IN SUPPORT OF AN ASPECT-ORIENTED APPROACH TO MIGRATING DISTRIBUTED APPLICATIONS

Devon M. Simmonds
Computer Science Department
University of North Carolina at Wilmington
Wilmington, NC 28403
Phone: (910) 962-3819, FAX: (910) 962-7457
\{simmonsd\}@uncw.edu

Abstract

Migrating distributed applications from one middleware to another is difficult in legacy software systems where middleware functionality is spread across and tangled with business logic. In these systems, the old middleware code has to be removed before the application is refactored and the new middleware functionality is added. However, when distributed applications are developed in an aspect-oriented model-driven development environment, the need to extract the old middleware before application migration proceeds may not exist. This paper motivates and illustrates the need for an aspect-oriented approach to distributed application migration in model driven development environments.

Keywords: aspect-oriented software development, distributed applications, application migration, middleware.

1 Introduction

Distributed systems [21] have become the norm for modern industries buoyed by several factors including [6, 11, 12]: the emergence of mobile and ubiquitous computing, service oriented architectures, web services and the rapid growth of the Internet. Modern distributed systems require the management of a variety of services such as security, transaction management and concurrency. These services are typically provided using middleware [3, 5, 22], a layer of software that occurs between application programs and the operating system. The primary purpose of middleware is to simplify the development of distributed systems by providing transparent access to infrastructural details such as operating system and hardware features; access to local, remote, and mobile resources; and a menu of standard services such as method invocation, events and quality of service.

With new and better technologies constantly emerging, the capacity to migrate a legacy application to a new technology is important for business competitiveness. Businesses migrate distributed applications to new middleware to protect investment, reduce costs and complexity, and to increase flexibility and performance. However, migrating a distributed application to a new middleware is difficult because middleware functionality is typically spread across and tangled with business logic. This difficulty in migrating applications reduces the portability, interoperability, reusability of distributed systems [5, 14, 15, 16, 17].

The model driven architecture [23] is a model driven engineering (MDE) initiative designed to address the proliferation of middleware technologies and the associated difficulties of managing distributed applications. The MDA advocated the vertical separation of platform independent business logic from platform-specific middleware technologies and the specification of transformations for converting a platform independent model (PIM) to a platform specific model (PSM). In addition to vertical separation of software concerns, horizontal separation of concerns are also required in MDE. Aspect-oriented software development (AOSD) techniques [1] support the horizontal separation of crosscutting software functionality. In AOSD functionality that crosses a design are modularized in aspects [2, 8, 9, 24].

In our previous work we designed an aspect-oriented model driven development framework (AOMDF) that supports vertical and horizontal separation of concerns [18, 19, 20]. This paper uses the AOMDF to motivate and illustrate the need for an aspect-oriented approach to the migrating distributed applications.
2 The Aspect-Oriented Model-Driven Development Framework (AOMDF)

Figure 1 is an activity diagram representation of the major actions and artifacts of the aspect-oriented model driven development framework (AOMDF) [18, 19, 20]. The framework supports the development of applications through the transformation of models and the separation of concerns. Both vertical and horizontal separation of concerns are supported. Vertical separation of concerns is realized by facilitating transformation of models from one level of abstraction to another. Horizontal separation of concerns is realized by (1) enabling the representation of crosscutting software features as aspects, separate from primary models that represent the core application functionality; and (2) facilitating separate specification of different software concerns at the same level of abstraction. The AOMDF is represented by two main composite activities: a Model Transformation activity and a Model Composition activity.

