# Towards a Method for the Development of Aspect-Oriented Generative Approaches

Uirá Kulesza  Alessandro Garcia  Carlos Lucena

*Software Engineering Laboratory – SoC+Agents Group*
*Computer Science Department*
*Pontifical Catholic University of Rio de Janeiro - PUC-Rio - Brazil*
*{uira, afgarcia, lucena}@inf.puc-rio.br*

## Abstract

Generative programming and aspect-oriented software development have been proposed over the last years aiming at increasing maintainability and reusability of software systems. While several research works have focused on the investigation of the individual use of each of these software engineering approaches, less attention has been paid to the integration of these two techniques. The integration of generative and aspect-oriented technologies can bring several benefits to the modeling and code generation of crosscutting features since early design stages. In this paper, we present some lessons learned from the development of an aspect-oriented generative approach for the domain of multi-agent systems. Based on this experience, we also present a preliminary version of a method to develop aspect-oriented generative approaches.

## 1. Introduction

Generative programming and aspect-oriented software development have been proposed over the last years aiming at increasing maintainability and reusability of software systems. Generative Programming (GP) [1] addresses the study and definition of methods and tools to enable the automatic production of software families from a high-level specification. Aspect-Oriented Software Development (AOSD) [2, 3] is an evolving paradigm which 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.

Each of these software engineering approaches can bring particular benefits to the development of software systems. GP promotes the separation of problem and solution spaces, giving flexibility to evolve both independently. Moreover, GP also brings benefits to the production and quality of system families by means of code reuse and generation. AOSD promotes improved separation of concerns, leading to the development of software systems that are easier to maintain and reuse.

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. Interesting questions need to be considered when developing an aspect-oriented generative approach, including: How to model crosscutting features in the problem space? How to design aspect-oriented architectures that address the crosscuting and non-crosscutting features modeled? Which technologies (domain-specific languages, frameworks) are appropriate to implement these aspect-oriented generative approaches?

Recent work explored the use of GP and AOSD together [4, 5, 6]. However, these reports did not cover nor describe in detail the typical phases (domain analysis, domain design and domain implementation) found while defining a generative approach.

In this paper, we describe our experience in the definition of an aspect-oriented generative approach for the domain of multi-agent systems (MASs). The development of the generative approach was organized by using the domain engineering method presented in [1]. However, the use of aspect-oriented techniques required the adaptation of several steps and notations existent in this method. Based on this experience, we also present some lessons learned and a preliminary version of a method to develop aspect-oriented generative approaches.

The remainder of this paper is organized as follows. Section 2 gives a brief description of the generative approach. Section 3 presents: (i) the development

process of the generative approach; (ii) the required adaptations of the domain engineering method to accommodate the use of aspect-oriented technologies; and (iii) some lessons learned. Section 4 illustrates a preliminary method for the development of aspect-oriented generative approaches. Section 5 discusses some related work. Section 6 presents our conclusion and points to directions for future work.

## 2. The Generative Approach for MAS

The aspect-oriented (AO) generative approach was developed aiming at exploring the Multi-Agent Systems (MASs) domain to improve the quality and productivity on the development of these systems. The generative approach, depicted in Figure 1, is composed of:

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

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

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

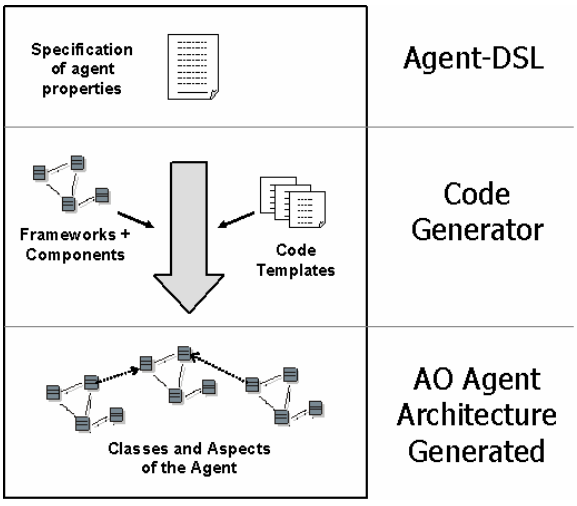


**Figure 1**. Generative Approach for MASs

Figure 1 also illustrates the use of the generative approach. MAS developers use the Agent-DSL 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, composed of an aspect-oriented framework and code templates. For further details about the generative approach, one can refer to [7, 8].

