JDSI: A SOFTWARE INTEGRATION STYLE FOR INTEGRATING MS-WINDOWS SOFTWARE APPLICATIONS IN A JAVA-BASED DISTRIBUTED SYSTEM*

Jim-Min Lin, Zeng-Wei Hong, Guo-Ming Fang
Department of Information Engineering and Computer Science, Feng Chia University, Taichung City 40724, Taiwan, ROC

Keywords: COTS software reuse, Software integration, Java-based distributed system, Architecture style

Abstract: Developing software systems by integrating the existing applications/systems over the network is becoming mature and practical. Microsoft Windows operating systems today support a huge number of software applications. It may accelerate the construction of components, if these commercial software applications could be transformed to software components. This paper proposes an architectural style to support a 3-phases process for migrating MS-Windows applications towards a distributed system using Java technologies. This style is aimed to provide a solution with clear documentation and sufficient information that is helpful to a software developer for rapidly integration of MS-windows applications. In final, an example parking lot management system that assembles two MS-Windows applications was developed in this work to demonstrate the usage of this style.

1 INTRODUCTION

Distributed computing has been an important trend for developing software systems. Client/server programming, distributed object systems and Internet/WWW are distributed computing technologies enabling a software system developed by integrating different software components across the boundaries of the network, operating systems, and languages. The research on the integration of distributed commercial off-the-shelf (COTS) software applications is becoming mature and practical. One of the important features of reusing COTS software is that software developers can rapidly integrate the well-established functionality in the new system rather than developing one from scratch. There have been a large number of COTS software applications running on a variety of Microsoft operating systems. It would facilitate the software component construction, if the software developers could utilize the functions of an existing MS-Windows application.

Nowadays Java is already a popular distributed computing paradigm widely adopted over industries, marketplace, and the Internet. An important feature of a Java program is that it can be deployed on heterogeneous OSs and platforms. It would be significant to assemble heterogeneous software components including diverse MS-Windows software applications using Java technologies. On the achievement of the above goal, a lot of Java-based services could be easily developed with the existing MS-Windows applications. Moreover, the diverse legacy applications under Microsoft OSs could be reused and included into a new Java-based software system.

In this study, we found it the major difficulty to obtain the source code and technical support from various vendors when reusing MS-Windows software applications. Therefore, it would be a key issue to overcome the incompatibility between MS-Windows application's operational interface and Java application's programmable interface.

Therefore, an architectural style named as Java-based Distributed Software Integration (JDSI) in this paper provides a migration process consisted of a bottom-up three phases: encapsulation, gluing, and interfacing. We firstly construct a server-side Java object by wrapping MS-Windows application(s) in the *encapsulation phase*. In this phase, a software wrapper will be used to overcome interface incompatibility. Then in the *gluing phase*, a coordinator program is used to coordinate the server-side wrapped objects and to integrate them into a client-server system. Finally, in the *interfacing*

* This research was partly supported by National Science Council, Taiwan, ROC under Grant NSC92-2213-E-035-044.

phase, we refer to the Model-View-Controller (MVC) pattern (Schmidt 1999) to construct the user-interactive system for the client-user's manipulation in the integrated software system.

By referring to the JDSI style, lots of valuable information, like design issue-solution pairs, participants' specification, and implementation strategy could be obtained to create applications with similar problem issues. To demonstrate the usage of the proposed style, a graphic parking-lot system that reuses and integrates MS-Windows software applications will be reported in this paper.

2 RELATED WORKS

Many researches have proposed the means to migrate the existing applications to a distributed system. However, most of them focused the source-available legacy applications.

De Lucia et al. (De Lucia 1997, Cimitile 1998, Canfora 1998, Aversano 2001) proposed a six-step process to decomposing the existing source code of COBOL systems into user interface component, application logic, and database. These components could be integrated into a Web-based information system (Aversano 2001). Serrano et al. (Serrano 2002) encompassed data mining and evolutionary migration techniques into reengineering a legacy program. They also proposed a feasible reengineering methodology to migrating non-object-oriented systems into the CORBA platform .

