An Aspect-Oriented Generative Approach

Uirá Kulesza
Computer Science Department
PUC-Rio, Brazil
55-21-2540-6915
uira@inf.puc-rio.br

Alessandro Garcia
Computer Science Department
PUC-Rio, Brazil
55-21-2540-6915
afgarcia@inf.puc-rio.br

Carlos Lucena
Computer Science Department
PUC-Rio, Brazil
55-21-2540-6915
lucena@inf.puc-rio.br

ABSTRACT
The integration of generative and aspect-oriented techniques is not a trivial task. This paper describes our experience in the definition of an aspect-oriented generative approach for the context of multi-agent systems. Our generative approach is composed of: (i) a domain-specific language called Agent-DSL, which allows to model crosscutting and non-crosscuting agent features; (ii) an aspect-oriented architecture that models a family of software agents; and (iii) a code generator that maps abstractions of the Agent-DSL to specific compositions of objects and aspects in specific implementations of agent architectures. The use of aspect-oriented techniques in the definition of our generative approach brought benefits to the modeling and code generation of crosscutting features since early design stages.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.11 [**Software Engineering**]: Software Architectures – Domain-Specific Architectures; D.2.13 [**Software Engineering**]: Reusable Software – Reuse models, Domain engineering.

General Terms
Design, Languages.

Keywords
Generative Programming, Aspect-Oriented Software Development, Multi-Agent Systems.

1. INTRODUCTION
Over the last years, generative programming and aspect-oriented software development have been proposed, aiming at increasing maintainability and reusability of software systems.

Generative Programming (GP) [1] is an approach based on domain engineering. It addresses the study and definition of methods and tools to enable the automatic production of software families from a high-level specification. GP promotes the separation of problem and solution spaces, giving flexibility to evolve both independently.

Aspect-Oriented Software Development (AOSD) [2] is an evolving paradigm to modularize crosscutting concerns that existing paradigms (e.g.: object-oriented) are not able to capture explicitly. Crosscutting concerns are concerns that often crosscut several modules in a software system. AOSD encourages modular descriptions of complex software by providing support for cleanly separating the basic system functionality from its crosscutting concerns. Aspect is the abstraction used to modularize the crosscutting concerns.

The use of aspect-oriented techniques in the definition of a generative approach can bring additional benefits for the development of system families, especially to the modeling and generation of crosscutting features since early development stages. However, the integration of generative and aspect-oriented technologies is not a trivial task. So far there is little understanding of the interplay between these techniques.

This paper describes an aspect-oriented generative approach for the development of multi-agent systems (MASs). The purpose of the generative approach is threefold: (i) to uniformly support crosscutting and non-crosscutting features of software agents starting at early development stages; (ii) to abstract the common and variable agent features; and (iii) to enable the code generation of aspect-oriented (AO) agent architectures.

2. THE GENERATIVE APPROACH
Figure 1 depicts our aspect-oriented generative approach that is composed of:

(i) a domain-specific language (DSL), called Agent-DSL, which is used to collect and model orthogonal and crosscutting features of software agents;

(ii) an AO architecture that models a family of software agents. It is centered on the definition of *aspectual* components to modularize the crosscutting agent features at the architectural level;

(ii) a code generator that maps abstractions of the Agent-DSL to specific compositions of objects and aspects in agent architectures.

Figure 1 also illustrates that MAS developers use the Agent-DSL language to describe models with the specific features of a given MAS. The code generator supports the generation of code associated with those features on the basis of our AO agent architecture. The development of the generative approach was organized into the typical phases of domain engineering processes: (i) domain analysis; (ii) domain design; and (iii) domain implementation.

2.1 Domain Analysis
In the domain analysis, we investigated the different concerns encountered in the development of software agents. These concerns were organized and modeled by using feature models. The main agent features found were: knowledge, interaction,

OOPSLA'04, Oct. 24–28, 2004, Vancouver, British Columbia, Canada.
ACM 1-58113-833-4/04/0010.

adaptation, autonomy and collaboration. Each of them is composed of specific subfeatures [3, 4].

In order to support the representation of crosscutting features, we have introduced a new kind of relation between features, called "crosscuts". We say that a feature A crosscuts a feature B, when either A or one of its subfeatures depends and inspects B or one of the subfeatures of B. The following agent features were characterized as being crosscutting: interaction, adaptation, autonomy and collaboration. Each of them inspects elements of the knowledge feature in order to exhibit a specific agent property. For example, the autonomy feature inspects changes on the knowledge feature in order to detect the need for autonomous proactive behavior.