## 3. From Domain Engineering To Aspect-Oriented Generative Approaches

In this section, we present the development process of the aspect-oriented generative approach. Following the guidelines presented by Czarnecki and Eisenecker [1], we have organized the development into three phases: domain analysis; domain design; and domain implementation. However, the use of aspect-oriented techniques required some changes during the accomplishment of these phases, such as: (i) the adaptation of modeling notations used in domain analysis and design; and (ii) the use of different techniques to accomplish the generative approach, such as, configuration DSLs and aspect-oriented frameworks. Sections 3.1, 3.2 and 3.3 detail and analyse each of these adaptations and new techniques used. Section 3.4 synthetizes the lessons learned of the development process.

### 3.1. Domain Analysis

In the domain analysis, we focused on the identification and modeling of common and variable features of the MASs domain concepts. Feature models [1] were used to represent the features and their respective relations. During the domain analysis, because we were also interested in modeling crosscutting agent concens, we identified the need to represent crosscutting features in a feature model. Thus, we have extended this notation to include the representation of crosscutting features.

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 adaptation feature inspects changes on the knowledge feature (goal instantiation) in order to detect the need for a plan to be executed. Figure 2 depicts the partial feature model resulted from the domain analysis. For further details about the agent features, one can refer to [7, 8, 9].

### 3.2. Domain Design

In the domain design, it was specified a generic and flexible AO agent architecture. Each feature modeled during domain analysis was considered. The AO architecture, depicted in Figure 3, is composed of two

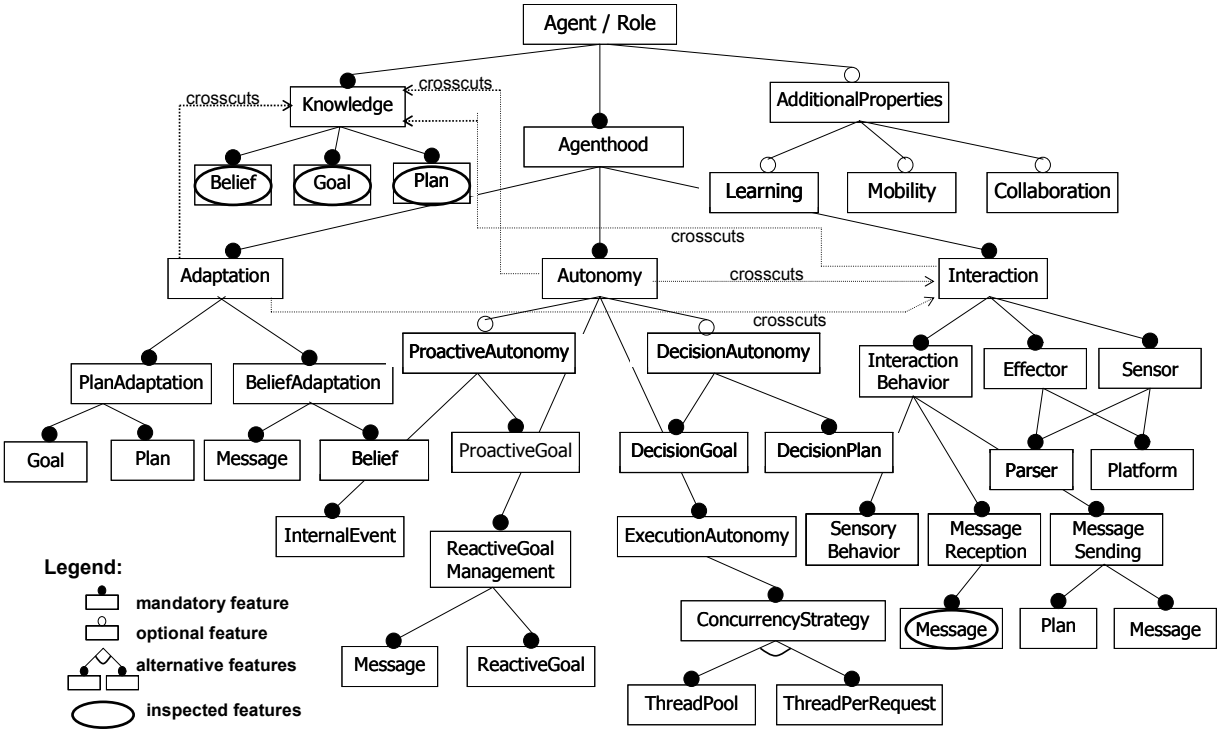


**Figure 2.** Partial Feature Model of a Agent / Role

kinds of components: (i) a central component that modularizes the non-crosscutting features associated with the agent knowledge; (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. Thus, there is a direct link between a crosscutting feature in domain analysis and an aspectual component in domain design.

