Towards an Aspect Oriented Modeling in Multi-agent Systems

Carla Silva[1], João Araújo[2], Ana Moreira[2], Jaelson Castro[1,3], Daniel Penaforte[1], André Carvalho[1]

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

{ctlls, jbc, dap4, allc}@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

Abstract. *Most approaches for developing multi-agent systems (MAS) do not take into consideration the separation and modularization of crosscutting concerns. The goal of this paper is to propose a systematic approach to both specify aspects in a generic fashion and use those specifications in MAS architectural design. To achieve this, we specialize an agency metamodel for MAS architectural diagram by using the concept of model roles. The proposed notation can be mapped to constructs of AspectJ and JADE. The ideas are illustrated using a content management system, called e-News.*

1. Introduction

The mechanisms provided by existing software development paradigms (e.g. structured, object-oriented and agent-oriented) are not sufficient to modularize some concerns that cut across several other concerns (called crosscutting concerns), found in most complex systems. The result is a decrease of system reusability and maintainability. These *crosscutting* concerns can be better addressed by adopting aspect-oriented software development (AOSD) techniques [Kiczales et al. 1997]. This paradigm has the purpose of localizing crosscutting concerns, avoiding their scattering through software artifacts or tangling these artifacts with specific concerns [Elrad et al. 2001].

Although many languages for multi-agent systems modeling have been proposed in the last few years, such as AUML [Odell et al. 2000] and MAS-ML [Silva and Lucena 2004], they do not consider the separation of crosscutting concerns. On the other hand, some languages for modeling separation of crosscutting concerns have been proposed, such as aSideML [Chavez 2004] and AODM [Stein 2002], but they do not address MAS directly. In this context, we propose a notation to support the separation of crosscutting concerns in MAS. To achieve this, we need to: (i) use the MAS architectural diagram, whose specification is based on the agency metamodel [Silva et al. 2006]; (ii) specialize the agency metamodel for architectural diagram by using the concept of model roles [Kim et al. 2002]; (iii) use both the architectural diagram and specialized agency metamodel to describe aspects; (iv) to map the proposed notation to the constructs of AspectJ and JADE implementation environments.

This paper is organised as follows: Section 2 presents some background on UML-based MAS notation and Pattern Specifications technique. Section 3 introduces our notation to support separation of crosscutting concerns in MAS. Section 4 exemplifies the use of our approach. Section 5 presents some related work. Finally, section 6 summarises our work and points out directions for future work.

2. Background

To present our approach, we first need to introduce the MAS Notation [Silva et al. 2006] based on UML [OMG 2005] and the Pattern Specifications technique [France et al. 2004].

2.1. UML-based MAS Notation

In this section, we present the MAS architectural diagram specified according to the agency metamodel introduced in [Silva at al. 2006] and reflecting the client-server pattern [Shaw and Garlan 1996] that we have tailored for MAS. We define the MAS architectural diagram (Figure 1) in terms of Agents and organizational architectural features which includes: Agent, Goal, Plan, AgentAction, OrganizationalPort, AgentConnector, Dependum, Dependee and Depender. The three first concepts extend the UML metaclass Class. The AgentAction extends both the Action and Operation UML metaclasses. The last five concepts extend respectively the UML metaclasses Port, Connector, Interface, InterfaceRealization and Usage. A Dependum defines an "agreement" of service offer between two agents that play the roles of Depender and Dependee. Thus, the agent responsible for providing the service is the dependee. The agent that requests the service providing is the depender. A dependum can be of four types: goals, softgoals, tasks and resources [Yu 1995]. 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 (depicted as a white square attached to the «Agent» class). A Goal is a concrete desire of an agent [Braubach et al. 2004]. A Plan encapsulates the recipe for achieving some goal. An AgentAction determines the steps to perform a plan.

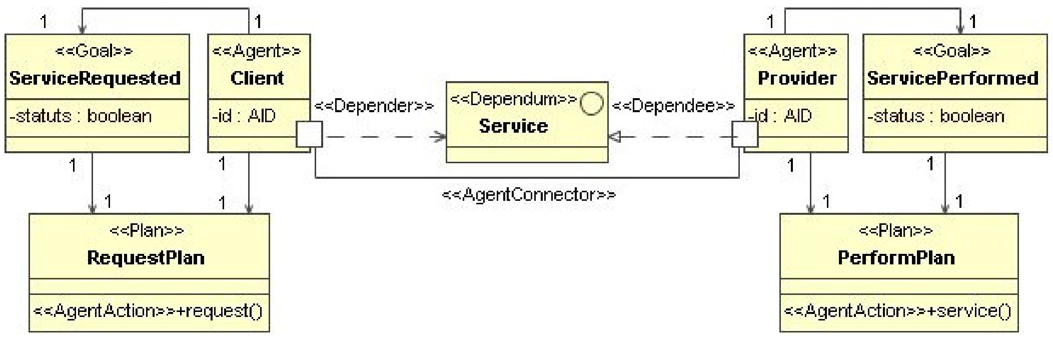


Figure 1. MAS Architectural Diagram

For example, in Figure 1 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

ServiceRequested 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.

