
# PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 32/08

# Extending PASSI to Model Multi-agent Systems Product Lines

**Ingrid Oliveira de Nunes**
**Uirá Kulesza**
**Camila Patrícia Bazílio Nunes**
**Elder José Reioli Cirilo**
**Carlos José Pereira de Lucena**

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**
**RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900**
**RIO DE JANEIRO - BRASIL**

Monografias em Ciência da Computação, No. 32/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena August, 2008

# Extending PASSI to Model Multi-agent Systems Product Lines [1]

**Ingrid Oliveira de Nunes[1], Uirá Kulesza[2], Camila Patrícia Bazílio Nunes[1], Elder José Reioli Cirilo[1], Carlos José Pereira de Lucena[1]**

[1] PUC-Rio, Computer Science Department, LES - Rio de Janeiro - Brazil
[2] Federal University of Rio Grande do Norte (UFRN) - Natal - Brazil

ioliveira@inf.puc-rio.br, uira@dimap.ufrn.br, camilan@inf.puc-rio.br,
ecirilo@inf.puc-rio.br, lucena@inf.puc-rio.br

**Abstract.** Multi-agent System Product Lines (MAS-PLs) have emerged to integrate software product lines (SPLs) and agent-oriented software engineering techniques by incorporating their respective benefits and helping the industrial exploitation of agent technology. Some approaches have been proposed in this context; however, they do not address development scenarios of traditional SPL architectures using agent abstraction. In this paper, we present a new approach for modeling MAS-PLs, focusing the domain analysis stage. Our approach is based on PASSI methodology and incorporates some extensions to address agency variability. A case study, OLIS (OnLine Intelligent Services), illustrates our approach.

**Keywords:** Multi-agent Systems, Software Product Lines, Methodology, Software Reuse.

**Resumo.** Linhas de Produto de Sistemas Multi-agents (LP-MASs) surgiram para integrar linhas de produto de software (LPS) e técnicas de engenharia de software orientada a agentes pela incorporação dos seus respectivos benefícios e auxiliando na exploração industrial da tecnologia de agentes. Algumas abordagens foram propostas neste contexto; entretanto, elas não atendem o desenvolvimento de cenários tradicionais das arquiteturas de LPS usando a abstração de agentes. Neste artigo, apresentamos uma nova abordagem para modelar LP-MASs, focando no estágio da análise de domínio. Nossa abordagem é baseada na metodologia PASSI e incorpora algumas extensões que focam na variabilidade de agência. Um caso de estudo, OLIS (*OnLine Intelligent Services*), ilustra a nossa abordagem.

**Palavras-chave:** Sistemas Multi-agentes, Linhas de Produto de Software, Metodologia, Reuso de Software.

[1]This work has been sponsored by Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

**In charge of publications:**
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

# Contents

# 1 Introduction

Over the last years, agents have become a powerful technology to allow the development of distributed complex applications. Software agents are a natural high-level abstraction that helps understanding and modeling this kind of systems. Agents usually present some particular properties (Wooldridge & Ciancarini 2000), such as autonomy, reactivity, pro-activeness and social ability; therefore they facilitate to develop systems that present autonomous behavior. A behavior is considered autonomous when it requires artificial intelligence techniques or the system accomplishes actions previously performed by users. Several methodologies (Sellers & Giorgini 2005) have been proposed in order to allow the development of Multi-agent Systems (MASs). However, most of them do not take into account the adoption of extensive reuse practices that can bring an increased productivity and quality to the software development.

Software product lines (SPLs) (Pohl, Böckle & van der Linden 2005, Clements & Northrop 2002) have emerged as a new trend of software reuse investigating methods and techniques in order to build and customize families of applications through a systematic method. Clements & Northrop (Clements & Northrop 2002) define a SPL as "a set of software intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". According to (Czarnecki & Helsen 2006), a feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among products in a product line. The main aim of SPL engineering is to analyze the common and variable features of applications from a specific domain, and to develop a reusable infrastructure that supports the software development. This set of applications is called a family of products.

Only recent research (Pena, Hinchey, Resinas, Sterritt & Rash 2007, Dehlinger & Lutz 2005) has explored the integration between SPL and MAS. The aim of these new approaches is to integrate SPL and agent-oriented techniques by incorporating their respective benefits and helping the industrial exploitation of agent technology. Nevertheless, these approaches use a SPL perspective for particular purposes and do not address the development of SPLs to derive MAS.

