PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 01/08

Developing and Evolving Multi-Agent System Product Lines

Camila Patrícia Bazílio Nunes
Ingrid Oliveira de Nunes
Uirá Kulesza
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. 01/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Janeiro, 2008

Developing and Evolving Multi-Agent System Product Lines [1]

Camila Patrícia Bazílio Nunes[1], Ingrid Oliveira de Nunes[1], Uirá Kulesza[2],[3], Carlos José Pereira de Lucena[1]

[1] PUC-Rio, Computer Science Department, LES - Rio de Janeiro - Brazil
[2] Recife Center for Advanced Studies and Systems - Recife - Brazil
[3] New University of Lisbon - Lisboa - Portugal

camilan@inf.puc-rio.br, ioliveira@inf.puc-rio.br, uira@cesar.org.br, lucena@inf.puc-rio.br

Abstract. Software Product Line (SPL) approaches motivate the development and implementation of a flexible and adaptable architecture to enable software reuse in organizations. The SPL architecture addresses a set of common and variable features of a family of products. Based on this architecture, products can be derived in a systematic way. A multi-agent system product line (MAS-PL) defines an SPL architecture that is modularized, also using software agents to model, design and implement its common and variable features. This paper presents the development of an MAS-PL for the web domain, describing its architecture, the agents that compose the system and details of the object-oriented implementation and design. This MAS-PL consists of the evolutionary development of the ExpertCommittee (EC) web-based system. Furthermore, this paper presents an analysis of this MAS-PL, reporting some lessons learned based on our experience in the development of the MAS-PL as: features types, crosscutting features where the aspect-oriented programming is relevant to improve the separation of concerns, and some problems with SPL methodologies.

Keywords: Product line, multi-agent systems, object-oriented, aspect-oriented.

[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

Software engineering aims to produce methods, techniques and tools to develop software systems with high levels of quality and productivity. Software reuse (Griss 1997) is one of the main strategies proposed to address these software engineering aims. Software reuse techniques provide many benefits, such as reduction of development costs and time to market, and quality enhancement. Over the last years many reuse techniques have been proposed and refined by the software engineering community. Examples of these techniques are: component-based development (Szyperski 2002), object-oriented (OO) application frameworks (Fayad et al. 1999) and libraries, software architectures (Shaw & Garlan 1996) and patterns (Buschmann et al. 1996, Gamma et al. 1995). One of the latest trends in software reuse is the product line approach.

Software product lines (Pohl et al. 2005, Clements & Northrop 2002) (SPLs) refer to engineering techniques for creating similar software systems from a shared set of software assets using a systematic method in order to build applications. Clements & Northrop (2002) define a software product line (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". The core idea of SPL engineering is to develop a reusable infrastructure that supports the software development of a family of products. A family of products is a set of systems that has some commonalities, but also variable features. 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.

Over the last years, agent-oriented software engineering (AOSE) has also emerged as a new software engineering paradigm to allow the development of distributed complex applications which are characterized by a system composed of many interrelated subsystems (Jennings 2001). Most of the current AOSE methodologies are dedicated to developing single multi-agent systems (Pena et al. 2006). New approaches (Pena et al. 2007, Dehlinger & Lutz 2005) have started to explore the adoption of SPL techniques to the context of multi-agent systems (MAS) development. The aim of these new approaches is to integrate SPL and AOSE techniques by incorporating their respective benefits and helping the industrial exploitation of agent technology. Nevertheless, there are still many challenges to be overcome in the development of multi-agent systems product lines (MAS-PLs) (Pena et al. 2006).

In this work, we relate our experience of development of a MAS-PL in a bottom-up fashion. Our MAS-PL has been developed and emerged from the evolution of a conference management web-based system. Each new version of the system includes the design and implementation of new features that the previous version does not address. Most of the new features are related to the introduction of new autonomous behavior in the original system using MAS technology, such as agents, roles and their associate behaviors. All the three versions of our MAS-PL share a common SPL core architecture. Our case study has allowed us to analyze and discuss relevant questions related to the integration of SPL and MAS technologies, such as:

(i) how to modularize agency features in a MAS-PL;

(ii) how to seamlessly incorporate autonomous behavior (agency features) in a traditional web-based system;

(iii) which SPL models are useful and essential to specify and document an MAS-PL, and how to adapt them to the context of MAS-PL development.

This paper is organized as follows. Some works related to multi-agent systems and product lines are described in Section 2. In Section 3 is presented our multi-agent system product line. We make an analysis about our work in Section 4. Finally, the conclusions are discussed in Section 5 and future works are related in Section 6.

2 Multi-Agent Systems and Product Lines

Only some recent research work has investigated the integration synergy of Multi-Agent Systems (MASs) and Software Product Lines (SPLs) technologies. Dehlinger & Lutz (2005) have proposed an extensible agent-oriented requirements specification template for distributed systems that supports safe reuse. Their proposal adopts a product line to promote reuse in multi agent systems, which was developed using the Gaia methodology. The requirements are documented in two schemas: the role schema and the role variation point. The first defines a role and the variation points that a role can play during its lifetime. The second captures the requirements of role variation points capabilities. The proposed approach allows the reuse of agent configuration along the system evolution. Each agent configuration can be dynamically changed and reused in similar applications.

Pena et al. (2007) describe an approach to describing, understanding, and analyzing evolving systems. The approach is based on viewing different instances of a system as it evolves as different "products" in a software product line. Following their approach, an SPL is developed with an AOSE methodology, and the system is viewed as MAS-PL. Their approach proposes a set of software engineering techniques based on an agent-oriented methodology called Methodology for analyzing Complex MultiAgent Systems (MaCMAS) (Pena 2005), whose main aim is to deal with complex unpredictable systems. The MaCMAS allows for the description of the same feature at different levels of abstraction. It presents a process to building the core architecture of an MAS-PL at the domain engineering stage, whose activities are: (i) to build an acquaintance organization; (ii) to build a feature model; and (iii) to analyze commonalities and to compose core features. The main advantage of the approach resides in the fact that it make it possible to derive a formal model of the system and each state that it may reach.

Our work also explores the combination of MAS and SPL techniques and technologies in the context of development and evolution of systems. We focus specifically in the web-based systems domain by proposing the insertion of autonomous behavior features inside traditional web layered architectures. Our main aim was to investigate and understand the benefits of the agency feature modularization during this process.

