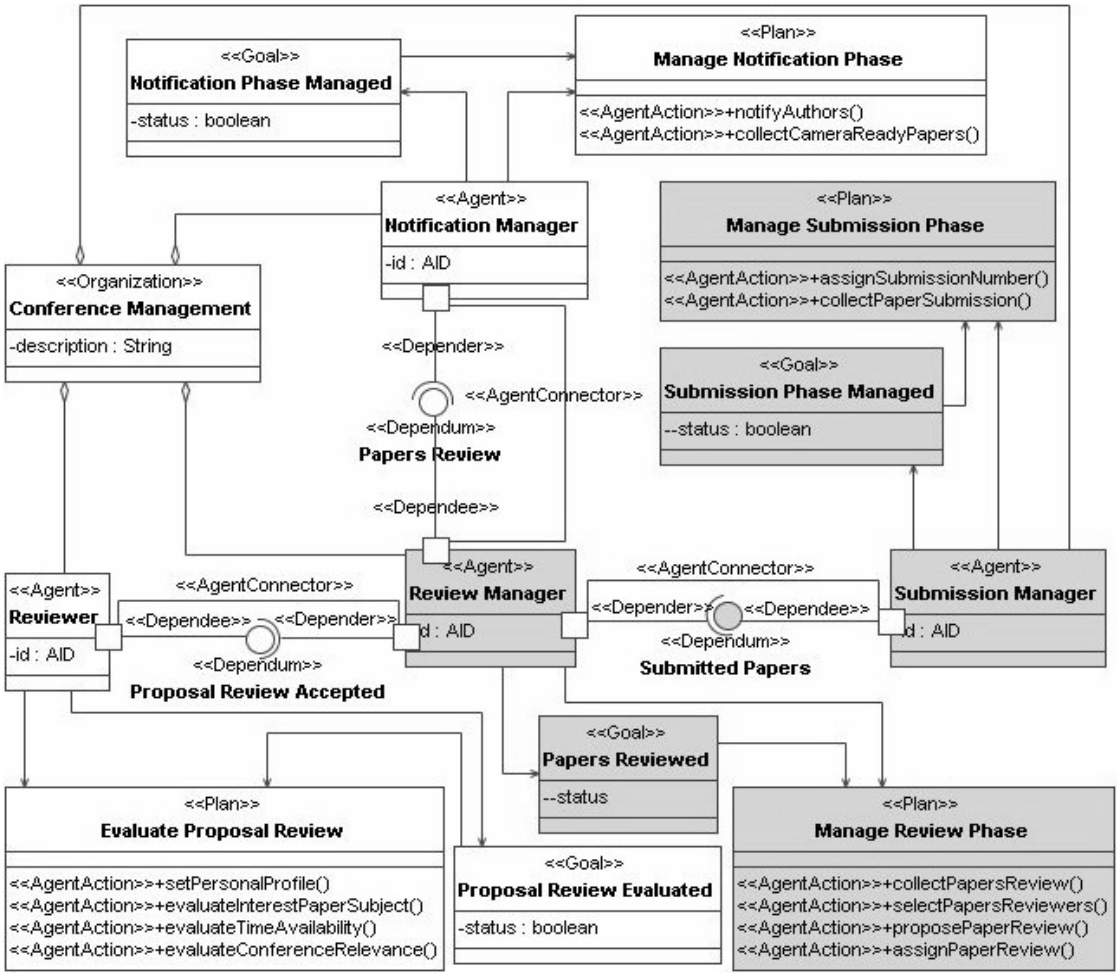

Figure 10. Conference Management Architectural Diagram

Each dependum of the Review Manager actor's dependencies becomes a «Dependum» interface. For example, the dependum of the *Submited Papers* resource dependency, between Review Manager and Submission Manager agents, becomes a *Submited Papers* «Dependum» interface, typed as a resource (colored area of Figure 10). The depender and dependee roles of the *Submited Papers* dependency become the

«Depender» dependency from Review Manager «Agent» class and the «Dependee» realization from Submission Manager «Agent» class, respectively. The *Submited Papers* resource dependency itself becomes a «Connector» between Review Manager and Submission Manager «Agent» classes. Ports are added to these «Agent» classes to enable the link through the «Connector». The same guidelines are applied to the other actors present in the i* model. As a result several classes are incorporated in the MAS architectural diagram depicted in Figure 10.