A new notation has been used to graphically represent our AO agent architecture [8, 9]. 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 in 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*.

Following the specification of the AO agent architecture, we refined their components to specific object-oriented and aspect-oriented structures. We have used the aSide [10] modeling language to represent the detailed design of each component. The knowledge component was refined as a set of class hierarchies and compositions. The agent aspectual components were refined to sets of aspects and auxiliary classes. The refinement of each component and their relations are documented by means of a pattern language. The pattern language shows how the domain features of MASs can be mapped to specific design structures of classes and aspects that represent the components of the AO agent architecture. A complete description of this pattern language can be found in [9].

Figure 4 shows the refinement of the adaptation aspectual component in two aspects: (i) `Adaptation` – which defines the common part of the adaptation aspectual component; and (ii) `SpecificAdaptation` – which represents the variable part. The `Adaptation` aspect defines two crosscutting interfaces: (i) `IBeliefAdaptation` – responsible for interpreting received messages from the environment and manipulating its beliefs based on the messages contents; and (ii) `IPlanAdaptation` – which determines the plan the agent must execute whenever a new goal needs to be achieved. Figure 4 also illustrates that the `Adaptation` component crosscuts the `Interaction` component, by means of the interception of the `IMessageReception` interface. The `SpecificAdaptation` is a subaspect of the `Adaptation` aspect. It defines specific knowledge and behavior

