An Aspect Oriented Model Driven Architectural Framework for Middleware Transparency

Devon Simmonds, Sudipto Ghosh, Robert France
Computer Science Department
Colorado State University
Fort Collins, CO 80523
Phone: (970) 491-4608
FAX: (970) 491-2466
{*simmonds,ghosh,france*}*@cs.colostate.edu*

February 24, 2003

Abstract

While complex distributed applications are becoming the norm for modern businesses, the underlying technologies are also changing constantly. Organizations are faced with a number of options for selecting middleware technologies that provide consensus on interfaces and interoperability on top of heterogeneous platforms, operating systems, network protocols and programming languages. Organizations seek to minimize the overall cost of middleware technology integration and evolution through the reuse of software artifacts. The Model Driven Architecture (MDA) initiative is targeted at meeting the needs of the software industry characterized by these challenges. The MDA initiative attempts to decouple the design of the application from the target middleware. To support the MDA vision we propose a novel approach that supports *middleware transparent development* of software. In our approach we incorporate mechanisms that separate application development from concerns related to the integration of the target middleware. This paper presents a framework for middleware transparent software development and its use in developing Jini applications.

Keywords: Aspect oriented software development, Jini, middleware, distributed software, transparency, MDA.

1 Introduction

Decades after the beginning of the modern computer industry and the introduction of "software engineering" as a discipline [9], software development continues to be a difficult task. With the advent and rapid growth of the Internet [2], distributed systems are quickly becoming the norm for modern business applications. While distributed systems enable the sharing of resources in spatially and temporally separated environments, they also bring in numerous challenges, such as heterogeneity, openness, security, scalability, failure handling and concurrency [3]. Heterogeneity is inevitable in distributed applications because it is impossible to achieve consensus on the foundational components of distributed systems — hardware platforms, operating systems, network protocols, and programming languages [10].

Middleware technologies such as CORBA [12], COM [8], Jini [15], .Net and SOAP [4, 7, 14] address these challenges and enable consensus on an application's component interfaces and interoperability. They help application developers achieve *transparency* in several contexts [3, 6]: (1) local and remote accesses, (2) the location of resources, (3) concurrency and synchronization, (4) mobility of resources, (5) scaling, (6) performance, (7) replication, and (8) failure handling.

The large number of available middleware technologies and the rapid growth in their development and evolution presents challenges to software developers. Applications need to evolve with changes in middleware technologies. Developers constantly need to keep up with changes that affect the entire software development process. Application design and implementation are closely coupled with the middleware technology that is incorporated. When a need arises to change the middleware, entire applications need to be redesigned and implemented to incorporate the changes.

The Model Driven Architecture [12, 13] (MDA) initiative is targeted at meeting the needs of the software industry that are faced with these challenges. The MDA initiative attempts to decouple the design of the application from the target middleware. To support the MDA initiative, we propose a novel approach that supports *middleware transparent* development of software. Middleware transparency separates application development from the integration of middleware and shields application architects from the details of specific middleware. The high level design architecture is independent of the middleware. Elements of the application that are specific to the middleware are modeled separately using aspects and seamlessly integrated (woven) into the application later in the development process.

The remainder of the paper is organized as follows. We describe the framework and illustrate its use in developing Jini applications in Section 2. In Section 3 we discuss related work and the successes and challenges of realizing middleware transparency using our approach. We present our conclusions in Section 4.

2 Framework for Achieving Middleware Transparency

The proposed framework consists of a five-stage process that is based on the MDA vision. The framework has two objectives:

1. Provide a platform specific model (PSM)
2. Provide transformation models that separate the development of the platform independent model (PIM) from the generation of the PSM.

In our framework, specifics of a middleware technology are treated as crosscutting concerns and modeled as aspects. A crosscutting concern is one that is scattered across many elements in a design model or code of the application, but achieves one purpose. For example, security-related code may be scattered across many classes in a Java application. In this paper, the models reflect the actual code that is present in the application. This is, however, not a restriction on our framework. One can use models at a higher level of abstraction as well.

Model elements constitute a family of components and this family is the target of the crosscutting analysis. In distributed applications that use a middleware technology, components of the application form a family for the middleware technology. The middleware crosscuts the clients and the servers (or peer processes) in the application. The focus of our work is to isolate and encapsulate the middleware related concerns.

In our framework, each client and server component is associated with a list of five models that are also called *generic components*. These are shown in Figure 1.

