Towards Aspect-oriented Model-driven Code Generation in the Formal Design Analysis Framework

Kendra Cooper
The Univ. of Texas at Dallas
kcooper@utdallas.edu

Lirong Dai
Seattle University
daia@seattleu.edu

Sergiu Dascalu
Univ. of Nevada
dascalus@cse.unr.edu

Nehal Mehta
The Univ. of Texas at Dallas
nehal.mehta@gmail.com

Sujala Velagapudi
The Univ. of Texas at Dallas
k_sujala@yahoo.com

Abstract

Model driven code generation has been investigated in traditional and object-oriented design paradigms; significant progress has been made. It offers many advantages including the rapid development of high quality code. Errors are reduced and the consistency between the design and the code is retained, in comparison with a purely manual approach. Here, we propose a model driven code generation approach for aspect-oriented development. The approach has three main steps. An aspect-oriented design is defined first. Previous work in the formal design analysis framework (FDAF) is used; the model is captured using FDAF's extended UML notation. Second, the aspect-oriented visual design model is translated into the formal text based notation XML. Third, the XML specification is translated into aspect-oriented code stubs, using the AspectJ programming language. FDAF has been extended in this work to support aspect-oriented model driven code generation. The approach is illustrated using an on-line banking system.

1. Introduction

The importance of modeling a high quality software architecture is well recognized in the software engineering community. The resulting quality of the implemented system, however, also relies on the correct and consistent refinement of the software architecture into code. Model driven code generation has been proposed as one technique to help accomplish this. The advantages of model driven code development include rapid code generation, reductions in errors, and consistency between the design and the code. Model driven code generation is within the scope of the Model Driven Architecture guide [10]. Starting from a *platform independent model* (*PIM*), transformations lead to *platform specific models* (*PSMs*), which include transformations into code.

Model driven code generation has been investigated in traditional and object-oriented development. In traditional development, models have been represented using formal notations including Petri Nets [10], Software Cost Reduction (SCR) [12], and SDL [12]. Full code generation has been accomplished; approaches for generating optimized code have also been proposed. In object-oriented development, models represented in UML have been used to generate fully executable code [6][8][9]. The generation of code "stubs" is an established feature of currently available commercial and open source object-oriented CASE tools, such as the IBM Rational Software Architect [6] and ArgoUML [1].

Significant progress has been made in automatic code generation for traditional and object-oriented development. However, to the best of our knowledge, model driven code generation in aspect-oriented development has not been considered. Aspect-oriented development is a relatively new paradigm, which provides a way to encapsulate capabilities that would tend to crosscut many elements in an object-oriented design, such as security [4]. At the code level, Aspect-oriented programming languages, such as AspectJ, provide linguistic mechanisms for separate expression of concerns (i.e., stakeholders' interests), along with implementation technologies for weaving these separate concerns into working systems. At the design level, a system's tangling concerns are encapsulated in a modeling element called an *aspect.* Subsequently, a

