Metrics for Migrating Distributed Applications

Devon M. Simmonds
Department of Computer Science
University of North Carolina, Wilmington
Wilmington, North Carolina, 28403
simmondsd@uncw.edu

Abstract

Managing ever changing technology is a significant challenge that has given rise to the model driven architecture and other efforts aimed at decoupling middleware technology from core business logic in enterprise applications. This paper presents a set of metrics for quantifying the effort required to migrate a distributed application from one middleware to another. The metrics may be used to quantify the benefits of a model driven development approach such as the AOMDE where crosscutting functionality is isolated. A case study that illustrates use of the metrics to migrate a Jini application to CORBA is presented. The results of the case study suggests that migrating an application to a new technology may involve significant human effort and that investing in aspect-oriented development could result in significant benefits.

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

1. Introduction

Software engineering [1] is a central sub-discipline of computer science centered on finding appropriate methods and tools for the systematic development and evolution of complex software systems. Software evolution [2] is itself a challenging undertaking especially when the software being managed has embedded crosscutting technologies [12]. The challenge stems in part, from the pervasiveness of such technology throughout enterprise applications and the resulting difficulty that this tangling and scattering of middleware technology presents when transition to a new middleware is desired. This difficulty in making the transition to new technologies has led to such efforts as the model driven architecture [4] and more recently, to Platform as a Service (PaaS) [8, 9].

The MDA is premised on the availability of principles, techniques and tools for decoupling middleware technology from core business logic in enterprise applications. In MDA, model driven engineering involves separating between platform independent models and platform-specific models and defining functional mappings that capture how PIMs are transformed into PSMs and vice versa. Several successful MDA stories have been reported [10].

Cloud-based platform as a service is a more modern solution to the problem of managing embedded technologies. In PaaS developers need not worry about changing technologies since the services that the technologies provide are now available from a vendor – as a service. This is quite an attractive option, assuming that organizations are free to change their PaaS providers. Against this backdrop, it is easy to forget that there are those organizations still bemused with the changing technology problem resulting from the presence old technology in their legacy applications. For these organizations, the transition to new technologies through an MDA-style approach or through PaaS remains a challenge and therefore, having some mechanism for systematically quantifying such transitions is beneficial.

This paper presents a set of metrics for quantifying the effort required to migrate a distributed application from one middleware to another. A case study that illustrates use of the metrics to migrate a Jini [3] application to CORBA [5] is presented. The rest of the paper is organized as follows. Section 2 introduces the migration metrics and a model for computing migration effort. Section 3 presents a case study of migrating an application from a Jini to a CORBA
platform. Finally, section 4 presents a discussion of the results and their implications.

2. Metrics for Application Migration

When a non-aspect-oriented distributed application is migrated [14, 16] to a new middleware, four activities must be undertaken:

1. The semantics of the new middleware must be learnt. This includes determining when, where and how each middleware feature is used as well as the inter-dependencies between middleware features and the distributed application (server or client). This is an important requirement even in cases where middleware features are pre-packaged as design or code aspects since it may be necessary to correct defects in the development and testing efforts.

2. The old middleware must be removed from the application code. This involves:
   a. Deleting statements from the application in cases where middleware statements do not have embedded business logic code.
   b. Modifying statements in the code in cases where business logic code statements have embedded middleware expressions.

   While this requirement is absent when an aspect oriented approach is used its computation is useful to determine the effort saved by aspect oriented and PaaS approaches.

3. The application code must be refactored [15] to facilitate the integration of the new middleware.

4. Code for the new middleware must be added to the application. This involves:
   a. Adding new statements to the application in cases where middleware statements do not require embedded business logic expressions.
   b. Modifying application statements in cases where business logic statements require embedded middleware expressions.

Using the four steps presented in the previous section, a model for quantifying the effort to migrate an application to a new middleware was developed using the Goal-Question Metric paradigm [13] and the concepts and metrics central to the application migration effort were identified. The GQM model is shown in Figure 1 classifies migration effort into two components: learning and development.

The learning element identifies the effort required to learn a new middleware technology. This element is represented by the first question in the GQM model. The development element identifies, (1) the effort required to modify the existing application and (2) the effort required to create new middleware artifacts. Each of these items is represented as a question in the GQM model.

2.1. Quantification of Learning Effort

The learning element of the model quantifies the effort expended to learn a new middleware technology in order to be able to write and deploy applications using the target middleware. We assume that developers are not familiar with the target middleware. Learning effort (LE) is quantified using the formula:

\[
LE = ISD + IFD + NOF
\]

where ISD represents inter-service dependencies, IFD represents inter-feature dependencies and NOF represents number of features that collaborate to
provide the middleware services used. Learning effort could apply to both source middleware and target middleware. That is, for legacy systems, the developer may need to learn both the old middleware as well as the new. Middleware features typically involve such things as such as security, transactions, persistence, and distribution. The intuition behind this formula is that while a developer may be familiar with a programming language syntax, the semantics of a new middleware will have to be learnt. In addition, language semantics is a function of structural semantics (interdependencies and when and where a feature is used) as well as algorithmic semantics (how a feature is used).

It should also be recognized that learning the semantics of and writing code for services that have more dependency relationships with other services are expected to be more difficult than services with fewer inter-service relationships. Similarly, a service with more intra-service collaborating features is expected to require more effort to learn and write code. Using this information, and the GQM model in Figure 1, the effort quantification model shown in Figure 2 was developed.

