Improving the Architectural Design of Multi-Agent Systems: The Tropos Case

Carla Silva¹, Jaelson Castro¹,², *, Patrícia Tedesco¹, João Araújo³, Ana Moreira³ and John Mylopoulos⁴

¹ Centro de Informática, Universidade Federal de Pernambuco, Recife-PE, Brasil, +55 81 2126 8430, 50732-970 {ctlls, jbc, pcart}@cin.ufpe.br
² Istituto Trentino di Cultura – ITC, Istituto per la Ricerca Scientifica e Tecnologica – IRST, Trento-Povo, Italy jaelson@itc.it
³ Departamento de Informática, FCT, Universidade Nova de Lisboa, Caparica, Portugal, +351 21 294 85, 2829-516 {ja, amm}@di.fct.unl.pt
⁴ Department of Computer Science, University of Toronto, Ontario, Canada, +1 416 978 5180, M5S 2E4 john@cs.toronto.edu

ABSTRACT
Agents provide developers with a flexible way of structuring systems around autonomous, communicative elements. In order to support the efficient development of such systems, design techniques need to be introduced. In this context, we propose a set of UML-based diagrams which can be used to capture structural and behavioral features of the Belief-Desire-Intention agent model. The approach aims at supporting detailed architectural design of multi-agent systems by providing a process to guide the description of agent roles according to the proposed diagrams in the context of the Tropos framework. To illustrate the approach we present a Conference Management System case study.

Categories and Subject Descriptors

General Terms
Design, Languages, Theory.

Keywords
Multi-Agent Systems, Architectural Design.

1. INTRODUCTION
Agent technology is believed to have the potential for becoming a mainstream solution of Software Engineering (as it was the case of object orientation). However, the expected benefits cannot be fully achieved yet because of the lack of suitable means for designers to clearly specify and structure their agent oriented systems.

In this work we provide an approach to detail the Multi-Agent Systems (MAS) architectural design, in the context of the Tropos framework [2, 5]. This approach includes (i) a notation to specify structure and interaction of software agent roles according to the Belief-Desire-Intention model [10] and FIPA standards [4], and (ii) a process to help the detailed specification of each agent role. This proposal is an improvement of our previous work [17] in that we now define a more complete notation for describing the agent roles and a process to guide their specification. In particular, the structural diagram introduced in [17] is complemented by a description of agent interaction protocols using an extension of the AUML sequence diagram [9]. The conceptual model presented in [17] has been redefined in order to improve the description of the agent roles’ features. Moreover, we provide a complete set of heuristics to identify the information related to the BDI agent architecture. It relies on a special way of conducting means-ends analysis of each agent role.

The rest of this paper is organised as follows. Section 2 reviews Tropos and some specific architectural concerns. Section 3 introduces our definition of agent role to be used in MAS development. Section 4 introduces the concept of agent role to support the organizational architectural features. Section 5 describes a Conference Management case study. Section 6 discusses related works. Finally, section 7 summarises our work and points out urgent and still open issues.

2. BACKGROUND
2.1 TROPOS
Tropos [2, 5] is an agent-driven framework aimed at building agent-oriented software that operates within a dynamic environment. Tropos adopts the concepts and models offered by

* Currently on leave of absence from UFPE.
\textit{i*} [20] framework which includes concepts such as \textit{actor} (actors can be \textit{agents, positions or roles}), as well as social dependencies among actors including \textit{goal, softgoal, task} and \textit{resource} dependencies. In order to promote an efficient development of agent-oriented systems, Tropos supports four phases of software development:

- Early requirements: It is concerned with the understanding of a problem by studying an organisational setting.
- Late requirements: It results in a specification of all functional and non-functional requirements for the system.
- Architectural design: A specific architectural style is going to be selected among alternatives using as criteria the desired qualities identified in the Late Requirement phase.
- Detailed design: It consists of defining how the components present in the architectural model, are going to fulfill its responsibilities according to design patterns.

