从面向对象的遗留系统到面向服务架构的迁移方法①

伍晓泉 白 琳 魏 峻 (中国科学院软件研究所 北京 100190)

Towards an Approach for Migrating Object-Oriented Legacy System to SOA Environment

Xiaoquan Wu, Lin Bai, Jun Wei (Institute of Software, Chinese Academy of Sciences, Beijing 100190)

Abstract: Migrating legacy system with web service is an effective and economic way of reusing legacy software in a SOA environment. In this paper, we present an approach for migrating a three-tie object-oriented legacy system to SOA environment. The key issue of the approach is about services identification from large numbers of classes. And we propose a bottom-up method to model the system with UML and identify services from UML then. This approach can be a reference to an auto-migrating process.

Key words: legacy system; web service; service identification

1 Introduction

Service-Oriented Architecture (SOA) has become an increasingly popular mechanism for achieving interoperability between systems. Because it has characteristics of loose coupling, published interfaces, and a standard communication model[1].

Web service is a keystone of service-oriented computing. Through the web service technology, legacy systems can be remained and integrated with other business functions in order to get improved efficiency[2].

There are generally two strategies to migrate legacy software into SOA environment. One is black-box strategy. It treats the legacy system as a whole and takes it as a part of the universe architecture. A common solution for that is to integrate legacy systems via service adaptors[3]. The role of the adaptors is to hide the complexity of calling backend functions which typically involve communication through proprietary protocols. Another solution is to provide the runtime environment for the legacy system. The communication between the legacy system and its requestor is mediated by the services deployed within the runtime environment[4]. The adventage of black-box strategy is that only the legacy interface is analyzed and the internals are ignored. But it is a short-term solution with high price of maintenance and management[2]. And it is not compatible when the internal logic needs to be changed for new requirements.

The other strategy is white-box strategy. We should analyze the architecture of the legacy system, extract useful code segments from it, identify services, and then migrate legacy code to new systems. The key is how to identify services from legacy code. In general, there are three approaches from the perspective of service identification process: top-down, bottom-up, and bidirectional[5]. In top-down approach, services are identified from the business rules. After the services have

① Supported by the National Grand Fundamental Research 973 Program of China under Grant No.2009CB320704; the National High-Tech Research and Development Plan of China under Grant Nos.2007AA010301, 2007AA01Z134

been defined, useful legacy code is migrated to new architecture [6].

Bidirectional process starts from both legacy system and application domain. When analyzing the application domain, a domain model can be built with UML and business functions identified. Then legacy system model is also built by the aid of converse engineering. Legacy system is reengineered by comparing these two models.

In bottom-up approach, services are identified only by analyzing the legacy systems according to the principle of low coupling and high cohesion. If the software entity is independent, self-contained, coarse-grained and loose-coupled, it turns to be a candidate service.

Bidirectional and bottom-up approaches are both invasive to legacy system, and different legacy system needs different modeling methods. Legacy code analysis is one of the most important steps in these approaches. It is troublesome to do it manually, especially for large legacy systems. Compared with bidirectional approach, bottom-up approach can be less human intervened, because all the information about the services comes from the legacy code. We argue this approach in this paper, and apply it to object-oriented legacy system.

2 Related Work

In Ref.[7], the author reports on the development of a methodology for the evolution of software towards new architectures. In this approach, it represents source code as graphs. This enables the use of graph transformation rules, allowing the automation of the transformation process. Prior to its model representation, the source code is subject to a preparatory step of semi-automatic code annotation according to the contribution of each of its parts in the target architecture. This paper first describes the overall methodology and then focuses on the code annotation and model transformation parts. It is a universal methodology and it doesn't refer to target architecture and specified technology.

In Ref.[8], the paper presents a concept framework for migration of legacy systems towards Service-Oriented Architectures. This consists of four steps:

- Code annotation categorizes
- Reverse engineering
- Redesign
- Forward engineering

First, block the source code according to the different elements of the target architecture they will be mapped to, then obtain a graph representation of the annotated code; use graph transformation rules to achieve the target architecture and then generate the target code. But it doesn't distinct the source code which will be mapped to different elements, like UI, business logic and data.

Ref.[2] proposes a reengineering approach which applies an improved agglomerative hierarchical clustering algorithm to restructure legacy code and to facilitate legacy code extraction for web service construction. It is bidirectional. A dendrogram is obtained after executing the algorithm on legacy code.

