Middleware Transparent Software Development & the MDA

Devon Simmonds, Sudipto Ghosh, Robert France
Computer Science Department, Colorado State University
Fort Collins, CO 80523
Phone: (970) 491-4608, FAX: (970) 491-2466

August 20, 2003

Abstract

A major source of complexity in distributed systems stem from the fact that the development and evolution of distributed systems are generally tightly 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. The problem is made significant due to the proliferation of middleware technologies and the pervasiveness of distributed systems. The Model Driven Architecture (MDA) is an exciting initiative that advocates decoupling the design of an application from the integration of the target middleware. To support the MDA vision we have developed an aspect-oriented *middleware transparent development* framework [1, 2, 3, 4]. This paper motivates the framework components and their use in Jini distributed systems development.

Keywords: Distributed systems, Middleware Transparent Software Development, MDA, middleware, Jini, CORBA, AOSD.

1 Introduction

The software development industry has seen significant growth and maturation over the last four decades of the 20th century. This growth and maturation can be seen in the evolution of programming paradigms, software architectures, software engineering and reengineering methods and tools, database and systems engineering, software and process metrics, and the diverse array of applications and application domains that have embraced software. Despite these positives, software development continues to be plagued by rising demand for software, increasing software complexity, and the inability to deliver quality products on time and within budget [5]. Distributed systems [6] have becoming the norm for modern industries, and indications are that this will continue to be so for the foreseeable future. The rapid growth of the Internet and the associated web services revolution, are expected to make distributed systems not only more pervasive but also more complex. This complexity will be driven by a need to integrate and expand application domains, and provide cross-platform and multi-middleware functionality.

Distributed middleware such as CORBA [7], COM [8], Jini [9], SOAP and .Net [10], is that most fundamental component of distributed systems, the purpose of which is to make distributed systems programmable by hiding infrastructural details including operating system, network and hardware specifics from application programs. Unfortunately, the pervasiveness of distributed systems and the proliferation of middleware technologies have conspired to create a very significant problem - the development and evolution of distributed systems have become coupled to continuously changing middleware technologies. This coupling presents problems because changes in the middleware necessitates changes in the distributed system. This results in undesirable constraints on the portability, interoperability, reusability,

and evolvability of distributed systems. Middleware has thus become a major source of challenge and complexity in distributed systems development and evolution.

1.1 The Model Driven Architecture (MDA)

The problem of constantly changing technologies, while not new, has become a much more significant problem in the last decade. During this time the Object management Group (OMG) [11], a large international trade association, has been championing the march to help reduce complexity, lower costs, and hasten the introduction of new software applications. In order to achieve this the OMG has provided a number of resources and standards, these include the Unified Modeling language (UML), the Meta-Object facility (MOF), the Common Warehouse Metamodel (CWM), the Object Management Architecture (OMA), and the Model Driven Architecture (MDA). The MDA and related research, is premised on the assumption that technological proliferation is inescapable in this pluralistic and multi-paradigmatic era. An immediate consequence of this is that a consensus will not be achieved on the foundational components of distributed systems including hardware platforms, operating systems, network protocols, and programming languages. The MDA advocates a modeling approach to software development that decouples the specification of software functionality from the specification of the target technology. The approach is model-driven, meaning that models are given first-class status im the MDA. The ultimate desire is to use software design models to control the development of software including code generation. The MDA proposes an architectural separation of concerns that articulates three views of a system, each represented by a model. These models are:

1. A Computation Independent (business/domain) Model (CIM) - a representation of the system requirements from a ”business” perspective without considering software concerns.
2. A Platform Independent Model (PIM) - a representation of a system that ignores details related to specific platforms. It captures those elements of a system that remain the same from platform to platform.
3. A Platform Specific Model (PSM) - a model of a system that combines platform independent information with information related to a specific platform.

The term platform refer to a specific technology for example CORBA, .Net, Jini, etc. The three primary goals of MDA are portability, interoperability and reusability.

