An MDA Framework for Middleware Transparent Software Development & Quality of Service

Devon M. 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
http://www.cs.colostate.edu/

Abstract. Software development is plagued by a number of perennial problems. Chief among these is software complexity. Experts agree that complexity is and will remain a serious problem for the foreseeable future. One approach to mitigating complexity is to apply the old *divide and conquer* concept by targeting areas of software development where significant gains are likely. One such area is distributed systems. A major source of complexity in distributed systems stem from the fact that the development and evolution of software are generally coupled to continuously changing middleware technologies. This coupling is undesirable because changes in the middleware necessitates changes in the application with resulting constraints on the portability, interoperability, reusability, and evolvability of systems. We present a Model Driven Architecture (MDA) framework that reduces complexity by decoupling application development from the target middleware. The framework is designed to facilitate quality of service specification.

1 Introduction

Distributed systems [1] 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 even more complex. This complexity will be driven by a need to integrate and expand application domains, and provide cross-platform and multi-middleware functionality. Middleware is that most fundamental component of distributed systems, the purpose of which is to make distributed systems programmable. Unfortunately, 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 distributed system. This places unnecessary constraints on the portability, interoperability, reusability, and evolvability of distributed systems. Middleware has thus become a major source of complexity for distributed systems.

Among the tasks that distributed systems are required to do is that of accommodating and amalgamating a variety of potentially conflicting quality of service (QoS) requirements for example performance versus security. QoS issues in distributed systems have come into vogue with the advent of the web services revolution [3], [4]. As more services become available on the Internet, there is a growing demand for mechanisms to assure the quality of the available services. From a simple teleconference to a life-critical or business-critical Internet collaboration, deterministic QoS are in demand. This paper addresses middleware transparent software development (MTSD) and QoS in distributed systems, with emphasis on MTSD.

1.1 Middleware Transparent Software Development

There are a variety of distributed middleware technologies, including CORBA [5], COM [6], Jini [7], SOAP and .Net [8]. The purpose of middleware is to make distributed systems programmable by hiding infrastructural details from the application program. Infrastructural details include operating system, hardware, and network specifics. The benefits of middleware include 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 access* as used in these examples, refer to the fact that infrastructural details are hidden from the application, for example, resources can be moved within a system without affecting the operations of users.. By contrast, in MTSD, the term transparent refer to the entire middleware rather than to middleware services. In being consistent with the Model Driven Architecture (MDA) [9], the fundamental philosophy of MTSD is that distributed systems development should be completely ignorant of middleware concerns, allowing any middleware to be chosen and integrated after the system has been developed.

The MDA and related research, have convinced most that technological proliferation is inescapable in this pluralistic and multi-paradigmatic era. If this is so, then a consensus will not be achieved on the foundational components of distributed systems including hardware platforms, operating systems, network protocols, and programming languages [10]. The absence of MTSD result in many problems. 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 [11].

1.2 Quality of Service

One of the foundational pillars of this research is aspect-oriented software development (AOSD) [12]. AOSD is a new development paradigm that emphasizes the separation of concerns and the encapsulation of crosscutting concerns or aspects. 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 this research, we assume that both middleware

concerns, and quality of service properties such as fault tolerance and performance are crosscutting concerns that can be modeled as aspects [13], [14]. 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 it both portable and adaptable.

QoS have become an important topic in distributed systems with the increase in web-based services, and the growth in the embedded systems market [15]. Conceptually, middleware provides a very attractive vehicle for the specification and composition of QoS properties. Two main reasons for this are as follows [16]:

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.

The remainder of the paper is organized as follows. In Section 2 we present the MTSD framework. Then is Section 3 a description of our application of the framework to distributed systems development using the Jini middleware is presented. The paper is completed with Sections 4 and 5 which present our final thoughts and conclusions respectively.

