An Aspect-Oriented Approach to Developing Middleware-based Applications

Sudipto Ghosh, Brahmila Kamalakar
Computer Science Department
Colorado State University
Fort Collins, CO 80523, USA
{ghosh, brahmila}@cs.colostate.edu

ABSTRACT

Middleware technologies provide features such as connectivity and dependability (e.g., security and fault tolerance services). A major challenge to software development organizations is the complexity of creating and evolving distributed systems resulting from the tangling of middleware-specific functionality with core business functionality in system designs. We present an MDA-compliant middleware transparent software development approach in which application designs are developed independently of the middleware platform. Middleware features are encapsulated as aspects and woven with artifacts that realize core functionality. Our approach enables easy replacement of one dependability mechanism by another, and easy migration from one middleware platform to another. The approach also promotes reuse of aspects in multiple applications.

Keywords

MDA, OMG, UML, aspect-oriented software development, distributed computing, modeling and meta-modeling, middleware technologies.

1. INTRODUCTION

The rapid growth of the Internet has resulted in widespread use of distributed applications that communicate with the help of middleware. Middleware platforms provide features and services that facilitate development of distributed applications. However, such features are often scattered across and tangled with modules providing core functionality. Using current software development techniques, application design and implementation becomes tightly coupled with the specific middleware technology that is incorporated into the application. Even though certain middleware services may be provided as components (e.g., connectivity) these components may be crosscut by other middleware features (e.g., security, events and transactions). The crosscutting nature of middleware makes understanding, analyzing and changing middleware features difficult. Since businesses need to keep up with advances in middleware technology, entire applications need to be redesigned and reimplemented to migrate from one middleware technology to another.

We propose a middleware transparent software development (MTSD) approach that decouples the design of middleware specific features from the design of core business functionality. The approach uses aspect-oriented modeling and programming techniques because they provide the necessary constructs for encapsulating crosscutting design and code elements. Software developers design primary models of the core application functionality. Other features of the application that will be realized as middleware services are modeled separately as aspects and seamlessly woven into the application later in the development process. MTSD supports incorporation of new dependability mechanisms and middleware features. It enables reuse of high-level application design and architectures that are independent of the middleware. MTSD supports the OMG's MDA initiative. In this paper, we present an overview of the MTSD approach.

2. RELATED WORK

Aspect-oriented software development [9] has introduced aspect languages like AspectJ and Aspect C# which help in abstracting and encapsulating crosscutting concerns at the programming level. Simmonds et al. [12] captured Jini middleware details in the form of code aspects. Pichler et al. [10] demonstrated the use of aspects with the Enterprise Java Beans container architecture. Zhang and Jacobsen [13] analyzed the use of aspects in middleware architectures and quantified crosscutting concerns in the implementations of middleware applications.

Bussard [1] described the encapsulation of CORBA fea-

OOPSLA and GPCE Workshop on Model Driven Software Development
2004 Vancouver, Canada

tures as code aspects and proposed the creation of a library of aspects for different CORBA features to ease the development of CORBA applications. Hunleth [5] proposed the creation of an Aspect-IDL for CORBA to support several new types of AspectJ introductions: interface method and field, interface, super class, structure field, oneway specifier, and IDL typedefs and enumerations.

The MDA initiative [11] employs design level abstraction to describe software systems. In the MDA approach, the Platform Independent Model (PIM) captures the functionality and behavior of the application free from the middleware technology. Integration of middleware technology specific mappings with the PIMs yields the Platform Specific Models (PSM).

Clarke et al. [2] used the subject-oriented modeling approach to capture reusable patterns of cross-cutting behavior at the design level. Each requirement is treated as a separate design subject. Design subjects are composed to obtain the complete system design.

France et al. [4] propose an aspect-oriented modeling (AOM) approach in which software designers specify primary models (base functionality), aspect models (non-orthogonal crosscutting functionality for dependability) and composition directives to obtain the integrated design. Cross-cutting design concerns are captured in aspect models using the Role-Based Metamodeling Language (RBML) [3]. We adopt the AOM approach for the development of middleware-based applications.