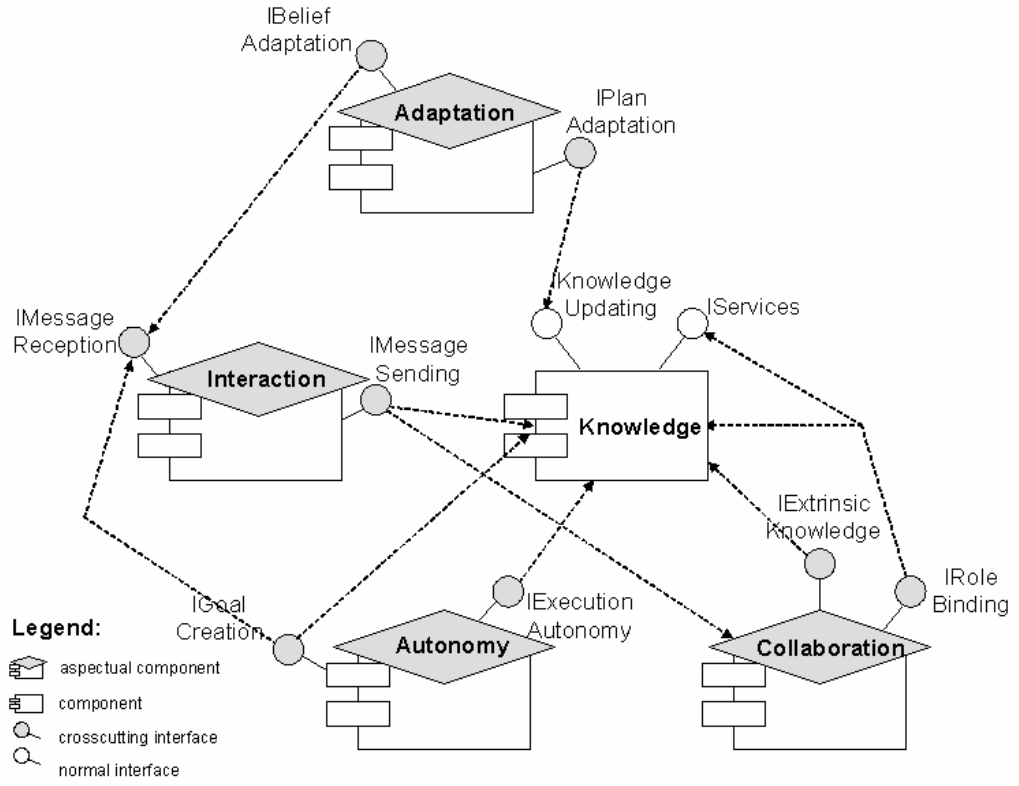


**Figure 3.** The Aspect-Oriented Agent Architecture

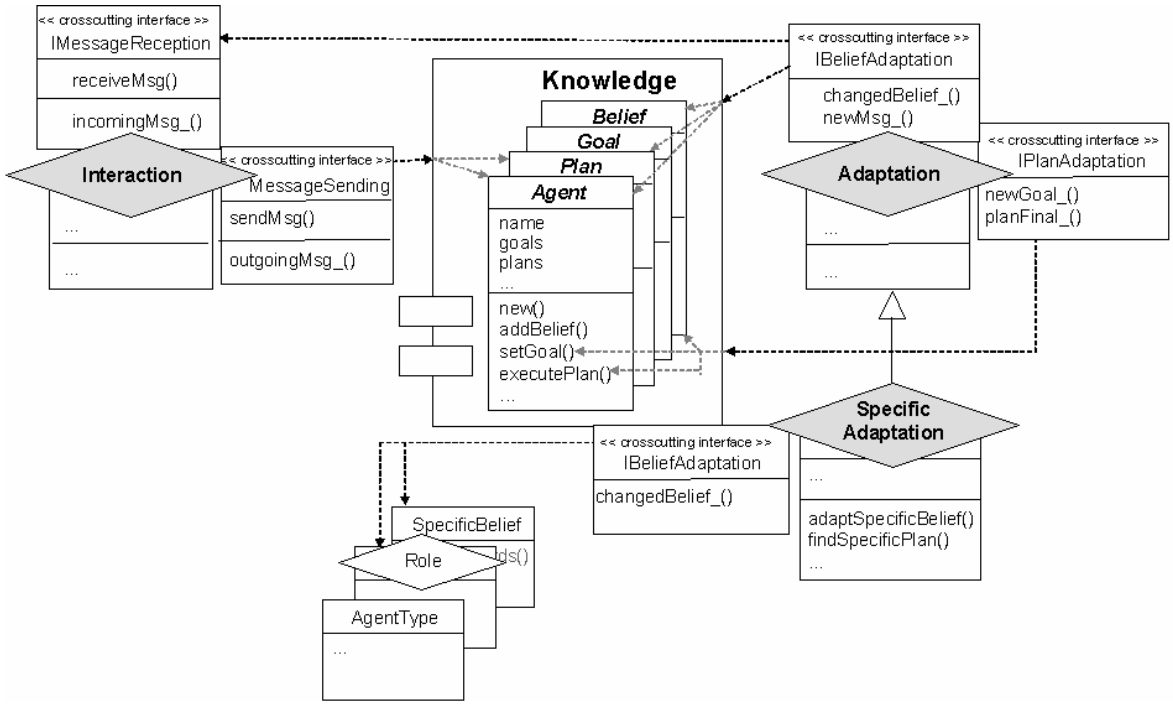


**Figure 4.** Detailed Design of the Adaptation Component

MAS. Thus, different `SpecificAdaptation` aspects can be specified.

### 3.3. Domain Implementation

In domain implementation, we used different technologies to accomplish the elements of our generative approach.

A configuration domain-specific language (DSL), called Agent-DSL, was specified based on the feature models defined in domain analysis. A configuration DSL allows to specify a concrete instance of a concept [1]. It can be directly derived from feature models. In this way, Agent-DSL allows modeling each of the agent crosscutting and non-crosscutting features captured by the feature models, such as, knowledge, interaction, adaptation, autonomy and collaboration. XML-Schema was the technology used to specify the Agent-DSL.

The aspect-oriented agent architecture was implemented by using Java and AspectJ [11] programming languages. The basis of the architecture implementation is an AO framework that contains hot-spots as classes and aspects. In the implementation of our framework it was used some of the AspectJ idioms presented in [12] (such as, *Abstract Pointcut*, *Chained Advice*, *Pointcut Method*, *Template Advice*). We found that they represent basic and recurrent constructions used to define AO frameworks in AspectJ.

