A Pattern-Oriented PSM Interoperability Framework

Di Zhou
Dept. of Computer Science and Software Engineering
University of Canterbury
dzh21@student.canterbury.ac.nz

Dr Richard T. Pascoe
Dept. of Computer Science and Software Engineering
University of Canterbury
richard.pascoe@canterbury.ac.nz

ABSTRACT
Software systems typically comprise multiple components which are developed using different technologies, thus the interoperability that enables the components to interact despite differences in programming languages and execution platforms is becoming a central issue. This paper describes how to use Object-Oriented design patterns to derive a cross-platform interoperability framework for bridging the different technology specific models (PSMs) within the Model Driven Architecture (MDA). In addition, we present a MDA based approach to applying the proposed PSM interoperability framework in an automatic fashion to generate a communication bridge between two PSMs in a multi-agent system.

Categories and Subject Descriptors
D.3.3 [Programming Languages]: Language Constructs and Features – Frameworks.
D.2.11 [Software Engineering]: Software Architecture – Patterns.

General Terms
Design

Keywords
MDA, Interoperability, Object-Oriented Frameworks, Design Patterns.

1. INTRODUCTION
The MDA [1], initially proposed by Object Management Group in 2001, is a model-driven approach to designing and developing reusable, interoperable software components and data models. It facilitates software development and improves model reusability by raising the software design level from implementation and Platform-Specific Models (PSMs) to Platform-Independent Models (PIMs).

A PIM is the reusable, portable, business description of a system that is specified independently of any particular technology. In contrast, a PSM is constructed by adding to the PIM the details of how an underlying implementation technology is to be used in the implementation of the target system. As software systems generally span multiple technologies, a number of PSMs, each involving a differing technology (e.g. Web model, EJB model and relational database model), are generated from a PIM.

In order to enable interoperability between different PSMs, described in Section 2, we propose a cross-platform interoperability framework that is designed and constructed using a sequence of Object-Oriented design patterns [2], which are like templates that capture design experience of experts and can be reused to solve recurring software design problems.

Section 3 presents existing framework definitions given by other authors. Section 4 discusses the proposed PSM interoperability framework including: the goals of the framework, the design of the framework and an evaluation of the framework. Adopting a MDA based approach can enable the framework to be reused in an automatic fashion by using transformation tools to generate not only PSMs but also bridges among them.

The adopted MDA based approach (known as elaborative MDA), described in Section 5, uses transformation tools to transform models at the different levels of abstraction i.e. PIMs, PSMs and implementation, elaborating some by inserting source code directly. Finally, conclusion and future work are presented in Section 6.

2. INTEROPERABILITY WITHIN MDA
In general, interoperability is defined as [6]: "...the ability of two or more systems or components to exchange information and to use the information that has been exchanged." In the research described here, interoperability is sought among multiple PSMs that have been derived from one PIM, despite any platform differences associated with each of these PSMs. This type of interoperability is classified as "syntactic interoperability" in [7]. Previous research related to interoperability between PSMs is summarized next.

Losange et al. [9] propose to (a) utilize a traceability model to keep links between the source and target model elements; and (b) record the communication information between the different models. A bridge between two PSMs is to be generated using a three step process: (1) automatic generation of the trace model; (2) analysis of the trace model to obtain the necessary conversions for the bridging purpose; and (3) realization of the bridge.

N. Bencomo and G. Blair [8] present a research problem related to interoperability among different platform specific applications and necessity for transformations and mappings of concepts between different PSMs in the context of specific domain problems. The focus of the authors is on "the identification of pertinent Domain Problems and the consequent..."
definition of mappings and transformation between the abstractions of different Middleware Technology Models."

Kleppe et al [5], refer to the interoperability problem as the need for different application components, each of which is built using the best technology for the job, to interact with each other. The authors’ proposal involves realizing cross-platform interoperability by using transformation tools to generate not only the PSMs, but the necessary bridges between them as well. The generated bridges are responsible for transforming and mapping the concepts from one platform into the concepts used in another platform. More specifically, the idea is that if one PIM is transformed into two PSMs that are targeted at two platforms then all of the information that is needed to bridge the gap between the two PSMs is available. Thus, the transformation tool can utilize the information to generate the bridges.

While a number of authors have tried to give their own solutions to solve the interoperability problem as described above, this paper proposes a pattern-oriented framework that acts as a generic interoperability mechanism by which different PSMs are to interoperate.

3. FRAMEWORKS

Object oriented frameworks as a means of reusing design have been discussed at length in the literature. Authors have defined and classified frameworks in different ways while some have provided design guidelines for developing reusable frameworks.