The architectural design of the Conference Management system (Figure 10) is performed by using the MAS architectural pattern depicted in Figure 4 to assign the system's responsibilities to the architectural components. The Conference Management system is composed of four agents: Submission Manager, Review Manager, Reviewer and Notification Manager. For example, in Figure 10 the colored area corresponds to the interaction between the Review Manager and Submission Manager agents to achieve the service Submitted Papers. The Review Manager agent intends to achieve the Papers Reviewed goal by means of the Manage Review Phase plan. However, to get the papers reviewed the Review Manager agent has to request the Submission Manager agent to perform the Submitted Papers service. This service provides to the Review Manager agent the submitted papers collected by the Submission Manager agent. The Submission Manager performs the requested service because it does not conflict with the achievement of the Submission Phase Managed goal. Hence, both the requested service and the goal achievement are accomplished by means of the Manage Submission Phase plan. The description of the Proposal Review Accepted and Papers Review services is achieved in a similar way.

4.3. Communication diagram

The communication diagram is defined in terms of instances of agents and the messages exchanged between them to achieve their responsibilities. For example, Figure 11 shows an interaction involving the Notification Manager, the Review Manager, the Reviewer and the Submission Manager agents.

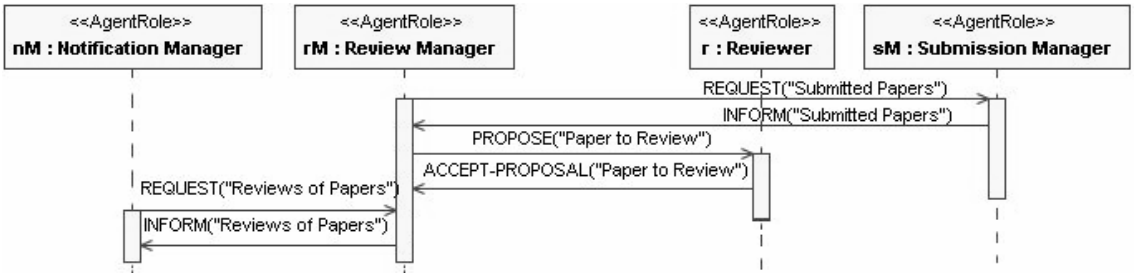

Figure 11. Conference Management Communication Diagram

The interaction specified using the communication diagram is asynchronous. Hence, the Review Manager agent sends a message requesting the submitted papers which are going to be provided through the execution of Submitted Papers service. Then, the Submission Manager answers by informing if the requested service has been performed successfully. The Review Manager agent sends a message to the Reviewer agent proposing a paper to review. If the Reviewer agent answers by accepting the proposal to review the paper, then the Proposal Review Accepted service is assumed to be achieved. The Notification Manager agent, in turn, sends a message to the Review Manager agent requesting the reviews of the papers which are going to be provided

through the execution of Papers Review service. Then, the Review Manager answers by informing if the requested service has been performed successfully.

4.4. Environmental diagram

The environmental diagram is defined in terms of agents composing an organization which is situated in an environment composed by resources which are accessed by the agents according to its rights. The heuristics presented in Section 3.5 continue to be used here to identify the properties which characterize that agent according to this diagram. Hence, all *Submited Papers*, *Papers Review*, *List of Reviewers* resource elements related to the Review Manager actor (Figure 9) become a «Resource» associated to the Review Manager «Agent» class in the environmental diagram presented in the Figure 12. In this diagram we have the *Review Manager* and *Submission Manager* agents composing the *Conference Management* organisation which is situated in the *Conference Management* environment. The Notification Manager and Reviewer agents have been omitted here due to the lack of space.