3 MAS-PL: Case Study

This section describes our experience in the development of a multi-agent system product line (MAS-PL) for the web domain. Our experience is presented along the next sections in a stepwise fashion. We initially present the ExpertCommittee (EC), a web-based conference management system that supports the paper submission and reviewing processes from a conference (Section 3.1). After that we describe an evolved version of this system in which

we have incorporated new agency optional and alternative features related mainly to the specification and implementation of user agents (Section 3.2). This new version of the EC system is characterized as a multi-agent system product line (MAS-PL), because it addresses a new set of optional and alternative agency features, which allows providing different customized configurations of the system. The EC MAS-PL architecture designed to address the new agency features is then presented in terms of the components and agents that compose the system (Section 3.3). Finally, we detail the OO design and implementation of the EC MAS-PL by describing the mechanisms adopted to implement its variabilities (Section 3.4).

In this case study, we develop some versions of the application in order to show how an evolving system can be viewed as a software product line. Each version of the application will contain new features. Moreover, we want to show the problems found when the features are added and how we deal with this evolution.

3.1 The ExpertCommittee Web-based System

The ExpertCommittee (EC) Case Study focuses on showing an evolving application. The EC is a typical web-based application whose aim is to manage the paper submission and reviewing processes from conferences and workshops. The EC system provides functionalities to support the complete process of conference management, which are listed in the Table 1. Each of these functionalities can be executed by an appropriate user type of the system, such as, conference chair, program committee members, authors and reviewers. In Figure 3(a), we show the EC feature model (Czarnecki 1998) relating the mandatory and the optional features present in the first version of the system.

The main classes of the system are the classes Conference, User and AssignedRole. The domain classes are presented in Figure 1. A conference has several important roles, as the coordinator, chair and committee members. This roles are played by users. An user can submit papers to the conference and it will be reviewed by committee members or reviewers according to the conference's review template.

The EC web-based system was structured according to the Layer architectural pattern (Fowler 2002). Figure 2 illustrates the base architecture of the EC web-based system, which is composed of the following components/layers:

(i) **GUI** - this layer is responsible to process the web requests submitted by the system users. It was implemented using the Struts[2] framework;

(ii) **Business** - is responsible to structure and organize the business services provided by the EC system. The transaction management of the business services was implemented using the mechanisms provided by the Spring[3] framework; and

(iii) **Data** - aggregates the classes of database access of the system, which was implemented using the Data Access Object (DAO) design pattern. The Hibernate[4] framework was used to persist the objects in a MySQL[5] database.

[2]http://struts.apache.org/
[3]http://www.springframework.org/
[4]http://www.hibernate.org/
[5]http://www.mysql.org/

Table 1: Expert Committee Functionalities.

Role	*Functionality*	*Description*
Coordinator	Create Conference	Create new conferences and assign a chair for it.
	View Conference	Show all details of the conference, including its papers
Chair	Define Basic Data	Define the basic data of the conference, as its name, its start and end dates and its review template.
	Define Areas of Interest	Defines the areas of interest of the conference.
	Define Committee Members	Define the committee members of the conference.
	Define Deadlines	Define all the deadlines of the conference.
	Assign Papers	Assign the papers to be reviewed by committee members.
	Notify Authors	Publish the review of the selected papers to their authors.
	View Conference	Show all details of the conference, including its papers. Here, the chair can accept or reject the paper.
Committee	Choose Areas of Interest	The areas of interest of the conference are displayed and the committee member should choose some of them.
	Accept / Reject Paper	Accept or reject to review a paper.
	Assign Paper to Reviewer	Assign the paper to be reviewed by reviewer.
	Review Paper	Review the paper.
Reviewer	Accept / Reject Paper	Accept or reject to review a paper.
	Review Paper	Review the paper.
Author	Subscribe Paper	Subscribe a paper to the conference, only some information about the paper is needed, the paper file should be uploaded later.
	Edit Paper	Allow the authors to edit the paper data, submit the paper file and submit the camera ready if the paper was accepted.
	View Review	Show the review of the paper not showing who review the paper.

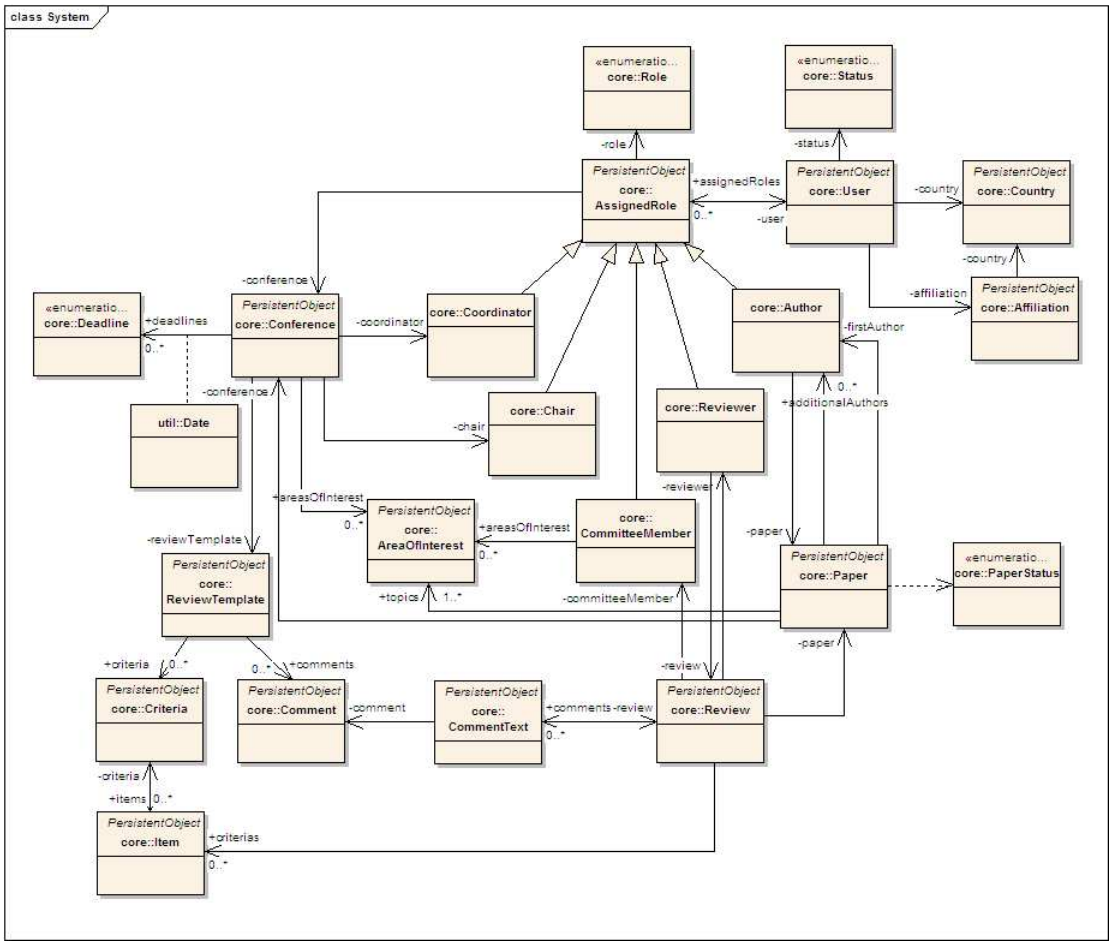


Figure 1: Domain Classes.

