Documenting and Modeling Multi-agent Systems Product Lines

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

[1] *PUC-Rio, Computer Science Department, LES - Rio de Janeiro - Brazil*
{ioliveira,camilan,lucena}@inf.puc-rio.br
[2] *Recife Center for Advanced Studies and Systems - Recife - Brazil*
uira@cesar.org.br
[3] *New University of Lisbon - Lisboa - Portugal*

Abstract

In this paper, we explore the use of existing software product line (SPL) approaches to document and model a multi-agent system product line (MAS-PL). Our analysis focuses specifically in the domain analysis and design stages of SPL development. The main aim of our study is to investigate the benefits, limitations and challenges of current SPL and MAS-PL approaches/methodologies to document and model MAS-PL features. Our investigation is illustrated and validated through the use of a web-based conference management system. As a result of our study, we propose the adaptation and extension of existing approaches to address the modeling of MAS-PL features.

1. Introduction

Nowadays, a common scenario in organizations is to develop similar products and to provide different customizations of these products to individual customers. This is typically addressed in an empirical way. Software product lines (SPLs) [18, 3] represent a new trend of software reuse that investigates methods and techniques to build and customize families of applications through a systematic method. Clements & Northrop [3] 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 ". According to [4], 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.

Over the past few years, several methods have been published to address the problems and challenges of SPL engineering [12, 18, 8]. Some of them only propose methodological guidelines, not specifying how to design or implement the SPL, meaning developers have to create their own way to develop the product line. Some of these methodologies propose a complete SPL development process based on existing paradigms, such as component-based [1] or object-oriented [8] software development. However, there are new trends, such as multi-agent systems (MASs) [10, 19], which are not considered by the current SPL methodologies. MASs have emerged as a new software paradigm to help in the development of complex software systems, which contain properties such as autonomy, reactivity, proactiveness and social ability. Recently, new approaches [16, 6] were proposed designed to explore the benefits of integrating SPL and Agent-Oriented Software Engineering (AOSE) techniques. Nevertheless, there are still many challenges to overcome in the development of multi-agent systems product lines (MAS-PLs) [17].

In this context, this paper investigates the adoption of proposed SPL and MAS-PL methodologies in the documentation and modeling of MAS-PL. During this process, we had to deal with challenges, such as how the features can be documented, modeled and modularized throughout the entire domain engineering process. A product line of conference management systems that includes the implementation of several optional agency features is used to illustrate and validate our study. Some adaptations and extensions of current SPL approaches are also proposed in order to address their identified deficiencies in the modeling and documentation of more complex agency features.

The remainder of this paper is organized as follows. Section 2 presents some existing SPL methodologies. In Section 3, an overview of the ExpertCommittee case study is presented, giving some details about its development. In Section 4, we show how we have modeled and documented our product line at the domain analysis and domain design phases. We present some discussions in the Section 5. Finally, the conclusions and directions for future works are discussed in Section 6.

2. Background

We have studied and compared some existing SPL methodologies. Almost all methods focus on the description of SPL properties at a very high level of abstraction and give no guidance on how the required flexibility should be realized at the implementation level. In our initial comparative study [14], we used the same evaluation framework of [13] to compare the SPL and MAS-PL methodologies. The goal of this evaluation was to obtain an overview of the methodologies and not necessarily to rate them. Subsequently, we analyzed how the investigated approaches could deal with the documentation and modeling of agency features. Table 1 presents partial results of this analysis. Due to space restrictions in this paper, we only reported the results of this second part of our study. For additional details of our study, please refer to [14].

Table 1. Methodologies Comparison.

