An MDA Framework for Middleware Transparent Software Development

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*

May 13, 2003

Abstract

The development and evolution of distributed systems are generally coupled to continuously changing middleware technologies. This coupling is undesirable because changes in the middleware necessitates changes in the application, resulting in unnecessary constraints on the portability, interoperability, reusability, and evolvability of systems. It is imperative that mechanisms be found to make the development and evolution of distributed systems a *middleware transparent software development* (MTSD) process. The Model Driven Architecture (MDA) is an exciting initiative specifically designed to facilitate technology independent software development. In the MDA vision, application development is decoupled from the target middleware. We are developing a MTSD framework that is consistent with the MDA vision. In this paper we present an overview of the framework and the results of applying the framework to Jini application development.

Keywords: Distributed systems, middleware, Jini, transparency, AOSD, MDA.

1 Introduction

Software development continues to be a very difficult task amidst an industry characterized by rising demand for software, increasing software complexity, the inability to deliver quality products on time and within budget, and the immaturity of the industry [9]. Distributed systems are becoming the norm for modern industries, and from all indications will continue to do so for the foreseeable future. As software spans the globe, facilitated by the rapid growth of the Internet [2], distributed systems will become more complex. This complexity will be driven by a need to integrate and expand application domains, provide cross-platform and multi-middleware functionality, and accommodate and amalgamate a variety of potentially conflicting quality of service (QoS) requirements e.g., performance versus security, and reliability versus efficiency. In this paper we address two important issues in distributed systems development: middleware transparent development and quality of service composition.

1.1 Middleware Transparent Development Issues

Distributed middleware technologies, such as CORBA [13], COM [7], Jini [15], SOAP and .Net [14], are designed to enable the development of complex distributed systems by hiding infrastructural details from the application program. Infrastructural details include operating system and network specifics. These technologies provide a number of benefits, including: transparent access to infrastructural details; a menu of standard services (e.g. security, transactions); and transparent access to local, remote, and mobile resources. The term *transparent* as used in these examples, refer to the fact that infrastructural details are hidden from the application. For example, accesses to local and remote resources are done using the same mechanisms. In our research transparency relates to the entire middleware, rather than to middleware services. Middleware transparent software development (MTSD), is the development of software without consideration for a specific middleware. It has become necessary due to the proliferation of middleware technologies, and the undesirable coupling between middleware and application. This coupling is manifested by applications being developed for a specific middleware, and middleware artifacts being deeply embedded in applications. The coupling is undesirable because the middleware technologies change rapidly and changes in the middleware necessitates changes in the applications.

It must be noted that technological proliferation is inescapable. Indeed it has been observed that a consensus will not be achieved on the foundational components of distributed systems including hardware platforms, operating systems, network protocols, and programming languages [10]. A number of problems result from this lack of transparent development. First, code, component, and application reuse are constrained; secondly, the desired levels of portability and interoperability cannot be achieved; and finally, system maintenance (corrective, adaptive, perfective, and preventive) becomes an even greater challenge [9].

1.2 Quality of Service Issues

In addition to the MTSD problem, distributed systems also suffer from the absence of QoS specification and composition mechanisms. QoS have become an important topic in distributed systems with the increase in web-based services, and the growth in the embedded systems market [6]. Middleware provides a plausible vehicle for the specification and composition of QoS requirements for the following reasons.

1. QoS specification at the middleware level have a neat conceptual fit with transparencies already provided by the middleware such as access and location transparencies, and will of necessity have to use many standard facilities already provided by the middleware.

2. QoS requirements may conflict and would require trade-off analysis to determine the best configuration of QoS properties that meet QoS goals. Having this analysis in the middleware would simplify the software development process and facilitate QoS reuse.

We assume that quality of service properties such as fault tolerance and real-time performance are crosscutting concerns that can be modeled as aspects. The quality of service aspects (QoSA) in the framework, are encapsulations of QoS properties. While our framework addresses both QoS and MTSD issues, this paper focuses mainly on MTSD since this is where we have done most work to date. Our framework is consistent with the MDA initiative making our framework both portable and adaptable. The remainder of the paper is organized as follows. In Section 2 we present the framework. We describe our application of the framework using Jini in Section 3. Final thoughts and our conclusions are presented in Sections 4 and 5 respectively.

2 Framework for Achieving Middleware Transparency

The objective of the framework is to provide a middleware transparent software development process that facilitates the specification and composition of QoS attributes.