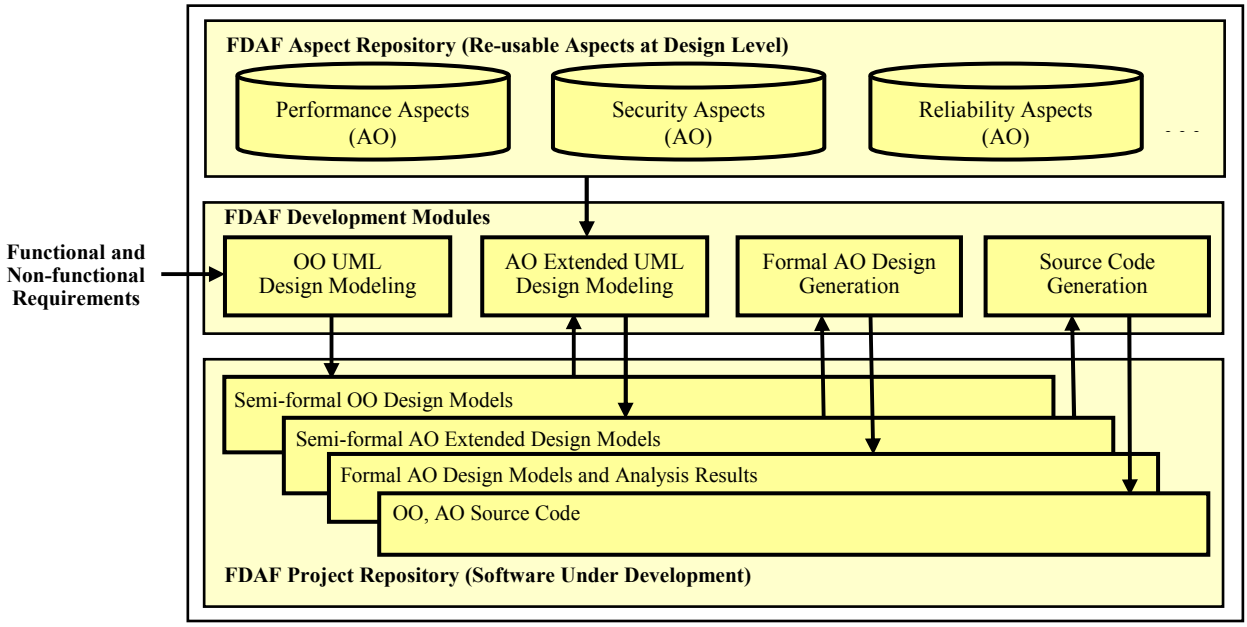

Figure 1. Overview of the Formal Design Analysis Framework

weaving process is employed to compose core functionality model elements (those realize the system's main functionalities) with these aspects, thereby generating a complete architecture design.

Here, we propose an aspect-oriented model driven code generation approach, which automatically generates AspectJ code stubs from an aspect-oriented design model. The approach is validated using an on-line banking system, in which an aspect is used to realize a security capability, Role-based Access Control (RBAC), in the design. Tool support has been developed to automatically generate the AspectJ code stubs from the input design. The long term goal of this research is to support full code generation in aspect-oriented development.

The remainder of this paper is organized as follows. Background on the Aspect-oriented design framework, FDAF, and the aspect-oriented programming language, AspectJ, are presented in Section 2. The code generation approach is presented in Section 3; it is illustrated using an on-line banking system. Conclusions and future work are in Section 4.

2. Background

An overview of the aspect-oriented design framework, FDAF, and the aspect-oriented programming language, AspectJ, used in this research are presented in this section. We recognize that alternative AOD and AOP techniques can also be used. FDAF has been selected because it provides a visual, UML based approach, tool support is freely available, which is based on the current version of the open source case tool ArgoUML, and has been validated on two substantial example systems. AspectJ has been selected because it has an established and growing user community; the tool support is well documented and freely available. The reader is referred to [2] for detailed presentations of AspectJ and [3][4] for FDAF.

FDAF is an aspect-oriented design and analysis framework. It supports the semi-formal visual modeling of an object-oriented (OO) base design and its subsequent extension with aspects to realize crosscutting (non-functional) capabilities such as security performance, etc. The aspect-oriented (AO) extended design can be used to a) automatically create formal designs, which can be rigorously analyzed, and b) automatically create code stubs, i.e., model driven code generation. The automatic code generation is the new extension presented in this work.

The meta-model for the UML extension is defined in [3]. The extension includes an AspectJClass, Pointcuts, Advice, etc. For example, the syntax and semantics for an element called AspectJClass is defined as part of the extension. This element has a name (Inherited from ModelElement in the UML core package), a list of Features (e.g., Attribute, Operation, Method), a list of Pointcuts, and a list of advice.

The FDAF approach includes an aspect repository, a set of modules that support development activities, and a repository to store project artifacts (refer to Figure 1).The aspect repository stores a collection of

predefined aspects. Architects can search the repository to select the appropriate aspect(s) according to the system's non-functional requirements and re-use them in their design.