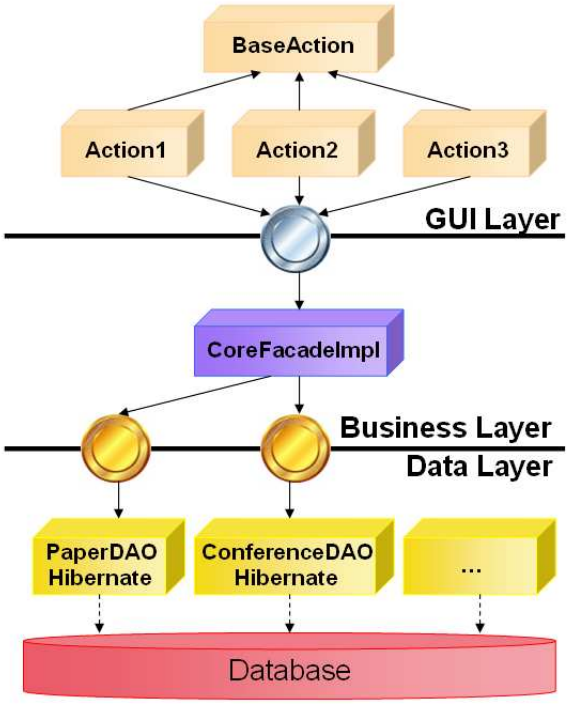


Figure 2: EC Base Architecture.

The first implemented version of our EC system is a common web application that has the features mentioned above. In the following versions, software agents were introduced on the EC system, adding autonomous behavior. This helps researchers and the event organization committee on the tasks that can be automated and make part of the review process and submission. Examples of features provided by the agents are the deadline monitoring and tasks management. Next sections detail these new versions and respective features present in their implementations.

3.2 Evolving the EC system to an MAS-PL

An evolving system can be viewed as a software product line, because the features that are common to all versions of the system comprise the core architecture of the product line. In our case study, we have evolved the original version of the EC System (Section 3.1) to incorporate new agency features. The main aim of these new features was to help the tasks assigned to all the system actors by giving them a user agent that addresses the following features:

(i) deadline and pending tasks monitoring; and

(ii) automation of user activities.

After producing this new version of the EC system, we have evolved it to improved the modularization of these optional and alternative agency features in order to enable the automatic (un)plugging of them from the original web system.

Table 2 summarizes the three different versions of our EC system. The first version was built without any autonomous behavior, in other words, without software agents. It was detailed in Section 3.1. The second version of the EC system contains features that are related to autonomous behavior and it has also some new features that add new functionalities to the system as well. The software agent abstraction was used to model and implement the autonomous behavior added to the original EC system. A software agent is an abstraction that enjoys mainly the following properties (Jennings 2001): autonomy, reactivity, pro-activeness and social ability. Thus, in this second version, we have used the agent abstraction and AOSE techniques to allow the introduction of new optional and alternative agency features in the system. Figure 3(b) illustrates the feature model of the second version of the EC system with new agency optional and alternative features.

Table 2: The three versions of ExpertCommittee.

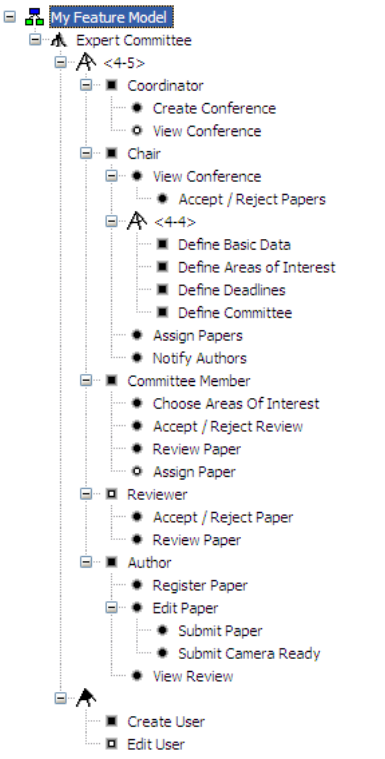
Version	*Description*
Version 1	Typical web-based application. It has the mandatory and some optional features to support the conference management process. These features are listed in Section 3.1
Version 2	Addition of autonomous behavior (agents) features, such as, automatic paper distribution, task management, deadline monitoring, and email notifications.
Version 3	Refactoring of Version 2 to improve the modularization of some agency features in order to make possible the automatic product derivation..

The third and last version of the EC system was implemented by applying a series of refactorings in Version 2. The system was restructured to make the (un)plugging of optional features possible. Each optional feature was modularized by using a combination of OO design patterns and techniques with Spring configuration files that allows the injecting of dependencies inside the variable points of the EC SPL architecture. It improves the capacity to produce and compose different configurations (products) of the SPL, and it also enables the automatic product derivation by means of model-based tools: software factories (Greenfield et al. 2004), generative programming (Czarnecki 1998), GenArch (Cirilo et al. 2007), pure::variants[6]. Product derivation is the process of constructing a product from the set of assets specified or implemented for a SPL (Deelstra et al. 2005). Each product is composed of the core features and a valid combination of optional and alternative features, according to the feature model. In an automatic product derivation process, the application engineer can generate a configuration (product) of the SPL by only selecting and choosing the features that are going to compose your product.