The code generator of the generative approach was implemented as an Eclipse plug-in [13]. This generator maps abstractions in the Agent-DSL to components and aspects of the agent architecture. The AO framework is used as the basis of the agent architecture. The main task of the generator is to instantiate the 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.

### 3.4. Lessons Learned

Based on the experience of development of an AO generative approach, we have identified some important requirements and techniques that are useful and relevant during the integration of GP and AOSD technologies. Below, we synthesize these lessons learned.

***Modeling of crosscutting features***. The modeling of crosscutting concerns in early design phases has been recognized recently as an important topic of research in AOSD [16, 17]. The extension of feature models to represent crosscutting features, presented in this paper (section 3.1), helps to become explicit the existence of crosscutting concerns in system families during domain analysis. We are developing new case studies in different domains to understand better how to model crosscutting features and their interactions.

***Specification and modeling of aspect-oriented architectures.*** A fundamental activity when constructing program families (and product lines) is the modeling of a software architecture that addresses common and variable features. During the development of our generative approach, we have used two complementary notations that allows to model and represent aspect-oriented architectures (section 3.2). The design of AO architectures encompasses aspectual components to address crosscutting features encountered during domain analysis. Normal components model the non-crosscutting features. In the refinement of AO architectures: (i) each normal component is mapped to a set of class hierarchies and compositions; and (ii) each aspectual component is mapped to a set of aspects and auxiliary classes. An important research topic that we are studying is the definition of principles and patterns to model AO architectures so that it is possible to address different crosscutting features.

***Specification of the configuration knowledge***. The configuration knowledge in a generative domain model defines how specific combinations of features in the problem space can be mapped to specific combinations of components in the solution space. The configuration knowledge was specified in our work by defining a pattern language [19]. Each one of the components of the AO architecture was specified as a design pattern of the pattern language. The description of each pattern emphasizes: (i) a specific problem in the context of agent architectures; and (ii) a flexible design structure to address that problem. The pattern languages shows in a stepwise fashion how to generate and instantiate the aspect-oriented agent framework.

***Construction of domain-specific languages***. During the implementation of our generative approach, we have defined a unique configuration domain-specific language to express non-crosscutting and crosscutting features of software agents (section 3.3). Configuration DSLs are recognized as a suitable alternative to define generative approaches that require to instantiate object-oriented frameworks. In our case, it was also suitable to represent the crosscutting features encountered in a definition of a software agent. An interesting work is the investigation of the combined use of a configuration DSL to express only the non-crosscutting features and aspectual DSLs to express each one of the crosscutting features existent in a domain.