This work aims at filling the gap between the architectural design and detailed design phases of Tropos. To support modeling and analysis during the architectural design phase, Tropos adopts the concepts and models offered by the \textit{i*} framework [20]. However, the use of \textit{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. To address this issue, we have proposed an approach to use the UML-RT (UML-Real Time) [11] as ADL in Tropos. This approach is going to be overviewed in the next section.

\section{2.2 ARCHITECTURE-SPECIFIC CONCERNS}

UML-RT [11] has been incorporated as an ADL in UML 2.0 [18] recently. In [3] we have presented an approach using UML-RT notation to represent organizational architectural styles [6], described initially using the \textit{i*} framework notation [20]. Figure 1 illustrates the mapping between \textit{i*} concepts and UML-RT.

In Tropos actors are active entities that carry out actions to achieve goals by exercising their know-how. In UML-RT, capsules are specialized active classes used for modeling self-contained components of a system. Hence, an actor is mapped to a capsule. A dependency describes an “agreement” (called dependum) between two actors playing the roles of dependee and dependee. The dependee is the depending actor, and the dependee, the actor who is depended upon. Dependencies have the form dependee $\rightarrow$ dependum $\rightarrow$ dependee. In UML-RT, a protocol is an explicit specification of the contractual agreement between its participants. Hence, a protocol captures the contractual obligations that exist between capsules. In our approach, a dependum is mapped to a protocol and the roles of dependee and dependee are mapped to protocol roles that are comprised by the protocol. A dependency (dependee $\rightarrow$ dependum $\rightarrow$ dependee) in Tropos is mapped to a connector in UML-RT.

The type of the dependency between two actors describes the nature of the agreement. Tropos defines four types of \textit{dependums}: goals, softgoals, tasks and resources. Each type of \textit{dependum} corresponds to different features in the protocol and therefore in ports that realizes its protocol roles. A Goal dependency will be mapped to an attribute with boolean type present into the port that realizes the Protocol Role dependee. A Softgoal dependency is mapped to an attribute with enumerated type present into the port that realizes the Protocol Role dependee. A Resource dependency is mapped to the return type of an abstract method placed on Protocol Role dependee. A Task dependency is mapped to an abstract method placed on the protocol Role dependee. Further details can be found in [3].

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1.png}
\caption{Mapping a dependency between actors to UML}
\end{figure}

\section{3. AGENT-SPECIFIC CONCERNS}

The conceptual model depicted in Figure 2 explains our agency view. The usage of the concepts and relationships depicted in Figure 2 has been motivated by a previous work [13] were we have established some agent properties required to specify MAS. A MAS can be conceived as an \textit{organization} which is composed by a number of roles. \textit{Norms} are required for the organization to operate harmoniously and safely. They define a policy and constraints that must be conformed to by all members [7]. Each agent can play one or more \textit{roles} and interacts with other agents according to protocols determined by the roles of the involved agents. A \textit{protocol} describes a sequence of \textit{messages} that can be sent or received by agents. In addition, the message must comply with an \textit{ontology}, i.e. the vocabulary of the terms used in the message content and their meaning (both the sender and the receiver must ascribe the same meaning to symbols for the communication to be effective). The MAS is typically immersed in exactly one \textit{environment} that the agents may need to interact with to play their roles [16]. A role has \textit{rights} to access resources which belong to the environment.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig2.png}
\caption{Agency Conceptual Model}
\end{figure}
We are currently investigating how Tropos agents can be designed using the Belief-Desire-Intention architecture [10]. Beliefs represent the information the agent has about its current environment and itself. Desires represent the possible courses of actions available to the agent. Intentions represent the state of affairs that the agent has committed to try to bring about [19]. In Figure 2, a goal is a concrete, momentary desire of an agent role [1]. A plan encapsulates the recipe for achieving some goal. An action determines the steps to perform a plan; and an action can contribute to accomplish different plans. By performing actions, agents can generate new goals to achieve and change the environment state. Therefore, that change is reflected in the agent’s beliefs. Beliefs are conditions for actions being performed. An agent which plays a role has the intention to achieve the role’s goals. These goals can be safety goals or liveness goals [21].

From this agency conceptual model, we have proposed a UML class diagram extension to capture the agency features, i.e. UML stereotypes [18] are used. Hence, in our view, an agent role consists of the structure depicted in Figure 3.

