Developing and Evolving a Multi-Agent System Product Line: An Exploratory Study

Ingrid Nunes[1], Camila Nunes[1], Uirá Kulesza[2, 3], Carlos J. P. de Lucena[1]

[1]PUC-Rio, Computer Science Department, LES
Rio de Janeiro - Brazil
{ioliveira,cnunes,lucena}@inf.puc-rio.br

[2]Recife Center for Advanced Studies and Systems - C.E.S.A.R.
Recife – Brazil
uira@cesar.org.br

[3]New University of Lisbon
Lisboa - Portugal

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 web-based system. Furthermore, this paper reports some lessons learned from this exploratory study of definition of a MAS-PL.

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

1 Introduction

Software engineering aims to produce methods, techniques and tools to develop software systems with high levels of quality and productivity. Software reuse [1] 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 [2], object-oriented (OO) application frameworks [3] and libraries, software architectures [4] and patterns [5, 6]. One of the latest trends in software reuse is the product line approach.

Software product lines [7, 8] (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 [8] 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 [9], 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 sub-systems [10]. Most of the current AOSE methodologies are dedicated to developing single multi-agent systems [11]. New approaches [12, 13] 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) [11].

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. The purpose of this case study was to create a scenario to identify problems related to the integration of the SPL and MAS technologies. Some of the questions that came over during the development of the MAS-PL were: (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. We also discuss possible solutions to these problems, such as adopts aspect-oriented programming (AOP) to isolate existing agency features.

This remainder of this paper is organized as follows. Some works related to multi-agent systems and product lines are described in Section 2. The ExpertCommittee MAS Product Line is presented in Section 3. The discussions and lessons learned are presented in Section 4. Finally, the conclusions and directions for future works are discussed in Section 5.

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 [13] 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 [12] 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) [14], 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 makes 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 The ExpertCommittee MAS Product Line

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).

3.1 The ExpertCommittee Web-based System

The ExpertCommittee (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, such as: (i) create conferences; (ii) define conference basic data, committee member, areas of interest and deadlines; (iii) choose areas of interest; (iv)

submit paper; (v) assign papers to be reviewed; (vi) accept/reject to review a paper; (vii) review paper; (viii) accept / reject paper; (ix) notify authors about the paper review; and (x) submit camera ready. Each of these functionalities can be executed by an appropriate user type of the system, such as, conference chair, coordinator, program committee members and authors. Figure 1 (a) shows the feature model [15] of the first version of EC system. This version was considered the core of our MAS-PL, created with the incorporation of new optional features in the subsequent versions.

The EC web-based system was structured according to the Layer architectural pattern [16] and 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[1] 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[2] framework; and (ii) Data – aggregates the classes of database access of the system, which was implemented using the Data Access Object (DAO) design pattern. The Hibernate[3] framework was used to make persistent the objects in a MySQL[4] database. Figure 2 illustrates the architecture of the EC web-based system and highlights the 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. The introduction of these new autonomous behaviors helps existing user tasks that can be automated using agent technology. Examples of features provided by the agents are the deadline monitoring and tasks management. We also added a new role, the reviewer, which can review a paper delegated by a committee member. Next sections detail these new versions and respective features present in their implementations.