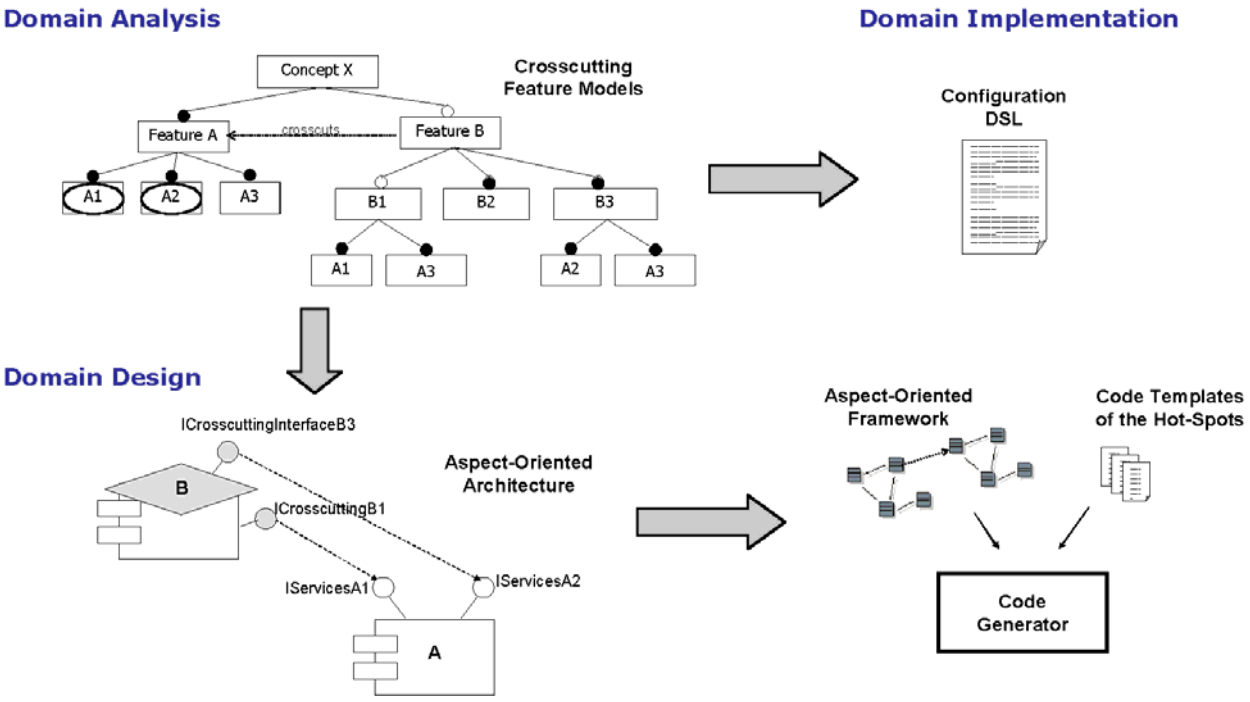


**Figure 5.** Artifacts Produced by the Development Activities of AO Generative Approaches

***Implementation of aspect-oriented frameworks***. Object-oriented frameworks are a common and useful technology to implement architectures for system families. They address the implementation of common (frozen-spots) and variable (hot-spots) features of system families. The use of the aspect abstraction in the definition of frameworks enables us to define common and variable behaviors of crosscutting features. In this work, an AO framework that models a set of crosscutting features encountered in agent architectures has been developed. We believe that AO frameworks are a fundamental technology to be used during the implementation of AO generative approaches.

## 4. Towards a Method to Develop Aspect-Oriented Generative Approaches

In this section, we outline a set of activities for the development of aspect-oriented generative approaches. We have adapted the method presented by Czarnecki and Eisenecker [1] based on our experience. These activities represent a significant step in the definition of a method to develop AO generative approaches. Table 1 presents the organization of these activities. Section 3 described the instantiation of these activities in the context of the multi-agent systems generative approach. We are currently refining each of these activities during development of a new case study. This case study comprises the domain of web-based information systems. Figure 5 illustrates the organization of the activities by presenting the different artifacts produced.

**(I) Domain Analysis**

(i) Definition of the domain.

(i) Identification and modeling of common and variable features of the domain concepts.

(iii) Identification and modeling of crosscuting features of the domain

**(II) Domain Design**

(i) Specification of the generic AO architecture.

(ii) Detailing of each normal and aspectual components of the AO architecture.

(iii) Identification and specification of DSLs

(iv) Specification of the configuration knowledge

**(III) Domain Implementation**

(i) Implementation of the DSLs.

(ii) Implementation of the AO architecture and components.

(iii) Implementation of the code generator.

**Table 1.** Development Activities of Aspect-Oriented Generative Approaches

## 5. Related Work

Some recent reports explored the integration of GP and AOSD. However, these reports have not covered or described in sufficient detail all the typical phases encountered in the development of a generative approach.

Pinto et al [5] have proposed DAOP-ADL, an architecture description language used to describe software architectures. This language supports the concepts of components and aspects. DAOP-ADL is interpreted by DAOP (Dynamic Aspect-Oriented Platform), a specific component and aspect-based middleware platform [27]. DAOP platform uses the information presented in the ADL to compose dynamically components and aspects of a application. DAOP-ADL enables the specification of AO architectures independent of programming languages and component platforms. The notation (section 3.2) proposed in this paper can also be formalized as an ADL.

Shonle et al [4] have developed XAspects, an extensible system that allows to define aspectual domain-specific languages. An aspectual DSL allows to express specific crosscutting concerns into separate constructs [1]. Examples of aspectual DSLs presented by them are [4]: (i) a coordination language used to specify thread coordination concerns; and (ii) a traversal language used to express collaborative behavior between classes. XAspects also supports the generation of Java and AspectJ code derived from one or more aspectual DSLs. In our work we have studied the domain of software agents to define a unique DSL that express non-crosscutting and crosscutting agent features. An interesting work related to the research of these authors is to investigate the use of XAspects to define one or more aspectual DSLs to express the agent properties in our generative approach.