2.2. Pattern Specifications

Pattern Specifications are introduced in [France et al. 2004] as a way of formalizing the structural and behavioral features of a pattern. The notation for PSs is based on the Unified Modeling Language (UML). The abstract syntax of UML is defined by a UML metamodel [OMG 2005]. PSs specialize this metamodel by specifying what model elements must participate in the pattern and is defined in terms of roles. Hence, each element in the specification of the patterns is a role, that is, a metaclass specialized by additional properties that any element fulfilling the role must possess. A PS can be instantiated by assigning UML model elements to the roles in the PS. A model conforms to a PS if its model elements that play the roles of the PS satisfy the properties defined by the roles. An example of a PS for class diagram and a conforming UML class diagram is given in Figure 2. Roles are identified by preceding them with a vertical bar, "|".

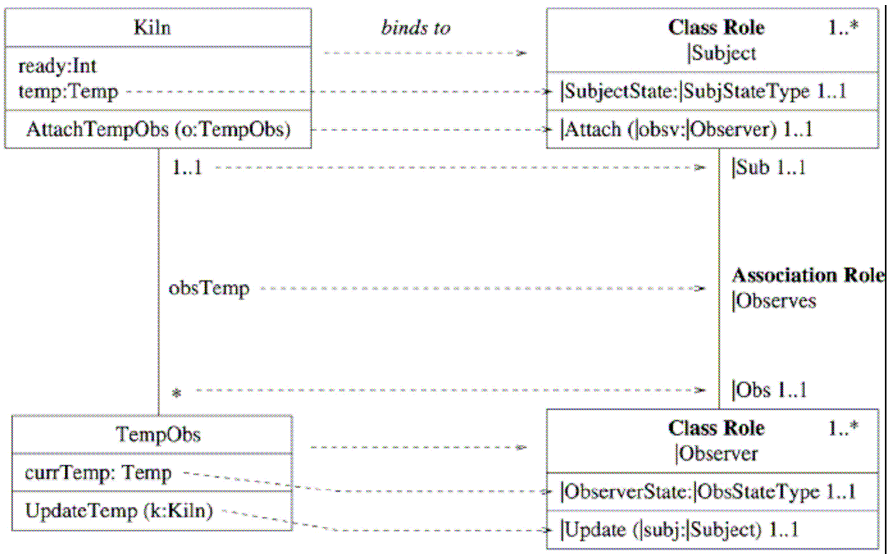


Figure 2. A PS class diagram (right) and a conforming UML class diagram (left). (Taken from [France et al. 2004])

The PS model is, in fact, a specialization of the metamodel for UML class diagrams but is presented graphically so that it resembles a class diagram not the metamodel. The right-hand side of Figure 2 is a PS model defining the structure of a general Observer pattern. The left-hand side shows a UML model that conforms to the PS model – *Kiln* plays the role of |*Subject*, *TempObs* plays the role of |*Observer*, *currTemp* plays the role of |*ObserverState*, *AttachTempObs* plays the role of |*Attach*, and so on. Note that the definition of conformance allows other model elements to appear in the conforming model as long as the original role constraints remain satisfied.

3. Aspects in MAS

In this section we present a motivation example for separating crosscutting concerns in MAS, an approach to model aspects in MAS and some guidelines to map the proposed notation to JADE and AspectJ code.

3.1. Motivation Example

JADE [Bellifemine et al. 2003] is a suitable agent platform to support the implementation of MAS. In JADE, a behavior represents a task that an agent can carry out. One of the most important features that JADE agents provide is the ability to communicate and the Yellow Pages service.

To exemplify a crosscutting concern in MAS we have the concern related to the DF (Directory Facilitator) agent defined in JADE. The DF provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals. An agent wishing to publish one or more services must provide the DF with a description including its AID and the list of its services. An agent wishing to search for services must provide the DF with a template description. The result of the search is the list of all the descriptions that match the provided template. A description matches the template if all the fields specified in the template are present in the description with the same values.