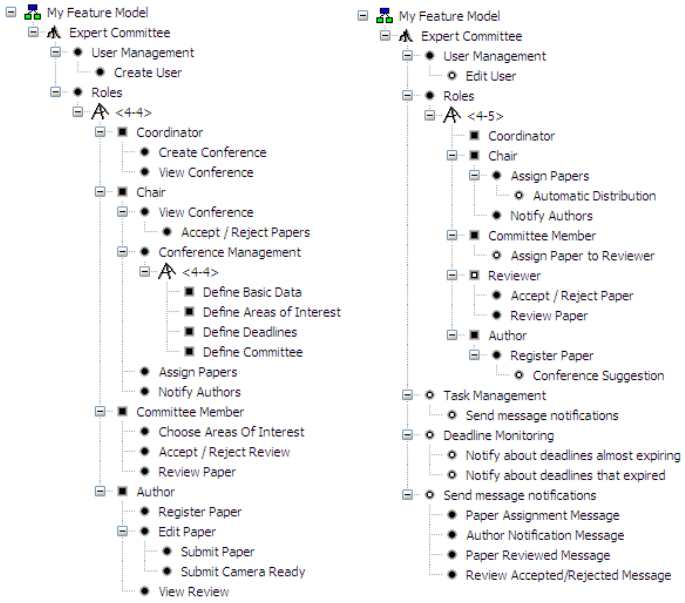
3.2 Evolving the EC system to an MAS-PL

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. In the third version of the EC system, we improved the modularization of these optional and alternative agency features to enable their automatic (un)plugging from the original system.

Table 1 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

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

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 [17]: 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. Fig. 1(b) illustrates the feature model containing the new agency optional and alternative features introduced in second version of the EC system.



(a) Mandatory Features of the EC **(b).** Optional Features of the EC.

Fig. 1. Expert Committee Feature Model.

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, such as: software factories [18], generative programming [19], GenArch [20], pure::variants[5]. Product derivation is the process of constructing a product from the set of assets specified or implemented for a SPL [21]. 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

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

application engineer can generate a configuration (product) of the SPL by only selecting and choosing the features that are going to compose your product.

Version	Description
Version 1	Typical web-based application that represents our MAS-PL core. It has the mandatory features that support the conference management process. These features are listed in Section 3.1.
Version 2	- Addition of the reviewer role and the functionalities related to it: accept/reject review and review paper. - Addition of the following functionalities: edit user and assign review to reviewer. - 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.

Table 1. The three versions of ExpertCommittee.

3.3 The EC MAS-PL Architecture

The EC Version 2 was implemented as an SPL architecture, which is illustrated in Figure 2. 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 these features. The JADE framework[6] 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 the introduction of the Observer pattern [6]. All the services that make part of the Business layer extends the `Observable` class. This class has a set of objects that implement the `Observer` interface. ,The EnvironmentAgent implements this interface, and is notified about changes in the system. Details about each agent that comprises the system are listed below:

Environment Agent: this agent monitors the EC system by observing the execution of specific business services. These monitored events of the EC system represent the environment in which the user agents are situated. Each user agent is specified to perceive changes in the environment and make actions according to them. The environment agent was implemented using the Observer design pattern [6]. When it is initialized, it registers itself as an observer of the services that compose the Business layer. These services are observable objects that allow the observation of their actions. That means that, for each call of the system business methods, the services not only execute the requested methods, but they also notify their respective observers. 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;

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

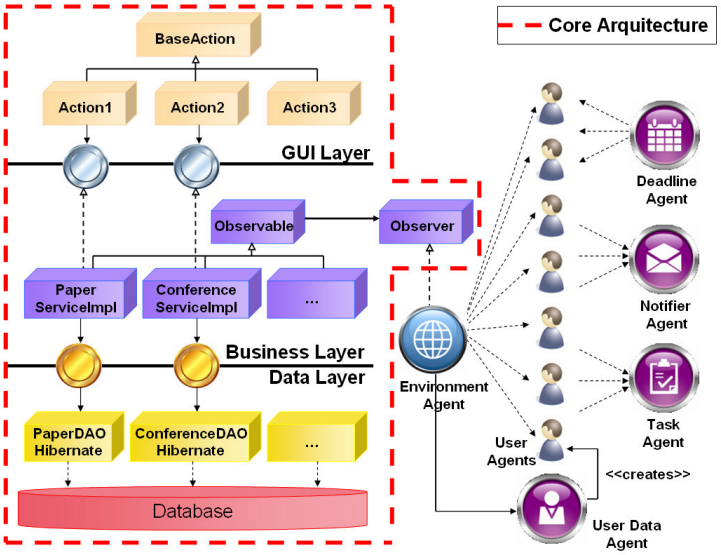


Fig. 2. MAS-PL Architecture.