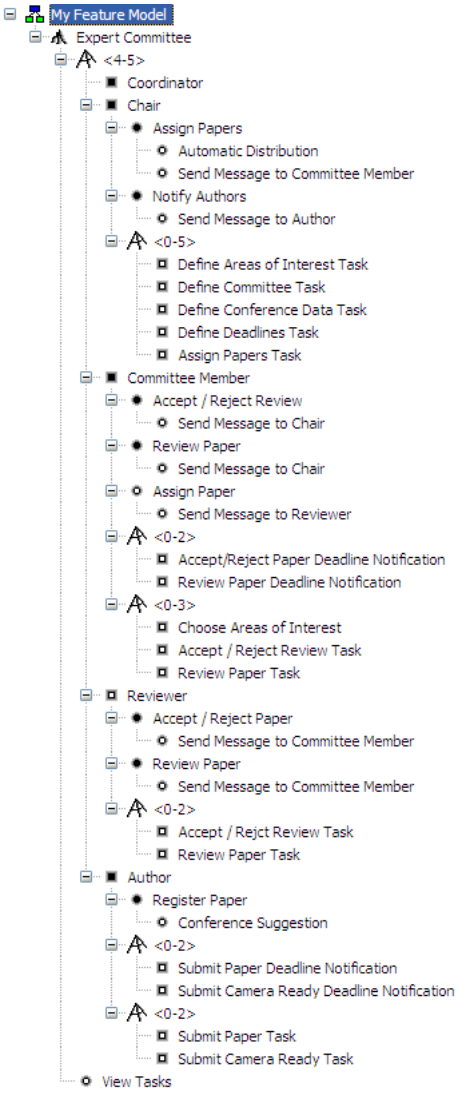
3.3 The EC MAS-PL Architecture

The EC Version 2 was implemented as an SPL architecture, which is illustrated in Figure 4. New features associated with the autonomous behavior of the system were added as a set of optional features. Different software agents and agent roles were specified to modularize

[6]http://www.pure-systems.com/



(a) First Version of the EC.



(b) Second Version of the EC.

Figure 3: Expert Committee Feature Model

these features. The JADE[7] framework was used as the base platform to implement our agents. These agents are responsible for monitoring the execution of different functionalities of the EC in order to provide their respective functionalities. The integration between the web architecture and the agents was accomplished by means of another implementation of the CoreFacade interface - the CoreFacadeProxyImpl. The CoreFacade interface delimits the Business layer, working as a Facade (Gamma et al. 1995). The CoreFacadeProxyImpl class acts like an interceptor of the requests to the CoreFacadeImpl class to notify the EnvironmentAgent about changes in the system. Details about each agent that comprises the system are listed below:

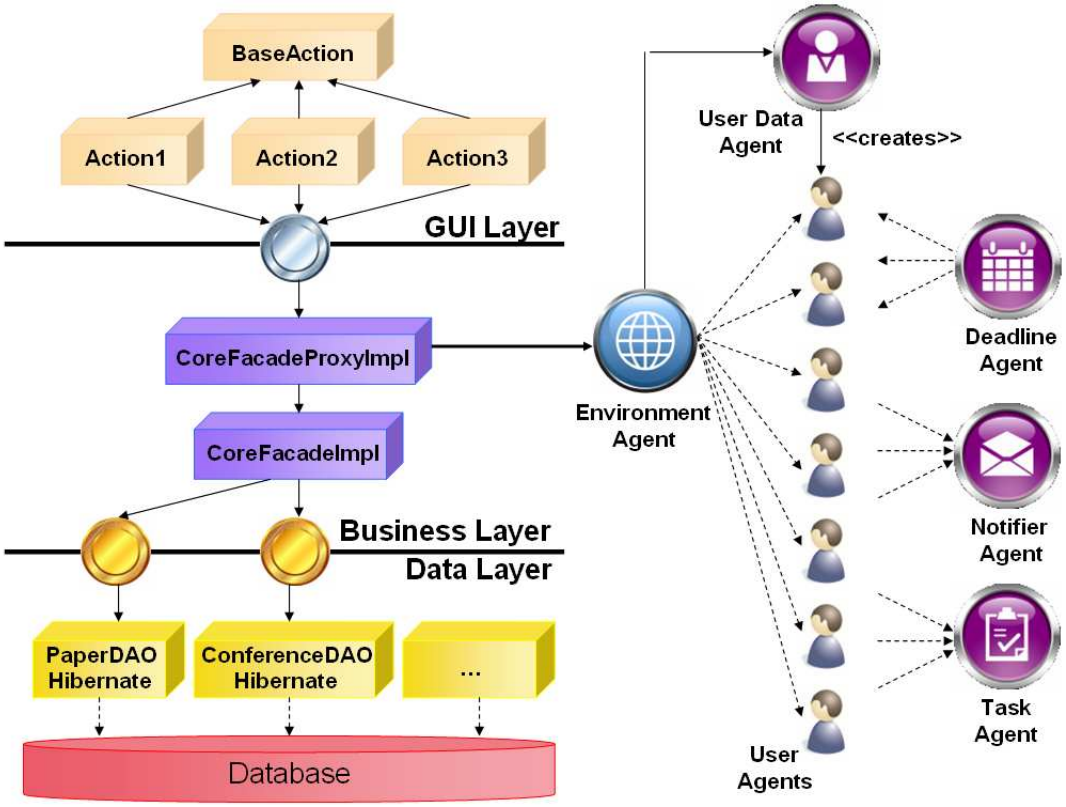


Figure 4: MAS-PL Architecture.

Environment Agent: this agent represents the environment of the system. Other agents perceive changes in the environment and make actions according to them. The environment agent was implemented using the Observer and the Proxy design patterns (Gamma et al. 1995) as depicted in Figure 5. When it is initialized, it registers itself as an observer of the CoreFacadeProxyImpl class. This is the implementation of the CoreFacade interface that allows the observation of its actions. That means that, for each call of the system business methods, the CoreFacadeProxyImpl class not only delegates the request to the original implementation of the CoreFacade,

[7] http://jade.tilab.com/

the CoreFacadeImpl class, to guarantee the execution of the business methods, but it also notifies the observers of the CoreFacadeProxyImpl class. The only observer in our implementation is the EnvironmentAgent, whose aim is to notify the other agents of the MAS-PL about the system changes;

User Data Agent: this agent receives notifications when new users are created in the database. When it happens, it creates a new user agent that will be the representation of the user in the system. The initial execution of the user data agent demands the creation of a user agent for each user already stored in the database;

User Agent: each user stored in the system has an agent that represents it in the system. This is the autonomous behavior, agents performing actions that the users should do. An example is when the paper submission deadline expires and the user agent in the chair role will automatically distribute the papers to the committee members. Besides this example, most of the user agents are responsible: (i) for analyzing and discovering pending tasks for user agents based on the roles the users play in the system; and (ii) for sending email notifications;

Deadline Agent: this agent is responsible for monitoring the conference deadlines. This monitoring serves two purposes: (i) to notify the user agents when a deadline is nearly expiring; and (ii) to notify the user agents when a deadline has already expired;

Task Agent: this agent is responsible for managing the user tasks. It receives requests for creating, removing and setting the execution date of tasks. The requests are made by the user agents;

Notifier Agent: this agent receives requests from other agents to send messages to the system users. In the current implementation, it sends these messages through email.