The services provided by the DF agent are usually used by all agents of a MAS implemented in JADE. Thus, all agents who need to publish its services in the Yellow Pages of the DF agent will present the code depicted in Figure 3.

```
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(genericAgent.getAID());
    ServiceDescription sd = new ServiceDescription();
    sd.setType(genericAgent.getServiceType());
    sd.setName(genericAgent.getServiceName());
    dfd.addServices(sd);
    try {
          DFService.register(genericAgent, dfd);     }
```

Figure 3. Code to register the service in the yellow pages

To unregister its services in the Yellow Pages an agent will present the code depicted in Figure 4. Analogously, all agents who need to find a specific provider agent in the Yellow Pages will present the code depicted in Figure 5.

```
try {
        DFService.deregister(genericAgent);     }
```

Figure 4. Code to unregister the service in the yellow pages

The *(un)register services* and *search provider agent* concerns (henceforth YPProvider and YPClient concerns, respectively) become tangled with the agents concerns and scattered in the system. To solve this issue, we can use aspect oriented abstractions both in the design and implementation phases. In the design phase we need to model crosscutting concerns separately from agent concerns and this can be achieved by using a proper notation. In the implementation phase, we need an implementation environment to codify crosscutting concerns separately from agents concerns and this can be achieved by integrating JADE and AspectJ, for example.