In this context, this work presents an approach for modeling MAS-PLs. We aim at proposing a methodology that covers the full SPL development process. However, in this paper we focus at the domain analysis stage. Our previous work (Nunes, Kulesza, Nunes & Lucena 2008) allowed us to identify particular kinds of variability of MAS-PLs and how effective the SPL methodologies are for documenting them. We now propose an approach that extends PASSI (Cossentino 2005), an agent-oriented methodology, to support the management of SPL variabilities. PASSI provides a useful way for specifying a MAS, although it considers the development of single systems. We motivate and illustrate this work with the OLIS case study, a product line of systems that provides different services to the user, such as calendar and events announcement.

The remainder of this paper is organized as follows. Some works related to multi-agent systems and product lines are described in Section 2. In Section 3, an overview of the OLIS case study is presented, giving some details about its development. In Section 4, we show how we have modeled our product line at the domain analysis stage, based on a PASSI extension. We present some discussions in the Section 5. Finally, the conclusions and directions for future works are discussed in Section 6.

## 2 Related Work

Over the past few years, several methods have been published to address the problems and challenges of SPL engineering. FORM (Kang, Kim, Lee, Kim, Shin & Huh 1998) extended FODA (Kang, Cohen, Hess, Novak & Peterson 1990) to cover the entire spectrum of domain and application engineering, including the development of reusable architectures and code components. Pohl et al (Pohl et al. 2005) propose a framework for SPL engineering that defines the key sub-processes of the domain engineering and application engineering process as well as the artefacts produced and used in these processes. PLUS (Gomaa 2004) provides a set of concepts and techniques to extend UML-based design methods and processes for single systems in a new dimension to address software product lines. In our previous work (Nunes, Kulesza, Nunes & Lucena 2008), we have identified that most of the SPL methodologies provide useful notations to model the agency features. However, none of them completely covers their specification. Agent technology provides particular characteristics that need to be considered in order to take advantage of this paradigm.

On the other hand, many agent-oriented methodologies have been proposed. Tropos (Bresciani, Perini, Giorgini, Giunchiglia & Mylopoulos 2004) provides guidance for the four major development phases of application development (Early requirements, Late requirements, Architectural design and Detailed design). PASSI (Cossentino 2005) brings a particularly rich development lifecycle that spans initial requirements though deployment and, in addition, emphasizes the social model of agent-based systems. Using the analogy of human-based organizations, Gaia (Wooldridge, Jennings & Kinny 2000) provides an approach that both a developer and a non-technical domain expert can understand. A complete overview and comparison of MAS methodologies is presented in (Sellers & Giorgini 2005). A particular objective of our study was to find out how these methodologies can be used to help on the development of MAS-PLs.

Recent research work has investigated the integration synergy of MASs and SPLs technologies. Dehlinger & Lutz (Dehlinger & Lutz 2005) have proposed an extensible agent-oriented requirements specification template for distributed systems that supports safe reuse. Pena et al (Pena et al. 2007) present an approach to describing, understanding, and analyzing evolving systems. The MAS-PL methodologies do not address development scenarios of traditional SPL architectures using agent technology. Instead, they adopt an existing MAS methodology as a base (Gaia and MaCMAS methodologies respectively) and extend it with SPL techniques for a particular purpose. The main problems that we have observed (Nunes, Kulesza, Nunes & Lucena 2008) in these MAS-PL methodologies to model and document MAS-PLs were: (i) they do not offer a complete solution to address the modeling of agency features in both domain analysis and design; and (ii) they suggest the introduction of complex and heavyweight notations that are difficult to understand when adopted in combination with existing notations (e.g. UML) and do not capture explicitly the separated modeling of agency features.

## 3 OLIS Case Study

According to (Krueger 2002), there are different SPLs adoption strategies. The proactive approach motivates the development of product lines considering all the products in the foreseeable horizon. A complete set of artifacts to address the product line is developed

from scratch. In the extractive approach, a SPL is developed starting from existing software systems. Common and variable features are extracted from these systems to derive an initial version of the SPL. Finally, the reactive approach advocates the incremental development of SPLs. Initially, the SPL artifacts address only a few products. When there is a demand to incorporate new requirements or products, the common and variable artifacts are incrementally extended in reaction to them. We have developed two MAS-PLs to drive our study and both followed the reactive approach. The first one is a conference management web system, the ExpertCommittee, on which we added autonomous behavior. More details can be seen in (Nunes, Nunes, Kulesza & Lucena 2008) and (Nunes, Kulesza, Nunes & Lucena 2008). The second one, the OLIS, is detailed in this section. OLIS case study explores the BDI (belief-desire-intention) model (Rao & Georgeff 1995). It will be used to illustrate our approach in the next section.