2.1 Framework Overview

In our framework, specifics of a middleware technology are treated as crosscutting concerns and modeled as aspects [4, 3]. A crosscutting concern is one that is scattered across many elements in a model, but achieves one purpose. For example, security-related concerns may be scattered across many classes in a Java application. In the framework, each client and server is associated with a list of components (see Figure 1). The meaning and roles of these components are described below. These descriptions represent our vision of the roles of these components since some are not yet implemented (e.g. aspect analyzer, QoSA), and others have been implemented only partially.

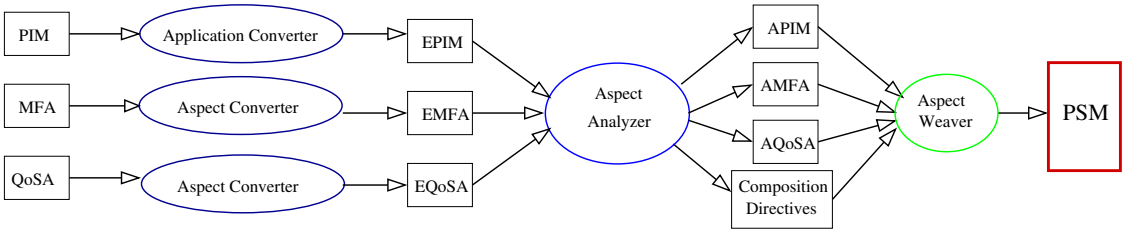

Figure 1: MTSD and QOS Framework for PSM Generation

1. *PIM*: The platform independent model of the application. The PIM is a generic model designed independent of middleware concerns.

2. *MFA*: A collection of middleware functional aspects (MFA). These aspects capture the middleware functional requirements for an application, for example, leasing, event handling and transactions. These are called *functional* to differentiate them from the QoS aspects (QoSA). MFAs are application independent generic aspects.

3. *QoSA*: A collection of middleware quality of service aspects, (QoSA), for example fault-tolerance and security. QoSAs are generic aspects developed independent of any application.

4. *Application Converter, Aspect Converter*: The application converter contains standardized mappings that transform platform independent models, to enhanced platform independent models (EPIM). The aspect converter contains standardized mappings that transforms generic aspects (MFA, QoSA) to application specific aspects (EMFA, EQoSA). Separate converters are required for each middleware. Converters perform architectural, design or code transformations to prepare a generic model for middleware specific aspect weaving.

5. *EPIM, EMFA, EQoSA*: These are the enhanced models. An independent model (PIM, MFA, QoSA) is developed without regard for a target middleware, and normally has to be transformed before aspect weaving is possible. An enhanced model is an independent

model that has been transformed to make it ready for aspect weaving. The enhanced middleware functional aspects (EMFA), and the enhanced quality of service aspects (EQoSA), are application independent, so they are transformed by the aspect converter to make them application specific.

6. *Aspect Analyzer*: The aspect analyzer contains standardized mappings to perform aspect composition analysis, composition conflict resolution, and trade-off analysis. Its role is mainly analytical. It outputs transformed models and composition directives that are used by the aspect weaver.

7. *APIM, AMFA, AQoSA, Composition Directives*: These are the analyzed models and the composition directives generated for the aspect weaver. APIM means analyzed platform independent model. AMFA means analyzed middleware functional aspects, and AQoSA means analyzed quality of service aspects. These models are analyzed for composition conflicts and QoS feasibility.

8. *Aspect Weaver*: The weaver uses the analyzed models and the composition directives to produce the final platform specific model(PSM).

9. *PSM*: A platform specific model. This is the final and complete model output by the framework with all the required middleware and QoS elements.

The process by which the PSM is generated, can be inferred from Figure 1 by reading left to right, where rectangles represent input and output and the circular components represent transformations. The aspect weaver produces the final model (PSM). The decision to weave is best made when conflicts and composition goals have been resolved satisfactorily.

2.2 MTSD and Automated and Component-Based Software Engineering

The MTSD and QoS framework we propose is expected to significantly affect the automation of distributed systems development. MTSD will allow tools to automatically customize applications for a specific middleware or platform. QoS specification and composition (QoSSC) directives will greatly enhance the automation of QoS specifications and realizations. MTSD and QoSSC together will facilitate the creation, analysis, and optimization of a variety of middleware related components, and component infrastructures. It is possible with large systems to have multiple versions of the same application running on the same or different platforms with completely different middleware and QoS requirements. MTSD will allow for the porting of applications while QoS composition semantics will allow for the configuration of systems and subsystems to meet client and systems goals and changing requirements.