The project repository stores the development artifacts, including semi-formal OO and AO design models, formal AO design models, and source code. The OO and AO semi-formal design models are manually created. Two development modules, OO UML Design Modeling and AO Extended UML Design Modeling, support these activities. The formal

```
aspect
:: 'ASPECT_NAME' '{' aspect_parts '}'

aspect_parts
:: aspect_part aspect_parts | ε

aspect_part
:: pointcut_def | advice_def

pointcut_def
:: 'POINTCUT_NAME' pointcut_body

pointcut_body
:: selection_expression

advice_def
:: 'POINTCUT_NAME': advice_body

advice_body
:: where_specifier '{'
code_in_programming_language '}'

where_specifier
:: 'BEFORE' | 'AROUND' | 'AFTER'

//CAPITALS - terminals,
lowercase_letters - nonterminals

Selection_expression is a primitive
pointcut chosen from the set of
predefined pointcuts of particular
aspect oriented (AO) programming
language, e.g., call(f), set(v),
meaning call to function f, or set a
value of v, respectively.
```

Figure 2. Simplified Syntax for an Aspect in EBNF [14]

AO design models (i.e., translations of the semi-formal AO design into formal notations) are automatically created. The formal notations include the architectural description languages Rapide, Armani, Æmilia, and Alloy in addition to Promela, the input language for the well-known model-checker SPIN. Source code stubs in AspectJ are automatically generated, using the extended AO design as input. This capability is new in FDAF.

AspectJ is one aspect-oriented programming language that is currently available. It is an aspect-oriented extension to the Java programming language. A simplified definition of the syntax definition for an aspect is presented using EBNF (extended Backus-Naur Form) in Figure 2 [14].

AspectJ extends Java with support for dynamic and static crosscutting implementations. The first makes it possible to define additional implementation to run at certain well defined points in the execution of the program. The second makes it possible to define new operations on existing types, i.e., it affects the static type signature of the program.

Dynamic crosscutting in AspectJ is based on a small but powerful set of constructs. Join points are well-defined points in the execution of the program; pointcuts are a means of referring to collections of join points and certain values at those join points; advice are method-like constructs used to define additional behavior at join points; and aspects are units of modular crosscutting implementation, composed of pointcuts, advice, and ordinary Java member declarations.

3. Code generation

The model-driven code generation approach has three main steps. The first step is to define an aspect-oriented, visual model of the design; this is the input for the code generation approach. The second step is to translate the design into a formal, text-based notation that can be readily processed. The third step is to transform the formal text based representation of the visual design model into the target programming language. Here, we use the FDAF, XML, and AspectJ notations in these three steps respectively; each step is described in more detail below. The approach is illustrated using a banking system example, in which a security feature Role-Based Access Control (RBAC) is needed. For illustration purposes, we follow the transformation of two elements. The aspect name is followed through the steps by highlighting it in yellow; the pointcut to the Controller class is highlighted in blue.

3.1. Define Aspect-oriented Design Model

An aspect-oriented visual design model is defined, which complies with the meta-model for the FDAF extension [3]. For this preliminary work, focused on generating code stubs, only an extended class diagram modeling the static view (i.e., the structure) of the design is needed. In the future, when full code

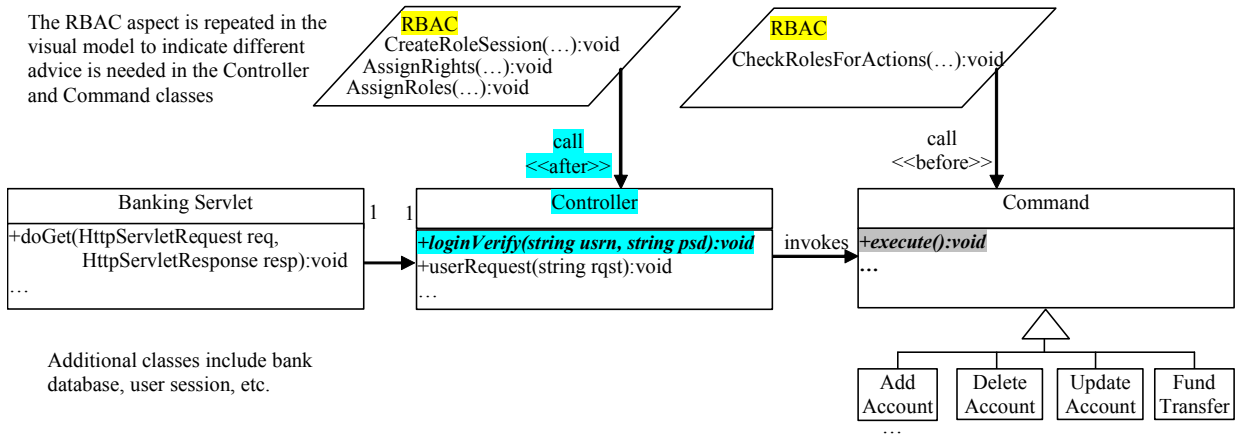

Figure 3 Banking System Example: FDAF Extended AO UML Design

generation is investigated, additional views and diagrams will be required, e.g., statechart, sequence, allocation, etc.).