The OLIS (**O**n**L**ine **I**ntelligent **S**ervices) case study is a web application that provides several personal services to users. The first version of the system is composed mainly by two services: the Events Announcement and the Calendar services. The Events Announcement service allows the user to announce events to other system users through an events board. The Calendar service lets the user to schedule events in his/her calendar. Announced events can be imported to the users' calendar. However, OLIS was designed in such a way that the system can be evolved to incorporate new services without interfere in the existing ones. The system has different flavors according to the type of event that it manages, such as: generic events, academic events and travel events.

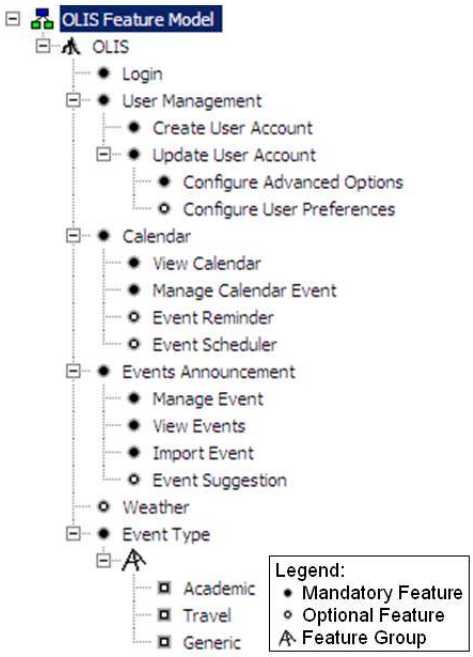


Figure 1: OLIS Feature Model.

After developing the first version of OLIS web application, new autonomous behavior features were introduced to automate some tasks in the system. We evolved the OLIS application, adding new features to it, which take advantage of the agents technology. The services become intelligent services. Figure 1 shows the feature model of the system.

The new features incorporated to the OLIS first version are: (i) *Events reminder* - the user configures how many minutes he/she wants to be reminded before the events, and the system sends messages to notify the user about events that are about to begin; (ii) *Events scheduler* - when an user adds a new calendar event that involves more participants, the system checks the schedule of the other participants to verify if this event conflicts with other existing ones. If so, the system suggests a new date for the calendar event that is appropriate according to all schedules from participants ; (iii) *Events Suggester* - when a new event is announced, the system automatically recommends the event after checking if it is interesting to the users based on their preferences. This feature is also responsible to check if the weather is going to be appropriate according to the place type where the event is going to take place; (iv) *Weather* - this is a new user service. It provides information about the current weather conditions and the forecast of a location. This service is also used by the system to recommend announced travel events.

The evolution of OLIS web application was accomplished by the introduction of software agents and their respective roles on the architecture. Agents were implemented using JADE[2] and Jadex[3] platforms. There are five different types of agents in the system: (i) EnvironmentAgent – perceives changes in the data model and propagates them to the other agents; (ii) FacadeAgent – retrieves information from agents to the business services, it is a facade between the web application and the agents; (iii) WeatherAgent – provides information about the weather and weather forecast; (iv) ManagerAgent – starts the UserAgents when the system starts up or when a new user is inserted in the system; (v) UserAgent – each user has an agent that represents him/her in the system. It acts in the name of the user. Each UserAgent is composed by roles, which implement agency features. For example, the EventScheduler and EventParticipant roles implement the Events scheduler feature.

# 4 Modeling OLIS in Domain Analysis with a PASSI extension

In this section, we present our approach for modeling MAS-PLs, which is based on the PASSI methodology. Due to the deficiencies and lack of expressivity for documenting variability, we propose extensions to document the agency features in MAS-PLs. We focus in this paper specifically on the domain analysis stage, yet the main aim of our work is to define a set of guidelines to model and document agency features along all SPL development stages. The case study previously presented (Section 3) is used to illustrate our extensions. It is important to notice that although the need to clearly modeling the agency features became from the incremental development of OLIS and ExpertCommittee, the extension proposed here can also be useful when adopting proactive and reactive development strategies.

PASSI (Process for Agent Societies Specification and Implementation) (Cossentino 2005) is an agent-oriented methodology that specifies five models with their respective phases for developing MASs. The methodology covers all the development process, from requirements to code. The five different models from PASSI are System Requirements

[2] vsis-www.informatik.uni-hamburg.de/projects/jadex/

[3] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

Model, Agency Society Model, Agent Implementation Model, Code Model and Deployment Model. The domain analysis stage corresponds to the System Requirements Model, which generates a model of the system requirements in terms of agency and purpose. PASSI follows one specific guideline: the use of standards whenever possible; and this justifies the use of UML as modeling language. However, the UML semantics and notation is extended to design specific needs of agents. PASSI methodology is designed for developing single systems, therefore we had to adapt it to express variability.