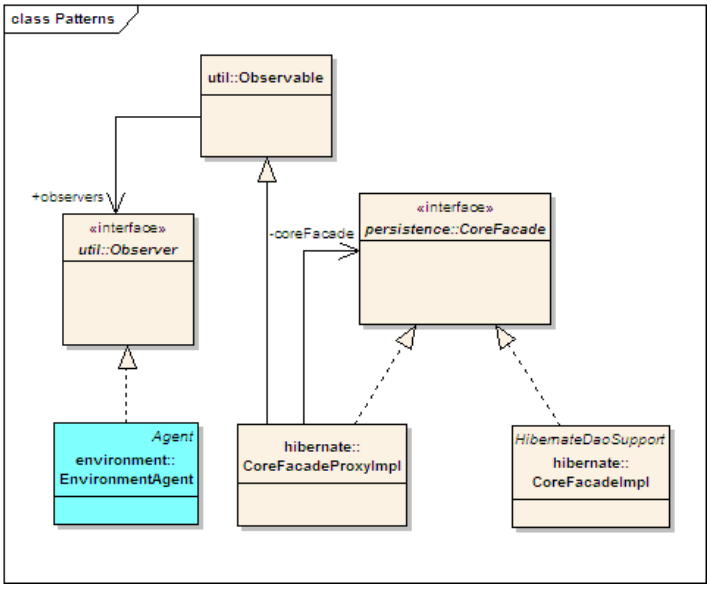


Figure 5: Observer and Proxy Patterns.

In this MAS-PL we use the Role Object Pattern (Bäumer et al. 1997) helped to separate better the agents features. The Role Object pattern models context-specific views of an object as separate role objects which are dynamically attached to and removed from the core object. in Figure 6 is depicted the pattern structure in our MAS-PL.

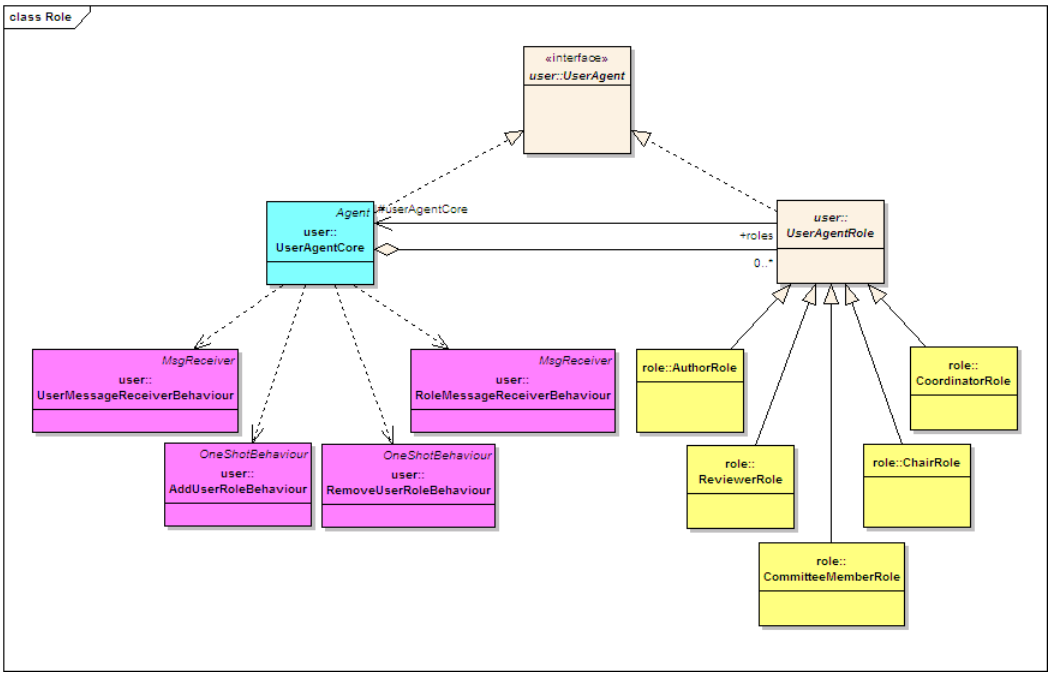


Figure 6: Role Pattern.

3.4 Evolving the EC MAS-PL Architecture

In versions 2 and 3 of the EC system, the MAS PL architecture was developed to provide the minimum impact when the new optional and alternative features must be added. In this way, different architectural and design decisions were accomplished to facilitate the creation of different configurations (products) of the MAS PL.

In the EC Version 2 of our MAS-PL, we adopt traditional design patterns to implement its variabilities (variable features). First, as we mentioned before, the integration between the web system and the environment agent was implemented using the Observer and Proxy design patterns Gamma et al. (1995). The Proxy pattern was used to allow the (un)plugability of the agency features and maintain the alternative to have all the agency feature as optional. Second, we use the Role Object pattern (Bäumer et al. 1997) to better modularize the implementation of each of our agents (Section 3.3). The Role Object pattern models context-specific views of an object as separate role objects, which are dynamically attached to and removed from the core object. This pattern was mainly used to provide a base implementation of the user agents whose behavior can be incremented by attaching new roles (such as chair, author, committee, reviewer) to be played by these agents.

Although the EC Version 2 already provides an improved modularization of the MAS PL optional and alternative features with the adoption of design patterns, when trying

to produce different configurations of our PL, we noticed the need to accomplish new adaptations in MAS-PL architecture, such as:

(i) to split the agent plans in small units to address only specific MAS features, because in the Version 2, the plans were implemented incorporating different features; and

(ii) to define a mechanism to provide an easier way to configure the different features, including fine-grained properties.

The EC Version 3 incorporated the implementation of these adaptations by providing:

(i) a feature-oriented modularization of the agent plans; and

(ii) a Spring-based mechanism to configure the main MAS-PL components.

Both implementation decisions enable the automatic instantiation of our MAS PL architecture using product derivation tools, such as: pure::variants, GenArch.

The customization of the MAS PL components using the Spring framework was accomplished by specifying a configuration file that aggregates different options of configuration of the MAS-PL, such as:

(i) different functional features of the conference management base system (edituser, paperdistribution) and respective properties;

(ii) the different agents and the respective plans; and

(iii) the different agent roles and respective plans.

These important elements of the system were modeled using the bean abstraction of the Spring framework. The bean abstraction is used to implement configurable component of systems. The code 1 illustrates a specific customization of our MAS PL configuration file that refers to a product that will contain all the optional features, except the task management.