1.2 Middleware Transparent Software Development (MTSD) Defined

Middleware provide many benefits to distributed systems, 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, and when applied to middleware, generally refer to the fact that infrastructural details are hidden from the application. Middleware therefore provides a number of transparencies including access, location, concurrency, replication, failure, mobility, performance, and scaling transparencies. By contrast, in MTSD, the term *transparent* refer to the entire middleware rather than to middleware services and facilities. The objective of MTSD is therefore to make the entire middleware invisible to the distributed systems developer. This vision is consistent with the MDA and allows for a number of signigicant benefits. First application developers are spared from managing the complexity of middleware concerns, secondly, the complexity of the software development process is reduced, and thirdly, the portability, interoperability and reusability of middleware and functional concerns and greatly enhanced. In addition costs will be reduced throughout the application life-cycle, there will be reduced development time for new applications, and MTSD will

facilitate improved application quality, the rapid inclusion of emerging technology benefits into existing systems, and an overall reduction in the complexity of application development, maintenance and evolution.

MTSD can be divided into two major categories of projects. First, there is the development of new distributed systems without the constraint of a target middleware. This is called middleware transparent software engineering (MTSE). Secondly, there is middleware transparent software reengineering (MTSR) - the modification and redesign of applications to make them middleware transparent. MTSR is an exciting research topic because of the myriad of legacy distributed systems in existence. If these systems are to take advantage of advances in new technologies, then they will have to be re-engineered to make it possible to integrate new and improved middleware technologies. MTSD subsumes both MTSE and MTSR.

2 The MTSD Framework

The objective of the framework is to provide a middleware transparent software development process that is MDA compliant. Presently only MTSE activities are addressed.

2.1 Framework Components

The framework is aspect-oriented and treats middleware as crosscutting concerns modeled as aspects [12, 13]. A crosscutting concern is one that is scattered across many elements in a model, but which achieves one purpose. For example, security-related concerns may be scattered across many classes in a Java application. The meaning and roles of these components are described in detail in [1, 2, 3, 4]. In this paper we present a simplified framework that ignores aspect analysis which is yet to be implemented. The other components of the framework are:

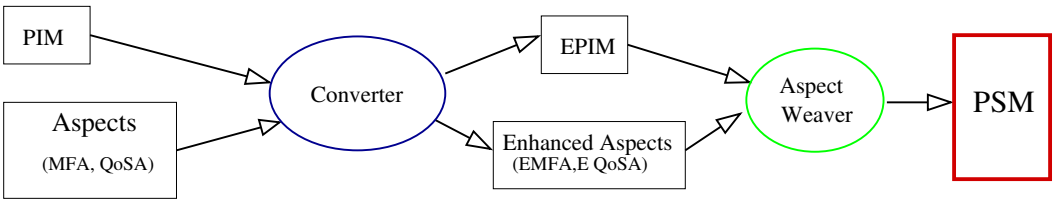

Figure 1: A Simplified Framework without Aspect Analysis

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

2. *Aspects* - MFA and QoSA: A collection of aspects that capture the middleware requirements (services, facilities, QoS) for an application, for example leasing, transactions, and security. MFA means middleware functional aspects, and QoSA means quality of service aspects. MFA includes all non-QoSA aspects.

3. *Converters* - Standardized mappings that transform PIMs and middleware aspects to their enhanced versions. Separate converters are required for each middleware. Converters perform architectural, design or code transformations to prepare a generic model (PIM, MFA, QoSA) for a specific context.

4. *The Enhanced Models* - EPIM, EMFA, EQoSA: Independent models (PIM, MFA, QoSA) are developed without regard for a target middleware (PIM), or a target application (MFA, QoSA), and have to be transformed by a converter before aspect weaving is possible. For example Section 3 shows a Jini PIM rearchitectured to contain additional inner classes required by Jini.

5. *Aspect Weaver*: The weaver produces the final platform specific model(PSM) by combining the enhanced aspects (EMFA, EQoSA) and the EPIM.