### 4.1 Feature Modeling

Feature modeling is an important activity in SPLs. It is the activity of modeling the common and variable properties of concepts and their interdependencies (Czarnecki 1998). The features are organized into a coherent model referred to as a feature model, which specifies the features of a product line as a tree, indicating mandatory, optional and alternative features. Features are essential abstractions that both customers and developers understand. Originally, the feature model was proposed in (Kang et al. 1990).

Figure 1 illustrates the OLIS feature model. It shows its different kinds of features: (i) Mandatory – features that are in all the versions of the system and are part of its core. Examples are the Calendar and Events Announcement features; (ii) Optional – features that made part of only some versions of the MAS-PL, such as the Event Scheduler and Event Suggester features; and (iii) Alternative – features that varied from one version to another one. There are different types of events and one of them must be chosen in the product derivation process (Cirilo, Kulesza & Lucena 2008). Besides the feature model, constraints express the feature interdependencies. Some features can depend on another, e.g. the Weather feature must be present if the Event Suggester and Travel Event Type are selected, and some features are mutually exclusive, the event type illustrates this constraint.

### 4.2 Domain Requirements Description

According PASSI, in the Domain Requirements Description phase, we make a functional description of the system composed of a hierarchical series of use case diagrams. In order to enable the variabilities modeling, we have adapted these PASSI diagrams using the PLUS method notation. In the PLUS approach, stereotypes are used to indicate if a use case is mandatory (kernel), alternative or optional. The method also proposes a feature dependency table to map use cases to each feature. Figure 2 shows a partial view of the OLIS MAS-PL use case model.

Besides the stereotypes, we also colored the use cases to indicate which feature they are related to. This indication is used in all artifacts. However, we also provide another view to the use cases to better express this relation: we grouped them into features with the UML package notation, as PLUS proposes. In addition, we have adopted the <<*agency feature*>> stereotype to indicate that the use cases of a specific package is related to an agency feature (see Figure 3). We preferred to use the common use case descriptions to explain use cases, instead of using sequence diagrams, as PASSI suggests. Use case descriptions are widely used in the literature, and are also adopted by PLUS.

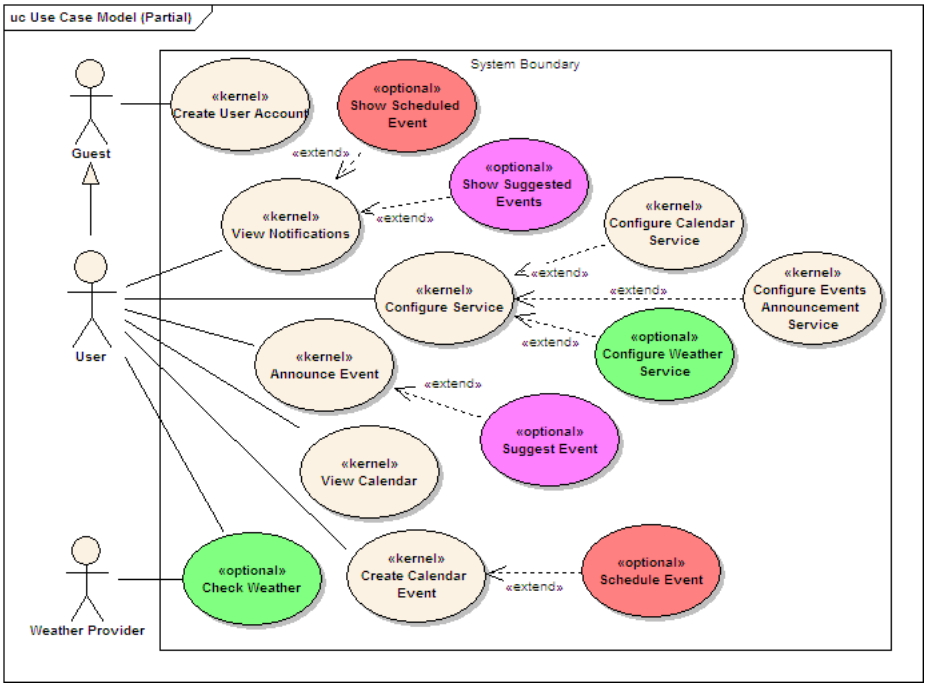


Figure 2: OLIS Use Case Diagram (Partial).

### 4.3 Agent Identification

The input of this phase are the use case diagrams generated in the Domain Requirements Description phase. Responsibilities are attributed to agents, which are represented as stereotyped UML packages. The OLIS Agent Identification Diagram is depicted in Figure 4.

