A Modeling Language for Advanced Separation of Concerns in Multi-Agent Systems

Carla Silva¹, João Araújo², Jaelson Castro¹, Ana Moreira², Márcia Lucena¹ and Leonardo Sarmento¹

¹ Centro de Informática, Universidade Federal de Pernambuco, 50732-970, Recife, Brazil
{ctl1s, jbc, mjnrl, lcms}@cin.ufpe.br
² Dept. Informática, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
{ja, amm}@di.fct.unl.pt

Abstract. The development of complex systems in open and dynamic environments benefits from the use of autonomous software architectures provided by multi-agent systems (MAS). Agent orientation offers a higher level of abstraction when thinking about software systems features and behavior, but this paradigm does not support the modularization of some properties that affect several system components. These properties are called crosscutting concerns. To properly describe the detailed design of MAS, it is required to model crosscutting abstractions. This paper proposes a metamodel to describe and modularize crosscutting concerns in detailed design of multi-agent systems. To illustrate the use of the proposed modeling language, we use an e-commerce example.

Keywords. Software Design, Modeling Language, Multi-agent Systems, Aspect-oriented Software Development

1 Introduction

Multi-agent based software engineering is appropriate to develop complex systems as autonomous entities that take the initiative to achieve system goals and represent software users. Several methodologies, specification languages and support tools have been proposed to support the development of such systems. In particular, this work aims at improving Tropos [10], a requirements-driven framework aimed at building multi-agent systems.

Multi-agent systems (MAS) are not free of crosscutting concerns, i.e., concerns that cut across the boundaries of other concerns. For example, it is recognized that the current use of design patterns can lead to the tangling and scattering of design patterns concerns with the functional modules concerns of the application [17] [11] [19] [2] [21]. Moreover, other systemic crosscutting concerns, such as exception handling, fault tolerance and context awareness, have been detected in MAS [9]. To handle

¹ Currently in post-doc position at Universidade Nova de Lisboa
these issues, we can use aspect orientation [12] which proposes means for modularization and composition of crosscutting concerns. In Aspect Oriented Software Development (AOSD), crosscutting concerns are encapsulated into separated units, called aspects. This improved separation of concerns increases software modularity and, consequently, software reusability, maintainability and understandability, as well as it reduces development costs. Several design modeling languages have been proposed to support the development of aspect oriented systems [4][5][24], but none of them is tailored for multi-agent systems using agent abstractions as first-order elements.

In this paper, we extend the agency metamodel, introduced in [20], with aspectual abstractions to modularize crosscutting concerns in multi-agent systems design. The proposed metamodel, called aspectual metamodel, is initially described as a UML profile [18] to benefit from the available tool support. A modeling language, based on the aspectual metamodel, is illustrated by a classical case study for multi-agent systems: the Medi@ System [3].

This paper is organized as follows. Section 2 reviews Tropos, social patterns, the agent-oriented notation, based on the agency metamodel, and the AOSD. Section 3 introduces the aspectual metamodel as well as the aspectual notation to describe MAS detailed design. Section 4 presents the Medi@ case study. Section 5 discusses related works. Last but not least, section 7 summarizes our work and points out future work.

2 Background

The purpose of this work is to propose a metamodel to support aspect oriented abstractions in MAS detailed design. Based on this metamodel, a modeling language is defined to separate and modularize crosscutting concerns in detailed design phase of Tropos framework, such as social patterns concerns. The proposed modeling language is based on the agent-oriented notation introduced in [20]. Thus, this section overviews Tropos framework and Social Patterns (section 2.1), the agent-oriented notation (section 2.2) and the aspect oriented software development (section 2.3).

2.1 Tropos and Social Patterns

Tropos supports five phases of software development: early requirements, late requirements, architectural design, detailed design and implementation [3].