Methodology	Domain Analysis	Domain Design
FORM	Feature diagram with composition rules	Subsystem, Process Model and Module Models
Framework [18]	Reusable, textual and model-based requirements, variability model	Reference architecture, refined variability model, mapping from design artifacts to requirements artifacts
PLUS	Requirements model consisting of a use case model and feature model	Static and dynamic models, feature/class dependencies, design of component-based software architecture
MacMAS extension [16]	Feature Model (features are goals)	Acquaintance Organization, Traceability and Role Models
Approach [6]	Role Schema, Role Variation Point	-

Commonly the SPL approaches adopt feature models as the typical notations to specify the SPL features. FORM [12] provides a feature modeling method for analyzing and capturing the common and variable features of SPLs and their respective interdependencies. The features are organized into a coherent model referred to as a feature model, which models 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. Pena et. al. [16] also proposes the use of feature models, but the features are the goals of the agents. Goals are not a detail of the system that is visible to the end user; therefore, they should not appear in a feature model. [18] document variabilities through a variability model, which models what varies from one product to another with the explicit indication of the variation points and variants. Furthermore, it also motivates the definition of explicit tracing links between the variations points/variants from the variability model and other analysis and design models (e.g., use cases and class diagrams). PLUS [8] proposes a feature model based on UML notation, but contains the same information of traditional feature models. Almost all the approaches do not address explicitly the modeling of the SPL requirements. The PLUS approach defines a customization of the use case model to specify and document the SPL requirements. Dehlinger & Lutz [6] adopt a product-line-like view of an agent-based software system and proposes a requirements specification template to capture and reuse dynamically changing configurations of agents for future similar systems.

In the domain design, most of the SPL approaches investigated only provide support to document and detail the SPL architectures in a very high-level manner. FORM proposes the modeling of an SPL architecture using three models: (i) *subsystem model* - presents the overall system structure; (ii) *process model* - details the dynamic behavior of the system; and (iii) *module model* - specifies each reusable component of the architecture. PLUS adopts traditional UML models marked with additional stereotypes to classify the system classes. It mentions the use of agents in the design of an SPL architecture, but it does not define a way to document it. Table 1 shows that the other investigated approaches [18, 6] do not provide explicit support to specify and model the SPL architecture and its respective components.

3. ExpertCommittee Case Study Overview

Our approach was developed based on our experience with the ExpertCommittee (EC) [15] case study, a multi-agent system product line for the web domain.

The EC is a conference management system, developed as 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. 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.

This MAS-PL was developed in an evolutionary way. We present details about the MAS-PL development (Section 3.1). After that, we discuss some MAS particular variability types that we have identified in our case study (Section 3.2).

3.1. The EC MAS-PL

We developed our case study considering that an evolving system can be seen as an SPL, because the features that are common to all versions of the system comprise the core architecture of the product line. Thus, each version of the system, which has new features, characterizes a new product.

Our MAS-PL was developed in an evolutionary way. There were three versions of the EC. The first version of the EC is a typical web-based application composed of the mandatory features that support the process of conference management. It was structured according to the Layer architectural pattern [7]. The second version of the EC system contains features that are related to autonomous behavior, such as deadline and pending tasks monitoring, 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.

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[1] configuration files that allows the injecting of dependencies inside the variable points of the EC SPL architecture, which can be seen in Figure 1.

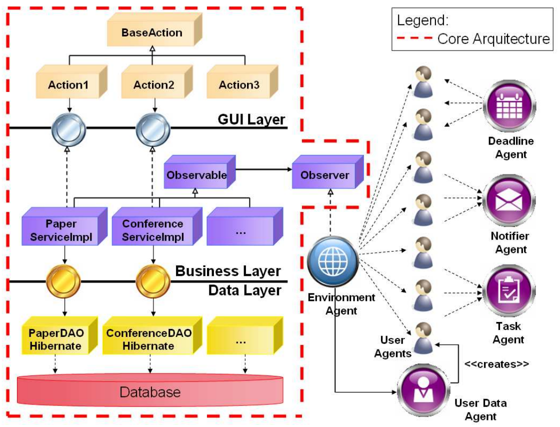


Figure 1. EC MAS-PL Architecture.