PASSI methodology considers that all functionalities of the system are performed by agents. Agents are entities that usually presents autonomy and pro-activeness, so only the functionalities that have these characteristics need to be performed by agents. Thus, the use cases that are into a UML package stereotyped with <<*agency feature*>> will be considered to be performed by an agent. These use cases will be grouped into <<*agent*>> stereotyped packages so as to form a new diagram. Each one of these packages defines the functionalities of a specific agent.

In OLIS case study, each user of the system has an agent that represents it. This agent is an instance of the `User Agent`. We did not find how to express the communication between different instances of the same agent in PASSI, so you draw an arrow between the two use cases that goes outside the UML package and then comes back (see Figure 4).

### 4.4 Role Identification

In the Role Identification phase, all the possible paths (a "communicate" relationship between two agents) of the Agents Identification diagram are explored. A path describes a scenario of interacting agents working to achieve a required behavior of the system. In different scenarios, agents can play different roles. The agent interactions are expressed through sequence diagrams.

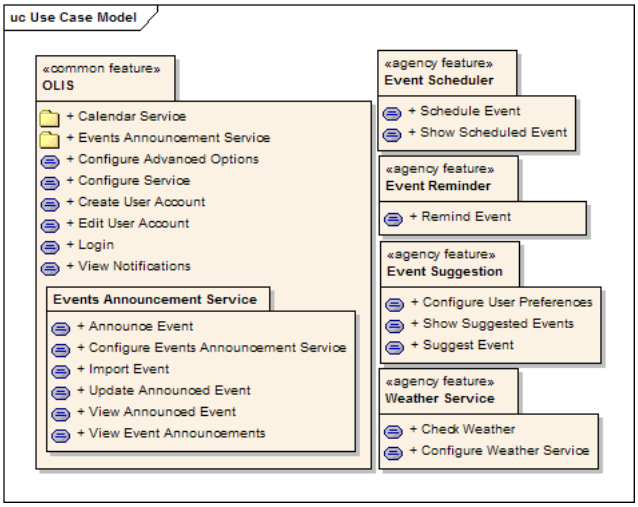


Figure 3: Grouped Use Cases (Partial).

Usually, each scenario corresponds to only one feature of the system. For these cases, a feature dependency table to map sequence diagrams to each feature is enough. However, there are some features that have impact in another feature. We say that the feature crosscuts the other. An example is the Event Suggestion feature of OLIS. When an event is inserted on the system, the UserAgent that represents the user who inserted it asks for the other user agents if they have interest on that event. So, the agent checks the availability of the user on the event date. Besides, if the event type is academic, the agent checks the areas of interest and the location of the event according to the user `AcademicPreferences`. And if the event type is travel, the agent checks the place type where the event is going to happen and the activities that can be done according to the user `TravelPreferences`; and the agent also consults the WeatherAgent to get the weather forecast and check if it will be good in the event date. Thus, according to the Event Type feature, the agents behavior change. The solution that we found for this problem is the use of UML fragments to express optional and alternative paths. Figure 5 illustrates this scenario. In this diagram, only the interaction among agents/roles are reported, the internal behavior of agents are specified in the next phase.

### 4.5 Task Specification

In the Task Specification phase, activity diagrams are used to specify the capabilities of each agent. According to PASSI, for every agent in the model, we draw an activity diagram that is made up of two swimlanes. The one from the right-hand side contains a collection of activities symbolizing the agent's tasks, whereas the one from the left-hand side contains some activities representing the other interacting agents.

In these diagrams, we have made three adaptations, some of them were already adopted in other diagrams: (i) instead of drawing only one diagram per agent, we split the diagram according to the features; (ii) use of UML fragments to show different paths when there is a crosscutting feature; (iii) a colored indication showing with which feature the task is related to. These adaptations can be seen in Figure 6. The main objective of this adaptations is to provide a better feature modularization and traceability. Splitting the