### 3.1 AGENT ROLE

The agent role class depicted in Figure 3 needs to be improved in order to support the organizational architectural features depicted in Figure 1, besides agency features. Observe that the agent role diagram (Figure 4) now supports the concept of organizational architectural port [3] through which architectural components (i.e., agents playing some architectural role in the MAS) can interact with each other by playing a specific role (depender or dependee) in some protocol (depicted by the stereotyped attribute <<Port>>).

![Figure 4 – An agent role diagram](image)

To describe the pattern’s dynamic behaviour we have decided to extend the AUML’s sequence diagram [15] to support the FIPA performatives [4] (Figure 5). The performative indicates what the sender intends to achieve by sending the message. The messages exchanged by Client and Provider agent roles obey the protocol described in Figure 6. Notice that the information between “[]” specifies the content of the message.

![Figure 5 - Protocol](image)

The Client agent role can send a message characterized by a REQUEST performative, indicating that it requests the Provider to perform a specific service. The Provider can reply with (i) a REFUSE performative, indicating that the service has not been performed; or (ii) an INFORM performative, indicating that the service has been performed.

The protocol specification provided in Section 2.2 is also extended to support the FIPA performatives [4] (see Figure 6). The <<Message>> stereotype in the <<Protocol Role>> class indicates the FIPA performatives that an agent role performing that Protocol Role may handle, i.e. the agent role communication interface. In the Client-Provider example, the <<incoming>> signal present in the <<Protocol Role>> Dependeec represents an activity that the <<Role>> Provider can be required to perform by incoming messages characterized by performatives defined in the <<Protocol Role>> Dependeec.
relationship must be further decomposed. The task can be operationalized by one or more tasks through the means-end link. Since we begin establishing a main goal and from this goal we possess. However, the process for performing the means-end analysis of the agent roles is a specialization of the original process, which produces a detailed specification of each agent role. Each resource related to the agent role in the i* model is the architecturally significant information. On the other hand, this new work introduces a notation to be used in detailed specification of agent roles according to BDI agent model. We also define a specialization of the original means-end analysis process which is appropriate for specifying the rationale of each agent role before the mapping of the i* concepts to the BDI concepts. Finally, we propose some heuristics for identifying the BDI concepts as well as other concepts which characterize agent roles in the architectural level.

4. SPECIFYING AGENT ROLES

We rely on the extended class diagram (see Figure 4) to describe each agent role which are required for the MAS architectural design. In order to identify the information related to the BDI agent architecture [10] it is necessary to perform a means-end analysis [20] of each agent role which belongs to the MAS architecture. This analysis helps to identify the reasons/motivations associated with each dependency that a role possesses. However, the process for performing the means-end analysis of the agent roles is a specialization of original process, since we begin establishing a main goal and from this goal we make the refinement of the agent role rationale. 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 relationship 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.

After performing this analysis, we can use the following heuristics to guide the specification of each agent roles:

H1 The architectural features of the extended agent role diagram can be identified by applying the mapping guidelines presented in section 2.2. Thus, we can identify the architectural ports, the protocols and the protocol roles played by the agent roles.

H2 Each resource related to the agent role in the i* model becomes a resource in the agent role class. It represents an environmental resource which the agent role needs to access to perform its responsibilities.

H3 Each goal (or softgoal) in the i* models becomes a goal in the agent role class. It represents the objectives the agent playing the role intends to satisfy.

H4 Each task in the i* models becomes a plan in the agent role class. It represents the means through which a goal is going to be achieved.

H5 Each leaf task in the i* models becomes an action in the agent role class. It represents each step which composes a plan.

H6 A belief is some condition for performing a sub-task (i.e., an action). It represents the knowledge the agent playing a role has about both the environment and itself.

H7 The organization is the MAS the agent role belongs to.