3.2. Dealing with Variability

Different kinds of variability were identified in the EC MAS-PL. In our case study, we mainly explored the variabilities related to autonomous behavior and their respective implementation using software agents. Throughout this paper, these kinds of features are called agency feature. Next we briefly describe them:

[1]http://www.springframework.org/

New Autonomous Behavior. We had to introduce agents into the architecture when we added autonomous behavior to the system. The *Task Management* feature implied the addition of a new agent in the system, which can be present or not, depending on the product being derived;

New Behavior for an Agent or Role. Some features have an impact inside the agent or the role. They allow defining agent internal variabilities by defining specific new behaviors of agents. The *Conference Suggestion Feature* is an autonomous feature; thus, the user agent, or more specifically the author role, performs it. When a paper is registered in a conference, the author role perceives it and sends suggestions of related conferences for the author who has registered his/her paper;

New Role for an Agent. Each role of the EC has a corresponding role in the user agent when a product has some autonomous behavior. However, not all roles are mandatory, such as the role *Reviewer*. Thus, roles must be modeled in a way that they can be (un) plugged.

Almost all the autonomous behavior features are accomplished by the collaboration of different agents. In our study, we have identified that many of these features are typically addressed by a different set of components and agents from the SPL architecture. In this way, a particular challenge of our study was to document and model the structure and behavior of these crosscutting features in domain analysis and design.

4. Modeling and Documenting Agency Features

In this section, we discuss the modeling and documentation of the agency features from the EC MAS-PL, presented in Section 3. We focus specifically on the domain analysis and design stages. We have initially analyzed how existing SPL and MAS-PL approaches can deal with the specification and modeling of agency features. Based on the deficiencies and lack of expressivity of these existing approaches, we propose new extensions to document the agency features of the EC MAS-PL. The main aim of our work is to define a set of guidelines to model and document agency features along all SPL development stages.

4.1. Domain Analysis

The domain analysis stage defines activities for eliciting and documenting the common and variable requirements of an SPL. It is concerned with the definition of the domain and scope of the SPL, and specifies the common and variable features of the SPL to be developed. In our study, we have analyzed how the modeling and documentation notations of current SPL approaches can deal with agency features. Table 1 shows the results obtained considering the SPL methodologies investigated in our study.

The EC MAS-PL features modeling and documentation was supported by the feature model proposed in [5]. It is an evolution of the original feature model proposed in [11] and also adopted by FORM. Figure 2 shows a partial view of this feature model. The features that were in all the versions are the mandatory ones. Features that made part of only some versions or varied from one version to another one are the optional features.

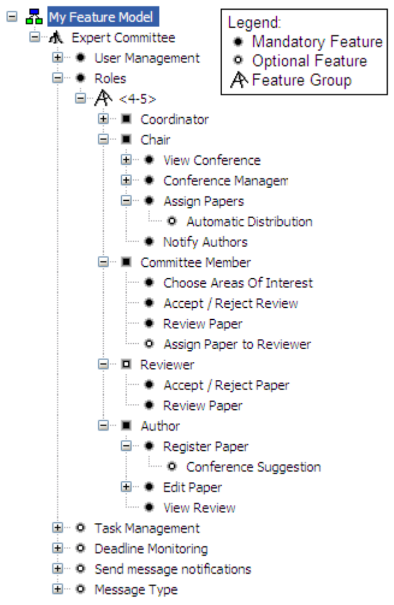


Figure 2. Feature Model.

The way proposed in the PLUS method was quite adequate to model our use cases. Use cases are grouped in packages according to the feature to which it was related. In this 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. We adopted these tables instead of the graphical notation of [18]. Figure 3 shows a partial view of the EC MAS-PL use case model. It contains three kernel use cases, one optional use case related to the reviewer role and two agency features: task management and conference suggestion. The following adaptations were applied to the use case notation proposed in [8] to better specify the agency features: (i) agents were represented with the same symbol as actors and are associated to the use cases with which they are involved; (ii) the <<*agency feature*>> stereotype was adopted to indicate that the use cases of a specific package is related to an agency feature.