In the Model Transformation activity, actions and artifacts are classified into three phases: a Model Acquisition Phase, a Transformation Specification Phase, and a Transformation Execution Phase. During the Model Acquisition Phase the source models that are to be transformed are acquired. Source models consists of a primary model that represents the core functionality of the application being developed and a set of aspect models, each representing a crosscutting software feature.
During the Transformation Specification Phase, the transformations required to change the source aspect models and the source primary model into target models are defined. Several classes of transformations are possible including: \[7, 13\]

1. Composition: a model transformation where multiple source models are integrated.
2. Anti-composition or Decomposition: a model transformation where a single source model is divided into multiple target models.
3. Refactoring: a model transformation where a model is reorganized into different parts at the same level of abstraction.
4. Refinement: a model transformation where more detail is added to a model.
5. Abstraction: a model transformation where detail is removed from a model.
6. Enhancement: The process of customizing a model to include required platform-specific features. The generated artifact is called an Enhanced Model. For example, transforming a platform-independent primary model to include features required by a middleware (e.g., CORBA), or transforming a generic transaction (two-phase commit) aspect model to include features required by CORBA.
7. Code Generation: The process of using input design models to output code for a target programming language.

Refinement, abstraction, and code generation are vertical transformations while composition anti-composition and refactoring are horizontal transformations. Enhancement may be either vertical or horizontal.

The Transformation Execution Phase involves applying the transformations defined during transformation specification to produce target models. The Model Acquisition Phase and the Transformation Execution Phase represent distinct abstraction levels in the model transformation process. The Model Acquisition Phase is referred to as the source level of a transformation while the Transformation Execution Phase is referred to as the target level of a transformation.

In aspect-oriented software development, a complete application is produced by composing the aspects with a primary model \[4\]. The Model Composition section of the AOMDF is designed to support the integration of models. The composition process is realized with the support of bindings and composition directives \[19, 10\]. Bindings identify the places in the primary model where the aspects are to be integrated, while composition directives are rules for determining how the aspect models and the primary model are to be composed. Model composition is presented as an independent AOMDF component because composition may be done at either the source level or the target level.

The AOMDF represents a family of model-driven development process instances. The distinction between these instances is determined by the values selected for four AOMDF variation points. The variation points are: (1) the level at which composition is done, i.e., source level or target level; (2) the nature of the levels, i.e., PIM to PIM transformation, PIM to PSM transformation, PSM to PIM transformation, or PSM to PSM transformation; (3) the types of transformations to be applied, for example, refactoring or anti-composition; and (4) the types of models being transformed, i.e., only aspect models, only primary model, both aspects and primary model. The perceived benefits of the AOMDF include: \[19\]

1. When aspects are separated from primary model, the specification of the transformation of either aspect or the primary model is likely to be less complex than the specification of the transformation of the corresponding composed model, since the composed model is likely to have more relationships and dependencies.
2. AOMDF supports the management of crosscutting concerns, which are isolated and encapsulated as aspects and therefore changes can be made to the aspects in one place, and trade-off analyses can be more easily effected by composing different aspects with the same primary model.
3. Application independent aspects, once defined, may be reused across multiple applications and application domains.

3 Application Migration Using The AOMDF

A primary benefit of aspect-oriented model driven engineering (AOMDE), in addition to the benefits noted in the previous section, is the capacity of AOMDE to support the migration of a distributed application from one platform technology to another. Application migration is a process where the technology with which an application was developed is replaced by a different, often better technology. Migration may happen at the design level or the code level. Figure 2 illustrates
Figure 2. AOMDF Migration Approach (adapted from [5] Fig 2).

application migration in the classic AOMDE context. The figure shows the migration of a distributed application from middleware M1 to middleware M2. The process is divided into two parts: (1) the development of the M1 application, Figure 2(a), and thereafter, (2) the development of the M2 application, Figure 2(b). The figure shows that when an application is to be migrated in AOMDE, three activities are necessary:

1. The aspect models for the new middleware must be created or obtained (Steps 5 to 7).
2. The original primary model must be enhanced to make it compatible with M2 (Step 8).
3. The M2 Enhanced Primary Model and the M2 Context-specific Aspects are then composed to form the new M2 application.