3 Applying the Framework to Jini

We developed and used a *stock broker* application to test our framework. The application allows clients to buy and sell stocks through a software broker which is a distributed Jini application. Figure 2 shows the components of the PIM, EPIM, and MFA that were integrated to produce the final PSM for our examples. In short, we composed a number of interfaces and classes with our PIM, and performed a variety of transformations using our converters to produce the EPIM. A number of our classes were inner classes required by Jini. The PSM was produced by combining the enhanced models (EMFA and EPIM) using weaving. We did not consider aspect analysis or quality of service aspects (QoSA)in these examples.

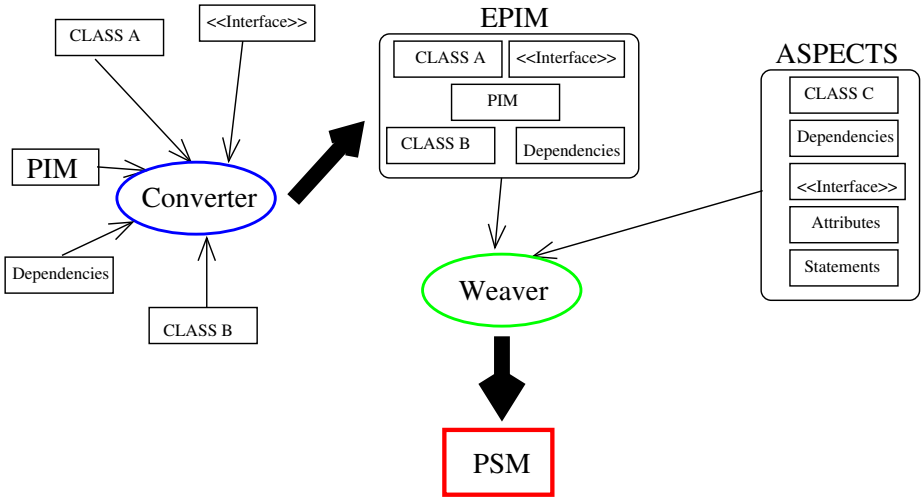

Figure 2: Application of Framework to Jini

4 Discussion

The relevance of this research can be seen from the number of research projects in QoS related issues and middleware [8]. These include Advanced Research Laboratory at Washington University St. Louis, British Telecom University Research Initiative at Lancaster University, The Control, Management and Telemedia (COMET) Group at Columbia University, Global Resource Management at SRI, Network Weather Service at the University of California, San Diego, Quality of Service for Objects (QuO) at BBN, and the MicroQoSCORBA Project at Washington State University. A few recent projects have examined the use of *Aspect Oriented Programming* to achieve middleware transparency. For example, 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++. In our previous work [12] we have proposed a framework for the middleware transparent development of distributed systems. We are testing our framework using CORBA, EJB, and Jini. There are many other notable research efforts for example [11].

5 Conclusion and Future Work

We presented a framework for middleware transparent software development and quality of service composition. We used aspects to model middleware technologies and decoupled the design of an application from its target middleware. We are working to implement the QoS composition component (aspect analyzer) of the framework. The framework frees developers from considering middleware concerns early in the software development life cycle, and holds great promise for application design. It raises the level of abstraction of distributed systems development, allowing the resolution of issues at design time rather than in coding, and facilitates the static analysis of design models. We believe that MTSD will make it possible to seamlessly migrate an application between middleware and to compose applications to serve the requirements of different middleware concurrently.

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] Geri George, Indrakshi Ray, and Robert France. Using aspects to design a secure system. In *The Eighth IEEE International Conference on Engineering of Complex Computer Systems - ICECCS*, Greenbelt, Maryland, USA, December 2002.

[4] Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck. Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM*, pages 87–93, October 2001.

[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] MicroQoSCORBA Home Page. Overview. *URL* `http://microqoscorba.eecs.wsu.edu/`, 2003.

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

[8] Object Services and Consulting, Inc. Internet Tool Survey. *URL* `http://www.objs.com/survey/QoS.htm`, 2003.

[9] Roger S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill series in computer science. McGraw-Hill, United States, 2001.

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

[11] Douglas C. Schmidt, Aniruddha Gokhale, Balachandran Natarajan, Sandeep Neema, Ted Bapty, Jeff Parsons, Andrey Nechipurenko, Jeff Gray, and Nanbor Wang. Cosmic: A mda tool for component middleware-based distributed real-time and embedded applications. In *Proc. OOPSLA Workshop on Generative Techniques for Model-Driven Architecture*, Seattle, WA USA, November 2002.

[12] Devon Simmonds, Sudipto Ghosh, and Robert France. "An Aspect Oriented Model Driven Architectural Framework for Middleware Transparency". In *Proc. of the Early Aspects Wprkshop at AOSD 2003*, Boston, Massachusetts, USA, August 2003.

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