The main difficulty raised in De Lucia's work is how to efficiently decompose an existing system (Cimitile 1998), Canfora 1998). However, lots of COTS MS-Windows software applications are seldom decomposable and even non-decomposable. Therefore, we prefer to wrap an entire MS-Windows software application through the simulation of a sequence of I/O operations. By this way, we need to write the new application logic as the job control program to call the wrapped application, and new user interfaces.

Sneed (Sneed 1998) had experiment on encapsulating the legacy system in five levels: job, transaction, program, module, and procedure. Similar to job and transaction level, our software wrapping is not to alert the source. But there are still little differences from Sneed's method. Sneed's job wrapper is a job control program. Compare with Sneed's wrapper in job level, our component wrapper is the glue between MS-Windows application and component interface. Sneed's transaction wrapper is applied on database system only, but our component wrapper is applied on MS-Windows applications with I/O simulation rather than limited to database systems.

3 JDSI ARCHITECTURAL STYLE

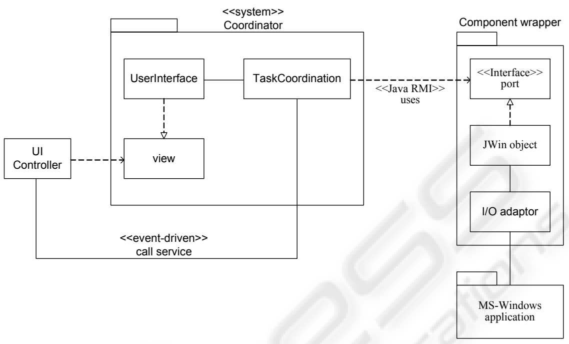

Figure 1: Structure of JDSI style

The structure of JDSI is shown in Fig.1. JDSI contains three main architectural components, *Component Wrapper*, *Coordinator*, and *UI Controller*. These components will accomplish the responsibility for three phases in JDSI respectively.

- *Encapsulation phase*. Lots of MS-Windows software products do not provide a programmable interface. A component wrapper has capability to provide a set of I/O interception/redirection functions to access a MS-Windows application (Lin 2004). The component wrapper also is used to encapsulate the wrapped COTS MS-Windows applications into Java objects.
- *Gluing phase*. JDSI style specifies a coordinator which actually plays the application logic that abstracts the functionality of this new software system. Coordinator is also a Java program controlling and interoperating with server-side MS-Windows software applications.

Interfacing phase. This integrated software system may commonly need a convenient operational user interface to the users. JDSI style adopts the concept of Model-View-Controller (MVC) model in designing user-machine interactive function.

3.1 Phase 1: Encapsulation Phase

The designated component wrapper defined in JDSI style (see Figure 2) has two participants to deal with encapsulation:

- ***I/O adapter.*** It supports I/O interception and redirection technique to transform a MS-Windows application's into a programmable component.

- ***JWin.*** This is an adaptee Java class that abstracts the functionality of the wrapped MS-Windows application. To deal with Java object encapsulation, JWin has a set of methods exposed on the interface and each method can invoke the corresponding procedures of I/O adapter.