- **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 him/her 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.

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 design pattern [6]. The Observer 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 [22] 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.

The EC Version 2 provides an improved modularization of the MAS-PL optional and alternative features through the adoption of design patterns. However, we noticed the need to accomplish new adaptations in the MAS-PL architecture in order to facilitate the production of different configurations of our MAS-PL. These required adaptations were: (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 easy 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 (edit_user, paper_distribution) 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.

4 Discussions and Lessons Learned

In this section we present 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 [15]. 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. According to [7], these two types of variability are classified as external and internal respectively. 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, the first one is external and the other ones internal. 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 AO Refactoring

Recent research work presents the benefits of adopting aspect-oriented programming (AOP) techniques to improve the modularization of features in SPL [23, 24], framework-based [25] or multi-agent systems [26, 27] 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 [28]. 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 [29] that is used to observe/intercept the execution of business methods of the services of the Business layer. 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 distributed among the services 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 [30] is thus a better option to improve the modularization and evolution of the agent roles features.

4.3 Adaptation of SPL Methodologies

Over the last years, many SPL methodologies have been proposed [7, 31, 32]. Many of them [7, 8, 31] 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 do not 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 a MAS-PL development methodology [12, 13]. These proposals consider MAS methodology as a base and adapt it to document features of a product line. Pena et al [11] identify current challenges to integrate the MAS and SPL software engineering approaches, such as management of evolving systems, need to provide 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 motivated by the growing need of this kind of systems to incorporate recommendations and alerts of pending tasks to their users. Based on the results of this exploratory study, we are currently investigating the need to propose new extensions to existing SPL methodologies in order to model and modularize each of different agency features specified. The main aim is to allow an explicit documentation and tracing of these agency variabilities along the SPL development process. In particular, we are focusing on these two research directions: (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.

Our case study has defined an architectural style to increment web-based systems with new agency features. It allowed introducing new agency features related to recommendations and alerts to the system users. Since many web-based system are typically structured following the guidelines of the Layer architectural pattern, we are currently exploring the usage of this same architectural style to different web-based systems in order to validate its applicability.

5 Conclusions and Future Work

This paper presented an exploratory study of development and evolution of a 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 the 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 a SPL architecture that allows to integrate agency features in traditional web-based system. Additionally, we presented a set of lessons learned from our study, related to: (i) the feature types encountered in our MAS-PL; (ii) the possibility of using aspect-oriented techniques to improve the modularization of agency features in our architecture; and (ii) the need of adaptation of existing SPL methodologies to allow the modeling of the different agency features during the SPL development.

We are currently extending the research work presented in this paper in several directions. We are working to define a base and lightweight SPL methodology that allows the specification and documentation of autonomous behavior along the domain and application engineering processes. We are refactoring our MAS-PL architecture using AOP based on the directions presented in Section 4.2. We are starting to organize an experimental empirical study [33] to compare the use of OO and AOP technologies in the development and evolution of MAS-PL. Finally, we plan to explore the use of our MAS-PL architecture to incorporate autonomous behavior in other web-based systems in order to validate it as an architectural style [4].

References

1. Griss, M.L.: Software Reuse: Architecture, Process, and Organization for Business Success. Proceedings of the ICCSSE '97, Washington, DC, USA, IEEE Computer Society (1997) 86.
2. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002).
3. Fayad, M., Schmidt, D., Johnson, R.: Building application frameworks: object-oriented foundations of framework design. John Wiley & Sons, Inc., New York, NY, USA (1999).
4. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (1996).
5. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad,P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley Sons (1996).
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley (1995).
7. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, New York,USA (2005).
8. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, MA, USA (2002).
9. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3) (2006) 621-645.
10. Jennings, N.R.: An agent-based approach for building complex software systems. Commun. ACM 44(4) (2001) 35-41.
11. Pena, J., Hinchey, M.G., Ruiz-Cortes, A.: Multi-agent system product lines: challenges and benefits. Communications of the ACM 49(12) (2006) 82-84.