Colyer et al [14] have presented some principles that guide the definition of flexible and configurable AO systems. They show the proper use of AOSD technologies to follow these principles. They also illustrate how the application of the principles can help in the configuration of different features in a product line. The work presented by these authors is an important step in the definition of guidelines for constructing program families in order to maximize configurability in AO architectures. This kind of principle is very useful to guide the specification of AO architectures that can be configured with different feature combinations in a generative approach.

Griss [15] presents some benefits of integrated use of aspect-oriented implementation technologies and feature engineering during development of product-lines. He also outlines a systematic approach for the combination of these two technologies. However, this work did not present a concrete case study which validates the suggested activities and techniques to develop product-lines.

## 6. Conclusions and Future Work

This paper reported our experience in the definition of an AO generative approach. We have organized the development of the generative approach using typical activities 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 feature model was extended to support the representation of crosscutting features. Also, a new notation was proposed to support the representation of AO architectures. Aspectual components have been used to model crosscutting features from the architectural point of view.

The definition of AO generative approaches brings important benefits to the development of software families. Generative programming allows: (i) to evolve the problem and solution spaces independently; and (ii) to define clearly the mapping between high-level features and implementation components. The integrated use of generative programming and AO techniques brought additional benefits, such as: (i) clear separation of non-crosscutting 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.

This work represents a significant step in the definition of a method to develop AO generative approaches. We are currently developing a new aspect-oriented generative approach for a different domain. The main purposes of this new case study are: (i) to refine the notations presented in Section 3; (ii) to detail the activities presented in Section 4; and (iii) to define a set of principles and guidelines to specify artifacts in a generative domain model considering the use of AOSD technologies.

**Acknowledgements**. This work has been partially supported by CNPq under grant No. 140252/2003-7 for Uirá, grants No. 141457/2000-7 and No. 381724/2004-2 for Alessandro, and by FAPERJ under grant No. E-26/150.699/2002 for Alessandro. The authors are also supported by the PRONEX Project under grant 7697102900, and by ESSMA under grant

552068/2002-0 and by the art. 1 of Decree number 3.800, of 04.20.2001. This research has also been partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## References

[1] K. Czarnecki, 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] P. Tarr, et al. "N Degrees of Separation: Multi-Dimensional Separation of Concerns". Proceedings of the 21st International Conference on Software Engineering, May 1999.

[4] M. Shonle, et al. "XAspects: An Extensible System for Domain Specific Aspect Languages". OOPSLA 2003 3D Track.

[5] M. Pinto, et al. "DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development", Proceedings of the GPCE´2003, Erfurt, Germany, September 2003.

[6] J. Gray, et al. "An Approach for Supporting Aspect-Oriented Domain Modeling", Proceedings of the GPCE´2003, Erfurt, Germany, September 2003.

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

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

[9] A. Garcia. "From Objects to Agents: An Aspect-Oriented Approach". PhD Thesis, PUC-Rio, April 2004.

[10] C. Chavez. "A Model-Driven Approach to Aspect-Oriented Design". PhD Thesis, PUC-Rio, April 2004.

[11] G. Kiczales, et al. "Getting Started with AspectJ". Communication of the ACM. October 2001.

[12] S. Hanenberg, R. Unland, A. Schmidmeier. "AspectJ Idioms for Aspect-Oriented Software Construction". Proceedings of the EuroPlop'03, Irsee, Germany, June 2003.

[13] S. Shavor, J. D'Anjou, S. Fairbrother, et all. The Java Developer's Guide to Eclipse. Addison-Wesley, 2003.

[14] A. Colyer, A. Rashid, G. Blair. "On the Separation of Concerns in Program Families". Technical Report, Computing Department, Lancaster University, January 2004.

[15] M. Griss. "Implementing Product-Line Features With Component Reuse". Proceedings of the International Conference on Software Reuse, Vienna, Austria, June 2000.

[16] Early Aspect Home-Page, Available at URL http://www.early-aspects.net/

[17] A. Rashid, et al. "Modularization and Composition of Aspectual Requirements". Proceedings of the AOSD´2003. 2nd International Conference on Aspect-Oriented Software Development. ACM. Pages 11-20.