Johnson and Foote [10] describe a framework as an object-oriented abstract design comprising a number of classes which can be specialized by application-specific classes to suit different applications. They classify frameworks into white-box and black-box frameworks. A white-box framework requires users to understand the structure of the framework before they can create application-specific subclasses. This type of framework is reused through inheritance by creating subclasses of framework base classes and providing the implementation of framework is reused through inheritance by creating subclasses that implement the interface of the components and abstract application logic. A user can create an application by implementing a specific configuration with the enterprise objects and binding the abstract application logic to the configuration.

The proposed PSM interoperability framework, described in the next section, is a white-box, application framework.

4. THE PROPOSED PSM INTEROPERABILITY FRAMEWORK

In this section: the goals for the proposed framework are given; patterns central to the framework are described; application of the framework to the development of a MAS is presented; and finally, the success with which the goals for the framework are achieved are discussed based upon the experiences of applying the framework.

4.1 Goals

The proposed interoperability framework should be:

1. A generic foundation for bridges that allow interaction among components implemented using distinct software platforms.
2. Easily reused, ideally in an automatic fashion, to a particular application.
3. Simple to avoid penalizing the application’s performance as a consequence of using the framework to implement the necessary bridges.

4.2 Design

We communicate our design of the framework using a set of design patterns as a number of experts have recommended to use patterns as a way to communicate design information as well as to document frameworks [2] [4].

The interoperability, described in Section 2, refers to the ability of two or more PSMs to interact. In particular, we focus on the methods allowing consumer PSMs to invoke services delivered by service provider PSMs. In order to achieve this goal, a number of issues need to be addressed as explained in the following sub-sections.

4.2.1 Exposing Service

A PSM's functionality is to be exposed to its clients (other PSMs) in such a way that: the provider PSM is easy to use; modifications of the provider PSM's implementation should not affect its clients; and the dependencies between the provider and its clients are minimized.

The Facade pattern [2] is applied to solve the above problem. A facade object encapsulates a provider PSM's functionality into a simplified interface and handles mapping/ transformation services to allow communication with the provider. The provider PSM is still available for clients who may need to use more specific interfaces. A client communicates with the provider through the facade object, which redirects calls from the client to the relevant objects in the provider PSM. A simple provider PSM may have one facade whereas a more complex provider PSM may contain multiple facades that expose different sets of functionality. Use of the Facade pattern decouples client PSMs from the provider PSM, such that varying the provider PSM will not affect its clients, and improves the portability of the provider PSM.

4.2.2 Handling Requests

Service provider PSMs share common steps to handle service requests. The steps refer to begin, redirect, and end. However, the implementation of each individual step may vary depend on each service provider PSM.

This problem can be solved by applying the Template Method pattern [2]. Template Method defines a fixed order in which the steps of an algorithm are performed (i.e. a blueprint of an
implementation), but it allows subclasses to vary the details of each step. This ensures that the structure of the algorithm remains unchanged, while subclasses provide the implementation. The template method is defined in an abstract class which provides the default implementation for one or more hook operations. A hook operation is a method that has a default implementation and may be overridden by subclasses.

4.2.3 Encapsulating Service Invocation

A client PSM needs to make requests to a service in such a way that it only knows to what service it wants and has no knowledge of how to implement these requests.

The Command pattern [2] is an appropriate solution to the problem. A command object encapsulates a request to perform some action on a receiver object. It separates the object that implements the request from the object(s) that receive and carry out the request. A requester makes a request by calling the execute() method of a command object. By applying the Command pattern, the service requester is decoupled from the service provider. The requester has no knowledge of how to make a request to a service; it just has a command object whose responsibility is to implement a request to the service provider.

4.2.4 Adapting Client Interface

- How to enable multiple client PSMs, each of which has a different interface, to communicate with a service provider PSM through a common interface?

- How to enable a client PSM to communicate with multiple service provider PSMs, each of which has a different interface?

The Adapter pattern [2] is applicable to solve these interoperability problems. The first interoperability problem can be addressed by creating one adapter for each client. Each adapter converts the client’s specific interface into the common interface that is compatible with the service provider’s façade. The second interoperability problem can be tackled by creating multiple adapters for the client. Each adapter is responsible for converting the client’s specific interface into the corresponding service provider’s interface. Interface adaptation enables a client interacts with one or more service providers that might expect different interfaces to the client. Adapters encapsulate the implementation details of the interoperability mechanism from the client.

4.2.5 Encapsulating Macro Service Invocation

A client PSM may need to make a sequence of requests to different service façades.

A MacroCommand is defined as [2]: “MacroCommand is a concrete Command subclass that simple executes a sequence of Commands.” This is exactly what is needed to solve the problem. The MacroCommand can be implemented using the Composite pattern. Thus, composite provides a way to create a part-whole hierarchy of objects. It simplifies clients code by allowing them to make a sequence of requests as the same as to make a single request. In order to enable clients to treat individual objects and composite objects uniformly, all objects in the structure need to have the same interface. This can be achieved by creating an abstract class that represents both of individual objects and composite objects.