Listing 1: XML Configuration File.

```
<bean id="ExpertCommittee"
class="br.puc.maspl.config.ExpertCommittee">
    <property name="optionalRoles">
        <list>
            <value>Reviewer</value>
        </list>
    </property>
    <property name="properties">
        <value>
            edit_user=true
            view_conference=true
            conference_suggestion=true
            paper_distribution=automatic
        </value>
    </property>
    <property name="agents">
```

```
        <list>
            <ref bean="DeadlineAgent" />
            <ref bean="NotifierAgentAgent" />
        </list>
    </property>
</bean>
<bean id="DeadlineAgent" class="br.puc.maspl.config.Agent">
    <property name="present" value="true" />
    <property name="optionalBehaviors">
        <list>
            <value>
            br.puc.maspl.agent.deadline.DeadlineMonitorBehaviour
            </value>
            <value>
            br.puc.maspl.agent.deadline.DeadlineReminderBehaviour
            </value>
        </list>
    </property>
</bean>
<bean id="NotifierAgentAgent"
class="br.puc.maspl.config.Agent">
    <property name="present" value="true" />
    <property name="agentProperties">
        <value>
    message_sender=br.puc.maspl.core.message.EmailMessageSender
    message_factory.show_sql=br.puc.maspl.agent.message.
        impl.MessageFactoryImpl
        </value>
    </property>
</bean>
```

4 MAS-PL Analysis

In this section we analyze and discuss several lessons learned from our experience of development and evolution of the EC MAS-PL. Our lessons learned are related to the following main points: features types, AO refactoring, and adaptation of SPL methodologies.

4.1 MAS-PL Features Types

SPL architectures address the implementation of different types of variable features, such as optional, alternative and OR features (Czarnecki & Helsen 2006). In our development experience, we have found that in a MAS-PL, the occurrence of variable features varies not only in term of its functional features, but it also depends and is structured based on the agency features that the MAS-PL needs to address. Since one of the main aims of the implementation of SPL architectures is to improve the modularization and management of their features, in a MAS-PL is essential to consider the agency features and evaluates how the existing technologies can help to address them.

In this particular study, we have identified three types of optional/extra features. We believe these three types can be considered in most of the MAS-PL, because they are really useful to improve the feature management of the MAS-PL. Next we briefly describe these three types:

- **New conference management features:** these features add new functionalities to the system, as creating new interfaces that users can access. This is the typical kind of feature addressed in SPL;
- **New autonomous behavior:** we had to introduce agents in the architecture when we added autonomous behavior in the system. The modularization of the autonomous behavior features using the agent abstraction enables us to (un)plug the features by only including or removing a set of agents;
- **New Behaviors and Roles for an Agent:** some optional features have impact inside of agents. They allow defining agent internal variabilities by defining specific new behaviors of agents. These features can be modularized as: (i) specific plans to be executed by the agent under specific conditions; and (ii) specific roles to be played by the agent in a specific context.

4.2 SPL Methodologies

Over the last years, many SPL methodologies have been proposed (Pohl et al. 2005, Gomaa 2004, Atkinson et al. 2000). Many of them (Clements & Northrop 2002, Pohl et al. 2005, Gomaa 2004) focus mainly on the requirement analysis, architecture and design modeling, and management processes. Some of them incorporate concepts and techniques from the object-oriented or component-based paradigms. However, these SPL methodologies don't detail or barely detail the modeling and documentation of agents or role features.

There is some recent research work that addresses initial proposals to define an MAS-PL development methodology (Pena et al. 2007, Dehlinger & Lutz 2005). These proposals consider MAS methodology as a base and adapt it to document features of a product line. Pena et al (Pena et al. 2006) identify current challenges to integrate the MAS and SPL software engineering approaches, such as management of evolving systems, necessity of new adapted techniques to cover distributed systems and the fact that agent-oriented software engineering does not cover some of the activities of SPL.

In our work, we have developed and evolved a web-based system by introducing the implementation of new variable agency features on its original architecture. We focused mainly on the use of OO techniques to modularize the implementation of the new agency features. The feature model was used to organize the SPL variabilities and guide us during the maintenance and refactoring of the different EC versions. The idea to introduce agency features in a web system was guided by the growing need of this kind of systems to incorporate recommendations and alerts of pendent tasks to their users. We believe that the architectural style adopted to structure and evolve the EC web-based system can be also used to increment the functionality of web-based system to other domains in order to introduce agency features related to recommendations and alerts to the system users. We intend to apply it to a different web-based system in order to validate our hypothesis. We are currently exploring two additional research directions related to MAS-PL methodologies: (i) documentation of MAS-PL architectures considering the integration of SPL and MAS proposed methodologies; and (ii) definition of a MAS-PL agile methodology to model their requirements and features.

4.3 AO Refactoring

Recent research presents the benefits of adopting aspect-oriented programming (AOP) techniques to improve the modularization of features in SPL (Alves et al. 2006) (Alves et al. 2005), framework-based (Kulesza et al. 2006) or multi-agent systems (Garcia et al. 2003) (Garcia et al. 2004) architectures. The increasing complexity of agent-based applications motivates the use of AOP. AOP has been proposed to allow well-modularized crosscutting concerns and it supports improved reusability and maintenance (Kiczales et al. 1997). Among the problems of crosscutting variable features we can enumerate:

(i) tangled code - the code of variable features is tangled with the base code (core architecture) of SPL;

(ii) spread code - the code of variable features is spread over several classes;

(iii) replicated code - the code of variable features is replicated over many places.

All these problems can cause difficulties regarding the management, maintenance and reuse of variable features in SPL. In order to promote improved separation of concerns, some crosscutting features that present the problems mentioned above are natural candidates to be designed and implemented using AOP. In our MAS-PL exploratory case study, we have found the following interesting situations to adopt AOP techniques:

(i) *modularization of the glue-code that integrates the web-based system (base code) with the agent features (new variable agency features)* - in our current implementation, this is addressed by the Observer design pattern (Hannemann & Kiczales 2002) that is used to observe/intercept the execution of business methods of the CoreFacade-ProxyImpl class. The Figure 7 depicts the implementation of the observer pattern with AOP. AOP can be used to modularize the intercepted code that allows the agents monitor the execution of the web-based system, it facilitates the (un)plug of the agency features in the system. In our case study, 17 methods of the CoreFacade-ProxyImpl class are intercepted to collect information for the agents; and

(ii) *modularization of the agent roles* - in the EC case study, we have used the Role OO design pattern to modularize the agent roles. We have noticed that the use of this pattern cannot provide an improved isolation of the agent role features, which is essential to SPL variability management. The implementation of the agent classes (e.g. UserAgentCore class) requires, for example, the activation and deactivation of the agent roles over different points of the execution of the agent behavior, such as agent initialization, execution of specific plans, etc. The adoption of AOP to modularize agent roles (Garcia et al. 2005) is thus a better option to improve the modularization and evolution of the agent roles features. This can seen in Figure 8, where the roles are created and associated to user agent object in *UserAgentCore* class. Thus, we separate the role codes in five aspects that crosscuts the *addRole* method in *UserAgentCore* class. With the aspect-oriented implementation gets easy to add, remove, or modify roles.