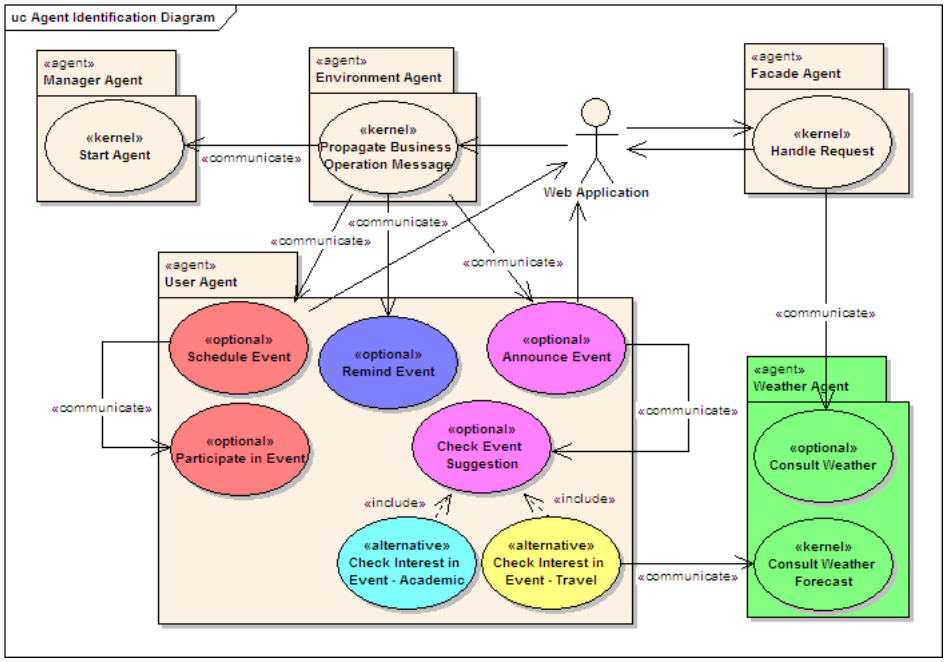


Figure 4: OLIS Agent Identification Diagram.

diagram in the way we proposed allow the selection of the necessary diagrams during the application engineering according to selected features. Nevertheless, crosscutting features could not be isolated from the others, and this is a challenge that we are still facing while developing of MAS-PLs (see Section 5).

Several PASSI extensions were proposed to model agency variabilities. Most of them came from PLUS (Gomaa 2004) approach, which provides useful notations to model SPLs (Nunes, Kulesza, Nunes & Lucena 2008). Table 1 summarizes the adaptations we proposed.

## 5 Discussions

In this section, we present and discuss some lessons learned while modeling agency features in MAS-PLs and challenges that we still have to face. These lessons learned offer directions for a methodology for developing MAS-PL that we are currently defining.

*Integration of SPL techniques with existing Multi-agent Methodologies.* Several MAS methodologies have been proposed (Bresciani et al. 2004, Cossentino 2005, Wooldridge et al. 2000). These methodologies have the same purpose of building agent-based systems; however each has its own unique perspective and approach to developing MASs. Though, no one methodology is useful in every system–development situation. One question that we had to deal while modeling MAS-PL is what methodology should be our start point. PASSI integrates concepts from object-oriented software engineering and artificial intelligence approaches. It uses an UML-based notation, and it brings facilities to the incorporation of notations already proposed for SPLs (Gomaa 2004). In addition, there are available tools for modeling and code generation. Some research work (Cossentino, Sabatucci & Chella 2003) promotes pattern reuse in PASSI methodology. It improves the possibility of code generation; nevertheless we did not discuss this in this paper.

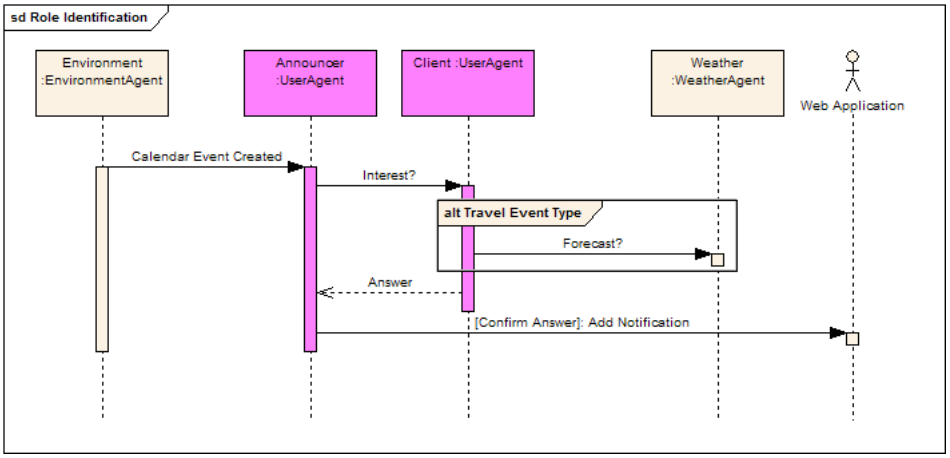


Figure 5: OLIS Role Identification Diagram.

