

A Multi-agent Systems Product Line Derivation Tool

Elder Cirilo¹, Ingrid Nunes¹, Uirá Kulesza² and Carlos Lucena¹

¹ PUC-Rio, Computer Science Department, Rio de Janeiro - Brazil
{ecirilo, ionunes, lucena}@inf.puc-rio.br

² Federal University of Rio Grande do Norte (UFRN) - Natal, Brazil
uira@dimap.ufrn.br

Abstract. Agent-oriented Software Engineering and Software Product Lines are two promising software engineering technologies, whose integration has been recently exploited in order to promote reuse and variability management in the context of complex and distributed systems. However, an automatic product derivation process and tools for supporting it are not addressed by existing research work. In this paper, we present a tool that allows modeling and managing variabilities of Multi-agent Systems Product Lines and automates their product derivation process. The tool is built on top of GenArch, our existing product derivation tool based on a model-driven approach.

Keywords: Multi-agent System, Software Product Lines, Product Derivation

1 Introduction

Multi-agent technology [1] has emerged as a promising technique to address the design and implementation of complex distributed systems. It provides a way of modeling complex problems in terms of abstractions of a higher level, e.g. agents and roles. On the other hand, Software Product Lines (SPLs) [2] have become a mainstream reuse practice that addresses the design and implementation of a set of domain related artifacts in order to deliver high quality software in a shorter time-to-market. Recent research work has exploited the integration between them, namely Multi-agent Systems Product Lines (MAS-PLs), to promote reuse and variability management in the context of complex systems. Research work associated with the MAS-PL development has proposed extensions of MAS methodologies [3, 4] and new processes [5] to support the analysis, design and implementation of MAS-PLs.

Although these proposed approaches provided substantial advances in the MAS-PL development, they mainly focus in the domain engineering process and little effort has been done in order to automate the application engineering process, in which the applications of the SPL are built by reusing domain artifacts and exploiting the SPL variability. Given that the success of the application engineering process is directly associated with the effectiveness of the SPL, there is a clear need of product derivation tools that automate the instantiation process by facilitating the selection, composition, configuration and integration of MAS-PL assets and their respective variabilities. In particular, MAS-PL calls for product derivation tools that must able to

deal with several concerns, such as trust, coordination, transaction, state persistence, that are usually implemented by different technologies, application frameworks or platforms. In addition, the configuration knowledge associated with agent abstraction is usually spread over two or more files [6]. In this context, current product derivation tools, such as Gears and pure::variants, are not suitable to manage the configuration knowledge of MAS-PLs. Thus, in this paper we present a product derivation tool that incorporates specific capabilities to manage variability and automatically derives products of MAS-PLs. It was built on top of GenArch [7], our existing product derivation tool based on a model-driven approach.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of our tool. Section 3 presents how our tool was implemented and technologies used to implement it, followed by Section 4, which concludes this paper.

2 MAS-PL Product Derivation Tool – Approach Overview

Our tool aims at making the mainstream software developer community able to use the concepts and foundations of agent-oriented software engineering in the development of MAS-PLs. It implements a variability management approach founded on generative programming [8]. The variability management is performed by means of four different models: (i) architecture model (A); (ii) agent-specific architecture model (M); (iii) feature model (F); and (iv) configuration model (C). Figure 1 shows an overview of our tool architecture and its models.

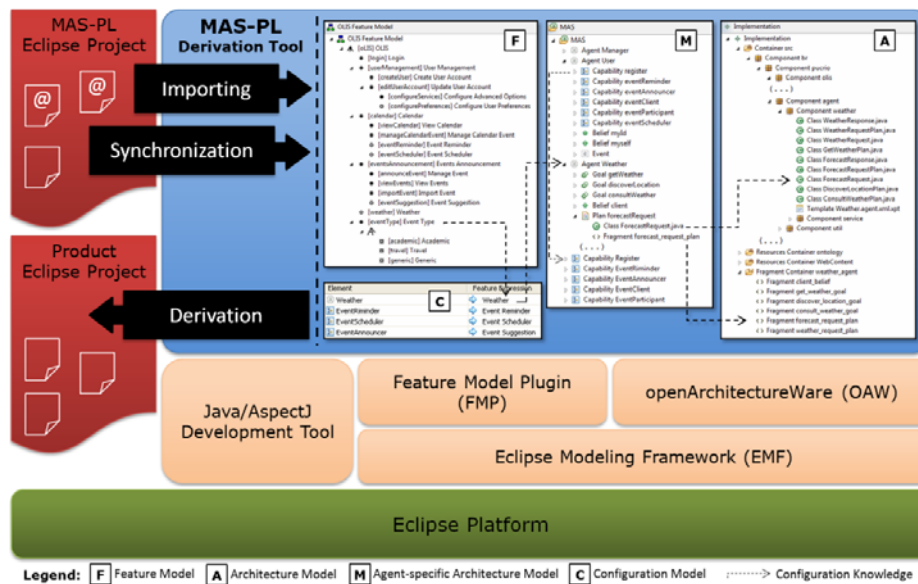


Figure 1. MAS-PL Derivation Tool Overview

The *architecture model* defines a visual representation of the MAS-PL elementary elements (classes, aspects, templates, folders, files, and code fragments). Fragments