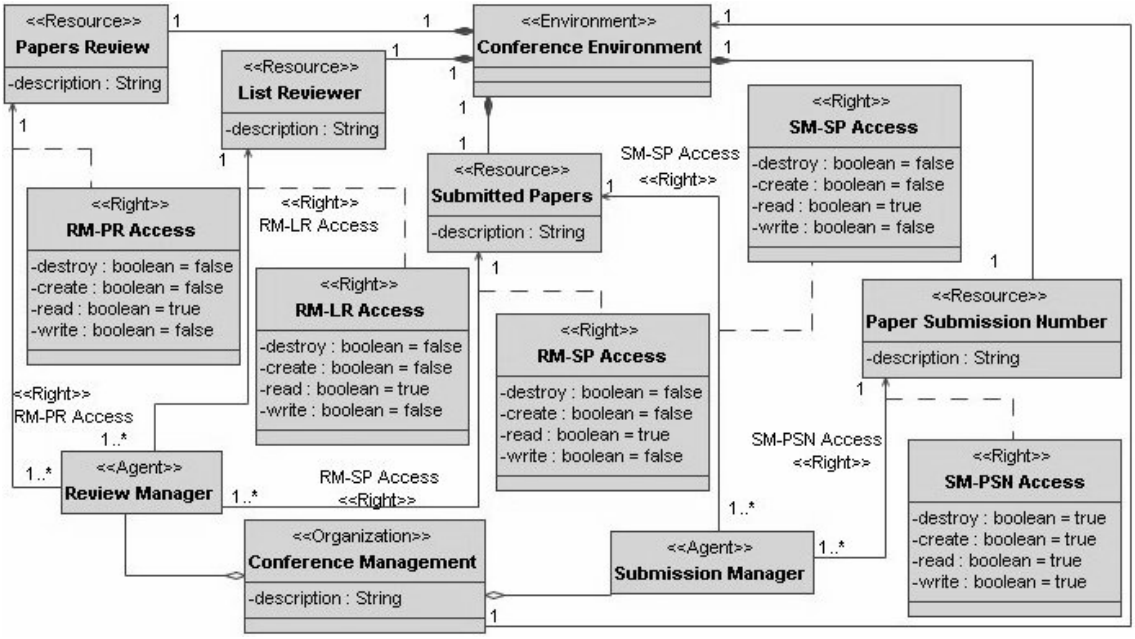

Figure 12. Conference Management Environmental Diagram

The *Submission Manager* agent needs to access the *Submitted Papers* resource available at the *Conference Management* environment to perform the Manage Submission Phase plan. The *Submission Manager* agent can only read the *Submitted Papers* resource, according to its *SM-SP Access* right (read *Submission-Manager-Submitted Papers Access* right). The *Review Manager* agent needs to access a *Papers Review* resource available in the *Conference Management* environment to perform the Manage Review Phase plan. The *Review Manager* agent can only read the *Papers Review* resource, according to its *RM-PR Access* right (read *Review Manager-Papers Review Access* right). The *Review Manager* agent also needs to access a *Submitted Papers* resource to perform the Manage Review Phase plan. His permission is for only reading the *Submitted Papers* resource, according to its *RM-SP Access* right (read *Review Manager-Submitted Papers Access* right). The description of the agent's rights to access the other resources is achieved in a similar way.

4.5. Intentional diagram

The intentional diagram is defined in terms of agents, their beliefs, goals, plans, norms and ontology. The heuristics presented in Section 3.5 are used here to identify the properties which characterize that agent according to this diagram. Hence, the condition to perform the *Collect Papers Review* task (Figure 9) is that the Review Deadline should have passed. The circumstance to perform the *Select "n" Reviewers of Paper Research Area, Propose Paper Review* and *Assign Paper to Reviewer* tasks is that the Submission Deadline should have passed. Each of these conditions becomes a «Belief» associated to the Review Manager «Agent» class. However, the intentional diagram is not shown in this paper because we have not defined yet the heuristics to derive both the MAS ontology and norms. These issues will be addressed in future work.