2 The MTSD Framework

The objective of the framework is to provide a middleware transparent software development process that facilitates the specification and composition of QoS properties. The framework is aspect-oriented in that both middleware and QoS concerns are treated as crosscutting concerns and modeled as aspects. The framework associates each client and server with a list of components (see Figure 1). The remainder of this section serves to delineate the meaning and roles of these components. These descriptions represent our vision of the roles of these components since the QoS components are not yet implemented and others have been implemented only partially.

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.

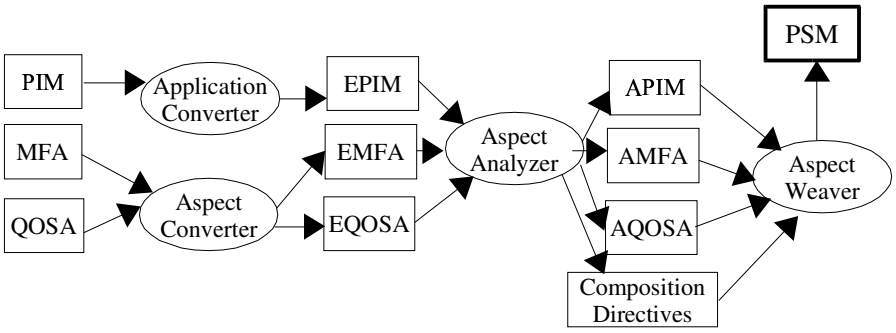

Fig. 1. MTSD and QoS Framework Component View

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. Converters - (application and aspect): 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. Independent models (PIM, MFA, QoSA) are developed without regard for a target middleware (PIM), or target application (MFA, QoSA), and normally have 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. Composition Directives: These are the logic and semantic statements generated by the aspect analyzer. The directives are used by the weaver to control and guide the weaving process.

8. APIM, AMFA, AQoSA: These are the analyzed models. 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.

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

10. 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 following is a simplified linear design overview of the PSM generation process. In this example we ignore QoS concerns.

1. Create the PIM for the application (server/client) – no middleware consideration necessary.

2. Select the target middleware and create the middleware aspects (MFA).

3. Transform the application to make it ready for the target middleware. This generates the EPIM.

4. Transform aspects (MFA) to make them ready for target application.

5. Weave the aspects (EMFA) into the application (EPIM) to produce the PSM.

3 Application: MTSD & Jini

To date most of the work we have done is with the Jini middleware. We have developed a simple *stock broker* distributed application and used it to test our application. The basic requirement of the application is the buying and selling of stocks through the broker. Figure 2 shows the actual components that were integrated to produce the final PSM. We did not consider aspect analysis or quality of service aspects in these examples. The application was written in Java and AspectJ was used as the aspect weaver. We followed the simplified design process presented earlier. This requires us to develop the Jini converters and MFA as a one-time exercise, and the PIM classes and interfaces for each required distributed application. These aside, the process is completely automated.

4 Discussion

Research in technology independent software development is a relatively new phenomena spearheaded by the OMG and their MDA initiative. A significant amount of work is also being done in web services and embebbed systems. QoS related research are numerous and include [17]: 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. AOSD has also spawned an exciting conglomorate of research projects [12], some of which are related to middleware and transparency. For example, Bussard [18] successfully encapsulated several CORBA services as aspects using AspectJ to make CORBA programming transparent to programmers. Hunleth, Cytron and Gill [19] suggest the creation of an AspectIDL for CORBA to complement the IDLs that are now available for other languages. There are many other notable research efforts for example [20].