3 A Bottom-up Approach for Migrating Object-Oriented Legacy System to SOA

In this bottom-up approach, we first analyze the architecture of the legacy systems, extract useful legacy code segments, model the legacy systems with UML class diagrams, and then identify services from UML diagrams. At last, we migrate the legacy code to web service environment with glue code. In this section, we describe the approach in steps.

In the object-oriented software, classes can be classified into five categories[9]:

- User Interface (UI) class

Encapsulate the elements, such as HTML pages, GUI screens, and printed/electronic reports that make up the user interface for your system.

- Business/domain class

Also known as entity classes, implementing the fundamental domain types within your application.

- Process class

Implement complex business logic, which can contribute to a potential service that pertains to several classes.

- Persistence class

Encapsulate access to your persistent stores,

including relational databases, flat files, and object bases.

• System class

Encapsulate technical features, such as your approach to inter process communication or error logging.

These classes can be mapped to a three-tier software architecture: UI, DATA, and LOGIC.

UI	User Interface (UI) class
DATA	Persistence class
LOGIC	Business/domain class
	Process class
(System dependent)	System class

System classes are platform-dependent. It is always related to the runtime environment of the software and is out of the scope of this paper.

We will discuss our migrating approach based on the left three tiers: UI, DATA and LOGIC.

3.1 User interface tier

For some complex UI system, such as GUI screens, UI is platform dependent. It will invoke operating system API, and create the windows graphs in local. So we can extract this part as the service requestor.

Another type of UI is web page. Various different technologies can be used in creating web pages. Some of these are: Hyper Text Markup Language (HTML), Extensible Markup Language (XML), Cascading Style Sheets (CSS) and Synchronized Multimedia Interaction Language (SMIL). Dynamic web pages can be using Java Server Pages (JSP), PHP or Active Server Pages (ASP). Extensible Style sheet Language for Transformations (XSLT) can be used for translating XML documents into other languages (like HTML or WML) or into other XML dialects. Multimedia and graphical content can be presented using Scalable Vector Graphics (SVG), Synchronized Multimedia Interaction Language (SMIL), Virtual Reality Modeling Language (VRML) or various other open standard based technologies. Some property formats are also commonly used, such as Flash and Shockwave by Macromedia Inc.

For these, we can also wrap it with web service. From the technology perspective, the idea is to identify the common control element, such as button and textbox, describe the layout of the control elements as XML forms, output it with web service, and then resolve it in local.

3.2 Data tier

The data tier stores long-term data, such as information on users, bank accounts, inventory information-typically mainly real world data. This data is stored normally in a Relational Database Management System (RDBMS) on its own server. In some instances, other data storage solutions can also be used such as fileservers, complete legacy systems or other remote systems. This tier may be implemented in one of several ways, including but not limited to hardcode SQL, data access classes such as Java Data Objects (JDOs), a service such as EJB's container-managed persistence (CMP), or one or more Web services. Take the view of web services, this tier need to handle with considerable amount of data, but the operation is relative simple, only containing Insert, Delete, Modify, Query and so on.

To wrap simple data source with web service, we should describe the data structure as XML forms, and exposure Insert, Delete, Modify and Query as the operations of the service.

But for multi data source, we should consider how to manage and access that data from inhomogeneous data source. Often the middleware is introduced to encapsulate data sources and mediate between them and the middleware [10].

Besides we should also take data coherence, security and performance into account.

Data coherence means we should use the legacy data without destroying it when other applications are handling the same data.

Data security means we should decide who can access the data. Legacy system often certifies the role internally. After wrapping it with web service, we may not only certify the role in service, but also identify the role who can invoke the web service. We can also encrypt the soap message to make it security in data transfers.

Because of the low performance of xml transfer, when handling with large amount of data, we should consider the performance, too.

3.3 Logic tier

This tier contains most business rules and often the

most complex parts of a software system.

Before the migration, we should assess the legacy systems. The assessment reveals the current status of a legacy system and specifies which phase it is in its lifecycle.

We first build the diagram description of the program in UML by converse engineering and extract useful code segments and discard dead code. And then identify the service by analyzing the contract of classes and domain packages. At last, we migrate the legacy systems to web service in the source-to-source level.

3.3.1 Source code decomposition

Because not all the software can be clearly divided into the five types of classes referred above, we should tidy the legacy code first. For example, separate business logic from presentation logic, and then categorize the source code according to the different elements of the target architecture they shall be mapped to.