5. Related Work

Several languages for MAS modeling have been proposed in the last few years, such as AUML [Odell et al. 2000], MAS-ML [Silva and Lucena 2004] and SKwyRL-ADL [Mouratidis et al. 2005].

The work presented in [Mouratidis et al. 2005] proposes a metamodel to define an architectural description language (ADL) to specify secure MAS. In particular SKwyRL-ADL includes an agent, a security and an architectural model and aims at describing secure MAS, more specifically those based on the BDI (belief-desire-intention) model [Rao and Georgeff 1995]. Moreover, the Z specification language is used to formally describe SkwyRL-ADL concepts. Our notation to model MAS also supports the BDI agent model. Furthermore, we also define a process to use the proposed notation in the MAS architectural design.

The proposal of a multi-agent system modeling language called MAS-ML is presented in [Silva and Lucena 2004]. It extends the UML metamodel according to the TAO (Taming Agents and Objects) metamodel concepts [Silva and Garcia et al. 2003]. TAO provides an ontology that defines the static and dynamic aspects of MAS. The MAS-ML includes three structural diagrams – Class, Organization and Role diagrams – which depict all elements and all relationships defined in TAO. The Sequence diagram represents the dynamic interaction between the elements that compose a MAS — i.e., between objects, agents, organizations and environments. However, this approach does not provide a detailed process for guiding the use of that modeling language in MAS development as we do.

AUML [Odell et al. 2000] provides extensions of UML, including representation in three layers of agent interaction protocols, which describe the sequence of messages exchanged by agents as well as the constraints in messages content. However, AUML does not provide extensions to capture the agent's cognitive map (individual structure) or the agent's organisation (system structure). We provide UML-based diagram to capture the agent internal structure and the MAS structure, as well as a detailed process for guiding the use of these diagrams in MAS modeling.

Summarizing, we are concerned with detailing the MAS architectural design by providing a standard notation and process to guide the specification of MAS architecture. The notation we have proposed here captures both the static and dynamic architectural agent features and considers the intentions associated with each agent communication protocol.

6. Conclusions and Future Work

This paper focuses on the Tropos architectural design phase and aims at providing a notation for designing MAS as well as a process to describe MAS architecture using such notation. To achieve this, we have defined an extension of UML metamodel to support agency features to provide a notation for specifying MAS design. This notation supports the specification of architectural features in the context of multi-agent systems. Moreover, we outline a process to guide the description of agents according to our notation in the context of the Tropos framework. We also applied our approach to a Conference Management System to illustrate its feasibility.

For future work we plan to investigate whether other UML 2.0 diagrams are useful for designing MAS. For example, the work presented in [Silva et al. 2005] could be used to complement our approach to model agent plans and actions using UML 2.0 activity diagrams in Tropos. We also need to improve the heuristics to (i) derive the system ontology from i* models and (ii) describe the MAS organizational norms. Applying our approach to other case studies is required to address these open issues.

Acknowledgements

This work was supported by several research grants (CNPq Proc. 304982/2002-4, CAPES Proc. BEX 1775/2005-7, Proc. BEX 3003/05-1, Proc. BEX 3014/05-3, Proc. BEX 3478/05-0 & CAPES/ GRICES Proc. 129/05).

8. References

Bellifemine, F., Caire, G., Poggi, A., Rimassa, G. (2003) "JADE - A White Paper**",** Special issue on JADE of the TILAB Journal EXP.

Braubach, L., Pokahr, A., Lamersdorf, W. (2004) "Jadex: A Short Overview", In 5th Annual International Conf. on Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays'04), AgentExpo

Castro, J. Kolp, M., Mylopoulos, J. (2002) "Towards Requirements-Driven Information Systems Engineering: The Tropos Project", Information Systems Journal. Volume 27. Elsevier, p. 365 – 89.

FIPA (2004) FIPA (The Foundation for Intelligent Physical Agents), Available: http://www.fipa.org

Giorgini, P., Kolp, M., Mylopoulos, J., Castro, J. (2005) "Tropos: A Requirements-Driven Methodology for Agent-Oriented Software", In Henderson-Sellers, B. et al. (eds.): Agent-Oriented Methodologies. Idea Group, p. 20 – 45.