This paper focuses on the detailed design, which is intended to introduce additional detail for each architectural component of a system. In this phase, designers can be guided by a catalogue of multi-agent patterns which offer a set of standard solutions. Tropos has defined a set of design patterns, called social patterns [14], focusing on social and intentional aspects that are recurrent in multi-agent and cooperative systems. Examples of social patterns include booking, subscription, monitor, broker, mediator, wrapper and matchmaker. However, it has been recognized that the use of design patterns can make application core dependent on patterns, decreasing the possibilities of reuse [17].
For example consider the Matchmaker pattern, which involves an intermediary agent (Yellow Pages Server) that receives requests from service Providers to subscribe/unsubscribe its services into the Yellow Pages maintained by it. A Client may need a specific service provided by an unknown Provider. The Yellow Pages Server also receives requests from Clients to locate some Providers which offer a specific service. If there is some Provider for the requested service, the Yellow Pages Server informs that Provider’s identification to the Client which, in turn, can directly interact with it [21].

JADE is a suitable agent platform to support the implementation of MAS. In JADE, a behavior represents a task that an agent can carry out. For example in the case of the Matchmaker pattern, the implementation of each pattern participant concern (i.e., Client, Yellow Pages Server and Provider concerns) will be provided by the JADE’s API (Application Programming Interface) [1]. In particular, the behavior of the main participant of the Matchmaker pattern [14], i.e., the Yellow Pages server agent, is encapsulated into the DF (Directory Facilitator) agent. Hence, it provides a service by means of which a client agent can find other agents providing the services it requires in order to achieve its goals. A provider agent willing to publish one or more services must inform the DF a description including its Agent Id (AID) and the list of available services. The services provided by the DF agent (Yellow Pages Server) are usually used by the MAS agents implemented in JADE. Thus, all agents who need to register or unregister its services in the Yellow Pages of the DF agent will implement the Provider’s concern. Similarly, all agents who need to find a specific provider agent in the Yellow Pages will implement the Client’s concern. Note that the Yellow Pages Server concern is implemented within the DF agent and its implementation is hidden in the JADE’s API. In this case, the Yellow Pages Server participant concern will not cross-cut the application functional modules.

However, both the Provider and Client concerns of the Matchmaker pattern are going to be implemented by several agents in a multi-agent system. Consequently, these concerns become tangled with the concerns of these agents (i.e., application core functionality) and scattered in the system, compromising reusability, maintainability and understandability. To address these issues, we advocate the use of aspect oriented abstractions and mechanisms in the detailed design of MAS.

### 2.2 Agent Oriented Notation

To support the detailed design phase of Tropos, in [20] we have proposed a metamodel wherein an extended version of UML 2.0 [18] is used as a notation to describe MAS design. The agency metamodel defines the constructs required to specify structural and dynamic features of MAS detailed design. Due to lack of space, in this paper we only review the structural diagram (for an example see Fig. 1). According to Tropos, multi-agent systems are designed in terms of AgentRoles. A Dependum defines an “agreement” of service providing between two AgentRoles. An AgentRole is an abstract characterization of the behavior of a social actor within some specialized context, domain or endeavor. It pursues a Goal, which is a condition or state of affairs in the world. A MacroPlan encapsulates a partial recipe for achieving a goal and is
composed of ComplexActions which, in turn, determines abstract steps to perform a MacroPlan. Some ComplexActions can be directly involved with the accomplishment of a Dependum, so that they are placed in the operation compartment of this Dependum. The AgentRole responsible for providing the service possesses a SocialPort playing the role of dependee and is related to the Dependum through a Dependee relationship. The AgentRole which requests the service possesses a SocialPort 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 [25], defined as the enumeration class DependumKind. A SocialPort is a property of a classifier that specifies a distinct interaction point between that AgentRole and its environment. To accomplish some Dependum an AgentRole exchange messages through SocialPorts connected by a SocialConnector. SocialConnector specifies a link that enables communication between two AgentRoles. Since a SocialConnector connector is an extension of assembly connector [18], we can use a dependency as a notation for SocialConnector.

Fig. 1. Structural Diagram

To illustrate the use of the modeling language based on the agency metamodel, let’s consider the situation where a client requests some services to a provider. The model in Fig. 1 shows the Provider agentRole, responsible for performing the service defined in the Dependum. This AgentRole aims at achieving the ServicePerformed goal by executing the PerformPlan MacroPlan, which, in turn, consists of performing the service() ComplexAction. The Client agentRole aims at achieving the ServiceRequest goal by executing the RequestPlan MacroPlan, which, in turn, consists of performing the request() ComplexAction. Therefore, the Client agentRole 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 SocialConnector.