In fact, a preliminary mapping of i* concepts into BDI concepts were originally proposed in [2]. However, at that stage we do not provide a process or a notation to capture the mapped information. On the other hand, this new work introduces a notation to be used in detailed specification of agent roles according to BDI agent model. We also define a specialization of the original means-end analysis process which is appropriate for specifying the rationale of each agent role. Finally, we propose some heuristics for identifying the BDI concepts as well as other concepts which characterize agent roles in the architectural level.

5. CASE STUDY

To illustrate the usage of our approach, we consider the domain of conference management introduced in [21] and modeled using the Tropos framework in [13]. 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 the 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, must be asked to produce a revised version of their papers. The Publisher has to collect these final versions and print the proceedings. In Figure 7 we present the Conference Management System – a solution developed as an example of MAS for the conference management domain. Due to lack of space, in this work we can not detail how the structure-in-5 architectural style [6] has been chosen and applied to the MAS architectural design. Our focus is on the detailing of the MAS architecture according to agent concepts depicted in the agent role diagram (Figure 4).

The initial version of the Conference Management System supports the submission, review and Notification phases of the process. The Conference Management System architecture is decomposed into four agent roles: Submission Manager, Review Manager, Notification Manager and Reviewer. Each of these agent roles is linked to the system through is-part-of relationship in Figure 7. 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 of process. The Reviewer agent role is caters for evaluating a paper proposal according to the reviewer preferences/skills.

The detailing process introduced in Section 4 is going to be used to produce a more detailed MAS architectural design model in the context of Tropos. We begin performing the means-end analysis for each agent role which belongs to the MAS architecture. Then, we rely on heuristics (see Section 4) to specify each agent role.

For example, in our case study we perform the specialized means-end analysis of the Reviewer, Review Manager, Notification Manager and Submission Manager agent roles in order to capture...
their rationale when pursuing their goals and dependencies. The Review Manager expects to have Papers Review Managed. One alternative to satisfy this goal is to perform the Manage Review Phase task. This task is decomposed into four sub-tasks (Figure 8): Collect Papers Review, Select "n" Reviewers of Paper Research Area, Propose Paper Review and Assign Paper Reviewer.

Figure 7 – A Conference Management architecture

The Reviewer agent role 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 agent role 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 agent role expects to obtain the Papers Notification Managed and to reach this goal it has one alternative which is performing the Notification Manager Phase task. This task is decomposed into the Notify Authors and Collect Revised Version of Accepted Papers sub-tasks.

Having concluded the means-end analysis of Reviewer, Submission Manager, Notification Manager and Review Manager (Figure 8) agent roles, we can now move on to identify the properties which characterize that agent according to the agent role diagram (Figure 4). The heuristics presented at Section 4 can be of some assistance to describe the Reviewer, Submission Manager, Notification Manager and Review Manager agent roles. For example, we can extract the information specified in the agent role diagram (Figure 4) and come out with a preliminary Conference Management MAS architecture (see Figure 9).

For example, the Papers Review Managed goal present in the means-end analysis of the Review Manager role becomes a <<Goal>> in the Review Manager role class. The Manage Review Phase task becomes a <<Plan>> in the Review Manager role class. Each of the Collect Papers Review, Select "n" Reviewers of Paper Research Area, Propose Paper Review and Assign Paper Reviewer tasks becomes an <<Action>> (Figure 9). All Submited Papers, Papers Review, List of Reviewers resource elements related to the Review Manager role become a <<Resource>> in the Review Manager role class.

Figure 8 - Means-end analysis for Review Manager

Each dependum of the Review Manager agent role’s dependencies becomes a protocol, according to the mapping guidelines provided in section 2.2. For example, the dependum of the Submited Papers resource dependency is mapped to a protocol and the roles of depender and dependee are mapped to protocol roles. This protocol determines the messages that the participants have to exchange with the Submission Manager agent role in order to send a paper (Submitted Papers resource dependency). Since de Review Manager agent role is the dependee of the Submited Papers resource dependency, the Review Manager agent role class possesses a <<Port>>, called portZ, which plays the Depender role of the SubmitedPapers protocol. Thus, the type of that <<Port>> is SubmitedPapers::Depender (Figure 9).