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(genericAgent.getServiceType());
template.addServices(sd);
try {
      DFAgentDescription[]  result  =  DFService.search(genericAgent,
template);
      for (int i = 0; i < result.length; ++i) {
        genericAgent.getProviders()[i] = result[i].getName();     }
```

Figure 5. Code to search in the yellow pages the agent which provides a service

3.2. Modeling Aspects in MAS

To model crosscutting concerns in MAS we propose an approach that specializes the agency metamodel for MAS architectural diagram [Silva et al. 2006] by using the concept of model roles [Kim et al. 2002]. In fact, the concept of model roles was used in the Pattern Specifications technique [France et al. 2004] to specialize the UML metamodel for specifying what model elements must participate in a pattern. The works presented in [Whittle and Araújo 2004] and [Araújo et al. 2004] use the Pattern Specifications (PS) technique [France et al. 2004] to address aspects modeling. Analogously, we can use the same principle of PS to promote the separation of crosscutting concerns in MAS modeling. In particular, our approach defines, through model roles, subtypes of agency metaclasses describing the MAS architectural diagram elements (e.g., Agent, AgentConnector agency metaclasses) which must participate in the aspect description. Formally, a role defines a subtype of an agency metaclass. The metaclass is called the base of the role. For example, a role that has the metaclass AgentConnector as its base specifies a subset of MAS agent connectors. A MAS model element conforms to (or plays) a role if it satisfies the properties defined in the role, that is, the element is an instance of the subtype defined by the role. Roles are identified by preceding them with a vertical bar, "|" (see Figure 6).

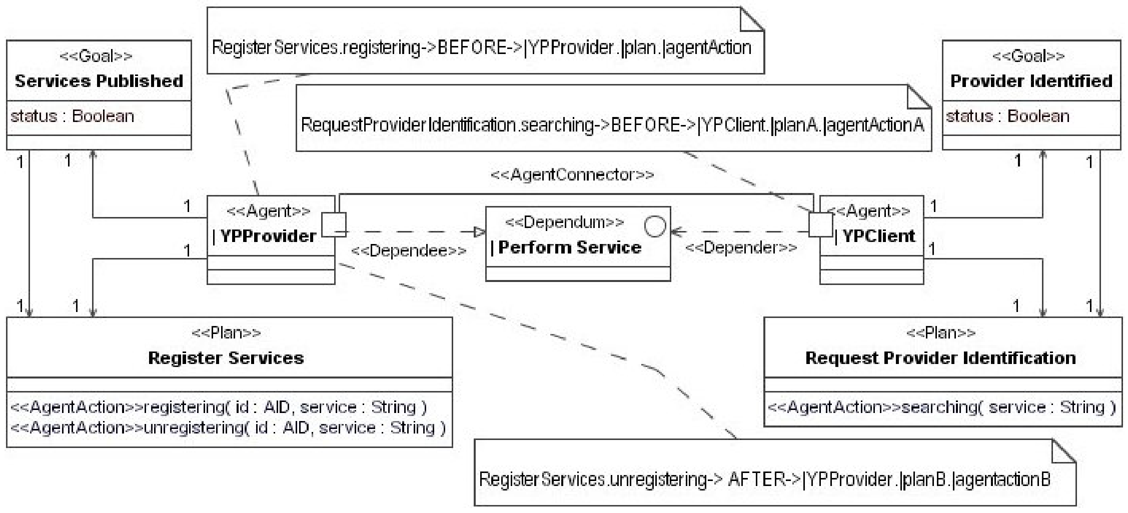

RegisterServices.registering->BEFORE->|YPProvider.|plan.|agentAction
RequestProviderIdentification.searching->BEFORE->|YPClient.|planA.|agentActionA
<<Goal>>
Services Published
status : Boolean
<<Goal>>
Provider Identified
status : Boolean
<<AgentConnector>>
<<Agent>>
| YPProvider
<<Dependum>>
| Perform Service
<<Agent>>
| YPClient
<<Dependee>>
<<Depender>>
<<Plan>>
Register Services
<<AgentAction>>registering(id : AID, service : String)
<<AgentAction>>unregistering(id : AID, service : String)
<<Plan>>
Request Provider Identification
<<AgentAction>>searching(service : String)
RegisterServices.unregistering-> AFTER->|YPProvider.|planB.|agentactionB
1
1

Figure 6. The DF Aspectual Diagram