2.3 Aspect Oriented Software Development

Crosscutting concerns can be found in most systems; many of them are related to software global constraints and system properties, such as security, reliability, persistence, exception handling, logging and distribution [12].

In Aspect Oriented Software Development (AOSD), crosscutting concerns are encapsulated into separated units, called aspects, to improve the modularization of these
Concerns [12]. Composition mechanisms are then responsible for weaving an aspect with the base units it affects. To do so, it is necessary to have a:

- **Join point**: indicating the point where the crosscutting concern should join the behavior of the base unit(s) (e.g., a class or even an aspect) it cuts across [13];
- **Composition rule**: expressing the sequential order into which each aspect must be composed with the base unit(s) it cuts across. It specifies if the aspect behavior is going to be applied before, after, or around the join point [13];
- **Before**: the composition happens whenever a join point is reached and before the actual computation proceeds;
- **After**: the composition happens after the computation “under the join point” finishes, i.e., after the method body has run, and just before control is returned to the caller;
- **Around**: the composition happens whenever a join point is reached, and has explicit control whether the computation under the join point is allowed to run at all.

Aspect oriented design modeling requires higher-level models that addresses the aspect oriented programming concepts but avoiding language specific details. It should allow the designer to work at a more abstract level during software construction and evolution. In fact, design models must conform to design principles, such as low coupling and high cohesion, as well as promote understandability of software solution [4]. Taking this premise into account, we propose the aspectual metamodel to provide an aspectual modeling language for MAS detailed design.

### 3 A Metamodel to Support Aspect Orientation in MAS Design

The mechanisms provided by agent-orientation are not sufficient to modularize properly the crosscutting concerns. In fact, some system properties such as context-awareness, fault-tolerance [9], security, as well as design patterns [21] cannot be located in separated modules using only agent-oriented abstractions. This issue can be solved by using abstractions and practices provided by the aspect orientation for separating system crosscutting concerns [12]. In doing so, we present an Aspectual Metamodel able to specify aspectual modules and how they must be composed with the base unit(s) it cuts across. To describe aspect oriented abstractions we need to define:

(i) a unit to encapsulate the crosscutting concern – the Aspect class; (ii) each behavior a crosscutting concerns comprises – the Advice operation; (iii) the definition of the distinct ways a crosscutting concern may affect a base unit – the Enhancement interface; (iv) the point where the crosscutting concern should join the behavior of the base unit it cuts across – the BaseOperation operation; (v) the specification of how and in which sequence the crosscutting behavior must be applied to the base units – the PointCut expression; (vi) the definition of a distinct interaction point between the crosscutting concern and the base unit – the AspectualPort port; and (vii) the explicit relationship between them – the Crosscut connector.

To make easier the understanding of the Aspectual Metamodel, in the following we present first the modeling language based on this metamodel. Thus, let us consider the
situation presented in Fig. 1, where a Client requests a Provider to perform a service. In particular, we intend to keep track of the Provider’s behavior. To do so, we can use the logging concern [16], which is a classical example of a crosscutting concern. Logging is an intrusive concern that watches what the base application does by sending messages to a logfile with information from the base application. In this case, we can create an aspect to encapsulate the logging concern and separate it from the other components. Observe in Fig. 2 that a stereotyped class is used to represent the Logging Aspect in the MAS detailed design.

Fig. 2. Aspectual Model

Fig. 2 shows the Provider AgentRole being affected by the Logging Aspect. It means that every execution of service ComplexOperation is going to be logged through record Advice. Observe that the Crosscut connector contains expressions responsible for composing the Aspect to the AgentRole. As specified in UML metamodel [18], an expression can be placed in the form symbol (operand1, operand2). In our approach, we have defined the PointCut element as an expression in which the symbol is a CompositionRule, the operand1 is an Advice and the operand2 is a ComplexOperation. It means that the operand1 will affect the operand2 in the situation stated by the symbol.

The presented notation is based on the Aspectual Metamodel depicted in Fig. 3. In particular, this metamodel is an extension of the agency metamodel [20] in which we added constructs required to specify aspectual modules, base modules and the relationship between them. For the sake of space, Fig. 3 presents only some of the concepts already defined in the agency metamodel (highlighted in gray color).