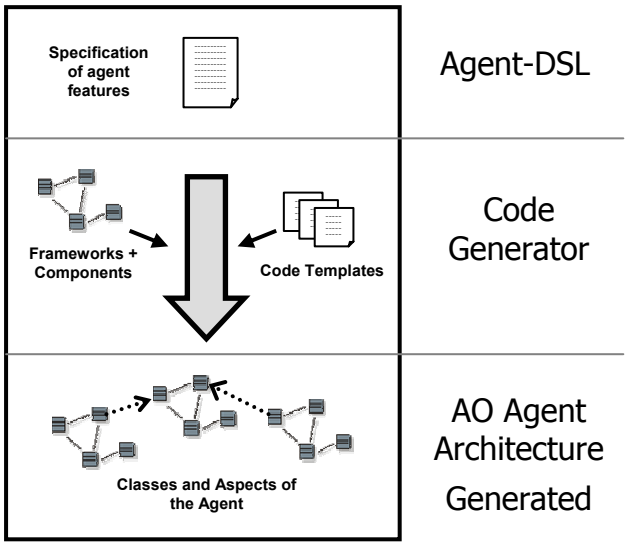

Figure 1. The Aspect-Oriented Generative Approach

2.2 Domain Design

In the domain design, we have developed a generic aspect-oriented (AO) agent architecture [3, 4] that considers each of the crosscutting and non-crosscuting features modeled during domain analysis. The architecture is composed of two kinds of components: (i) a central `Knowledge` component that modularizes the non-crosscutting subfeatures associated with the knowledge feature; and (ii) the *aspectual components* that separate the crosscutting agent features from each other and from the `Knowledge` component. Aspectual components represent crosscutting features at the architectural level.

A new notation has been used to graphically represent our AO agent architecture. It was developed to enable the representation of aspectual components. In the proposed notation, an aspectual component may crosscut other aspectual or non-aspectual components using its crosscutting interfaces. A *crosscutting interface* may add new state or behavior to other components or intercept (and modify) the existent behavior of components. Non-aspectual (normal) components are represented in a similar way to UML and offer their services through the *normal interfaces*.

2.3 Domain Implementation

In the domain implementation, we have used different technologies to implement the central components of the generative approach. First of all, the Agent-DSL was specified by using the XML-Schema technology. This DSL is used to specify the agency properties that agents could have to accomplish their tasks.

The AO agent architecture was implemented by using the Java and AspectJ programming languages. The basis of the architecture implementation is an AO framework that contains hot-spots as classes and aspects. The use of the aspect abstraction in the definition of frameworks enabled us to define common and variable behaviors of several crosscutting features.

In the configuration knowledge of the generative approach, we implemented a code generator as an Eclipse plug-in. This generator maps abstractions in the Agent-DSL to components and aspects of the agent architecture. The main task of the generator is to instantiate the AO framework, creating subclasses and subaspects for specific hot-spots of the framework. Depending on the agent descriptions provided, new types of agents (or roles) with their respective agent properties can be generated. Java Emitter Templates (JET), a generic template engine of the Eclipse Modeling Framework (EMF), has been used to write the code templates that specify the classes and aspects of the agent architecture to be generated.

3. CONCLUSIONS

This work presented the definition process of an AO generative approach. The goal of this approach is to explore the horizontal domain that MASs represent in order to enable the code generation of agent architectures. We organized the development of the generative approach using typical phases encountered in domain engineering processes. During the development process of the generative approach, it was necessary to adapt modeling notations used in generative programming due to the adoption of the AO paradigm.

The integrated use of generative programming and AO techniques brought additional benefits to the development of software families, such as: (i) clear separation of orthogonal and crosscutting features starting at early design phases; (ii) direct mapping of crosscutting features in aspectual components; (iii) simplified implementation of code generators, because the composition of crosscutting concerns is accomplished by the aspect weavers, and (iv) improved reuse of artifacts associated with crosscutting agent features.

4. ACKNOWLEDGMENTS

The authors have been partially supported by CNPq and FAPERJ.

5. REFERENCES

[1] K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000.

[2] G. Kiczales, et al. "Aspect-Oriented Programming". Proc. of ECOOP'97, LNCS 1241, Springer-Verlag, Finland, June 1997.

[3] U. Kulesza, A. Garcia, C. Lucena. "Generating Aspect-Oriented Agent Architectures". Proceedings of the 3rd Workshop on Early Aspects, AOSD'2004, March 2004, Lancaster, UK.

[4] U. Kulesza, A. Garcia, C. Lucena, A. von Staa. "Integrating Generative and Aspect-Oriented Technologies". Proceedings of the Brazilian Symposium on Software Engineering (SBES´2004), Brasilia, Brazil, October, 2004.