The detailed description of the EC MAS-PL use cases

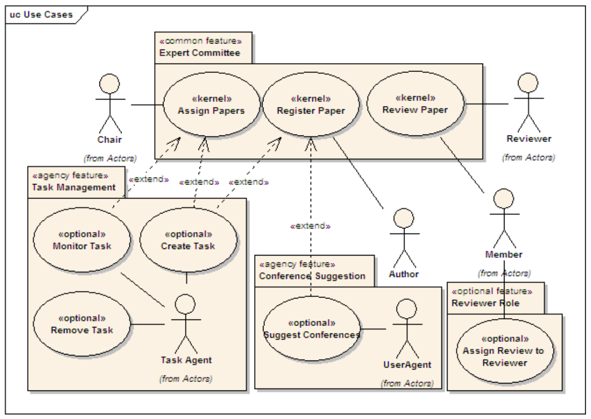


Figure 3. Use Case Diagram.

was carried out in the following way: (i) the kernel use cases were described using the common documentation provided by existing UML methods; and (ii) the agency features were documented using the template depicted in Table 2. This new template details important information to understand the interactions between the agency feature and other ones, such as: the event that starts the use case, the agents and roles that are involved and if the feature is mandatory, optional or alternative. We did not used the template proposed in [6] because it is a too low-level specification and it addresses the internal variability of the agents.

Table 2. Agency Feature Description.

Agency Feature: Conference Suggestion
Reuse Category: Optional
Dependency: Extends *Register Paper* Use Case
Description: When a paper is registered to a conference
Event: paper was registered to a conference
Agent/Roles: user agent / author role, notifier agent
Main Flow: 1. User registers a paper to a conference. 2. User Agent perceives the change in the environment. 3. Author role detects the conferences that have areas of interest similar to the ones of the registered paper and creates a message to be sent to the user. 4. Author role sends a message to the Notifier Agent requesting to send the message to the user. 5. Notifier Agent sends the message.

4.2. Domain Design

The domain design aims at defining an architecture that addresses both the common and variable features of an SPL. A set of components and core assets can be specified as part of the SPL architecture. The modularization of features must also be taken into account during the design of the architecture core assets to allow the (un) plugging of features.

The EC MAS-PL architecture was documented in our case study in two different levels: (i) a component view - that illustrates the main components (or subsystems) of the

SPL architecture; and (ii) a logical view - that details the different components defined for the SPL architecture in terms of UML class diagrams. Figures 1 and 4 show, respectively, the component and logical view of the EC architecture. The component view details the web system layers and the deployed agents that execute inside this system. The component view gives not only an overall overview of the SPL architecture components and agents, but also expresses their organization in runtime.

The logical view details the architecture components and agents in terms of UML class diagrams. Similar to PLUS, we used stereotypes to classify the classes, but our classification was mandatory (kernel), optional or alternative. The classes of different components can be organized in packages, or they also can be colored to characterize a specific component. Figure 1 shows the main components of the EC MAS-PL (GUI, Business and Data Layers), and the different agents responsible for implementing the autonomous behavior of the system. Each different agency feature of the MAS-PL can be detailed using: (i) a separate class diagram that only contains the classes responsible for implementing that feature and alternatively the classes that are related with it; (ii) a colored indication in the main class diagram that shows the elements (classes, interfaces, methods) related to the implementation of that feature. It is exemplified in Figure 4; and (iii) a specific design template that details the components and agents involved in the realization of an agency feature, and their respective interactions.

Table 3 shows the design template of the Conference Suggestion agency feature. It details the goals, entities, events and execution plan related to the conference suggestion feature provided by a set of agents. It complements the agency feature description provided in domain analysis (Figure 2) by detailing the communication of the different system agents and the environment. While the class diagrams of an agency feature describe the elements that modularize it, our template design details the dynamics of the agents involved in its realization.

Table 3. Agency Feature Design Description.

Agency Feature: Conference Suggestion	
Goal: Send conference suggestions to users	
Entities: EnvironmentAgent, UserAgent, NotifierAgent, AuthorRole and ConferenceService.	
Events Generated: SendMessage **Events Perceived:** RegisterPaper	
Plan:	
Environment Agent	*Action:* send message to User Agents *Message Content:* paper registered
User Agent	*Action:* creates Author Role and adds it to the agent *Condition:* user is the first author of the paper
Author Role	*Action:* send message to Conference Service *Message Content:* conferences related to the conference the author has registered
Conference Service	*Action:* send message to Author Role *Message Content:* related conferences
Author Role	*Action:* creates user message with conferences returned *Action:* send message to Notifier Agent *Message Content:* user message to be sent to the user
Notifier Agent	*Action:* send user message