4.2.6 Summary

Figure 1 describes the design of the interoperability framework, including all patterns applied. Each Class is annotated using one or more stereotypes to show the pattern name. ServiceFacade, Command and MacroCommand are the framework base classes that are expected to be extended or instantiated in a particular application.

Figure 1: Interoperability Framework Design Summary

4.3 Evaluation of the Framework

A multi-agent system (MAS) was developed in order to evaluate the applicability of the framework in terms of the three design goals specified earlier in Section 4.1. A MAS is a community of autonomous agents [3] that operate to achieve individual goal(s) whilst collaborating and negotiating with each other to fulfill an overall goal.

A prototype MAS, depicted in Figure 2, has been developed to investigate the performance of various Straddle Carrier dispatching rules in the container unloading operation in a port terminal. The physical resources that are being used in the port terminal such as Quay Cranes, Straddle Carriers are modeled as Crane Agents, Straddle Carrier Agents respectively. Similarly, the ship controller is modeled as the Ship Controller Agent. Each agent has a capability to perform database related operations (e.g. updates and queries) on a local database. Jadex 1 is used to construct Belief Desire Intension (BDI) structured agents and the object-oriented database management system of JADE 2 is used for persistency.

Figure 2: Usage of the framework in the MAS application.

Goal 1 requires the proposed framework to act as a generic interoperability mechanism by which different software platforms are to interoperate. The MAS application consists of two software platforms: Jadex and JADE, which can not directly interact with each other. Applying the proposed framework to this problem, a communication bridge, comprising the Façade class and the Adapter class, is built.

1 http://vsis-www.informatik.uni-hamburg.de/projects/jadex/
2 http://www.jadeworld.com/
between the two subsystems as shown in the above figure. The Façade class that extends the ServiceFacade abstract class is created in order to (a) expose the service provided by the JADE PSM; (b) handle mapping/transformation services to allow communication with the provider. The hook methods (i.e. begin(), redirect(), and end()) defined in the ServiceFacade class are overridden in the Facade class. The Adapter class that extends the Command abstract class is created in order to (a) encapsulate a request to the JADE’s façade; (b) convert the requester agent’s interface into the interface that the façade expects. As it is satisfied by the effect of the communication bridge that successfully enables Jadex agents to interoperate with the JADE’s database management system, the goal 1 is achieved.

Goal 2 is to reuse the proposed framework in an automatic fashion. The adopted MDA approach, described in Section 5, enables the reuse of the proposed framework in an automatic fashion by using transformation tools to generate a communication bridge between the Jadex PSM and the JADE PSM. More specifically, a common interface between the consumer (i.e. Jadex PSM) and the service provider (i.e. JADE PSM) is specified as part of the PIM. Therefore, the communication bridge, comprising a façade and an adapter, can be generated automatically using transformation tools. This goal is achieved due to the fact that the proposed framework is automatically reused in the prototype MAS.

Simplicity is the third goal. As the framework comprises three classes each of which has a small number of methods, we say that the goal is achieved.

5. BRIDGE GENERATION
This section presents the adopted MDA approach to automatically reuse the proposed framework to generate bridges among PSMs. The bridge generation process encompasses (1) the creation of a common interface between a service consumer and provider in the PIM; (2) the generation of bridge PSM, comprising Facade and Adapter classes, from the PIM; and (3) the generation of executable code from the PSM.

Creating a PIM in a UML modelling tool is the starting point of the MDA process. The completed PIM is exported to an XMI document which is transformed into another XMI document representing the PSMs according to a set of transformation rules, which provide the details about PIM-to-PSM model transformations. The resulting PSMs in XMI format may be then imported and refined using a UML modeling tool. Finally, the refined PSMs expressed in XML are transformed into executable code using a PSM-to-Code transformation tool.

6. CONCLUSION AND FUTURE WORK
This paper describes a pattern-oriented framework that is used to bridge different PSMs in a MDA application. We have adopted the approach of using design patterns as building blocks for the framework design in order to make the output framework more reusable.

A prototype MAS has been developed and will be applied to enhance container unloading operations in a port terminal. The MAS consists of two components that are developed using different technologies i.e. Jadex is used to construct BDI agents and JADE is used for persistence. The proposed framework is applied in the MAS as a generic interoperability mechanism by which the two distinct components are able to successfully interoperate.

The proposed framework is a white-box framework that is reused through inheritance. It requires users to understand its internal structure before they can apply it in a particular application. Although this can be a difficult task when the design of the framework is complicated, it is not the case for the proposed framework because it only consists of three base classes each of which has a small number of methods.

In our adopted MDA approach, the generation of the two PSMs is a semi-automatic process since the system’s dynamic behavior needs to be specified manually in the source code. In the future, we will shift our focus to capture the system’s dynamic behavior in PSMs.

7. REFERENCES