1
Introduction

Over the past decades, agents have become a powerful software abstraction to support the development of complex and distributed systems (Jennings 2001). They are a natural metaphor to understand systems that present some particular characteristics such as high interactivity and multiple loci of control. These systems can be decomposed in several autonomous and pro-active agents comprising a Multi-agent System (MAS). This autonomy property refers to agents able to act without the intervention of humans or other systems: they have control both over their own internal state, and over their behavior (Wooldridge 1999). In order to develop agent-based systems, adequate techniques that explore their benefits and their peculiar characteristics are required (Zambonelli 2000). In this context, Agent-oriented Software Engineering (AOSE) has emerged as a new software engineering paradigm to support the development of MASs and to help on the industrial exploitation of agent technology; and then several research work has been proposed in this direction, such as methodologies (Wooldridge 2000a, Cossentino 2005, Bresciani 2004), modeling languages (Silva 2007, Silva 2003a, Silva 2004a, Silva 2008, Bauer 2001, Bauer 2002), development platforms (Bellifemine 2007, Pokahr 2005, Bordini 2007, Howden 2001), tools and programming languages. As a consequence, MASs are gaining wide acceptance in both industry and academia; however, although many agent-oriented methodologies have been proposed, none is mature enough to be used in industrial and commercial environments (Shehory 2001).

The main aims of software engineering is 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 goals. Software reuse techniques have contributed for significant improvements to reduce both time and cost of software development. 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 application frameworks (Fayad 1999) and libraries, software architectures

(Shaw 1996) and patterns (Gamma 1995, Buschmann 1996). In addition, some research work has been published in order to bring the advantages of software reuse to the MAS development. Several of these works are exploiting patterns reuse (Cossentino 2002, Cossentino 2003, Gonzalez-Palacios 2004, Lind 2002); however, most of the agent-oriented methodologies do not take into account the adoption of extensive reuse practices, and address the development of single systems (Girardi 2002).

In the context of software reuse, the concepts of system families and Software Product Lines (SPLs) have gained a significant popularity throughout the software industry and research community, leading to the emergence of a new field called Software Product Line Engineering (SPLE). In this approach, reuse evolves from an *ad-hoc* to a systematic way. A SPL (Clements 2002) refers to a family of systems sharing a common, managed set of features that satisfy the needs of a selected market. The systems are developed from a common set of core assets in a prescribed way. SPLE aims at exploring existing common and variable features of the set of systems, in such way that different customized applications can be developed from a reusable set of artifacts with a reduced time-to-market. A feature (Czarnecki 2006) is a property or functionality of the SPL that is relevant to some stakeholder. The development of a SPL is typically divided into two key processes: (i) domain engineering – the commonality and the variability of the SPL are defined and realized; and (ii) application engineering – the applications of the SPL are built by reusing domain artifacts and exploiting the product line variability.

Over the last years, several approaches for developing system families and SPLs have been proposed (Weiss 1999, Czarnecki 2000, Atkinson 2002, Clements 2002, Gomaa 2004, Pohl 2005). The main goal of most of these approaches is to define, model and implement a common and flexible architecture, which addresses the common and variable features of the SPL. In order to develop such architecture, current approaches either provide high-level guidelines to implement SPLs (Weiss 1999, Clements 2002) or use different existing technologies, such as component-based (Atkinson 2002), object-oriented (Gomaa 2004), code generation (Czarnecki 2000) and Aspect-oriented Programming (AOP) (Alves 2006, Kulesza 2006a).

Only recent research (Dehlinger 2007, Pena 2006a) has explored the integration synergy of SPLs and MASs technologies, by incorporating their respective benefits. This integration results in product lines that present features that take advantage of agent technology, comprising Multi-agent System Product Lines (MAS-PLs). These features, named agent features, present an autonomous or pro-active behavior; therefore the agent abstraction

is appropriate for their development.

1.1 Problem Statement and Limitations of Existing Work

As stated in the previous Section, both MASs and SPLs can provide several benefits for the software development. Nevertheless, little effort has been made in order to combine these technologies. In (Pena 2006b), there are many challenges in the MAS-PL development to be overcome. After reviewing SPL, MAS and MAS-PL approaches, we have identified some issues that need to be addressed in the MAS-PL development.