represent pieces of code (or text). They are used mainly to represent variability that may exist in configuration files, e.g. XML or properties files. The *agent-specific architecture model* provides a visual representation of the MAS-PL architecture in terms of agent abstractions (agent, capability, goals, beliefs, etc.). It provides a modular solution to document and trace features to lower level elements (classes, files, code fragment), enabling an automatic analysis of features covering and the impact of changing requirements. The *feature model* [8] is used in our tool to represent commonalities and variabilities that exist in the SPL architecture. Finally, the *configuration model* is responsible for defining the mapping between features and implementation elements. It represents the configuration knowledge from a generative approach [8], being fundamental to link the problem space (features) to the solution space (implementation elements) and enable automatic product derivation. This is the main advantage of our tool – features may be mapped to agent concepts and then to lower level elements, such as classes and fragments, and therefore these higher level concepts do not get diffused into the configuration knowledge.

The tool provides functionalities to create and maintain these models (automatic importing and synchronization) and to use them to automatically derive products. An initial version of the derivation models can be automatically created by parsing the code assets that implements the MAS-PL architecture. Specific Java annotations (`@Feature` and `@Variability`) can also be used in code assets (Java classes and AspectJ aspects) to characterize that these elements address a specific feature and represent a variation point. In addition, our tool is able to parse files of a specific agent platform, the Jadex, in order to create an initial version of the agent-specific architecture model. Jadex is based on Java and XML files. The last is used to define agents and their related concepts, such as beliefs, goals and plans.

The automatic product derivation process is driven by an instance, also known as configuration, of the feature model and by the configuration knowledge expressed in the configuration model. The feature model configuration informs which functionalities the product must contain. The configuration model provides the information needed to resolve which code assets must be instantiated and customized. The classes, aspects or files that must be customized are implemented using the template technology. The variable pieces from a template are customized based on the information provided by the derivation models (architecture and agent-specific architecture models). The derivation process is concluded with code generation based on processing templates and loading the selected and generated code assets into specific folders of a new Eclipse Java project, which represents the derived product.

3 Tool Architecture and Implementation

Figure 1 shows the general structure of our tool architecture based on Eclipse platform technologies. It was developed as an Eclipse plug-in using different toolkits available at this platform, such as Eclipse Modeling Framework (EMF); and openArchitectureWare (oAW).

Derivation models were specified using the EMF, which is a Java/XML framework that enables building model-based tools using structured data models. It generates a

set of Java classes to manipulate and persist models. The feature model is implemented by the Feature Modeling Plugin (FMP). This plug-in allows modeling the feature model proposed by Czarnecki et al. [8], which supports mandatory, optional, and alternative features, and their respective cardinality.

Our tool encompasses three main functionalities structured over three modules: (i) automatic models construction – importing module; (ii) artifacts synchronizations – synchronizing module; and (iii) automatic product derivation – the derivation module. (i) uses the Java Development Tooling (JDT) and AspectJ Development Tools (AJDT) to browse the Abstract Syntax Tree (AST) of Java classes and AspectJ aspects, respectively, in order to process the Java annotations to build derivation models. These plug-ins are also used by the synchronization module to keep consistency between models and code assets. Our tool adopts the XPand language of openArchitectureWare (OAW) plug-in to specify code templates that are used by the derivation module to perform code asset customization.

4 Conclusion and Future Work

In this paper, we presented a tool for supporting automatic product derivation of MAS-PLs. This tool provides a set of models to specify the configuration knowledge associated with agent concepts and code assets of the MAS-PL architecture and to enable automatic product derivation. We have experienced our tool with MAS-PLs implemented using the Jadex framework. A preliminary evaluation shows that it is able to reduce significantly the amount of lines of XML code that must be manually manipulated during the derivation process [6]. As future work, we intend to incorporate new architecture models to this tool in order to deal with different concerns, such as persistence, user interface, and service composition.

References

- [1] Wooldridge, M.: *An Introduction to MultiAgent Systems*. Second edn. John Wiley & Sons (2009)
- [2] Clements, P., Northrop, L.: *Software Product Lines: Practices and Patterns*. Addison-Wesley (2002)
- [3] Pena, J., Hinchey, M.G., Ruiz-Corts, A., Trinidad, P.: Building the core architecture of a multiagent system product line: with an example from a future nasa mission. In: *Agent-Oriented Software Engineering VII*. Volume 4405 of LNCS. (2006) 208-224
- [4] Dehlinger, J., Lutz, R.R.: Supporting requirements reuse in multi-agent system product line design and evolution. In: *ICSM*. (2008) 207-216
- [5] Nunes, I., Lucena, C., Kulesza, U., Nunes, C.: On the development of multi-agent systems product lines: A domain engineering process. In: *AOSE'09*. (2009) 109-120
- [6] Cirilo, E., Nunes, I., Kulesza, U., Lucena, C.: Automating the product derivation process of multi-agent systems software product lines. In: *SBES'09* (to appear). (2009)
- [7] Cirilo, E., Kulesza, U., de Lucena, C.J.P.: A product derivation tool based on model-driven techniques and annotations. *J.UCS* **14**(8) (2008) 1344-1367
- [8] Czarnecki, K., Eisenecker, U.W.: *Generative programming: methods, tools, and applications*. Addison-Wesley, USA (2000)