12. Pena, J., Hinchey, M.G., Resinas, M., Sterritt, R., Rash, J.L.: Designing and managing evolving systems using a MAS product line approach. Science of Computer Programming 66(1) (2007) 71-86.
13. Dehlinger, J., Lutz, R.R.: A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems. Proceedings of SELMAS '05, New York, NY, USA, ACM Press (2005) 1-7.
14. Pena, J.: On improving the modeling of complex acquaintance organisations of agents. A method fragment for the analysis phase. PhD thesis, University of Seville (2005).
15. Czarnecki, K.: Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. PhD thesis, Technical University of Ilmenau (1998).
16. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley (2002).
17. Wooldridge, M., Ciancarini, P.: Agent-Oriented Software Engineering: The State of the Art. Proceedings of AOSE´2000, LNCS 1957. Springer-Verlag, Berlin (2000) 1-28.
18. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with atterns, Models, Frameworks, and Tools. John Wiley and Sons (2004).
19. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison Wesley Longman (2000).
20. Cirilo, E., Kulesza, U., Lucena, C.: GenArch: A Model-Based Product Derivation Tool. Proceedings of the Brazilian Symposium on Components, Architectures and Reusability (SBCARS´2007), Campinas, Brazil (2007) 17-24.
21. Deelstra, S., Sinnema, M., Bosch, J.: Product derivation in software product families: a case study. Journal of Systems and Software 74(2) (2005) 173-194.
22. Bumer, D., Riehle, D., Siberski, W., Wulf, M.: The Role Object Pattern. Proceedings of Pattern Languages of Programming (PLoP´ 97).
23. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., Lucena, C.: Refactoring Product lines. In: GPCE '06: Proceedings of GCPE´2006, New York, NY, USA, ACM (2006) 201-210.
24. Alves, V., Matos, P., Cole, L., Borba, P., Ramalho, G.: Extracting and evolving mobile games product lines. In Proceedings of SPLC´2005, LNCS 3714, pages 70--81. Springer, 2005.
25. Kulesza, U., Alves, V., Garcia, A.F., de Lucena, C.J.P., Borba, P.: Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming. In: ICSR'06, Torino (2006) 231-245.
26. Garcia, A., Sant'anna, C., Chavez, C., Silva, V., Lucena, C., Staa, A.: Agents and Objects: An Empirical Study on the Design and Implementation of Multi-Agent Systems. In: Proceedings of SELMAS´2003, Portland, USA (2003) 11-21.
27. Garcia, A., Lucena, C., Cowan, D.: Agents in Object-Oriented Software Engineering. Software: Practice & Experience, Elsevier, 34(5) (2004) 489-521.
28. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In: Proceedings of ECOOP´2007, LNCS 1241., Berlin, Heidelberg, and New York, Springer-Verlag (1997) 220-242.
29. Hannemann, J., Kiczales, G.: Design pattern implementation in Java and aspectJ. In: Proceedings of OOPSLA '02, New York, NY, USA, ACM (2002) 161-173.
30. Garcia, A., Chavez, C., Kulesza, U., Lucena, C.: The Role Aspect Pattern. In: 10th European Conference on Pattern Languages of Programs (EuroPLoP2005), Isree, Germany. (2005).
31. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004).
32. Atkinson, C., Bayer, J., Muthig, D.: Component-based product line development: The kobrA approach. In Donohoe, P., ed.: Proceedings of SPLC´2000. (2000) 289-309.
33. Figueiredo, E., Cacho, N., SantAnna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F., Dantas, F.: Evolving software product lines with aspects: An empirical study on design stability. In: Proceedings of the ICSE´2008, Leipzig,(2008).