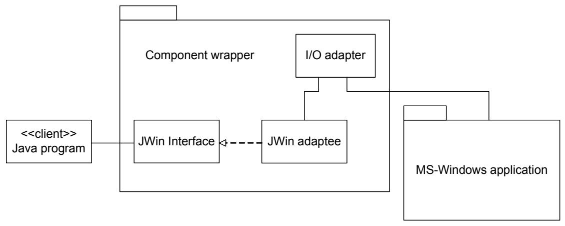

Figure 2: Structure of component wrapper

3.1.1 I/O Interception and Redirection

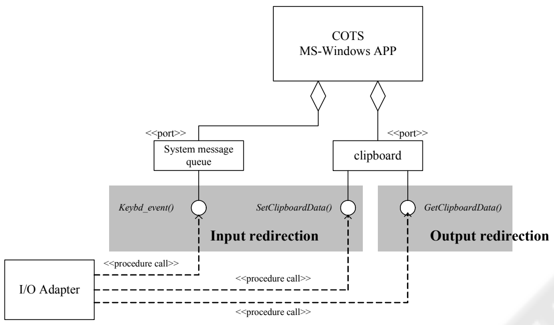

Figure 3: I/O interception and redirection

An MS-Windows application shown in Figure 3 could be regarded as a software component which has two ports including *System Message Queue* and *Clipboard Space*. Three important APIs provided in Microsoft SDK, *keybd_event()*, *SetClipboardData()*, and *GetClipboardData()*, are the procedures for other program's invocation.

For input interception/redirection, there exists a system message queue to store external events such as keyboard and mouse events. Once the keyboard is clicked on a window, this window becomes active and receives the program focus. A thread will receive event messages in system message queue only as this thread receives focus. The API, *keybd_event()*, provided by MS-Platform SDK can send keyboard events to an active MS-Windows software application. If the inputs are not device events, Microsoft supports clipboard mechanism used to intercept input data. Another API in MS-SDK, *SetClipboardData()*, can be used to intercept/redirect input data from I/O adapter.

For output redirection, the MS-Windows Clipboard mechanism is also applied. First, the I/O adapter triggers the wrapped COTS MS-Windows application to save the output data in the clipboard space. Then the I/O adapter acquires the output data from the clipboard space through the API, *GetClipboardData()*.

3.1.2 Java Object Encapsulation

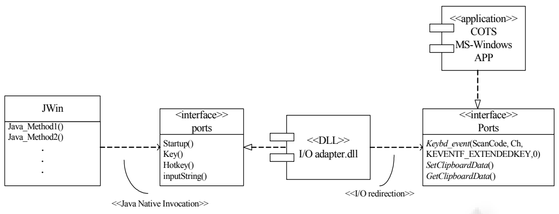

Figure 4: Collaboration of Java and I/O adapter in object encapsulation

JWin is an adaptee class which exposes the existing interface to abstract the wrapped MS-Windows software application. A set of methods on this interface would serve the functionality of the MS-Windows application.

In the encapsulation phase, the I/O adapter is firstly programmed in Microsoft Visual C++ language with MS-SDK APIs and formed into the Dynamic Linking Library (.dll) file format. This *I/O adapter.dll* provides procedures like *startup()*, *key()*, *hotkey()*, *inputString()* and so on as the imports. These import procedures would call the *keybd_event()*, *SetClipboardData()*, and *GetClipboardData()* to access the COTS MS-Windows application. Then JWin uses its Java methods to call the procedures provided in I/O adapter.dll as shown in the Figure 4 After these step, a COTS MS-Windows application seems to be wrapped and encapsulated in Java class.

Language incompatibility is however a problem in this step. Fortunately, Java provides JNI (Native Invocation) technique to invoke methods written in other programming languages, like C++.

3.2 Phase 2: Gluing Phase

To assemble the server-side wrapped MS-Windows applications, JDSI style defines a coordinator which has two responsibilities to deal with the integration. First, the designated coordinator is used to interoperate individual MS-Windows applications in performing user requests. Second, coordinator provides a user-interactive system for client users' manipulation.

The coordinator defined by JDSI is illustrated in Figure 5. Coordinator has two counterparts: *TaskCoordination* and *UserInterface*.

TaskCoordination is a Java class which abstracts the new integrated software system. TaskCoordination is also an application logic which exposes software services designed by the software integrator for client-side users, but these services are implemented by remotely calling the JWin's methods in the server-side.