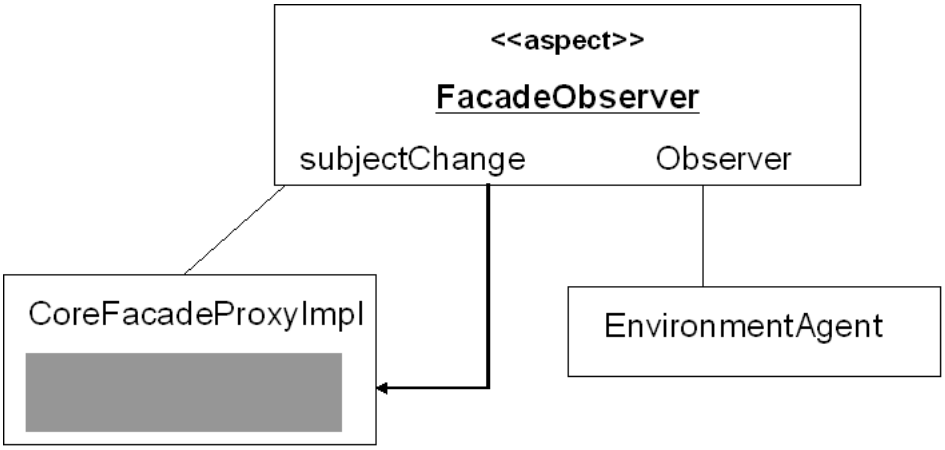


Figure 7: Observer Pattern Aspect-Oriented Implementation

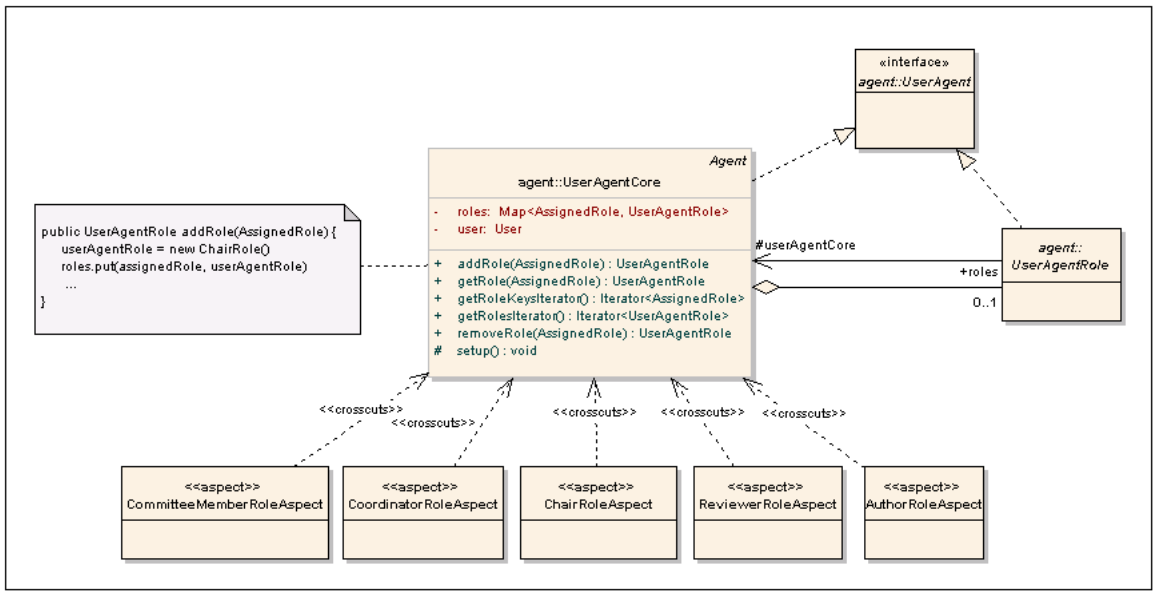


Figure 8: Roles Aspect-Oriented implementation.

4.4 Comparing our Experience to Other MAS-PL Initiatives

The multi-agent system product line presented in the paper focuses in the development of a web system. The works mentioned in Section 2, show stand-alone applications, usually applications related to NASA. We tried to expose a situation that could happen in software factories, where there is an web application that, in further versions, autonomous behavior is added. With the use design patterns, we showed that agents can be introduced on existing applications with a low impact. A commonality between Pena's and our work is viewing an evolutionary system as being a software product line.

The way that we adopted to construct the product line is also different from the others work. They were worried about the specification of the MAS-PL in order to make a formal specification of the system viable. In our case study we developed the system in a bottom-up fashion. This helped as to see the problems that are not present in the development of a SPL implemented only with object and that not have autonomous behavior. The related works also do not present how the derivation of products is done.

Our contributions are the development of an evolving system, viewed as a product line, on which was added autonomous behavior (agents). Besides that, crosscutting features were identified indicating aspect-oriented programming should be used in order to improve the separation of concerns and facilitate the product instantiation. Furthermore, we pointed out some deficiencies in the current SPL methodologies to develop a MAS-PL, bringing up the necessity of extending them.

5 Conclusion

This work presented an exploratory study of development and evolution of an MAS-PL. We initially developed a traditional web-based system to support the process of conference management. After that, we evolved this system to incorporate a series of new agency features, which addresses autonomous behavior associated with recommendations to their system users. Different user agents and roles were implemented to modularize these features. The feature model was also adopted to drive the incorporation of the new features. As a result of our study, we presented an SPL architecture to integrate agency features in traditional web-based system, and we discussed some lessons learned regarding the feature types encountered in our MAS-PL, the need of using AOP techniques to improve the modularization of features in our architecture and directions of adapting SPL methodology to allow the specification of SPLs that present autonomous behavior implemented with software agents.

6 Future Works

6.1 The automatic instantiation using GenArch

In order to allow the automatic instantiation, we will use a model-based tool named GenArch (Cirilo et al. 2007). GenArch is centered on the definition of three models (feature, architecture and configuration models) which enable the automatic instantiation of software product lines (SPLs) or frameworks. In order to use the GenArch tool, it is necessary to annotate the existing code of MAS-PL. These annotations indicate the imple-

mentation of features and variabilities in the code of artifacts from the SPL. After that, we will be able to make an automatic product derivation.