Kolp, M., Giorgini, P., Mylopoulos, J. (2002) "Information Systems Development through Social Structures", In 14th International Conference on Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy.

Minsky, N and Muarata, T. (2004) "On Manageability and Robustness of Open Multi-Agent Systems", In: Lucena, C. et al. (eds.): Soft. Eng. for Multi-Agent Systems II: Research Issues and Practical App.. LNCS, Vol. 2940, Springer-Verlag, p. 189 – 206.

Mouratidis, H., Faulkner, S., Kolp, M., Giorgini, P. (2005) "A Secure Architectural Description Language for Agent Systems", In 4th Autonomous Agents and Multi-Agent Systems (AAMAS'05). Uthrecht, The Netherlands.

Odell, J., Parunak, H. V. D, Bauer, B. (2000) "Extending UML for agents", In Proc. of the 2nd Int. Bi-Conf. Workshop on Agent-Oriented Information Systems at the 17th National Conf. on Artificial Intelligence, Austin, USA. iCue Publishing, p. 3 – 17.

Rao, A. S. and Georgeff, M. P. (1995) "BDI agents: from theory to practice", Technical Note 56, Australian Artificial Intelligence Institute.

Selic, B. and Rumbaugh, J. (1998) "Using UML for Modeling Complex Real - Time Systems", Rational Whitepaper, Available: www.rational.com.

Silva, C., Castro, J., Mylopoulos, J. (2003) "Detailing Architectural Design in Requirements Driven Software Development: The Tropos Case", In Proceedings of the XVII Simpósio Brasileiro de Engenharia de Software, Manaus, Brasil, p. 85 – 93.

Silva, V., Garcia, A., Brandão, A., Chavez, C., Lucena, C., Alencar, P. (2003) "Taming Agents and Objects in Software Engineering", In: Garcia, A. et al. (eds.): Soft. Eng. for Large-Scale Multi-Agent Systems. LNCS, Vol. 2603. Springer-Verlag, p. 1 – 25.

Silva, V. and Lucena, C. (2004) "From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language", In: Sycara, K. et al. (eds.): Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers, 9, 1-2, p. 145 – 189.

Silva, C., Tedesco, P., Castro, J., Pinto, R. (2004) "Comparing Agent-Oriented Methodologies Using a NFR Approach", In Proc. of the 3rd Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'04). Edinburgh, Scotland, p. 1 – 9.

Silva, V., Noya, R., Lucena, C. (2005) "Using the UML 2.0 activity diagram to model agent plans and actions", In 4th Autonomous Agents and Multi-Agent Systems (AAMAS'05). Uthrecht, The Netherlands, p. 594 – 600.

Silva, C., Castro, J., Tedesco, P., Araújo, J., Moreira, A., Mylopoulos, J. (2006) "Improving the Architectural Design of Multi-Agent Systems: The Tropos Case", In 5th Soft. Engineering for Large-Scale Multi-Agent Systems at ICSE'06 (to appear).

Shaw, M. and Garlan, D. (1996) Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.

Sommerville, I. (2001) Software Engineering – Ed.6. Addison Wesley.

OMG (2004) Meta Object Facility (MOF) 2.0 Core Specification, Available: http://www.omg.org/docs/ptc/04-10-15.pdf.

OMG (2005) Unified Modeling Language (UML): Superstructure. Version 2.0, Available: www.omg.org/docs/formal/05-07-04.pdf.

Wooldridge, M. (2002) An Introduction to Multiagent Systems. John Wiley and Sons, Ltd. England, p. 15 – 103.

Yu, E. (1995) Modelling Strategic Relationships for Process Reengineering. Ph.D. thesis. Department of Computer Science. University of Toronto, Canada.

Zambonelli, F., Jennings, N., Wooldridge, M. (2003) "Developing Multiagent Systems: the Gaia Methodology", In ACM Transactions on Software Engineering and Methodology, 12, 3, p. 317 – 370.