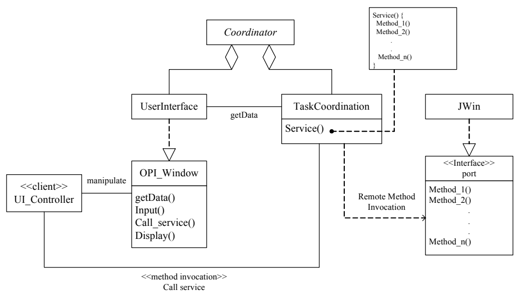

Figure 5: Structure of coordinator

3.3 Phase 3: Interfacing

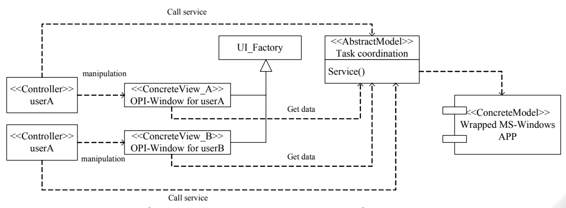
Call service
UI_Factory
<<AbstractModel>> Task coordination
Service()
manipulation
<<Controller>> userA
<<ConcreteView_A>> OPI-Window for userA
Get data
<<ConcreteModel>> Wrapped MS-Windows APP
<<Controller>> userA
<<ConcreteView_B>> OPI-Window for userB
manipulation
Get data
Call service

Figure 6: MVC pattern in JDSI

A new integrated software system may commonly need a operational interface such as GUI for client-side users. Due to the old user interface of the wrapped MS-Windows application is hardly separated. We prefer to build a new one for client users rather than reusing the old one.

JDSI style includes a Model-View-Controller (MVC) pattern into coordinator. UserInterface in coordinator provides user interface (i.e. OPI_Windows in Figure 5) for end-user's operation through UI_Controllers, such as monitor, mouse, keyboard, and other I/O devices. Each operation would call the services of TaskCoordination to request the sever-side MS-Windows applications.

According to MVC pattern, the UserInterface could be regarded as an abstract UI_Factory (see Figure 6), which declares an interface for operations that create concrete OPI-Windows for users.

The TaskCoordination in coordinator could be regarded as the "Model" component in MVC, because it implements the functionality of the integrated system. However, TaskCoordination is not a concrete Model, because each of services in coordinator is operated by remotely calling the server-side wrapped MS-Windows software applications.

4 CONCLUSION

JDSI style specifies a software wrapper for wrapping MS-Windows applications as Java objects. JDSI style could be applied to COTS-based system development and even for legacy system reengineering. The valuable design knowledge could help a software engineer to deal with the related problems when reusing COTS-based systems and legacy systems. Our current work is focused on how to promote the performance and robustness in JDSI style.

REFERENCES

Aversano, L., Canfora, G., Cimitile A., De Lucia, A., 2001. Migrating Legacy Systems to the Web: An Experience Report. *Proceedings of the 5th European Conference on Software Maintenance and Reengineering*, 148 –157.

Canfora, G., Cimitile, A., De Lucia, A., Di Lucca, G.A., 1998. Decomposing Legacy Programs: A First Step towards Migrating to Client-Server Platforms. *Proceedings of the 6th International Workshop on Program Comprehension*, 136 –144.

Cimitile, A., De Carlini, U., De Lucia, A., 1998. Incremental Migration Strategies: Data Flow Analysis for Wrapping. *Proceedings of the 5th IEEE Working Conference on Reverse Engineering*, 59 –68.

De Lucia, A., Di Lucca, G.A., Fasolino, A.R., Guerra, P., Petruzzelli, S., 1997. Migrating Legacy Systems towards Object-Oriented Platforms. *Proceedings of IEEE International Conference on Software Maintenance*, 122 –129.

Lin, J.M., et. al., "Reengineering Windows Software Applications Into Reusable CORBA Objects", *Journal of Information Software and Technology*, Vol.46, No.6, pp. 403-413, May 2004.

Schmidt, D., Stal, M., Rohnert, H., Buschmann, F., 1999. *Pattern-Oriented Software Architecture: Volume 2*. Reading, MA: Addison-Wesley.

Serrano, M., Carver, D., Montes De Oca, C., 2002. Reengineering Legacy Systems for Distributed Environments. *Journal of Systems and Software*, 37-55.

Sneed, H.M., 1998. A Case Study in Software Wrapping. *Proceedings of the International Conference on Software Maintenance*, (1998), 86-94.