6.2 Development Methodology

The *Expert Committee* Case Study was developed in a bottom-up fashion. There are a lot of software product lines methodologies, but they usually are based on object-oriented software design. Moreover, there are also methodologies for agent oriented software design, but they are focused on the development of single applications. Thus, the idea is to propose a new development methodology for multi-agent systems product lines. It will be based on the SPL methodologies, and MAS techniques will be introduced on it, as the Figure 9 shows.

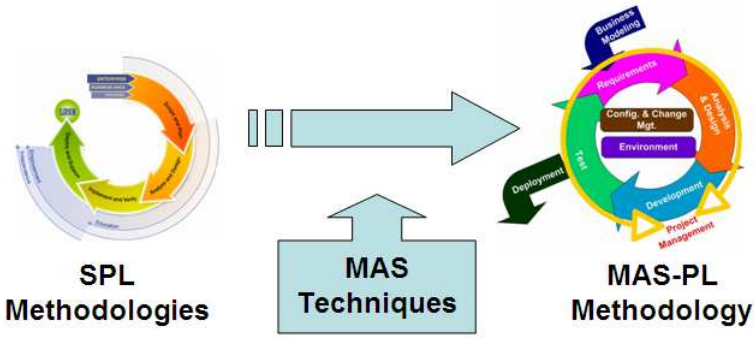


Figure 9: Development of a MAS-PL Methodology.

6.3 The Metrics

After refactoring the MAS-PL using aspect-oriented programming, there will be two versions of the product line: one implemented only with objects, another with aspects used in convenient points. This will allow an empirical study comparing the implementations object-oriented (OO) and aspect-oriented (OA). In other words, we will analyze quantitatively and qualitatively the impact of using OO and OA in MAS-PL maintenance scenarios. Then, we can check what are the impacts of adding new functional features, autonomous behavior features (Agents and its Roles) and behavior for specific Agents and Roles. Initially, we will use the metrics suite defined in (Sant'anna et al. 2003).

References

Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P. & Lucena, C. (2006), Refactoring product lines, *in* 'GPCE '06: Proceedings of the 5th international conference on Generative programming and component engineering', ACM, New York, NY, USA, pp. 201–210.

Alves, V., Matos, P., Cole, L., Borba, P. & Ramalho, G. (2005), Extracting and evolving mobile games product lines, *in* 'Proceedings of the 9th International Conference of Software Product Lines', Springer, pp. 70–81.

Atkinson, C., Bayer, J. & Muthig, D. (2000), Component-based product line development: The kobrA approach, *in* P. Donohoe, ed., 'Proceedings of theFirstSoftware Product Line Conference', pp. 289–309.

Bäumer, D., Riehle, D., Siberski, W. & Wulf, M. (1997), 'The Role Object Pattern', citeseer.ist.psu.edu/baumer97role.html. submitted for Pattern Languages of Programming (PLoP) 97.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P. & Stal, M. (1996), *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley Sons.

Cirilo, E., Kulesza, U. & Lucena, C. (2007), GenArch: A Model-Based Product Derivation Tool, *in* 'Proceedings of the 1º. Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS 2007)', Campinas, Brazil, pp. 17–24.

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

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.

Deelstra, S., Sinnema, M. & Bosch, J. (2005), 'Product derivation in software product families: a case study', *Journal of Systems and Software* **74**(2), 173–194.

Dehlinger, J. & Lutz, R. R. (2005), A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems, *in* 'SELMAS '05: Proceedings of the fourth international workshop on Software engineering for large-scale multi-agent systems', ACM Press, New York, NY, USA, pp. 1–7.

Fayad, M., Schmidt, D. & Johnson, R. (1999), *Building application frameworks: object-oriented foundations of framework design*, John Wiley & Sons, Inc., New York, NY, USA.

Fowler, M. (2002), *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-oriented Software*, Addison-Wesley.

Garcia, A., Chavez, C., Kulesza, U. & Lucena, C. (2005), The role aspect pattern, *in* '10th European Conference on Pattern Languages of Programs (EuroPLoP2005)', Isree, Germany.

Garcia, A., Lucena, C. & Cowan, D. (2004), 'Agents in object-oriented software engineering', *Software Practice Experience* **34**(5), 489–521.

Garcia, A., Sant'anna, C., Chavez, C., Silva, V., Lucena, C. & Staa, A. (2003), Agents and Objects: An Empirical Study on the Design and Implementation of Multi-Agent Systems, *in* 'ACM International Conference on Software Engineering, Proceedings on 2nd International Workshop on of Software Engineering for Large-scale Multi-Agent Systems (SELMAS)', Portland, USA, pp. 11–21.

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

Greenfield, J., Short, K., Cook, S. & Kent, S. (2004), *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, John Wiley and Sons.

Griss, M. L. (1997), Software Reuse: Architecture, Process, and Organization for Business Success, *in* 'ICCSSE '97: Proceedings of the 8th Israeli Conference on Computer-Based Systems and Software Engineering', IEEE Computer Society, Washington, DC, USA, p. 86.

Hannemann, J. & Kiczales, G. (2002), Design pattern implementation in Java and aspectJ, *in* 'OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications', ACM, New York, NY, USA, pp. 161–173.

Jennings, N. R. (2001), 'An agent-based approach for building complex software systems', *Commun. ACM* **44**(4), 35–41.

Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. & Irwin, J. (1997), Aspect-Oriented Programming, *in* 'Proceedings European Conference on Object-Oriented Programming', Vol. 1241, Springer-Verlag, Berlin, Heidelberg, and New York, pp. 220–242.

Kulesza, U., Alves, V., Garcia, A. F., de Lucena, C. J. P. & Borba, P. (2006), Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming, *in* 'ICSR'06', Torino, pp. 231–245.

Pena, J. (2005), On improving the modelling of complex acquaintance organisations of agents. A method fragment for the analysis phase., PhD thesis, University of Seville.

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.

Pena, J., Hinchey, M. G. & Ruiz-Cortés, A. (2006), 'Multi-agent system product lines: challenges and benefits', *Communications of the ACM* **49**(12), 82–84.

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

Sant'anna, C., Garcia, A., Chavez, C., Lucena, C. & von Staa, A. (2003), On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework, *in* 'Proceedings XVII Brazilian Symposium on Software Engineering', Manaus, Brazil, pp. 19–34.

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

Szyperski, C. (2002), *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.