Similarly, the Submission Manager agent role class includes a <<Port>>, called portW, which plays the Dependee role of the SubmitedPapers protocol. Thus, the type of that <<Port>> is SubmitedPapers::Depender. The dependencies between the agent roles become connectors, represented as associations between their respective role classes. For example, the dependencies between Review Manager and Submission Manager roles becomes a connector between the Submission Manager and Review Manager role classes. The planSelecting(), beliefUpdating() and goalUpdating() basic actions are always present in any agent role specification. The same guidelines are applied to the other agent roles. As a result several agent roles classes are incorporated in the agent role diagram (Figure 9).

The communication protocol shared by the Review Manager and Submission Manager roles is specified in Figure 10. For example, the Review Manager agent role (which possesses the portW) sends a message characterized by a REQUEST performative indicating a request to get a paper (Submitted Papers resource) to the Submission Manager agent role (which possesses the portZ). The Submission Manager agent role can reply with a message characterized by either: (i) a REFUSE performative, indicating that the resource can not be provided; (ii) an INFORM performative, indicating not only that the resource can be
provided but also providing the resource itself. The <<incoming>> signal present in the <<Protocol Role>> Dependee of Figure 10 represents a resource (information) that the <<Role>> which possesses the <<Port>> PortW can be required to provide by incoming messages characterized by performatives defined in the <<Protocol Role>> Dependee.

Figure 10 – SubmitedPapers communication protocol between Review Manager and Submission Manager

6. RELATED WORK

Several languages for MAS modeling have been proposed in the last few years, such as AUML [9], MAS-ML [14] and SKwyRL-ADL [8].

The work presented in [8] proposes a metamodel which defines 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. Moreover, the Z specification language is used to formally describe SkwyRL-ADL concepts. Our approach also proposes a notation to model MAS specification according to the BDI agent model. Furthermore, we also define a process to be used.

The proposal of a multi-agent system modeling language called MAS-ML is presented in [14]. It extends the UML metamodel according to the TAO (Taming Agents and Objects) metamodel concepts [12]. 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 [9] provides extensions of UML, including representation in three layers of agent interaction protocols, which describes 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). On the other hand, although we use an extension of AUML sequence diagram in our notation, we provide UML-based diagram to capture the agent role 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 closing the gap between the software architectural design and detailed design by providing a process to guide a detailed derivation of the MAS architecture. The notation used by that process captures both the static and dynamic architectural agent features and considers the intentions associated with each agent communication protocol.
7. CONCLUSIONS AND FUTURE WORK
This work aims to address current issues related to the lack of proper techniques and notation to model specification in MAS design. We propose UML-based diagrams to capture agent features in the software architectural level and, therefore, support MAS architectural design modeling in more detail. Our notation provides a specification of an agent role diagram which supports the Beliefs-Desires-Intentions agent model and is extended to accommodate organizational architectural properties (such as ports, connectors, protocols for goal, task, softgoal and resource dependencies). Moreover, we propose an extension of the AUML’s sequence diagram to capture the FIPA performatives used in the messages exchanged by the agents. Finally, we outline a process to guide the description of agent roles according to our notation in the context of the Tropos framework. We also applied our approach to a Conference Management System in order to indicate the feasibility of both the proposed notation and process.

Our proposal is under development and consequently much work is still required to improve both the notation and the heuristics to specify MAS in detail. In fact, some information associated to the agent roles is not yet properly addressed by the current heuristics. For example, a strategy to derive the system ontology from i* models is also required. Therefore, applying our approach to other real case studies is crucial to address these open issues. We are also working in an approach for using aspect orientation in MAS development. To achieve this purpose, we need to extend the agency conceptual model to support abstractions of aspect-oriented paradigm. Moreover, other concepts such as commitment, trust and communication language can be added in a next version of the paradigm. Furthermore, other concepts such as commitment, trust and language can be added in a next version of the paradigm. Future work also includes investigating whether other UML 2.0 diagrams are useful for designing MAS. For example, the work presented in [16] could be used to complement our approach to model agent plans and actions using UML 2.0 activity diagrams in Tropos.

8. ACKNOWLEDGMENTS
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 & CAPES/ GRICES Proc. 129/05).

9. REFERENCES