3.3.2 System modeling with UML

The second step is to model the legacy software. The advantage of obtaining a graph representation of the legacy code is that graph transformation rules can be used to identify the services. UML is the universal language for modeling an object-oriented system. Some software tools can convert source code to UML class diagram automatically. The UML diagram presents the system more intuitionist than source code. From the model of the legacy systems, we can extract the useful classes which need to be migrated to web services, and omit other classes in the diagrams.

3.3.3 Service identification

In Ref.[9], Ambler proposed a method for deriving web services from UML models. Ambler's method is an example of a bottom-up web service design approach that defines web services on top of existing components or objects and is useful for migration to service-oriented architectures. The main focus of this method is grouping highly coupled classes into coarser components called domain packages, and refining the resulting component interfaces to produce larger grained services that are exported as web services. The process of identify contract and domain packages is manual. We use his definition of contract, and import graph theory to analyze the UML class diagrams and identify the services semi-automatically.

A contract is any behavior of an object that other objects can request. In other words, it is an operation that directly responds to a message from other classes.

In the UML class diagram, classes can be seemed as vertexes, and the lines between classes are the edges.

• Simplify hierarchies

For the sake of identifying services, inheritance and aggregation hierarchies can often be simplified. If we only take inheritance into account, the class diagram will contain many separate trees. Analyze the public functions of the classes. If they don't introduce a new contract, they will be ignored. And sometimes the tree can be seened as a single vertex. For aggregation hierarchies, it is an undirected graph. If the sub graph contains nothing about business rules, it should be ignored as "part classes".

• Identify class contracts

For classes, contracts define the external interface, also known as the public interface of a class. You can ignore all the operations that aren't class contracts for the sake of simplicity, because they don't contribute to communication between classes distributed in different packages

• Identify potential domain packages

In the graph theory, the outdegree and indegree of a vertex represent the relationship with other vertexes. And in the class diagram, more lines between classes mean higher cohesion. We could separate the classes into different packages, and minimize the edges connected with other packages. One of the key goals is to organize the design into several packages in such a way as to reduce the amount of information flowing between them. At the same time, we should modify the partition to avoid reasonless partition.

• Define domain-package contracts

Domain-package contracts are those class contracts that are accessed by classes outside of a domain package. It's important to make sure that contracts of a package are cohesive, that is, it makes sense to put them together. If the contracts do not make sense as a group, then you should distribute them into multiple packages.

• Define services

The domain packages and sub graphs are all the candidate services. To define services, we should first

analyze the domain-package contracts, and then take the proper contracts as the operations of a service. You can also combine and split, and make the service definition more reasonable.

3.3.4 Code migration

Code migration is a source-to-source transformation. The aim of this process is to extract interface of WSDL (Web Services Description Language), which is used to describe the web service and wrap parameters of the operations with XML data. And glue code should be designed to combine some interfaces as an operation.

Now there are many tools which can generate WSDL file from class file or COBOL automatically, but usually the information is required to come from only one file. So you should write a new class, and invoke the operations identified in this class.

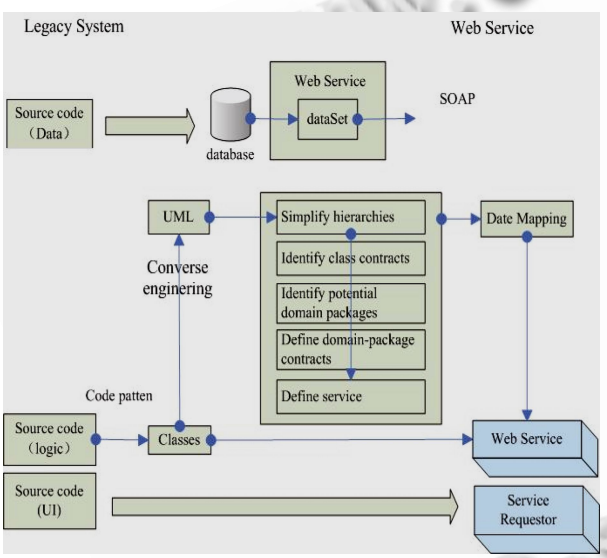

Fig.1 The framework of the wrapping method

The general approach is shown in Fig.1. We first use converse engineering to model the legacy software, then analyze the class contract among the class from UML diagram. After that, we identify services with the aid of domain packages and wrap them in source code level.