3. MTSD APPROACH

The MTSD approach is illustrated in Figure 1. In this approach, the application developer models the application free from middleware concerns in a *Middleware Transparent Design* (MTD) model. The MTD model contains UML class diagrams and interaction diagrams. Other views (e.g., statecharts and activity diagrams) may also be used. In the MDA terminology, the MTD is the PIM.

Middleware specific features are localized in aspect models. Generic aspect models in the form of aspect libraries are provided by the middleware platform vendor. These models are described using the RBML [3] in terms of static and interaction pattern specifications. There is one aspect model for each feature (e.g., connectivity, directory service, security, and replication).

The application developer specifies the bindings from the generic aspect models to the application context and generates the context-specific aspect models. The generation can be automated in part; it still requires binding information as input from the application developer.

These models are implemented in an aspect language (currently in AspectJ). The developer transforms an aspect model to code aspects with the help of mappings that convert aspect model constructs to AspectJ constructs. This task may also require input from the application developer. For example, for the authorization aspect, we need to provide information in the form of application-specific role names and their access privileges.

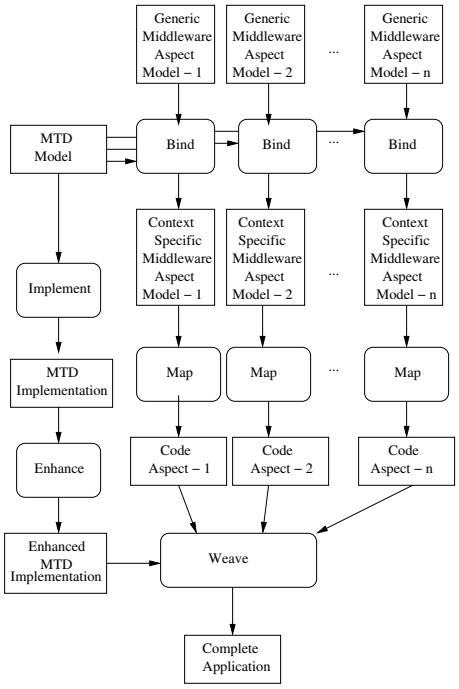

Figure 1: The MTSD Approach.

The application developer implements the MTD model. The MTD implementation then needs to be converted into an *Enhanced MTD implementation* to make it ready for aspect weaving for a specific middleware platform. Different middleware platforms need different application architectures. For example, Jini requires that a proxy object be implemented by the developer. The code for the proxy object is usually written as a Java inner class. Java RMI requires proxy stubs to be automatically generated from the server implementation. The client implementation in the MTD does not possess knowledge about the nature of proxies and assumes that the server object is local. Thus, the client (and the server in some cases) may need to be modified to adapt them to the middleware platform used in the application. The enhanced MTD implementation is woven with code aspects using the AspectJ compiler.

The generic aspect models can be reused to develop other applications. The mappings from the context-specific aspect models to code aspects in AspectJ are described in terms of generic aspect models and AspectJ constructs. Thus, these mappings can also be imple-

mented in a tool and reused. The mappings defined to enhance an MTD implementation and convert it into a form required for weaving are also reusable. The MTD models and implementations can be reused with aspects for other middleware technologies when migrating from one middleware technology to another.

4. CURRENT RESULTS

In our work we have used of the MTSD approach for incorporating remote connectivity, security, and transaction features into CORBA-based distributed applications [6, 7, 8]. The CORBA features were treated as reusable patterns for use with any application. The core designs for these features were specified using the RBML as generic aspect models independent of the application context. RBML static pattern specifications were used to specify the static structure of the feature and RBML interaction pattern specifications were used to specify the behavior. The generic aspect models were mapped to the application context and transformed to AspectJ code using the transformations developed as a part of this research. The code aspects were woven with the MTD implementation after making the required enhancements to ensure CORBA compliance.