The DF Aspectual Diagram in Figure 6 consists of two Agent roles (YPClient and YPProvider), and one Dependum role (Perform Service). The YPClient and YPProvider roles define subtypes of Agent metaclass, while the Perform Service role defines a subtype of Depedum metaclass in the agency metamodel. Moreover, we have the model roles present in the notes which describe the agents' points affected by the aspects. To weave the aspects with the MAS, we need to instantiate each model role present in the Aspectual Diagram (e.g., |YPClient, |PerformPlan, |plan, etc.).

3.3. From UML-based Agent Notation to JADE and AspectJ code

In this section we provide some guidelines to map our notation to represent MAS (Figure1) and aspects in MAS (Figure 6) in constructs of JADE and AspectJ implementation environments, respectively.

In the affected Architectural Diagram: (i) each «Plan» presented in Figure 1 becomes a Behavior in JADE; (ii) each «AgentAction» becomes a common method; (iii) each «Agent» becomes an Agent in JADE; (iv) the goal element is not mapped to any JADE construct, since JADE does not support the implementation of cognitive agents.

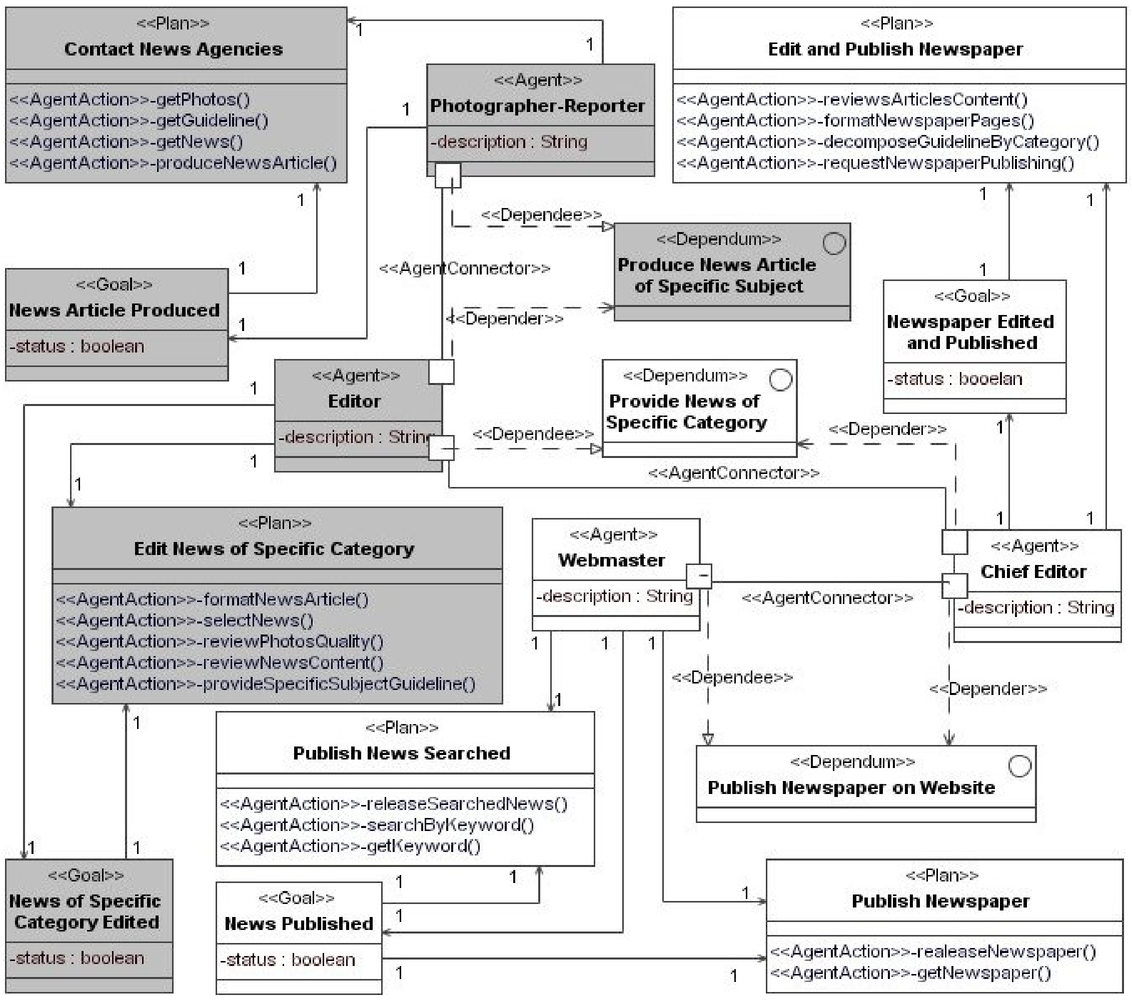
In the Aspectual Diagram: (i) each «Plan» presented in Figure 6 becomes an Aspect in AspectJ; (ii) each «AgentAction» becomes a Join Point in the aspect it belongs to; (iii) each «Agent» is a module affected by a aspect; (iv) each note attached to an agent becomes a Pointcut; (v) the goal element is not mapped to any AspectJ construct.

4. An Example

To illustrate the usage of our approach, we consider the domain of Newspaper Office introduced in [Silva, 2005]. The e-News system (Figure 7) enables a user to read news by accessing the newspaper website maintained by a Webmaster agent which is responsible for updating the published information. The information to be published is provided by the Chief Editor agent. The Chief Editor agent depends on the Editor agent to have the news of a specific category. For example, an Editor may be responsible for political news, while another one may be responsible for sports news. Each Editor contacts one or many Photographers-Reporters which can find the news of specific categories (e.g., about sport news). The Chief Editor then edits the Editor' news and forwards them to the Webmaster to publish them.

4.1. Architectural diagram

We start by proposing the architectural solution for the e-News problem (Figure 7) by using the MAS architectural pattern depicted in Figure 1. The e-News system is composed of four agents: Editor, Webmaster, Chief Editor and Reporter. For example, in Figure 7 the shaded area corresponds to the interaction between the Editor and Photographer-Reporter agents to achieve the service *Produce News of Specific Subject*. The Editor agent intends to achieve the *News of Specific Category Edited* goal by means of the *Edit News of Specific Category* plan. However, to edit the news the Editor agent has to request the Photographer-Reporter agent to perform the *Produce News of Specific Subject* service. The Photographer-Reporter performs the requested service because it does not conflict with the achievement of the *News Article Produced* goal. Hence, both the requested service and the goal achievement are accomplished by means of the *Contact News Agencies* plan.