Implicit in these three steps are several benefits inherent in AOMDE approach. These benefits are summarized in the next section.

4 Benefits of Application Migration Using The AOMDF

To underscore the benefits of an aspect-oriented approach to application migration, consider a scenario in which a legacy application developed with middleware M3 in a non aspect-oriented manner, is to be migrated to middleware M2. This migration scenario is illustrated in Figure 3. In the figure, migration is accomplished in two main steps: (1) The M3 aspects are developed (Steps 1a, 1b and 2a), and (2) a version of the legacy primary model specifically enhanced for M2 is prepared from the M3 PSM (Steps 1 and 2). It is in step 1 the AOMDE approach is superior to the traditional non-AOMDE approach. Specifically, all the M3 artifacts in the M3 PSM must be found and extracted before migration can proceed. This presents

Figure 3. Migrating A Legacy Application.
several non-AOMDE difficulties:

1. The PSM may be very complex. For example, if the PSM is a design model, it could consist of hundreds of classes (for object-oriented software) with scattered and tangled middleware artifacts. Even worst, if the PSM is code, the application could consist of potentially millions of lines of code.

2. The M3 middleware must be removed from the application. This involves:
   (a) Deleting middleware artifacts that have no embedded application statements.
   (b) Modifying application artifacts that have embedded middleware statements.

3. In order to be able to correctly remove the M3 artifacts from the legacy application, developers are required to be conversant with the M3 middleware. Therefore either they are familiar with M3 or they must learn:
   (a) The syntax and semantics of the M3 middleware.
   (b) When, where and how each middleware feature is used as well as inter-dependencies between middleware features and application features.

This may require the expenditure of financial and other resources to train developers. However, in this case, the efforts expended to train developers and to learn the syntax and semantics of the M3 middleware are essentially undesirable from a business perspective, since the M3 middleware is being discarded. Thankfully, in an AOMDE environment, these expenditures are unnecessary.

4. The M2 middleware must be added to the application, so developers may also have to learn the M2 middleware.

5 Discussion and Conclusion

The ability of AOMDE to support migration of distributed applications as presented in this article, is premised on one important assumption about the nature of AOMDE environments. In an AOMDE environment where code is generated from models, it is important that modifications to the application are model-driven rather than being code-centric. If modifications are code-centric, it is critical that the design models from which the code was generated be updated to correctly and faithfully reflect the changes made at the code level. Without this symbiosis between models and code, any future enhancement of the design model and associated transformation into code for a different middleware will be faulty, since the transformed software would not faithfully reflect all the features represented in the old code.

The resources and human effort that must be expended to learn a new middleware, may be much more than it appears at first thought. Consider for example, a scenario in which developers must become familiar with CORBA distribution. Then they need to understand: the number of dependencies among the CORBA features used to effect distribution, and the number of dependencies between CORBA distribution features and the core functionality of the application. Developers must therefore learn the following CORBA features and relationships: the role of the portable object adapter (POA), naming service and object request broker (ORB); the relationship between the POA and the ORB, the relationship between the ORB and the naming service and the relationship between naming service components.

In general, the resources and effort required to migrate a distributed application in a non aspect-oriented environment may be particularly exhaustive in cases where the developers must first gain an understanding of the old middleware. This is the case because a complex distributed application may include not one but several middleware features with complex inter-dependencies, for example, transactions, security, events, fault tolerance and concurrency. Even in cases where developers are already familiar with the old middleware, the effort required to correctly locate and extract all the artifacts of the old middleware may be quite exhaustive.

AOMDE has the capacity to significantly reduce the effort required to migrate distributed applications, since minimal effort and resources are expended on the old middleware. In addition, the effort expended on the new middleware is also minimized since required middleware features are encapsulated in aspects. Therefore, with respect to the new middleware, developers may only need to correctly follow configuration instructions. By contrast, in a non-AOMDE environment, developers may be required to learn both old and new middleware.

References