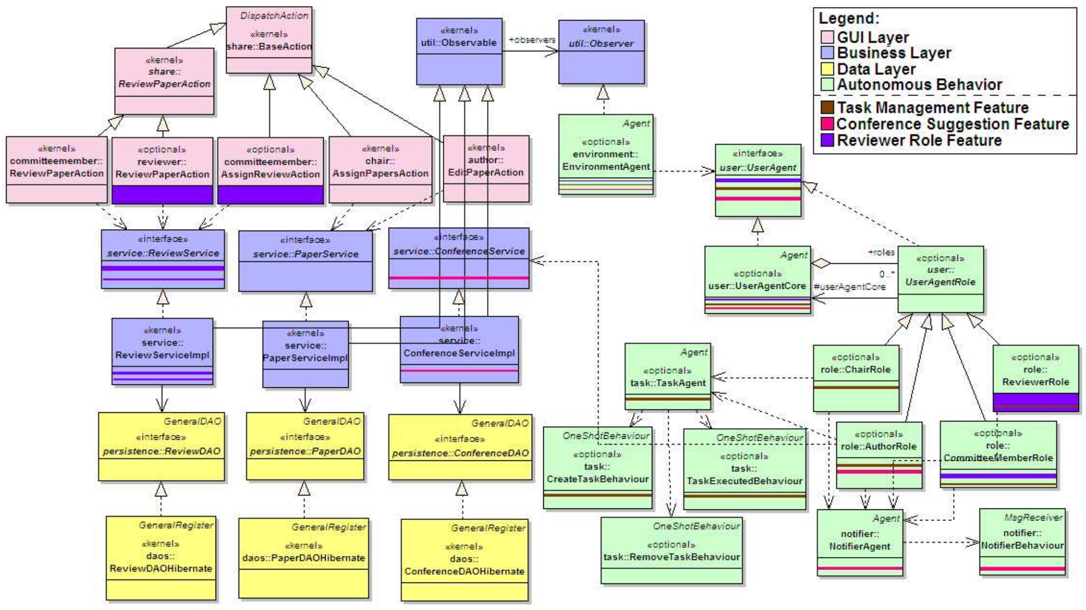
5. Discussions

In this section, we discuss some lessons learned and challenges that we have found when documenting the agency features of EC MAS-PL. These lessons learned offer directions for a methodology for developing MAS-PL that we are currently defining.

Agency Feature Documentation using SPL methodologies. During the modeling and documentation of the EC MAS-PL, 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. In our case study, we adopted a different strategy to model the SPL agency features. We started modeling the agency features using only the notations provided by SPL methodologies to investigate their expressivity. After that, we adapted and complemented the selected notations to improve the documentation of the agency features. The domain analysis and design templates were created in this context.

MAS-PL methodologies. The investigated 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 and extend it with SPL techniques for a particular purpose. Pena et. al. [16] adapt the Methodology for analyzing Complex MultiAgent Systems (MaCMAS) to deal with evolving systems. Dehlinger & Lutz [6] 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 MASs, which was developed using the MaCMAS and the Gaia methodologies. The main problems that we have observed when using these MAS-PL methodologies to model and document the EC MAS-PL 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.

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. In our study, we observed that the current SPL methodologies do not provide clear support to deal with the