<<Plan>>
Contact News Agencies
<<AgentAction>>-getPhotos()
<<AgentAction>>-getGuideline()
<<AgentAction>>-getNews()
<<AgentAction>>-produceNewsArticle()
<<Agent>>
Photographer-Reporter
-description : String
<<Plan>>
Edit and Publish Newspaper
<<AgentAction>>-reviewsArticlesContent()
<<AgentAction>>-formatNewspaperPages()
<<AgentAction>>-decomposeGuidelineByCategory()
<<AgentAction>>-requestNewspaperPublishing()
<<Dependee>>
<<Dependum>>
Produce News Article of Specific Subject
<<AgentConnector>>
<<Goal>>
News Article Produced
-status : boolean
<<Depender>>
<<Goal>>
Newspaper Edited and Published
-status : booelan
<<Agent>>
Editor
-description : String
<<Dependum>>
Provide News of Specific Category
<<Dependee>>
<<Depender>>
<<AgentConnector>>
<<Plan>>
Edit News of Specific Category
<<AgentAction>>-formatNewsArticle()
<<AgentAction>>-selectNews()
<<AgentAction>>-reviewPhotosQuality()
<<AgentAction>>-reviewNewsContent()
<<AgentAction>>-provideSpecificSubjectGuideline()
<<Agent>>
Webmaster
-description : String
<<AgentConnector>>
<<Agent>>
Chief Editor
-description : String
<<Dependee>>
<<Depender>>
<<Plan>>
Publish News Searched
<<AgentAction>>-releaseSearchedNews()
<<AgentAction>>-searchByKeyword()
<<AgentAction>>-getKeyword()
<<Dependum>>
Publish Newspaper on Website
<<Goal>>
News of Specific Category Edited
-status : boolean
<<Goal>>
News Published
-status : boolean
<<Plan>>
Publish Newspaper
<<AgentAction>>-realeaseNewspaper()
<<AgentAction>>-getNewspaper()

Figure 7. MAS Architectural Diagram for the e-News System

4.2. Weaving the DF Aspect Model

We finish by weaving the DF aspect with the architectural design of e-News system. To achieve this we need to instantiate each model role present in the structure of the aspect (e.g., |Client, |Provider). For example, the following bindings represent the instantiations of the model roles present in the DF Aspectual model (Figure 6) for the e-News system:

1. Bind |*YPClient* to *Editor*
2. Bind |*YPProvider* to *Photographer-Reporter*
3. Bind |*Perform Service* to *Produce News Article of Specific Subject*
4. Bind |*plan.*|*agentAction* to *ContactNewsAgencies.getGuideline*
5. Bind |*planA.*|*agentActionA* to *EditNewsSpecificCategory. provideSpecificSubjectGuideline*
6. Bind |*planB.*|*agentactionB* to *ContactNewsAgencies.produceNewsArticle*