Defining the design is an iterative process, which involves designing a base design and then extending the design with cross-cutting aspects. A partial model for the banking system is illustrated in Figure 3. The base model uses standard UML class diagram elements (classes, association, etc.). The aspect oriented extension uses a new graphic icon, the parallelogram, to represent the aspect. Here, the aspect is the RBAC (the name is highlighted in yellow in Figure 3). A new association is provided to visually represent the crosscuts from aspects to classes (i.e., pointcuts and joinpoints). In the example, classes in the base design including a Controller, Command, and Banking Servlet. The capabilities represented in the RBAC are needed (i.e., crosscut) in two places: it crosscuts the Controller and Command classes. The RBAC aspect is included in two places in the model to indicate that different advice is needed in the classes. A variety of pointcut types are currently supported including call, execute, this, and handler.

3.2. Transform Aspects in the Visual Model into XML

An XML schema for the aspect has been defined, which represents the privilege, name, modifier, and pointcuts. The tags and organization of the schema have been derived from the definition of AspectJ syntax. For example, the AspectJ syntax definition begins with the production rule for an Aspect; our corresponding XML schema begins with the tag <Aspect>, which is composed of tags for pointcuts. Tags have not been defined for advice, which are the bodies of the signatures in the aspect, as they are empty in the design phase.

The data used to populate the XML specification are extracted from the visual design model using syntactic analysis. For example, the name of the aspect in the visual design model is extracted and used to populate the <name> element in the XML specification (the string "Aspect" is appended to the name); the crosscuts in the visual design model are used to extract the pointcut definition and populate the pointcut specification, including the expression type of the pointcut (e.g., simple or a compound expression), the name of the pointcut (created using the class name and signature name as a base), the type of the pointcut (e.g., call, execute), etc. An example of a partial XML specification for the example banking system is presented in Figure 4. Here, the name of the aspect is highlighted in yellow: RBACAspect; a pointcut in blue. The sample represents the RBAC aspect crosscutting the Controller class in Figure 3.

3.3. Transform XML into Programming Languages

The capability to generate Java code stubs from a UML class diagram is already available in ArgoUML [1]. Here, we focus on the generation of AspectJ code stubs. Once the XML specification has been created, the generation of code stubs is straightforward. The elements of the specification are retrieved from the specification (e.g., get the name) and are printed out to a file, considering the concrete syntax requirements of AspectJ (e.g., the use of keywords, braces, parentheses, colons, etc.). An example of the code automatically generated is presented in Figure 5. Here again, the name of the AspectJ aspect is highlighted in yellow and elements of a pointcut in blue. A developer would then

```
<Aspect>
 <isPrivileged>false</isPrivileged>
 <name>RBACAspect</name>
 <packagename />
 <modifier>public</modifier>
  <PointCut>
    <PointCutExpr>
        <exprType>simple</exprType>
  <pCutName>ControllerLoginPointCut
  </pCutName>
          <pCutType>call</pCutType>
          <memberModifier />
    <memberClassTypePattern>Controller
    </memberClassTypePattern>
    <memberNamePattern>loginVerify
    </memberNamePattern>
    <methodReturnType>void
     </methodReturnType>
     <fieldTypePattern />
     <parameterList>
       <parameter>
          <parameterType>String</parameterType>
           <parameterId />
       </parameter>
       <parameter>
          <parameterType>String</parameterType>
           <parameterId />
       </parameter>
     </parameterList>
   </PointCutExpr>
  </PointCut>
   …
```

Figure 4 XML Representation of RBAC Aspect

complete the implementation of the Aspects by developing the advice code.