The bindings for obtaining the context-specific diagram elements such as classes, interfaces, operations, message parameters, return types and attributes from the corresponding roles in the generic aspect model are specified by the application developer. In some cases, the application developer needs to specify the actual values corresponding to certain message parameters in the generic aspect models. For example, in authorization, the application developer needs to specify the value of `SERVER_QOP_CONFIG_TYPE` that is needed to create a new policy to override default policies.

The types of bindings required for CORBA are, *stamp out*, *bind*, *generate(stamp out)* and *generate(bind)*. *Generate(bind)* and *generate(stamp out)* are the bindings that are applied to obtain the context specific diagram elements corresponding to the elements that are generated on IDL compilation. These two types of bindings will not be applicable to other middleware technologies because IDL compilation is specific to CORBA. Hence, the types of bindings may differ with the type of middleware used.

The approach allows multiple features to be incorporated into the application without having to deal with one complex design model that realizes all the features. This is because each context-specific aspect model is separately transformed to code and then woven with the MTD implementation.

The context-specific aspect model for a particular feature may constitute context-specific aspect models that conform to more than one pattern. For example, the context-specific class diagram for connectivity in a Bank application [6] used in the transaction service illustration was obtained by composing the context-specific class diagrams corresponding to the *Naming Service with Inheritance* and the *ServiceCreator-InType* patterns. The context-specific interaction diagrams corresponding to both the patterns were used to describe the context-specific interaction diagrams of the connectivity feature. Thus, there is one context-specific aspect model per middleware feature.

AspectJ transformations to map the context-specific aspect models to AspectJ code were described in terms of AspectJ language constructs and the generic aspect models pertaining to the middleware feature. AspectJ offers the constructs necessary for transforming the CORBA aspect models to code. The code aspects were created using the AspectJ transformations and context-specific aspect models. Aspect ordering had to be ensured when multiple CORBA features were incorporated into the application. For example, authentication is performed before remote connectivity and authorization. The aspect ordering was ensured using aspect *precedence*, an aspectJ construct, in the aspect code. The precedence was specified in cases where multiple advices were defined on the same pointcut. Different aspect precedences are required for different combinations of middleware aspects (e.g., authentication and remote connectivity, or authorization and remote connectivity) that are woven with the application. The correct aspect precedence has been worked out for the combination of the features used in our work.

User-defined exceptions that need to be raised and handled by a service method in an aspect are obtained from the IDL specification of the method. Specific CORBA exceptions that have to be handled in the aspect are specified by the application developer. The application developer defines the exception handling behavior.

Before the code aspects can be woven with the MTD implementation, certain middleware feature-specific enhancements are necessary to be made to the MTD implementation. The enhancements are needed for ensuring the presence of the required pointcuts in the MTD implementation for advices to take effect and for compliance with the middleware technology. In some cases, the MTD implementation enhancements are reusable with different middleware feature mechanisms. For example, connectivity using the IOR mechanism and the naming service with inheritance and tie require the same MTD implementation enhancements. By studying the enhancements required for various middleware features and middleware technologies, the enhancements reusable with different features and different middleware technologies can be determined. Construction of the server and client objects has been used as the pointcut for many of the advices. In the connectivity aspect, when the ORB is initialized, the command-line arguments have to be sent as parameter to the constructor. For authentication, there is no need to send the command-line arguments when the constructor is called. Hence,

the execution of the single argument constructor with command-line arguments is used as the pointcut for both the advices. By detailed observation, such multiple use pointcuts can be identified. The pointcuts defined independent of the application architecture make them reusable across applications.