SPL methodologies do not address agent features. In (Nunes 2008a), we have identified that most of the SPL methodologies provide useful notations to model the agent 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.

MAS methodologies do not address the development of system families. Several MAS methodologies have been proposed (Sellers 2005); however they are devoted to developing single products. Therefore, these approaches do not cover some of the activities of SPL. These are mainly concentrated on commonality analysis, and its implications for the SPL approach.

MAS methodologies do not consider the integration with other existing approaches. MAS methodologies usually propose to distribute all the system functionalities and responsibilities among agents. Agents are an abstraction that provides some particular characteristics, such as autonomy and pro-activeness. Therefore, we claim that features of the SPL that do not take advantage of agent technology can be modeled and implemented using traditional design and programming techniques.

MAS-PL approaches fail to provide a complete solution for MAS-PL development. MAS-PL approaches do not address development scenarios of traditional SPL architectures using agent technology. Instead, they adopt an existing MAS methodology as a basis and extend it with SPL techniques for a particular purpose. The main problems that we have observed (Nunes 2008a) in these MAS-PL approaches to model and document MAS-PLs were: (i) they do not offer a complete solution to address the modeling of agent features in both domain analysis and design; (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 (iii) do not explicitly capture the separate modeling of agent features.

1.2 Proposed Solution and Contributions Overview

This dissertation contemplates the definition of a domain engineering process for developing MAS-PLs. The process focuses on system families and includes domain scoping and variability modeling techniques. Our approach is the result of an investigation of how current SPL, MAS and MAS-PL approaches can model MAS-PL. Based on this experience, we propose our process, which is built on top of some MAS and SPL approaches. We have combined some techniques and notations of these approaches (Gomaa 2004, Cossentino 2005, Silva 2007) and added some extensions/adaptations. The scenario that we have explored is the incorporation of autonomous or proactive behavior into existing web systems. The main idea is to introduce software agents into existing web applications in order to allow the (semi)automation of tasks, such as autonomous recommendation of products and information to users (Holz 2008). Due to the existence of many web applications already developed and deployed on application servers, our MAS-PL approach aims at extending these web applications with the aim to bring the minimum impact to their provided features and services that are adequately structured according to classical architectural patterns, such as Layer and model-view-controller (MVC) (Buschmann 1996).

Therefore, the main contributions of this dissertation are:

(i) to define a whole domain engineering process for developing MAS-PLs, detailing its activities and their respective artifacts;

(ii) to provide notations to model and document agent variability;

(iii) to provide models to capture agent features traceability;

(iv) our approach models agent features independently, promoting a low impact in the incorporation of agents into existing systems designed with other technologies, such as object-oriented; and

(v) to present design and implementation guidelines to help in the development of MAS-PLs, in particular an architectural pattern that provides a general structure to add autonomous behavior to existing web applications using agent technology.

1.3 Dissertation Outline

The remainder of this dissertation is organized as follows.

Chapter 2 presents the state-of-art of the two development technologies combined in this dissertation: Multi-agent Systems and Software Product Lines. Besides presenting them, we point out some of their relevant approaches.

Chapter 3 presents the proposed domain engineering process. First, it gives an overview and details the key concepts of the process, and later describes each one of the phases and activities that compose it.

Chapter 4 describes two MAS-PL case studies for the web-domain, which were developed in order to help in the elaboration of our process and to evaluate it. Besides detailing the case studies, it presents artifacts generated during their development.

Chapter 5 presents lessons learned during the development of our process and case studies. We show guidelines to design and implement agent features in MAS-PLs. These guidelines aims at providing modularization to agent features in order to support variability. In addition, we discuss identified limitations of our approach.

Chapter 6 presents the related work of this dissertation. Two kinds of work are shown: MAS-PL approaches and integration approaches of web applications and software agents.

Chapter 7 presents concluding remarks and directions for future work. The main contributions of the dissertation are also detailed in this Chapter.