4. Conclusions and Future work

Model driven code generation has a number of benefits, including the rapid development of source code and the potential to reduce errors. It has received a great deal of attention in traditional and object-oriented development communities. In this work, we present our approach to model driven code generation in aspect-oriented software development. To the best of our knowledge, this has not been presented in the literature. The approach uses a visual, design model (UML based extended with aspect-oriented elements) as input and automatically generates AspectJ code stubs. The design is translated into an XML specification using syntactic analysis. The XML specification is then translated into code stubs.

```
public aspect RBACAspect{
pointcut ControllerLoginPointCut():
call(void
Controller.loginVerify(String,String));

pointcut CommandExecutionPointCut():
call(void Command.execute());

after() : ControllerLoginPointCut()
    {
    }

before() : CommandExecutionPointCut()
   {
   }
}
```

Figure 5 Example of Automatically Generated AspectJ Code Stub for the RBAC Aspect

The initial results have been very promising. We plan to validate the approach using additional, publicly available AspectJ applications. The applications will be reverse engineered into design models (providing inputs to test the approach); the complete source code will be simplified down to code stubs. In combination, the reverse engineered design and simplified code stubs will form test cases. In addition, the rigorous definition of the transformation algorithm and the formalization of the traceability relationships among the models are under development.

We have observed that the automatic code generation of code stubs is helpful, but a substantial amount of manual development is left to the programmers. Our long term research goal is to extend the approach to support full code generation. This will require the use of syntactic and semantic analysis techniques, which have been successfully applied in traditional and object oriented code generation research. As shown in the literature, full code generation techniques based on UML require the use of additional diagrams such as statechart, activity, sequence and, for distributed systems, allocation diagrams [6][8][9]. We propose the use of the extended activity diagram from FDAF to support the full code generation for non-distributed systems. Distributed systems will require the extension of the FDAF to include allocation diagrams.

The formal verification of the generated AspectJ code (stubs or full code) with respect to the UML model is an important issue. This requires a formal semantic definition of the AspectJ language, which is currently under investigation in the AspectJ community. When this definition is available, the formal verification of the code generated by the approach can be done.

References

[1] ArgoUML homepage, available at: http://argouml.tigris.org/

[2] AspectJ homepage, available at: http://www.eclipse.org/aspectj/downloads.php

[3] Dai, L., "Formal design analysis framework: an aspect-oriented architectural framework", The University of Texas at Dallas, Ph.D. Dissertation, 2005.

[4] Dai, L. and Cooper K., "Modeling and Performance Analysis for Security Aspects", Journal of Science of Computer Programming, Volume 61, 2006, pp. 58 – 71.

[5] Filman R., Elrad T., Clarke S., and Aksit M., Aspect-Oriented Software Development. Addison Wesley Professional, 2005.

[6] Ho, W., Jézéquel, J., Le Guennec, A. and Pennaneac'h. F., "UMLAUT: An Extendible UML Transformation Framework," in Proceedings of the 14th IEEE Int'l Conf. Automated Software Eng. (ASE '99), Oct. 1999.

[7] IBM Rational Software Architect, available at: www-128.ibm.com/developerworks/rational/ products/rsa/

[8] Long, Q., Liu, Z., Li, X. and Jifeng, H., "Consistent code generation from UML models" in Proceedings of the Australian Software Engineering Conference, Mar. 29-April 1, 2005., pp. 23-30.

[9] Mellor, S. and Balcer, M., *Executable UML: A Foundation for Model Driven Architecture.* Addison-Wesley, 2002.

[10] OMG Model Driven Architecture Guide V1.0.1, June 12, 2003, available at www.omg.org

[11] Philippi, S., "Automatic code generation from high level Petri Nets for model driven systems engineering", Journal of Systems and Software, Oct. 2006, vol. 79, no. 10, pp. 1444-1455.

[12] Rauchwerger, Y., Kristoffersen, F., Lahav, Y., Cinderella SLIPPER: An SDL to C-Code Generator, Lecture Notes in Computer Science, Volume 3530, 2005, pp. 210-223.

[13] Rothamel, T., Liu, Y.A., Heitmeyer, C.L., Leonard, E.I. "Generating optimized code from SCR specifications", Proceedings of the ACM/SIGPLAN/SIGBED Conference for Languages, Compilers, and Tools for Embedded Systems, 2006, pp. 135-144.

[14] Tóth, M. and Beličák, M., "Dynamic Restructuralization of Software Systems using Aspect-oreinted Programming", Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics January 25-26, 2007, pp. 457-468.