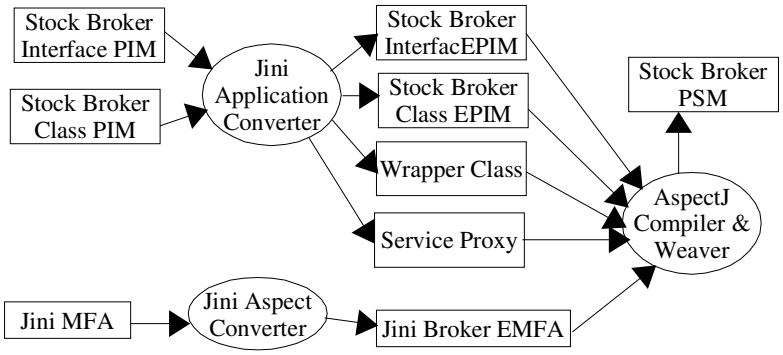

Fig. 2. The Jini Stock Broker MTSD Project

5 Conclusion and Future Work

The MTSD and QoS framework we propose is expected to have a positive effect on the automation of distributed systems development. MTSD will allow tools to automatically customize applications for a specific middleware or platform. Many other exciting possibilities, such as configuring an application to work with multiple middleware are also on the horizon. In this paper 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 the target middleware. In the future we expect to apply the framework to other middleware technologies such as CORBA, SOAP, .and Net. In addition we also expect to implement the QoS and aspect analysis components of the framework. The framework facilitates the static analysis of design models. We believe that MTSD will make it possible to seamlessly migrate applications between

middleware and to compose applications to serve the requirements of different middleware concurrently.

References

1. George Coulouris, Jean Dollimore, and Tim Kindberg. 2001. Distributed Systems Concepts and Design. 3rd Edition, Addison-Wesley.
2. CenterSpan. Internet Tutorial.. URL: http://www.centerspan.org/tutorial/net.htm
3. InfoEther LLC. White Paper: Enabling the Web Services Revolution: Leveraging the Transforming Power of the Personal Computee. URL: http://www.infoether.com/
4. Daniela Florescu, Andreas Grunhagen, and Donald Kossman. XL: An XML Programming Language for Web Service Specification and Composition. Proc. The Eleventh Int'l World Wide Web Conference, Honolulu, (May 2002).
5. The Object Management Group. The Common Object Request Broker Architecture CORBA/IIOP 2.6. URL http://www.omg.org/corba/
6. Microsoft Inc., "Microsoft's COM+ Technology," URL http://www.microsoft.com/complus
7. Jim Waldo. Alive and Well: Jini Technology Today. IEEE Computer, Vol. 33, pages 107-109, (June 2000).
8. The W3Schools.com Web Site. URL: http://www.w3schools.com/
9. The Object Management Group. The Model Driven Architecture (MDA). URL http://www.omg.org/mda/
10. Richard Soley: MDA, An Introduction. URL: http://www.omg.org/mda/presentations.htm
11. Roger S. Pressman. Software Engineering: A Practitioner's Approach (4th Edition). McGraw-Hill (2001).
12. The AOSD Web Page. URL: http://aosd.net/
13. Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck. Handling Crosscutting Constraints in Domain-Specific Modeling. Communications of the ACM, (October 2001).
14. Geri George, Indrakshi Ray, and Robert France. Using Aspects to design a Secure System. Proc. The Eighth IEEE International Conference on Engineering of Complex Computer Systems – ICECCS, Greenbelt, Maryland, USA, (December 2002).
15. microqoscorba.eecs.wsu.edu. The MicroQoSCORBA Home Page: Overview.
16. D. M. Simmonds, S. Ghosh and R. B. France. An MDA Framework for Middleware Transparent Software Development. To appear in Proceedings of RTAS 2003 Workshop on Model-Driven Embedded Systems, Washington, D.C., (May 27-30, 2003).
17. Object Services and Consulting, Inc. Internet Tool Survey. URL http://www.objs.com/survey/QoS.htm
18. Elisa L. A. Baniassad, Gail Murphy, Christa Schwanninger and Michael Kircher. Where are Programmers Faced with Concerns?, Workshop on Advanced Separation of Concerns in Object-oriented Systems at OOPSLA 2000.
19. Frank Hunleth, Ron Cytron and Christopher Gill. Building Customizable Middleware using Aspect Oriented Programming . OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems, Tampa FL, USA. (October 14, 2001).
20. 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. Proc. OOPSLA Workshop on Generative Techniques for Model-Driven Architecture, Seattle, WA USA, (November 2002).