4 Case Study

OnceBPD is a business process designer. Users can draw BPMN diagram with it. It can also translate BPMN files to BPEL files and deploy the process on a BPEL engine. It is developed with JAVA language in Eclipse platform. We migrate it to a SOA architecture and take it as an example of applying and validating the whole method.

The architecture of this software includes two tiers: UI and Logic. First we separate user interface from business logic part. We extract the function of drawing BPMN diagram and take this part as the client to invoke service. Then wrap the other part as web services.

Then we model the business logic classes with UML class diagram. We use eclipse plug-in Agile[11] to build the UML class diagrams automatically. The diagram is shown in Fig.2. After reorganizing the classes in the diagram, we can find three sub diagrams. (Some operations executed by eclipse do not appear in the diagram.) It shows that the three parts are low-coupled and can be viewed as service candidates.

Now we just take one of the sub diagrams to practice the other steps which is selected in Fig.3. To make it clear, the operations of the classes are ignored.

We enlarge the selected sub diagrams in Fig.4. The “part classes” WSClient can be omitted. After reorganizing, we can clearly see that the classes named Mapping, NSStack, Utility and FileReader form a class hierarchy, which can be treated as a single class.

The next step is to identify class contracts. They are usually the public operations which are invoked by other objects. The lines in the diagram show the relationships between classes. And we could identify domain packages according to the lines, because they represent the cohesion and coupling between classes.

We can separate the classes into two domain packages shown in Fig.5. Two services Deploy and ParseWSDL are identified. The operations of the services are listed as follows:

Deploy	deploy
	undeploy
PraseWSDL	getPortTypes
	getPortType
	getParterLink
	……

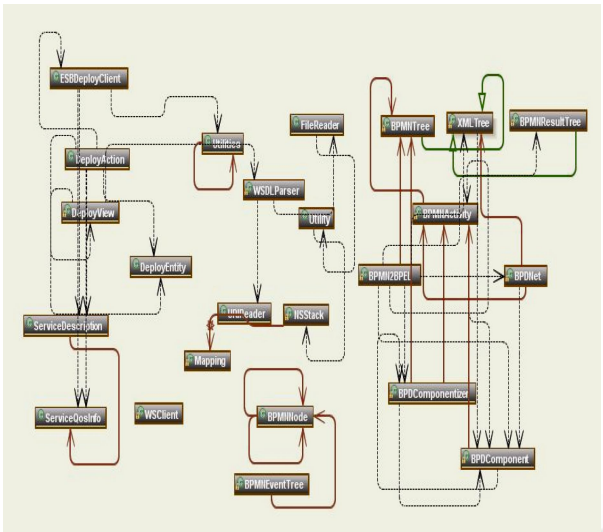

Fig.2　Auto-generated UML class diagrams of OnceBPD

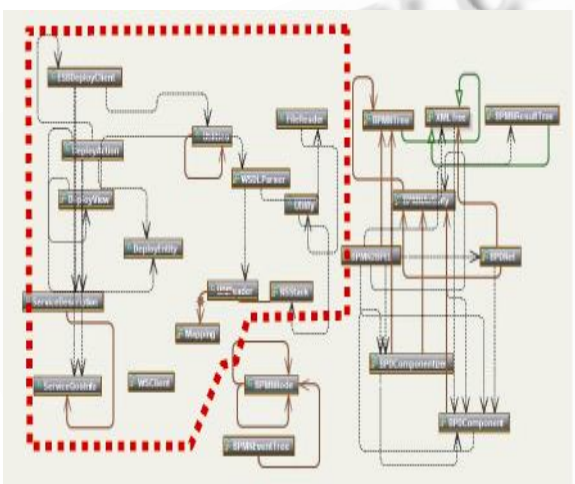

Fig.3　Select one of the sub diagram

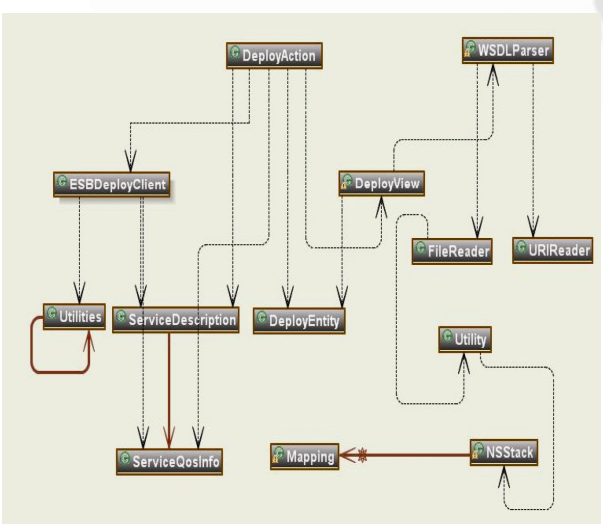

Fig.4　Reorganized sub diagram

At last, we transform the source code to the new architecture. Create a new class for each service and create the identified operations to invoke the related operations in other classes. This class is built for the convenient of generate WSDL with software tools. We examine the parameters of the operation and present them with XML form. Then we create the WSDL description of services with axis2 java2wsdl [12].

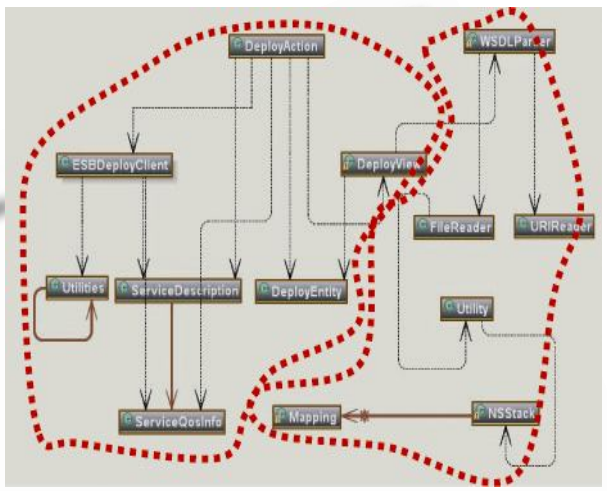

Fig.5　Domain packages

5　Conclusion and Future Work

In this paper, we presented and implemented an approach for migrating object-oriented software with web services in a bottom-up way. When analyzing the architecture of the legacy systems, the program can be mapped to different elements like UI, data and logic tier. We can migrate them to web service environment separately. Now the process can be finished semi-automatically with the aid of software tools in the phases of modeling legacy systems with UML diagrams and generating WSDL files.

In this approach, we take cohesion and coupling into account to identify services. By the help of graph theory, the process can be realized automatically. At least a candidate solution can be given. The advantage of the approach is that it can be an effective assistant when the scale of the system is too large to cope with.

But there are still many difficulties in realizing the

process in a fully automated manner such as how to identify the operations of the service exactly, how to convert parameters to XML forms, how to generate source code automatically and so on.

Another problem is how to apply this method universally to more legacy systems with various types of architectures. We will try to solve these problems in the future.

References

1 Balasubramaniam S, Ed Morris GAL, Simanta S, Smith D. SMART: Application of a Method for Migration of Legacy Systems to SOA Environments. Heidelberg SB. ICSOC 2008. Springer Berlin Heidelberg, 2008.

2 Zhang Z, Yang H. Incubating services in legacy systems for architectural migration. Software Engineering Conference, 11th Asia-Pacific, 2004.

3 Bhattacharya S. Integrate legacy systems into your SOA initiative. http://www.ibm.com/developerworks/webservices/library/ws-soa-legacyapps/S_CMP=cn-a-aix&S_TACT=105AGX52.

4 Bali B, Bubak M, Wegiel M. A Solution for Adapting Legacy Code as Web Services. Component Models and Systems for Grid Applications, 2005:57－75

5 Nuffel DV. Towards a Service-Oriented Methodology: Business-Driven Guidelines for Service Identification. Meersman R, Tari Z, Herrero P. OTM Workshops(1).Springer:294-303.

6 Linthicum DS. Next Generation Application Integration: From Simple Information to Web Services. AddisonWesley Longman Publishing Co., Inc, 2003:512.

7 Correia R, Matos C, Heckel R, El-Ramly M. Architecture Migration Driven by Code Categorization. Software Architecture, 2007:115－122.

8 Matos C. Service Extraction from Legacy Systems. Graph Transformations, 2008:505－507.

9 Ambler. Deriving Web services from UML models, 2002.

10 Roth, M. T. A Wrapper Architecture for Legacy Data Sources. http://www.almaden.ibm.com/cs/garlic/vldb97wraprj.ps.

11 http://www.agilej.com/

12 http://ws.apache.org/axis2/