6. *PSM*: The final and complete model output by the framework with all the required middleware elements.

2.2 Framework Design Process Overview

The design process used by the framework to create applications is a simple three-stage semi-linear process where the activities of stage 1 can be done concurrently as can the activities of stage 2. The three stages consists of five activities as follows:

1. **Stage 1**: Create the PIM for the server/client - no middleware consideration necessary.

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

3. **Stage 2**: Transform PIM to EPIM using the application converter.

4. Transform the generic aspects to enhanced aspects (EMFA, EQoSA) using the aspect converter.

5. **Stage 3**: Weave the enhanced aspects into the EPIM to produce the PSM. A separate PSM is produced for each client and for each server.

2.3 The Rationale for Enhanced Models

The question of the purpose and relevance of enhanced models is an important one that continues to generate interest. Figure 1 shows that there are two main input to the proposed framework - a PIM and aspects (MFA, QoSA). The same figure also shows that there are enhanced models for each PIM and each aspect. Each PIM has a particular architectural signature (structure) consisting of a collection of classes with specific relationships, a collection of interfaces used by the classes, and a collection of methods particular to each class and interface. Although all PIMs are middleware independent, one PIM may differ significantly from another PIM in its architectural signature. This difference and the fact that middleware aspects require a PIM with a specific architecture in order for aspect weaving to be successful, provide the rationale for an Enhanced Platform Independent Model (EPIM). An EPIM is simply a PIM that has been refactored to produce the appropriate architectural signature required for a specific middleware aspect. The particular refactoring algorithm applied depends on the selected aspect (e.g. transaction) and the selected middleware (e.g. Jini). For a given middleware, different aspects will require different architectural signatures and for a specific aspect (e.g. transaction) the required architectural signature will differ from middleware to middleware. In our framework the refactoring algorithm is encapsulated in a *converter*. There are significant benefits to this separation. These refactoring transformations cannot be specified in the PIM as that would make it platform specific, and it cannot be specified in the weaver (e.g. AspectJ) as this would make the weaver application specific which is completely unacceptable. Specifying the transformation in a separate component is therefore the only sure way to be MDA compliant and provide the flexibility and structural coherence needed for the framework. A similar argument can be provided for the need of enhanced aspects (EMFA, EQoSA). A generic middleware aspect (MFA, QoSA) is intended to be used for any relevant application. At the time of its creation it is unnecessary to

determine the specific application to which a generic aspect will be applied. For practical purposes this information would be unavailable for numerous situations. Generic middleware aspects are therefore application independent. The application independent nature of generic aspects coupled with the fact that each application has its own architectural signature, necessitates that generic aspects are transformed before usage to make them application specific. A useful example is provided in the following section.

3 Application of Framework to Jini

Most of the work we have done is with Jini but some work has also been done with CORBA. Our primary application is a simple stock broker distributed application. Its functional requirements includes the registration of clients, and the buying and selling of stocks. We did not consider aspect analysis or quality of service aspects in this example. 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. Figures 2, 3, and 4 give a graphical view of the PIM, EPIM and the PSM for the application. The specific activities that produced these models during the design phase will now be delineated.

1. Creating the PIM & MFA - The MFA were developed and reported in [1]. The PIM (see Figure 2) was developed and tested as a stand alone Java application. It consists of a single interface and three classes.

2. Generating the Enhanced Models (EPIM, EMFA).

   A number of differing Jini design models are possible. The one we chose to use required that the services to be made available by the server be captured in a inner class and that a proxy for this inner class be created as well. The aspect converter implemented a simple string matching algorithm that replaces generic class names and class attributes in the aspects with the actual class names and attributes from the Stock Broker application. The application converter was written using the Java Tree Builder software [14]. It generated the EPIM (see Figure 3) by a refactorization of the PIM to include seven new components all of which are required by Jini. The tasks performed by the application converter are as follows:

   (a) Create a remote interface to be used by the server

   (b) Create two inner classes: a server (from the PIM) and a proxy for the server.