Varying degree of dependence of the aspect code on the application context was observed with different CORBA features. For example, the naming service and the IOR mechanism aspects are generic in nature. The naming service aspects depend on the application context for the service name and service object. The IOR mechanism aspects depend on the application context for the location of the IOR file and the service object. However, the transaction service code aspects depend on the application context to a higher degree. For example, the implementation of the *Resource* interface is dependent on the application business functionality. Suitable MTD implementation enhancements and input from the application developer are needed to develop such aspects. Similarly, input from the application developer is needed to implement operations such as the `domain`, `getRunAsRole`, and `getRequiredRoles` in the authorization aspect.

MTSD allows the reuse of the base application design and implementation for introducing different middleware features. For example, the Bank application that was used to illustrate CORBA remote connectivity using OSAgent was reused in the CORBA authentication and CORBA authorization illustrations. MTSD also allows replacement of one middleware feature mechanism with another, thereby making the middleware feature mechanism reusable. For example, the IOR connectivity feature mechanism for remote connectivity was replaced by naming service with inheritance in the HelloWorld example. With MTSD, more than one middleware feature may be incorporated into the same application. This was demonstrated by incorporating naming service with inheritance and transactions in the CORBA transaction service illustration. The middleware feature aspects also can be reused in multiple applications. This was demonstrated by reusing naming service with inheritance in the Bank example for CORBA transactions. The transformations from context-specific aspect models to AspectJ code are described in terms of generic aspect models and AspectJ constructs, thus making them reusable.

5. FUTURE WORK

There are several areas of research that can be investigated in the MTSD context.

5.1 Applying MTSD to CORBA Features

In our work, secure CORBA applications and CORBA transaction-based applications were developed using the MTSD approach. The next step is to apply the approach to other CORBA features such as trader service, interface repository, dynamic invocation interface, dynamic skeleton interface, event service, and fault tolerance. Aspect models may be developed for fault tolerance mechanisms using object replication. The fault-tolerance aspects are responsible for maintaining replica consistency, replication transparency and failure transparency. There are two types of fault-tolerance mechanisms: those that are provided by the middleware infrastructure, and those that are provided by the application. In both cases, the MTSD approach can provide the advantages resulting from the decoupled design of business functionality and fault tolerance concerns. In infrastructure-controlled fault tolerance, the aspect implementations will utilize the API and services provided by the middleware platform vendor. In application-controlled fault tolerance, the aspect implementations will be developed by the application developer.

The MTSD approach has to be applied to different vendor implementations of CORBA (e.g., IONA and BEA). The MTSD approach needs to be applied to C++ implementations of CORBA using AspectC++.

The application architecture in which one CORBA service is created and activated by another CORBA service, which in turn is activated by the *Server*, conforms to the *ServiceCreator-InType* pattern. An instance of this pattern was found in the Bank application, where the *Bank* service created and activated accounts. The *Bank* service was created and activated by the *Server*. There may be other patterns for other application architectures present in CORBA applications. These patterns need to be identified.

CORBA requires the declaration of the service interfaces in an IDL format. Hence, the Java interface written by the developer needs to be converted to the CORBA IDL format. An IDL file corresponding to the Java interface may be generated automatically with the help of a tool. Such a tool needs to be developed.

5.2 Applying MTSD to Other Middleware

The MTSD approach needs to be applied to other middleware technologies such as Java RMI, Jini, SOAP-RPC, and EJB. This will help in characterizing properties of middleware features that make them isolatable as aspects. This will also help in evaluating the MTSD approach in terms of the ease of migrating applications from one middleware technology to another.

The aspect models developed in our work are CORBA-specific. Another step may need to be introduced in the MTSD process, which specifies the middleware feature (e.g., security and transactions) as aspect models independent of the middleware technology (e.g., CORBA and Jini). These aspect models may then be refined to aspect models specific to the required middleware technology and implementation (e.g., JacORB and Visibroker for CORBA).

5.3 A Variant of the MTSD Approach

In our work, the generic aspect model for a particular middleware feature is mapped to the application context using the bindings specified by the application developer. In the next step, the context specific aspect models are mapped to code aspects in AspectJ, using the transformations identified in this research. Alternatively, the generic aspect models may be mapped to code aspects in AspectJ and then later mapped to the application context. Abstract aspects and abstract pointcuts may be used to map the generic models to code. The MTSD approach may be tried out for this method of transforming the aspect models to code.