In the Aspectual Metamodel (Fig. 3) an Aspect encapsulates a crosscutting concern and each of its distinct behaviors is defined in an Enhancement interface provided by the Aspect. Each Aspect possesses at least one AspectualPort which specifies a distinct interaction point between that Aspect and the base modules. Thus, an Aspect will affect a base module through a Crosscut connector which is attached to its AspectualPort by a CrosscutEnd and to the base module’s port by a BaseConnectorEnd. Observe that the base module affected by the Aspect can be either an AgentRole (Fig. 1) or an Aspect, since the BaseConnectorEnd can be specialized in SocialConnectorEnd
or CrosscutEnd. Considering the base module as an AgentRole, the other extreme of the Crosscut connector is attached to a SocialPort through a SocialConnectorEnd. If the base module is an Aspect, the other extreme of the Crosscut connector is also attached to an AspectualPort through a CrosscutEnd.

Fig. 3. Aspectual Metamodel

A Crosscut connector is an extension of the Assembly Connector (a kind of connector) [18], but it overwrites the constraint of the Assembly Connector, i.e., instead of being defined from a required port to a provided port, a crosscut connector is defined from a provided port to another provided port (in the case of an Aspect affecting another Aspect) or from a provided port to a dependee port (in the case of an Aspect affecting a AgentRole).

Observe that the Enhancement interface realized by the Aspect is composed of at least one advice which comprises a crosscutting behavior to be added to the base modules. Other feature that can compose an Enhancement interface is the IntertypeDeclaration. This declaration specifies a new member (property or operation) to the classes to which the aspect is attached, or changes the inheritance relationship between classes [8]. Thus, IntertypeDeclarations can be of three types: parent, property and operation. A parent IntertypeDeclaration changes the inheritance hierarchy of a base module by changing or adding a supertype to it. A property IntertypeDeclaration changes the static structure of a base module by adding a new property, while an operation IntertypeDeclaration changes it by adding a new operation.

An important feature of the Crosscut connector is that it possesses the information related to the binding between the Aspect and the base module. This information is a PointCut expression which maps an Advice operation (provided by the Aspect) to an Operation (performed by the base module) by using a CompositionRule (after, around
or before). This expression informs the point and the situation in which the Aspect will affect the base module.

In the next section we present a case study based on an e-commerce multi-agent system to illustrate the use of our design modeling language to modularize both systemic crosscutting concerns, such as security, and patterns crosscutting concerns, such as the matchmaker social pattern.

4 Medi@ System Aspectual Detailed Design

Media Shop [3] sells and ships different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like. To increase market share, Media Shop has decided to open up a B2C retail sales front on the Internet. The basic objective for the new system is to allow an on-line customer to examine the items in its catalogue and place orders. Fig. 4 shows one view of the Medi@ detailed design by using the aspectual notation.

Medi@ system is structured into three AgentRoles: Front Store, Back Store and Billing Processor. Front Store is responsible for supplying a customer with a web shopping cart to keep track of items the customer is buying when visiting Medi@. Back Store keeps track of all web information about customers and other data of strategic importance to Media Shop. Billing Processor is in charge of the management of orders, bills and other financial data [3].

As expected to an e-commerce system, security is an essential non-functional requirement to be achieved in this system. In fact, clients exposed to the internet are, like servers, at risk in web applications. It is possible for web browsers and application servers to download or upload content and programs that could represent a certain degree of risk to the system and the information it manages. Thus, a security requirement might be operationalized by requesting client authorization, encrypting personal data or checking data consistency for each business transaction.

4.1 The Security Concern

The security concern is tangled and scattered with all Medi@ agentRoles. For example, when the customer places an order for the first time in the Medi@ system, he has to fulfill a profile form. It contains the Customer’s personal data such as address, credit card number, phone number, etc. Thus, all the customer data might be encrypted by the Front Store agentRole before sending it to the Back Store agentRole.

To finish the order of items added to customer’s shopping cart, the Front Store agentRole must request specific customer’s information for authorization. Moreover, to process the billing information, it is needed to access customer’s information such as the credit card number and delivery address. This data must be also encrypted by the Back Store agentRole before sending it to the Billing Processor agentRole.