   (c) create a wrapper class as the outer class for the two classes just mentioned.

   (d) Insert import statements from the interface into the wrapper class.

   (e) Add throws clauses and/or exceptions statements specific to Jini to the client and server code.

3. PSM Generation. AspectJ was used as the weaver. This is both an asset (AspectJ is Java compatible) and a challenge (our directives are limited to those of AspectJ).

The MFA is not shown but it includes all the components in Figure 4 (the PSM) that are not found Figure 3 (the EPIM). These components include the *discovery, lookup, lease, and activation* packages.

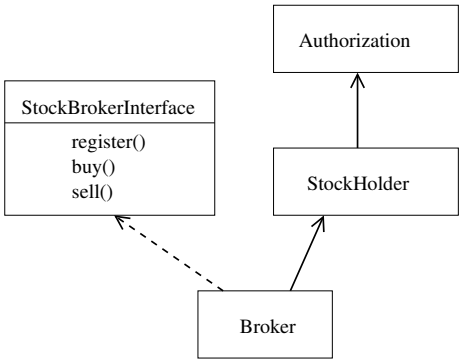

Figure 2: The jini StockBroker PIM

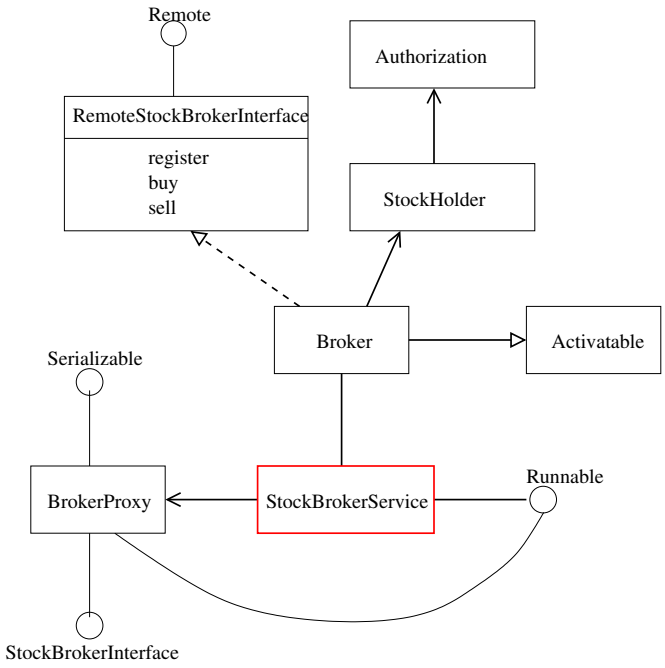

Figure 3: The Jini StockBroker EPIM

4 Discussion and Conclusion

MDA related research and success stories [15] are numerous and progressively increasing. They include for example research on problems in electric power generation, transmission, distribution and industrial use at the Asea Brown Boveri (ABB) Research Center Heidelberg, Germany; and the Open System Architecture for Condition-Based Monitoring/Maintenance (OSA-CBM) Project. Apart from our work [1, 2, 3, 4], research specifically related to MTSD exist but under differing nomenclature, for example the ArcStyler [15] project. Other researchers have examined the use of *Aspect Oriented Programming* to achieve middleware transparency for example, Bussard [16] successfully encapsulated several CORBA services as aspects using AspectJ to make CORBA programming transparent to programmers. Significant related work have also been done by Hunleth, Cytron and Gill [17], Douglas C. Schmidt et al [18], Jeff Gray et al [12], Robert France et al [13], and Charles Zhang et al [19].