1. *A platform independent model (PIM)*: The PIM is the model that is written independent of middleware considerations. The PIM for a server is called a platform independent

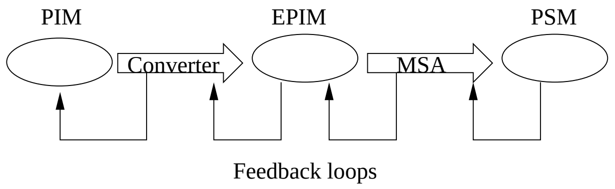

Figure 1: A Component View of the PSM Generation

server model (PISM), and that for a client, a platform independent client model (PICM). At the programming level, these are simply an expression of our functional requirements expressed using the target programming language.

2. *An enhanced platform independent model (EPIM)*: The EPIM is an enhanced version of the PIM that is made ready for aspect weaving. The EPIM for a server is called an enhanced platform independent server model (EPISM), and that for a client, an enhanced platform independent client model (EPICM). The EPIM is required because current aspect oriented programming languages (AOPLs) are not able to meet all the needs of an application (e.g. creating inner classes) prior to the weaving of the middleware specific aspects.

3. *A converter*: It provides a standardized mapping that transforms a platform independent model (PIM) to its enhanced version, (EPIM). A client converter is called a *c-converter*, while a server converter is called a *s-converter.* A separate converter is required for each middleware. Converters perform architectural, design or code transformations which cannot be done by the AOPL being used, for example code transformations such as creating and inserting inner classes.

4. *A collection of middleware specific aspects (MSA)*: These aspects capture the middleware requirements for an application. MSAs are the main mechanism for separating PIMs from PSMs. Aspect weaving transforms the enhanced model to its platform specific derivative. The MSA for a server is called the middleware specific server aspects (MSSA), and that for a client, the middleware specific client aspects (MSCA).

5. *A platform specific model (PSM)*: It is the complete application integrated with the middleware. The PSM for a server is called a platform specific server model (PSSM), and that for a client, a platform specific client model (PSCM).

The components can be classified into developed and generated items. Developers are required to build only three components — PIM, S-Converter, and MSA. The other two components (EPIM and PSM) are generated as part of the transformation process. The transformation is a three-stage process as described in Figure 2.

The rationale for the three-stage process is that current aspect oriented languages do not provide the facilities to make all the modifications necessary to the platform independent models (PICM and PISM). However, if this was possible, we would have a simpler two-stage process as shown in Figure 3:

The process is conceptually simple. First, the PIM is created using the chosen development paradigm. This could include for example aspect oriented analysis (AOA) and aspect oriented design (AOD), or any development paradigm (e.g., object-oriented methods). Depending on the target middleware, converters are created. The PIM is then fed into the converter to produce the EPIM. Aspect weaving is then used to add the MSA to the EPIM and produce the PSM.

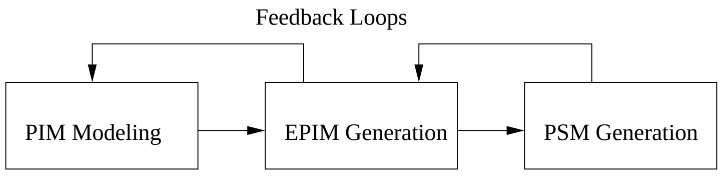

Figure 2: A Process View of the PSM Generation

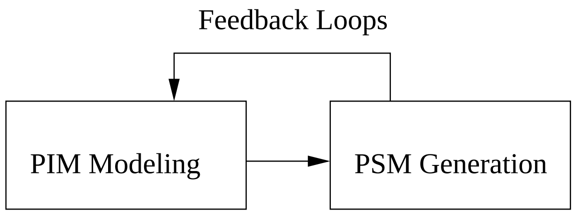

Figure 3: Two Stage Process

2.1 Jini application development

Examples of each component from the development of a Jini-based application are given below. AspectJ was used to write aspects that encapsulated Jini specific concerns. The models refer to the code artifacts.

1. PISM — Java server code independent of any middleware.
2. EPISM — Java server code made ready for Jini.
3. s-converter — a standardized mapping that performs transformations on the PISM that cannot be done using the MSSA. These include the insertion of `import` statements, and the creation and insertion of inner classes.
4. MSSA — AspectJ aspects that capture the Jini requirements for Java server applications. These represent the EPIM to PSM mappings.
5. PSSM — complete server code including all required middleware code. This code is produced in two stages, first the PISM is converted to the EPISM, then weaving adds the MSSA.