In order to avoid that scattering and tangling of the security with base modules of MAS, a proper notation to locate crosscutting concerns in separate modules is required. Thus, we can use the aspectual notation to properly describe the Medi@ detailed
design. In the model depicted in Fig. 4, the Security concern is modularized into an aspect which encapsulates two crosscutting behaviors: the Encryption and Authorization behaviors. They are defined in each Enhancement interface provided by the Aspect, namely the Encryption enhancement and Authorization enhancement, which will affect differently each module of the Medi@ system.

For example, the Security aspect crosscuts the Front Store agentRole with both the Encryption and Authorization enhancements, as well as the Back Store agentRole with the Encryption enhancement. The Encryption enhancement adds to the Front Store agentRole the ability to encrypt data before collecting the customer’s profile. It also adds to the Back Store agentRole the ability to encrypt data before sending payment information to the Billing Processor agentRole. The Authorization enhancement, in turn, adds to the Front Store agentRole the ability to authorize the customer after per-
forming checkout of the shopping cart. This improved separation of concerns in MAS produces software artifacts that are easier to maintain and reuse, since each concern is located into separate components.

For the sake of simplicity, some of the dependencies between the AgentRoles composing Med@, as well as the «ComplexAction» stereotypes were omitted. Observe that each ComplexAction defined in a MacroPlan must be available in some Dependium implemented by an AgentRole. An interested reader can find the complete specification of the Med@ design in [22].

Some systemic properties are recognized as being crosscutting concerns, such as error handling, fault tolerance [9] and security, as illustrated in this section. In addition, other concerns present crosscutting characteristics such as design patterns concerns [11] [19] [2] [21]. Next section we use the proposed aspectual notation to modularize the Matchmaker pattern concern in the Med@ detailed design.

4.2. The Matchmaker Pattern Concern

The Med@ system has the Availability requirement, i.e., the system has to ensure easier recovery of the system if some agent in the system stops running. This requirement could not be shown in the case study because we did not present the requirements models of the Med@ system. However, the interested reader can find it in [3].

Analyzing the description of several patterns [14], we have concluded that the most suitable pattern to address the Availability requirement is the Matchmaker, since it enables the search for another agent to replace the one that has stopped. Thus, the focus of the aspectual model depicted in Fig. 5 is to present the Matchmaker Aspect and its composition with the agentRoles of Med@ system. For this reason, in this view of the aspectual model, we have omitted the Security Aspect for the sake of simplicity. The Matchmaker Aspect encapsulates two crosscutting behaviors: the Client and the Provider behaviors. They are defined in each Enhancement interface provided by the Aspect, namely the Client enhancement and Provider enhancement. Each of these behaviors is going to affect the agentRoles in the system in a distinct way. For example, the Client enhancement will affect an agentRole by adding the ability to request the identification of a specific service provider to the Yellow Pages Server agentRole. The Provider enhancement, in turn, will affect an agentRole by adding the ability to subscribe or unsubscribe in the yellow pages of the Yellow Pages Server agentRole.

Since we have chosen JADE as the target agent implementation environment, we are going to consider the JADE’s Directory Facilitator (DF) agent as the implementation of the pattern’s Yellow Pages Server participant. For this reason, the Matchmaker Aspect does not present a third enhancement, called Yellow Pages Server enhancement. Thus, we do not need to worry about the separation of the Yellow Pages Server participant concern since it is already implemented in the JADE’s API.

According to the Fig. 5, the Matchmaker Aspect crosscuts the Front Store agentRole with the Provider enhancement, as well as the Back Store agentRole with the Client enhancement. The Provider enhancement adds to the Front Store agentRole the ability to request its subscription in DF’s yellow page before collecting the customer’s profile. It also adds to the Front Store agentRole the ability to request its subscription in DF’s yellow page after performing checkout of the shopping cart. The Client
enhancement, in turn, adds to the Back Store agentRole the ability to request the information of a specific provider to the DF while it is recording a profile. For the sake of simplicity we have omitted some of the crosscut relationships. For example, the Matchmaker aspect also crosscut the Billing Processor agentRole with Client enhancement since it needs to locate the Back Store to request the Customer’s payment information. Therefore, the Back Store agentRole is also affected by the Matchmaker aspect with Provider enhancement.