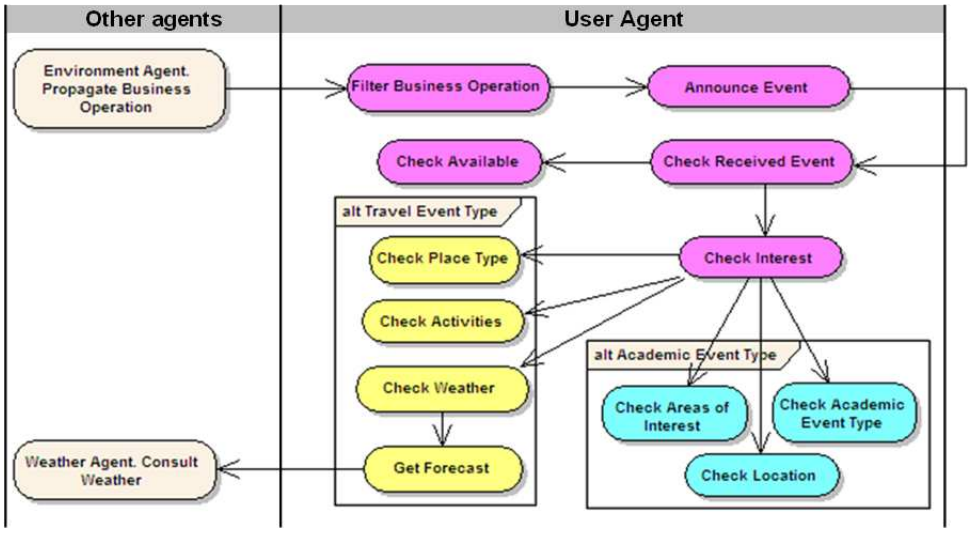


Figure 6: OLIS Task Specification Diagram.

*Explicit Separation of the Modeling and Implementation of MAS Features from other Technologies.* MAS methodologies usually propose to distribute all the system functionalities / responsibilities among agents. Agents are an abstraction that provides some particular characteristics, such as autonomy and pro-activeness. Therefore, we claim that features of the SPL that do not take advantage from agent technology can be modeled and implemented using typical programming techniques. In our approach, we adopted the *<<agency feature>>* stereotype to indicate the features that present autonomous behavior and should be modeled using agent abstraction. This let us use the several technologies that exist for improving the development of web applications, such as design patterns and persistence and web application frameworks.

*Granularity in Software Product Lines.* In the literature, there a many examples of SPLs with features with coarse granularity. This means that these features can be implemented wrapped in a specific unit, such as a class, a method or an agent. Although,

fine-grained extensions, e.g. adding a statement in the middle of a method, usually require the use of techniques, like conditional compilation, which obfuscate the base code with annotations. Though many SPLs can and have been implemented with the coarse granularity of existing approaches, fine-grained extensions are essential when extracting features from legacy applications (Kästner, Apel & Kuhlemann 2008). Our scope in this study was dealing with coarse-grained features; however we are currently extending OLIS MAS-PL by adding new fine-grained features to explore this scenario.

Table 1: PASSI extensions.

| **Phase** | **Extensions** |
|---|---|
| Feature Modeling | New phase |
| Domain Requirements Description | Use of Stereotypes (kernel, alternative or optional)<br>New use cases view (grouped in features)<br>Use of <<agency feature>> stereotype to indicate autonomous behavior features<br>Use of colors to trace features |
| Agent Identification | Only use cases in <<agency feature>> stereotyped packages are distributed among agents<br>Use of Stereotypes (kernel, alternative or optional)<br>Use of colors to trace features |
| Role Identification | Use of UML fragments (crosscutting features)<br>Use of colors to trace features |
| Task Specification | One diagram per agent and feature<br>Use of UML fragments (crosscutting features)<br>Use of colors to trace features |

*Crosscutting agency features.* Many of the agency features are implemented by a set of different system components, agents and classes. They are characterized as crosscutting features, because their design and implementation are typically spread and tangled along different system modules. Our approach do not provide clear support to deal with the documentation of these crosscutting features, however we are currently investigating how existing aspect-oriented approaches (Jacobson & Ng 2004, Clarke & Baniassad 2005) can help the visual documentation of the agency features. We already have some results in this direction, by making studies about agency features modularity (Nunes, Kulesza, Sant'Anna, Nunes & Lucena 2008).

## 6 Conclusions and Future Work

In this paper, we presented an approach for modeling MAS-PLs at the domain analysis stage. Our approach is based on PASSI methodology, which supported the specification of software agents. We have extended this methodology to address agency variabilities in product lines. An important phase that needs to be added in the methodology is the feature modeling, which is the activity that identifies the SPL common and variable features. In addition, we have extended the PASSI notation, using stereotypes to indicate the variable abstractions and components of the systems. Since PASSI is based on the UML notation, it allowed us to adopt notations from PLUS, an existing SPL approach. We also discussed some important topics that arose from our study, such as the use of object-oriented techniques in agent-based applications and the need of a clear support for crosscutting features.