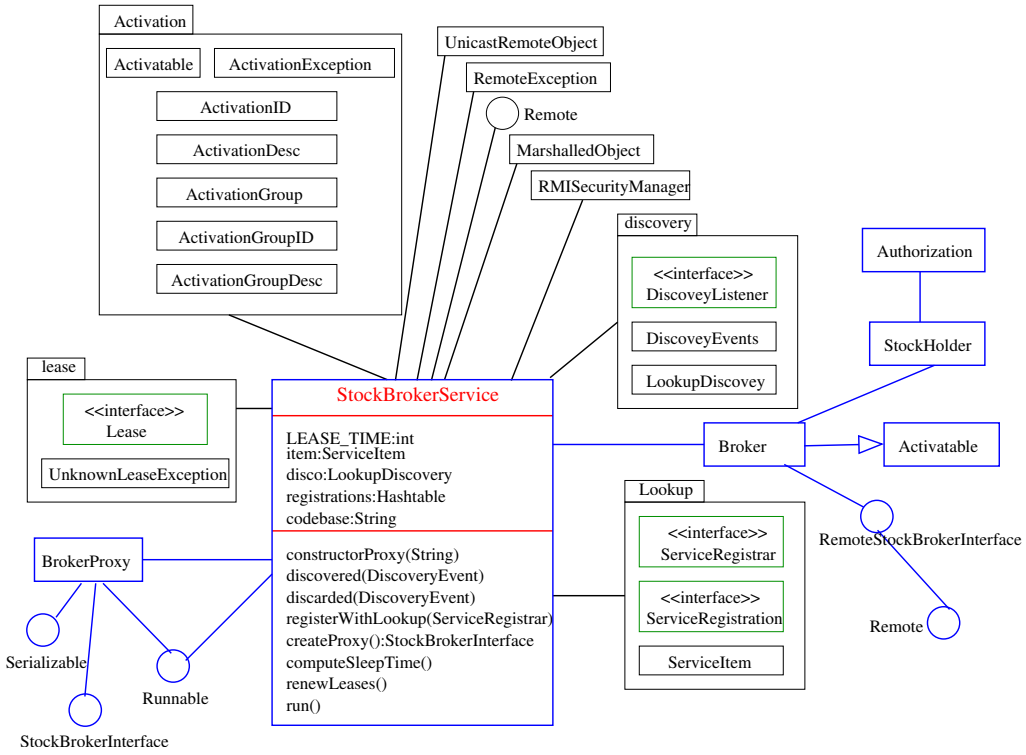

Figure 4: The Jini StockBroker PSM

The MTSD framework we presented is designed to facilitate quality of service composition. This is important because of the close afiinity between middleware and QoS. The framework frees developers from considering middleware concerns early in the software development life cycle, and facilitates the MDA goals of portability, interoperability and reusability. It raises the level of abstraction of distributed systems development, allowing the resolution of issuesat design time rather than in coding, and fosters the static analysis of design models and the rapid inclusion of emerging technology benefits into existing systems.

References

[1] 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.

[2] 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.

[3] Devon Simmonds, Sudipto Ghosh, and Robert France. "An MDA Framework for Middleware Transparent Software Development". In *Proc. of RTAS 2003 Workshop on Model-Driven Embedded Systems*, Washington, D.C., USA, May 2003.

[4] Devon Simmonds, Sudipto Ghosh, and Robert France. "An MDA Framework for Middleware Transparent Software Development & Quality of Service". In *To appear in Proceedings of QoS in CBSE Workshop at RST2003*, Toulouse, France, June 2003.

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

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

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

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

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

[10] W3Schools.com. The w3schools.com web site. *URL* `http://w3schools.com/`, 2003.

[11] The Object Management Group (OMG). The OMG Web Page. *URL* `http://omg.org/`, 2003.

[12] 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.

[13] 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.

[14] Jens Palsberg, Kevin Tao, and Wanjun Wang. The Java Tree Builder. *Department of Computer Science, Purdue University, URL: http://www.cs.purdue.edu/jtb/.*

[15] The Object Management Group. MDA Success Stories. *URL: http://www.omg.org/mda/products-success.htm*, 2003.

[16] 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.

[17] 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.

[18] 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.

[19] Charles Zhang and hans Arno. Jacobsen. Quantifying aspects in midleware platforms. In *Proc. of AOSD 2003*, Boston, Massachusetts, USA, August 2003.