5.4 Formalization and Tool Support

Currently, the transformations from the aspect models to code are not formally defined. A specification language with formal semantics needs to be developed for describing the transformations. The transformations specified in a formal language will be useful for testing the conformance of the aspect code that is generated using the transformations.

The approach may be implemented in a toolset. A tool needs to be developed for applying the context-specific bindings specified as input from the application developer to obtain context-specific aspect models from the generic aspect models in an automated manner. Another tool can automate the development of code aspects from the context-specific aspect models using the code transformations developed in this research.

5.5 Evaluation of the MTSD Approach

The MTSD approach needs to be applied to various application architectures and the effect of using multiple potentially conflicting aspects with one primary model needs to be investigated. Resolving such conflicts by preserving the properties of the aspects in the woven code has to be studied.

An evaluation of the MTSD approach in terms of a cost-benefit analysis can be conducted by developing distributed applications with and without using the MTSD approach. The approach should be evaluated with the model bindings and code transformations being applied automatically with the help of a tool. The time and effort required for applying the MTD implementation enhancements for different middleware features have to be measured.

Verification techniques to ensure that the woven code actually preserves the specified properties of the aspects have to be investigated. Validation techniques to ensure the correctness of the generic aspect models, and the context-specific aspect models obtained from the generic models have to be developed.

6. REFERENCES

[1] L. Bussard. Towards a Pragmatic Composition Model of CORBA Services Based on AspectJ. In *Proceedings of ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, Sophia Antipolis and Cannes, France, June 2000.

[2] S. Clarke. Extending Standard UML with Model Composition Semantics. *Science of Computer Programming*, 44(1):71–100, July 2002.

[3] R. France, D.-K. Kim, S. Ghosh, and E. Song. A UML-based pattern specification technique. *IEEE Transactions on Software Engineering*, 30(3), March 2004.

[4] R. B. France, I. Ray, G. Georg, and S. Ghosh. An aspect-oriented approach to design modeling. *To be published in IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, to appear*, 2004.

[5] F. Hunleth, R. Cytron, and C. Gill. Building Customizable Middleware Using Aspect Oriented Programming. In *OOPSLA Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa, Florida, USA, October 2001.

[6] B. Kamalakar. Aspect-Oriented Development of CORBA-based Applications. *Master's Thesis (in preparation), Department of Computer Science, Colorado State University, Fort Collins, Colorado, USA*, 2004.

[7] B. Kamalakar and S. Ghosh. An Aspect-Oriented Approach for Developing CORBA Applications. *Submitted to Information and Software Technology. Available at* `http://www.cs.colostate.edu/homes/ghosh/papers/ist.pdf`, 2004.

[8] B. Kamalakar, S. Ghosh, and P. Vile. Middleware transparent development of dependable corba applications. In *15th IEEE International Symposium on Software Reliability Engineering*, page to appear, St. Malo, France, November 2004.

[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '01)*, pages 327–353, Budapest, Hungary, June 2001.

[10] R. Pichler, K. Ostermann, and M. Mezini. On Aspectualizing Component Models. *Software Practice and Experience*, 33(10):957–974, August 2003.

[11] Richard Soley. MDA, An Introduction. *URL* `http://omg.org/mda/presentations.htm/`, 2002.

[12] D. Simmonds, S. Ghosh, and R. B. France. Middleware Transparent Software Development and the MDA. In *UML 2003 Workshop on*

SIVOES-MDA, to appear in Proceedings SIVOES 2003, Electronic Notes in Theoretical Computer Science, Elsevier, San Francisco, CA, October 2003.

[13] C. Zhang and H.-A. Jacobsen. Refactoring Middleware with Aspects. *IEEE Transactions on Parallel and Distributed Systems*, 14(11):1058–1073, November 2003.