Our focus in this paper was in the domain analysis stage, but we are currently working on the development of a methodology that allows an explicit documentation and tracing

of agency features throughout the SPL development process. Some SPL methodologies are not used in practice due to their high complexity. Thus, we aim at developing an agile and adaptable methodology. Our methodology is being organized as a process framework composed of: (i) a core - that defines a set of mandatory activities and artifacts; and (ii) specific customizations - that specify additional activities and artifacts to the core according to specific scenarios that need to be addressed. Tool support for the methodology based on model-driven engineering techniques is also under development.

## References


Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. (2004), 'Tropos: An agent-oriented software development methodology', *AAMAS 2004* **8**(3), 203–236.

Cirilo, E., Kulesza, U. & Lucena, C. (2008), 'A Product Derivation Tool Base on Model-Driven Techniques and Annotations', *Journal of Universal Computer Science* .

Clarke, S. & Baniassad, E. (2005), *Aspect-Oriented Analysis and Design: The Theme Approach (The Addison-Wesley Object Technology Series)*, Addison-Wesley Professional.

Clements, P. & Northrop, L. (2002), *Software Product Lines: Practices and Patterns*, Addison-Wesley, USA.

Cossentino, M. (2005), *From Requirements to Code with the PASSI Methodology*, Idea Group Inc., Hershey, PA, USA, chapter IV.

Cossentino, M., Sabatucci, L. & Chella, A. (2003), Patterns reuse in the passi methodology, *in* 'ESAW'03', Springer-Verlag, pp. 29–31.

Czarnecki, K. (1998), Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models, PhD thesis, Technical University of Ilmenau.

Czarnecki, K. & Helsen, S. (2006), 'Feature-based survey of model transformation approaches', *IBM Systems Journal* **45**(3), 621–645.

Dehlinger, J. & Lutz, R. R. (2005), A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems, *in* 'SELMAS 2005', ACM Press, USA, pp. 1–7.

Gomaa, H. (2004), *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, Addison Wesley Longman Publishing Co., Inc., USA.

Jacobson, I. & Ng, P.-W. (2004), *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series)*, Addison-Wesley Professional.

Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E. & Huh, M. (1998), 'Form: A feature-oriented reuse method with domain-specific reference architectures', *Ann. Softw. Eng.* **5**, 143–168.

Kang, K., Cohen, S., Hess, J., Novak, W. & Peterson (1990), Feature-oriented domain analysis (foda) feasibility study, Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie-Mellon University.

Kästner, C., Apel, S. & Kuhlemann, M. (2008), Granularity in software product lines, *in* 'ICSE '08', ACM, New York, NY, USA, pp. 311–320.

Krueger, C. W. (2002), Easing the transition to software mass customization, *in* 'PFE '01', Springer-Verlag, London, UK, pp. 282–293.

Nunes, C., Kulesza, U., Sant'Anna, C., Nunes, I. & Lucena, C. (2008), On the modularity assessment of aspect-oriented multi-agent systems product lines: a quantitative study, *in* 'SBCARS 2008', Porto Alegre, Brazil.

Nunes, I., Kulesza, U., Nunes, C. & Lucena, C. (2008), Documenting and modeling multi-agent systems product lines, *in* 'SEKE 2008', Redwood City, USA, pp. 745–751.

Nunes, I., Nunes, C., Kulesza, U. & Lucena, C. (2008), Developing and evolving a multi-agent system product line: An exploratory study, *in* 'AOSE 2008', Estoril, Portugal, pp. 177–188.

Pena, J., Hinchey, M. G., Resinas, M., Sterritt, R. & Rash, J. L. (2007), 'Designing and managing evolving systems using a MAS product line approach', *Science of Computer Programming* **66**(1), 71–86.

Pohl, K., Böckle, G. & van der Linden, F. J. (2005), *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer-Verlag, New York,USA.

Rao, A. S. & Georgeff, M. P. (1995), BDI-agents: from theory to practice, *in* 'ICMAS 1995', San Francisco.

Sellers, B. H. & Giorgini, P., eds (2005), *Agent-Oriented Methodologies*, Idea Group Inc.

Wooldridge, M. & Ciancarini, P. (2000), Agent-Oriented Software Engineering: The State of the Art, *in* P. Ciancarini & M. Wooldridge, eds, 'AOSE 2000', Vol. 1957, Springer-Verlag, Berlin, pp. 1–28.

Wooldridge, M., Jennings, N. R. & Kinny, D. (2000), 'The gaia methodology for agent-oriented analysis and design', *AAMAS 2000* **3**(3), 285–312.