### 2.2. Middleware Development Effort

Middleware development involves middleware coding and middleware modification. The middleware coding element is used to quantify the effort required to write code for the new middleware. Middleware coding effort (MCE) is quantified as the number of new lines of code (NLOC) that must be written for the new middleware. The formula used is:

\[
\text{MCE} = \text{sum (NLOC added for each method)}
\]

The middleware modification element quantifies the effort required to:

1) Delete old middleware code where the middleware code does not have any embedded business logic expressions. For example the Jini statement `ActivationGroup.createGroup (gid, group, 0)` does not require any embedded business logic expression. The statement is used to create a Jini activation group.

2) Delete old middleware code where the middleware code does not have any embedded business logic expressions. For example the Jini statement `ActivationGroup.createGroup (gid, group, 0)` does not require any embedded business logic expression. The statement is used to create a Jini activation group.

The middleware modification effort (MME) is quantified as the number of middleware lines of code that must be changed (i.e., modified or deleted) in each method. The formula used is:

\[
\text{MME} = \text{sum(MLOC changed for each method)}
\]

The middleware development effort (MDE) is given by:

\[
\text{MDE} = \text{MCE} + \text{MME}
\]

In summary the application migration effort (AME) for a specific application is computed as a combination of learning effort (LE) and development effort (MDE) using the formula:

\[
\text{AME} = \text{LE} + \text{MDE} \quad \text{………} \quad (1)
\]

Figure 2. Migration Effort Quantification Model.
where \( LE = ISD + IFD + NOF \) \( \ldots \ldots \) (2)

and \( MDE = MCE + MME \) \( \ldots \ldots \) (3)

3. Case Study: Jini to CORBA Migration

Using formulas 1, 2 and 3, the effort quantification model requires the computation of the learning and development components.

3.1 Learning Effort Computation

In order to compute learning effort, the number of services used, their inter-relationships and the number of features and their inter-relationships must be computed. Only one service, (‘distribution’) was used in this case study.

![Service and Feature Interaction Model](image)

![Server and CORBA Feature Interaction](image)

Figure 3. Jini to CORBA Migration Quantification Example.

As a result, inter-service dependency (ISD) is 0. In order to determine learning effort for CORBA distribution, the following metrics must be computed:

1) The number of middleware services used for CORBA distribution. For this case study only one service (‘distribution’) was used.
2) The number of dependencies among the CORBA features used to effect distribution.
3) The number of dependencies between CORBA distribution features and the business functionality.
4) The number of lines of CORBA code that would normally be written for the migration effort.
5) The number of lines of application code edited or deleted to eliminate Jini code.

The inter-dependencies of the features that collaborate to provide CORBA distribution is illustrated in Figure 3a. The figure shows that a CORBA server requires use of seven different CORBA features. Learning requires an understanding of:

1) The role of the portable object adapter (POA), naming service and object request broker (ORB).
2) The relationship between the POA and the ORB.
3) The relationship between the ORB and the naming service.
4) The relationship between naming service components.

Overall a total of six inter-feature dependencies (IFD) must be understood. In this example the CORBA naming service is used.

The program-feature dependencies (PFD) of the CORBA features and the business functionality is illustrated in Figure 3b. In this case the semantics of seven distinct relationships need to be understood. The learning effort is therefore computed as:

\[
LE = ISD + NOF + IFD + PFD = 0 + 6 + 7 + 7 = 20 \text{ units of learning.}
\]

This number indicates that before migrating to CORBA, a developer must understand 13 relationships and seven features. This includes comprehending the structural feature semantics (when and where a feature is used) as well as algorithmic semantics (how a feature is used).

3.2 Computing Middleware Development Effort

Middleware development effort involves writing code for CORBA, and modifying or deleting application code with embedded Jini statements. The middleware coding effort (MCE) is quantified as the total number of CORBA lines of code (MLOC) that must be added to an application. An estimate for MCE was computed as the LOC written for the CORBA aspects. This is a reasonable assumption since the functionality provided by the aspects must also be provided by the developed code. The total lines of code written for CORBA server aspects amount to: 26 LOC
for the server and 19 LOC for the client. In summary:

\[ MCE = 26 + 19 = 45 \text{ LOC} \]

The total number of lines of Jini code to be eliminated amount to 199. This is the number of LOC written for the Jini aspects. We assume that no middleware modification effort (MME) will be required for CORBA distribution. The total middleware development effort is therefore:

\[ MDE = MCE + MME = 45 + 199 = 244 \text{ LOC} \]

In summary the application migration effort (AME) for the application is computed as:

\[ AME = LE + MDE \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (1) \]

\[ = 20 \text{ units of learning and} \]

\[ 244 \text{ units of development.} \]

4 Discussion and Conclusion: Making Sense of Metrics

This result suggests that migrating application across middleware may involve significant human effort. This inference is based on the following observations:

1. The learning and development efforts for a typical distributed application will be much more than for this example because a distributed application may use any number of middleware services, for example, transaction, security, events, fault tolerance and concurrency.
2. In this case study we used pre-tested aspects that were pretested and would therefore eliminate some errors that would normally be uncovered during a development project where middleware code is written from scratch.
3. Our model ignores the cost of inter-service connectivity and inter-feature connectivity and their varying degrees of complexity. That is, some inter-connectivity is more complex and requires more time to grasp. For example, understanding distribution is less complex than understanding transaction management.

This simple case study provides some indication of why platform as a service is such a growing phenomenon. However, there are many legacy systems where one or more middleware are endemic and for these organizations making the transition to a new middleware or to PAAS will not occur without some pain.

References


2015.