We used a converter to supply those conversions which cannot be enforced using AspectJ. Conversions used in the Jini s-converter were:

1. Insert all required import and extends statements.
2. Create interface — insert a remote interface that corresponds with the server interface.
3. Modify constructor — modify the constructor to supply the codebase for the application.
4. Create classes — create two inner classes: a service class (backend) and a proxy inner class for the backend class.
5. Any other required modifications.

2.2 Roles of developers

The co-dependent and interacting components in our framework necessitate complementary but differing roles for groups of software developers. The roles must be clearly defined and appropriate standards specified. The standards and responsibilities imposed on each group may vary depending on the target middleware and the sophistication of the available AOPL. In general, PIM developers create an application independent of middleware considerations except for an imposed programming standard. Converter developers must bridge the gap between the AOPL and MSA. MSA developers for Jini must address such concerns as groups, leasing, naming service, transactions, and security, as well as the parameterization of these concerns. Problem resolution and troubleshooting could become be a large part of the job description of EPIM and PSM composers since aspects composition can result in problems [1].

3 Discussion

A few recent projects have examined the use of *Aspect Oriented Programming* to achieve middleware transparency. Bussard [1] successfully encapsulated several CORBA services as aspects using AspectJ to make CORBA programming transparent to programmers. Hunleth, Cytron and Gill [5] suggest the creation of an AspectIDL for CORBA to complement the IDLs that are now available for languages such as Java and C++. The proposed AspectIDL would support several new types of *introductions* (using the AspectJ definition): interface method and field, interface super class, structure field, oneway specifier, and IDL typedef and enumerations. They also introduce the concept of concerns that crosscut programming languages. In our previous work [11] we used AspectJ to successfully encapsulate Jini specific code into a number of aspects that greatly simplified the application development process, and achieved a fair measure of transparency.

We are yet to see other approaches that incorporate aspect oriented modeling to achieve middleware transparency. Incorporating the use of aspects early on in the software development process is expected to reduce development time for the incorporation of new middleware technologies.

While our framework is promising and provides a model for reasoning and analysis of middleware transparency issues, a number of challenges and issues remain. First, the framework needs to be used with other middleware standards such as CORBA, SOAP, and .Net. This should result in a much more evolved and useful framework. Secondly, we applied the framework at the coding level for our Jini application. We need to analyze the framework in the context of aspect oriented analysis (AOA) and aspect oriented modeling (AOM).

4 Conclusion

We presented a framework for software development incorporating the use of aspects to model middleware technologies and decouple the design of an application from its target middleware. This framework evolved from research in which we are currently engaged and is based on the MDA vision. This work will provide many benefits such as reducing design complexity, reduced development time when incorporating new middleware technologies, maintainability, and adaptable software evolution.

References

[1] Laurent 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] CenterSpan. Internet Tutorial. *URL* `http://centerspan.org/tutorial/net.htm/`, 2003.

[3] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems Concepts and Design*. International Computer Science. Addison-Wesley/Pearson Education, USA, 2001.

[4] Robert Englander. *Java and SOAP*. O'Reilly, California, USA, May 2002.

[5] Frank Hunleth, Ron Cytron, and Christopher 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] Manish Parashar. Ece 451- Introduction to Parallel and Distributed Programming, Fall 2001. *Department of Electrical and Computer Engineering, Rutgers University*, 2001.

[7] Microsoft Corporation. .Net. *URL* `http://microsoft.com/net/`, 2003.

[8] Microsoft Inc. Microsoft's COM+ Technology. *URL* `http://www.microsoft.com/complus`.

[9] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. Software Engineering and Database. McGraw-Hill, United States, 1997.

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

[11] Devon Simmonds and Sudipto Ghosh. "Middleware transparency through aspect-oriented programming using AspectJ and Jini". In *Proc. of the Java/Jini Technologies II Conference at ITCOM 2002*, pages 133–141, Boston, Massachusetts, USA, August 2002.

[12] The Object Management Group. The Common Object Request Broker Architecture CORBA/IIOP 2.6. *URL* `http://omg.org/corba/`, 2003.

[13] The Object Management Group. The Model Driven Architecture. *URL* `http://omg.org/mda/`, 2003.

[14] W3Schools.com. The W3Schools.com Web Site. *URL* `http://w3schools.com/`, 2003.

[15] Jim Waldo. Alive and Well: Jini Technology Today. *IEEE Computer*, 33(6):107–109, June 2000.