Figure 8 shows the aspect applied to the problem. Thus, the Editor agent can search for Photographer-Reporter agents at run time, the Photographer-Reporter agent can publish its services in the yellow pages service, which ensures the decoupling among agents. Hence, if a Photographer-Reporter contacted by an Editor stops running, for example, the Editor may replace that Photographer-Reporter by locating another one in the yellow pages.

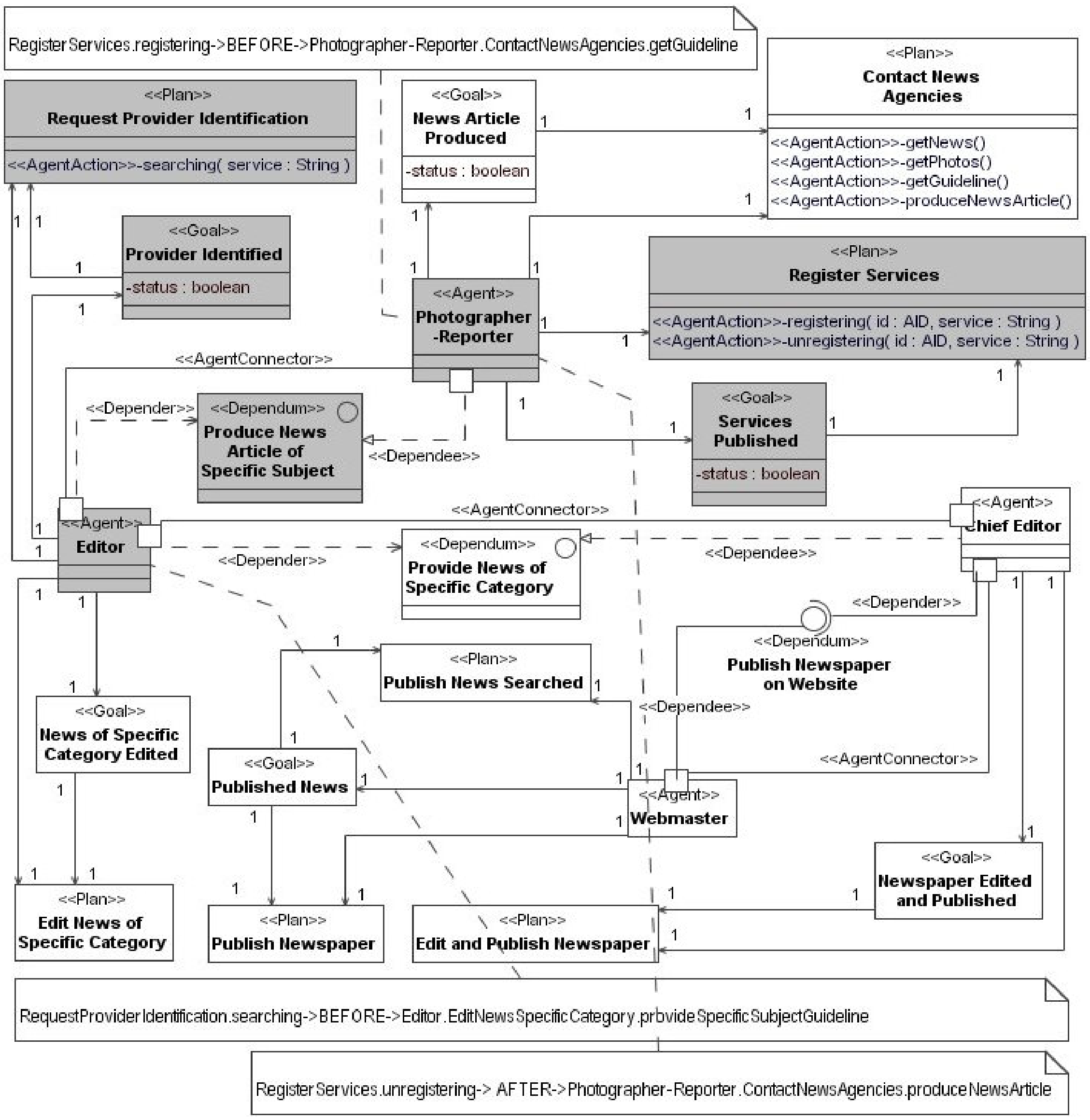

RegisterServices.registering->BEFORE->Photographer-Reporter.ContactNewsAgencies.getGuideline
<<Plan>> Request Provider Identification
<<AgentAction>>-searching(service : String)
<<Goal>> News Article Produced
-status : boolean
<<Plan>> Contact News Agencies
<<AgentAction>>-getNews()
<<AgentAction>>-getPhotos()
<<AgentAction>>-getGuideline()
<<AgentAction>>-produceNewsArticle()
<<Goal>> Provider Identified
-status : boolean
<<Agent>> Photographer-Reporter
<<Plan>> Register Services
<<AgentAction>>-registering(id : AID, service : String)
<<AgentAction>>-unregistering(id : AID, service : String)
<<AgentConnector>>
<<Depender>>
<<Dependum>> Produce News Article of Specific Subject
<<Dependee>>
<<Goal>> Services Published
-status : boolean
<<Agent>> Editor
<<Agent>> Chief Editor
<<Dependum>> Provide News of Specific Category
<<Dependum>> Publish Newspaper on Website
<<Plan>> Publish News Searched
<<Goal>> News of Specific Category Edited
<<Goal>> Published News
<<Agent>> Webmaster
<<Goal>> Newspaper Edited and Published
<<Plan>> Edit News of Specific Category
<<Plan>> Publish Newspaper
<<Plan>> Edit and Publish Newspaper
RequestProviderIdentification.searching->BEFORE->Editor.EditNewsSpecificCategory.provideSpecificSubjectGuideline
RegisterServices.unregistering-> AFTER->Photographer-Reporter.ContactNewsAgencies.produceNewsArticle

Figure 8. MAS Architectural Diagram weaved with the Aspectual Diagram

4.3. The Partial AspectJ Code for the e-News System

In this section we show how the Aspectual Diagram introduced in Figure 6 can be codified using the AspectJ environment. To achieve this, we use the mapping guidelines presented in section 3.1. According to these guidelines, the plans Register Services and Request Provider Identification become aspects in AspectJ. The agent actions for registering, unregistering and searching become join points in its corresponding aspects. The agents IYPProvider and IYPClient are the agents affected by the aspects, i.e., the Photographer-Reporter and Editor agents. The notes become pointcuts in their corresponding aspects. For example, in Figure 9 we present the RegisterServices aspect weaved with the PhotographerReporter agent in the e-News System. The code after the comments “// Register…” and “// Unregister…” are the ones presented in Figure 3 and Figure 4, respectively.