Legend:
GUI Layer
Business Layer
Data Layer
Autonomous Behavior
Task Management Feature
Conference Suggestion Feature
Reviewer Role Feature
DispatchAction «kernel» share::BaseAction
«kernel» share::ReviewPaperAction
«kernel» committeemember::ReviewPaperAction
«optional» reviewer::ReviewPaperAction
«optional» committeemember::AssignReviewAction
«kernel» chair::AssignPapersAction
«kernel» author::EditPaperAction
«kernel» util::Observable
+observers
«kernel» util::Observer
Agent «optional» environment::EnvironmentAgent
«interface» user::UserAgent
Agent «optional» user::UserAgentCore
+roles
0..*
#userAgentCore
«optional» user::UserAgentRole
«interface» service::ReviewService
«interface» service::PaperService
«interface» service::ConferenceService
«kernel» service::ReviewServiceImpl
«kernel» service::PaperServiceImpl
«kernel» service::ConferenceServiceImpl
GeneralDAO «interface» persistence::ReviewDAO
GeneralDAO «interface» persistence::PaperDAO
GeneralDAO «interface» persistence::ConferenceDAO
GeneralRegister «kernel» daos::ReviewDAOHibernate
GeneralRegister «kernel» daos::PaperDAOHibernate
GeneralRegister «kernel» daos::ConferenceDAOHibernate
Agent «optional» task::TaskAgent
«optional» role::ChairRole
«optional» role::ReviewerRole
OneShotBehaviour «optional» task::CreateTaskBehaviour
OneShotBehaviour «optional» task::TaskExecutedBehaviour
OneShotBehaviour «optional» task::RemoveTaskBehaviour
«optional» role::AuthorRole
«optional» role::CommitteeMemberRole
Agent notifier::NotifierAgent
MsgReceiver notifier::NotifierBehaviour

Figure 4. Class Diagram of the EC product line.

documentation of these crosscutting features. In domain design, we have proposed a template design to help the documentation of the agency features. It allows specifying how the different design elements interact to address a specific agency feature. We are currently investigating how existing aspect-oriented approaches [9, 2] can help the visual documentation of the agency features in combination with our templates.

6. Conclusions and Future Work

In this paper, we presented an exploratory study that analyzed and discussed how existing SPL approaches can help the documenting and modeling of multi-agent system product lines (MAS-PLs). Different agency features were presented, which were added to an existing web-based conference management system as optional features. Three types of agency variabilities were addressed in our paper: addition of agents; addition of plans; and addition of roles. Most of the MAS-PL documentation was supported by the PLUS approach, showing the effectiveness of current SPL approaches to document MAS-PL. However, the documentation of the agency features required the creation of additional templates to specify: (i) the interdependencies and relationships between core functionalities (mandatory use cases) and optional agency features (optional use cases) in domain analysis; and (ii) the elements and dynamics responsible to address a given agency feature in domain design.

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. The proposed methodology aims to be simple and systematic. We believe that due to the high complexity of many SPL methodologies, many of them are not used in practice. Different and new abstractions have been proposed in these methodologies, making the understanding and adoption of them difficult. 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. Our approach aims to be systematic in the sense of providing clear and detailed guidelines about how developers should use it.

References

[1] C. Atkinson, J. Bayer, and D. Muthig. Component-based product line development: The KobrA approach. In P. Donohoe, editor, *SPLC*, pages 289–309, 2000.

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

[3] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, USA, 2002.

[4] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[5] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration using feature models. In *SPLC*, pages 266–283, 2004.

[6] J. Dehlinger and R. R. Lutz. A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems. In *SELMAS*, pages 1–7, New York, NY, USA, 2005. ACM Press.

[7] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[8] H. Gomaa. *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.

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

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

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

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

[13] M. Matinlassi. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada. In *ICSE*, pages 127–136, Washington, DC, USA, 2004. IEEE Computer Society.

[14] I. Nunes. Towards a multi-agent product line development methodology, 2008. http://www.inf.puc-rio.br/ ioliveira/maspl/.

[15] I. Nunes, C. Nunes, U. Kulesza, and C. Lucena. Developing and evolving a multi-agent system product line: An exploratory study. In *AOSE (to appear)*, 2008.

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

[17] J. Pena, M. G. Hinchey, and A. Ruiz-Cortés. Multi-agent system product lines: challenges and benefits. *Commun. ACM*, 49(12):82–84, 2006.

[18] K. Pohl, G. Bckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, New York,USA, 2005.

[19] M. Wooldridge and P. Ciancarini. Agent-Oriented Software Engineering: The State of the Art. In P. Ciancarini and M. Wooldridge, editors, *AOSE*, volume 1957, pages 1–28. Springer-Verlag, Berlin, 2000.