Fig. 5. Second view of the Aspectual Model for Medi@

The Matchmaker pattern ensures the decoupling among modules contributing to the achievement of the system’s availability. For example, if a Front Store agentRole contacted by a Back Store agentRole stops running, the Back Store may replace it by requesting to the Directory Facilitator to locate another one in its yellow pages.
5 Related Work

Aspect oriented design requires models that facilitate the specification of systems in the presence of multiple aspects and crosscutting, as well as reduce the gap between system design and implementation [4]. In this light, several modeling languages have been proposed to support the design of aspect oriented systems. For example, Aspect Oriented Design Model (AODM) [24] enhances the existing UML specification with aspect oriented concepts that reproduces the crosscutting characteristics of the AspectJ language. The aSideML [4] is an aspect oriented modeling language based on the UML that provides notation, semantics and rules for specifying aspects and crosscutting at the design level of OO systems. Theme/UML [5] is an approach that handles crosscutting concerns at design level by means of themes. In this proposal, each model contains its own theme, i.e., the design of an individual requirement. A composition relationship specifies how thematic design models are to be composed by identifying overlapping concepts in different models and specifying how models should be integrated. These proposals do not support MAS development.

A particular approach, proposed by Garcia [8], promotes separation of specific agent concerns by using aspect oriented abstractions and methods. These specific agent concerns include interaction, adaptation, autonomy, knowledge, collaboration, roles, learning, and mobility. In particular, we consider these concerns as agent abilities. In [8], aspects were used to capture these agent abilities that are hard to modularize with existing abstractions of object-oriented software engineering. This proposal uses the aSideML notation [4]. We are also interested in multi-agent systems development. However, we are not concerned with the separation of agent abilities since we consider them as internal, i.e., already built-in and supported by some agent implementation platform. We use aspect oriented abstractions to support the separation and modularization of agent-oriented design patterns concerns instead of agent abilities.

In Silva et al. [21], we have proposed a technique to describe social patterns by promoting the separation and modularization of its concerns and the latter systematic composition with the MAS base modules. This technique is called Agent Pattern Specification (APS) and was developed by specializing the agency metamodel [20] with the use of model roles [7]. In contrast to our current work, the APS technique does not consider the aspectual abstractions as first-order elements. Moreover, the APS technique was used only to modularize social patterns concerns and other crosscutting concerns, such as security, were not taken into account until that moment. However, the authors intend to investigate if the APS technique can be used to modularize systemic crosscutting concerns and compare the results to our current proposal.

6 Conclusions

Aspect orientation complements current paradigms aiming at improving the separation of crosscutting concerns in software development and, therefore, providing better understandability, maintainability and reusability to the artifacts generated during software development lifecycle.
This paper has focused on a detailed design modeling language for complex software systems developed with agent technology. We have introduced a metamodel to describe the detailed design of multi-agent systems by separating and modularizing crosscutting concerns in aspectual abstractions. Modeling MAS behavior has already been proposed in [20] as an extension of the UML’s sequence diagram. However, modeling the dynamic composition between the aspect and the base modules in MAS is required. This is theme for future work.

A problem that needs to be handled when developing MAS is related to models’ scalability. In fact, modeling even simple agent-oriented systems tends to produce complex models, since there is much information to capture. This issue can be handled by using views of the models to the extent that the models’ complexity increases (such as we did in Fig. 4 and Fig. 5).

Besides supporting detailed design phase of Tropos with proper modeling language, our purpose in defining the Aspectual Metamodel also includes using the principles of Model Driven Engineering [23] to generate models through model transformations rules.

In this work we have chosen to define the Aspectual Metamodel as an extension of the UML metamodel, in doing so we can benefit from the available CASE tool support. However, we intend to create our own concrete syntax (graphical representation) for the modeling language defined by the Aspectual Metamodel, rather than using the UML concrete syntax. To achieve this, we can describe the proposed metamodel using the eCore metametamodel [6] and use other features provided by the Eclipse to support automatized checking of completeness and consistency of models.

Acknowledgements

This work was supported by the CAPES-GRICES (Project 129/05) and PICDT-CAPES research grants.

References