```
public aspect RegisterServices {
pointcut     registering()     :     call     (void     Photographer-
Reporter.getGuideline());
pointcut    unregistering()     :     call     (void     Photographer-
Reporter.produceNewsArticle ());
before() : registering() {
Photographer-Reporter    genericAgent     =     (Photographer-Reporter)
thisJoinPoint.getThis();
// Register the PhotographerReporter service in the yellow pages
...}
after() : unregistering() {
Photographer-Reporter    genericAgent     =     (Photographer-Reporter)
thisJoinPoint.getThis();
// Unregister from the yellow pages
...}
```

Figure 9. AspectJ Code for RegisterServices Aspect

Figure 10 presents the RequestProviderIdentification aspect weaved with the Editor agent in the e-News System. The code after the comments “// Update…” is the one presented in Figure 5. For the sake of space we could not illustrate the use of the mapping guidelines for the JADE environment.

```
public aspect RequestProviderIdentification {
     pointcut searching() : call (void Editor.provSpecSubjGuide(..));
     before() : searching() {
     Editor genericAgent = (Editor) thisJoinPoint.getArgs()[0];
// Update the list of PhotographerReporter agents
...)
```

Figure 10. AspectJ Code for RequestProviderIdentification Aspect

5. Related Work

Aspect- oriented modeling requires the use of a higher-level aspect model that addresses the aspect oriented programming concepts at a preliminary design stage (avoiding language specific details), and allows the designer to work at a more abstract level during software construction and evolution.

Aspect-Oriented Design Model (AODM) [Stein 2002] enhances the existing UML specification with aspect-oriented concepts that reproduces the crosscutting characteristics of the AspectJ language. The aSideML [Chavez et al. 2005] 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 object-oriented systems. Composition patterns [Clarke and Walker 2001] is an approach that handles crosscutting concerns at design level, by means of templates. However, these approaches do not address MAS directly.

[Garcia 2004] argues that the design and implementation of agent internal architecture concerns (e.g. interaction, adaptation, autonomy, knowledge, collaboration, roles, learning and mobility) tend to affect or crosscut many classes of the system design and code, including those representing the basic agent functionality. To address this issue, [Garcia 2004] proposes an aspect-oriented approach to support the separate handling and modularization of multi-agent systems specific concerns. The proposed approach encourages the separate handling of agent properties, and provides a disciplined scheme for their composition. In our approach, we are concerned with the separation of crosscutting concerns which are not related to the internal characteristics of an agent, since we consider the agent already built-in and supported by some implementation platform.

In [Silva et al. 2006a], we have extended the aSideML [Chavez et al. 2005] to incorporate agency features to be used in both the separation of crosscutting concerns in MAS and its latter weaving with the system agents. However, this approach becomes complex because we extended the aSideML by using agency features, architectural features and model roles. On the other hand, our current work is quite simple since to model crosscutting concerns in MAS we have created an agency metamodel and specialize it by using model roles.

6. Conclusions and Future Work

This work presents an approach to separate crosscutting concerns in multi-agent systems. This approach comprises a notation to describe the crosscutting concerns separately from MAS functional modules (i.e. agents) and a systematic way for weaving the crosscutting concerns with the agents. We have exemplified the presence of a crosscutting concerns in a MAS, implemented in JADE. This crosscutting concern was separated and modularized by using model roles to specialize the agency metamodel for MAS architectural diagram. To weave the aspects with the system agents we just need to instantiate the model roles present in the specification of the aspects by using specific elements of the application. We also present how a system specified according to our approach can be implemented using an integration of JADE and AspectJ.

Future work includes developing a tool to support both the notation and the generation code, as well as to apply our approach in other case studies. Moreover, we are applying our approach for crosscutting concerns in the description of agent-oriented design patterns.

Acknowledgements

This work was supported by several research grants (CNPq Proc. 304982/2002-4, Proc. 142248/2004-5, CAPES Proc. BEX 1775/2005-7 & CAPES/ GRICES Proc. 129/05).

References

Araújo, J., Whittle, J. and Kim, D. (2004) “Modeling and Composing Scenario-Based Requirements with Aspects”, In: RE’04, Kyoto, Japan.

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: Net.ObjectDays’04, AgentExpo.

Chavez, C., Garcia, A., Kulesza, U., Sant’Anna, C., Lucena, C. (2005) “Taming Heterogeneous Aspects with Crosscutting Interfaces”, In: SBES’05, Uberlândia, Minas Gerais, Brasil, p. 216 - 231.

Clarke, S. and Walker, R. (2001) “Composition Patterns: An Approach to Designing Reusable Aspects”, In: ICSE’01, Toronto, Canada, p. 12 – 19.

Elrad, T., Filman, R. and Bader, A. (2001) “Aspect-Oriented Programming”, In: Introduction, Communications of the ACM, 44, 10, p.29 – 32.

France, F., Kim, D., Ghosh, S., Song, E. (2004) “A UML-Based Pattern Specification Technique”, In: IEEE Transactions on Software Engineering, 30, 3, p. 193 – 206.

Garcia, A. (2004) “From Objects to Agents: An Aspect-Oriented Approach”, PhD Thesis, Computer Science Department, PUC-Rio, Rio de Janeiro, Brazil.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J. (1997) “Aspect-Oriented Programming”, In: ECOOP’97, Springer-Verlag, Finland.

Kim, D., France, R., Ghosh, S. and Song, E. (2002) “Using Role-Based Modeling. Language (RBML) as Precise Characterizations of Model Families”, In: IEEE ICECCS’02, Greenbelt, MD.

Odell, J., Parunak, H. and Bauer, B. (2000) “Extending UML for agents”, In: AOIS’00 at the 17th National Conference on AI, Austin, TX, USA. iCue Publishing, p. 3 – 17.

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

Silva, V. and Lucena, C. (2004) “From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language”, In: JAAMAS’04, Edited by K. Sycara et al., Kluwer Academic Publishers, 9, 1-2, p. 145 – 189.

Silva, I. (2005) Design and Implementation of Multi-agent Systems: The Tropos Case. (In portuguese: Projeto e Implementação de Sistemas Multi-Agentes: O Caso Tropos), Master Thesis, CIn, Universidade Federal de Pernambuco.

Silva, C., Castro, J., Araújo, J., Moreira, A., Alencar, F. and Ramos, R. (2006) “Separation and Modularization of Crosscutting Social Patterns in Detailed Architectural Design”, In: CAiSE’06 Forum, Tudor, Luxemburgo.

Silva, C., Araújo, J., Moreira, A., Castro, J., Tedesco, P., Alencar, F. and Ramos, R. (2006) “Modeling Multi-Agent Systems using UML”, In: SBES’06, Florianópolis, Santa Catarina, Brasil (to appear).

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

Stein, D. (2002) An Aspect-Oriented Design Model Based on AspectJ and UML, Master Thesis, University of Essen, January.

Whittle, J., Araújo, J. (2004) “Scenario Modeling with Aspects”, In: IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Eng. and Architecture Design, Edited by A. Rashid, A. Moreira, B. Tekinerdogan.

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.