Creating Web Services from Legacy Code

Vinay Goyal
Professor (CSE)
Jind Institute of Engineering & Technology,
Jind. (Haryana)

Amit Jain
Research Scholar (Computer Science),
Teerthanker Mahaveer University,
Moradabad. (Uttar Pradesh)

ABSTRACT

Over the past two decades, lots of people have forecast that legacy systems would soon be a craze of the history. In bare disparity, companies are now apprehending greater repayment from their legacy systems as they tie them to distributed systems. However, companies face quite a lot of major disputes in managing their legacy systems in these rapidly changing environments. The role of web services in migration of legacy systems to service-oriented architecture is of extreme importance for research field. Web Services are software system designed to support interoperable machine-to-machine interaction over a network. It has an interface designed in a format that systems can work upon. In this paper, an emphasis has been laid onto the study of web services and their role in context to extraction of several components from legacy systems.

Keywords

WSDL, SOAP, W3C, Packaging, Assembling.

1. INTRODUCTION

Today's computing requires promising services accessible as promptly as probable but this is not so simple to attain since vast quantity of functionality is somehow masked in billions of dollars value of existing code. Web services standards—including SOAP, WSDL, UDDI, and BPEL—are based on the readily and openly available internet protocols XML and HTTP, and thus are easier and cheaper to be apprehended.
Web Services, along with a group of protocols, has become de facto incorporation expertise for apprehending service-oriented computing. Simple Object Access Control Protocol (SOAP) is generally used as the communication protocol. Web Services Description Language (WSDL) has become the de facto standard for unfolding the boundary of a web services. And further the concept of Universal Description, Discovery and Integration (UDDI) is related to defining a directory service for web services. UDDI enables web service clients to locate candidate services and discover their details. Service consumers and service providers utilize these standards to perform Service-Oriented Architecture's basic procedures.
Web Services Description Language (WSDL) is a layout for relating a web services interface. It is a way to explain services and how they should be related to precise network addresses. The price for that is modeling the evolution of legacy systems to the WSDL interface, setting the parameters according to the interface specification [1].
WSDL has three parts:

- Syntactic Description
- Operations
- Service Bindings

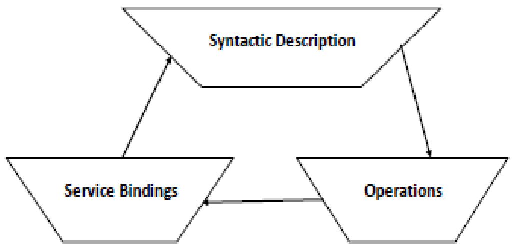

Fig 1. Components of WSDL

Syntactic Description is usually done by using XML and comprises both of data type description and message details that make use of the data type descriptions.
These descriptions are usually based upon some contracted XML terminology. This conformity could be in existence within an organization or between organizations. Terminologies within an organization could be intended purposely for that organization. They may or may not be based on some industry-wide terminology. If data type and message descriptions need to be used between organizations, then most likely an industry-wide terminology will be used.
Processes describe procedures for the messages supported by a web services. There are four types of processes:

- One-Way: Message sent without acknowledgement.
- Request/Response: The sender sends a message and gets acknowledgement.
- Solicit Response: A request for a response.
- Notification: Messages sent to multiple receivers.

Processes are grouped into port types. Port types define a set of processes supported by the Web service. Service bindings join port types to a port. A port is defined by relating a network address with a port type. A collection of ports defines a service.
This binding is commonly created using - SOAP which provides the envelope for sending web service messages over the internet. It is part of the set of standards specified by the W3C - The World Wide Web Consortium, which is involved in developing interoperable technologies (specifications, guidelines, software, and tools) and serves as a forum for information, commerce, communication, and collective understanding.

2. ARCHITECTURE OF WEB SERVICES

The architecture for web services provides a framework that can be represented with more powerful illustrations and practices taken from recognized computer science approaches. A set of services can be poised to form another service called amalgamated service that represents a selected business process.
Presently, a business process using web services can be depicted in Business Process Execution Language (BPEL)

and can be executed by the assistance of business process engine.

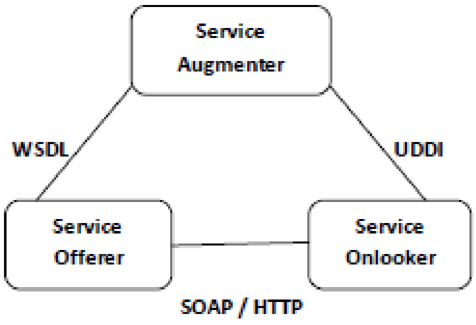

Fig 2. Architecture of web services

There are several set of steps which are from a part of Web Services. The UDDI registry is intended to eventually serve as a means of "discovering" Web Services described using WSDL. The Web Services Description Language (WSDL) forms the basis for Web Services. SOAP essentially provides the envelope for sending the Web Services messages. SOAP generally uses HTTP, but other means of connection may also be used. In the above fig 2, a layout of various procedures alongwith the services and description language has been shown. The depiction of a business process and its implementation decreases the information technology space between business and software professionals.

3. FRAMEWORK FOR EXTRACTING WEB SERVICES

There are three fundamental steps essential to produce web services from legacy code.

- Recovering the legacy code.
- Packaging the recovered code.
- Assembling the code as a web service.

Recovering the Legacy Code

To be able to recover code from an accessible legacy code base, it is prerequisite essential to situate that code and to conclude if it is appealing to be reused. At times, there is no issue in scrutinizing and appraising the code of a few miniature programs. That can be done by any programmer well-known with the code using an easy text editor. It is relatively dissimilar to investigate numerous programs in search of a few reusable chunk of code. Also domain expertise is required, but he/she should be supported by computerized reverse engineering tools. The method to determine the business policy is the outcome which they generate. By recognizing the variables which are returned by the functions dispensing the business rules, one can also classify the functions.

If the code was written in structured manner and then it would have been easy to assign it to single block of code. But the actual scenario is somewhat different from the usual. It is altogether scattered as business functions in several blocks of code. There is also a possibility that an individual block of code may be involved in executing numerous business functions. So in that situation there exist a close association between the code structures and the business rules.

By having a data flow analysis based on the final outcome, it is possible to outline the outcome back in the course of all of the statements, which put in the effort toward fabricating it. Thus, on recognition of the instructions, it is easy to locate what exactly is stored in code blocks, i.e. the procedure, paragraphs, subroutines, etc., they are interlinked with the variables they relate to. This technique is known as Code Slicing. It is originally used in testing to validate the path leading to a given outcome. Nevertheless, it relates uniformly to the assignment of eliminating business policies. The indispensable point is that a business regulation is defined as an algorithm for calculating a given outcome. There may also be several outcome formed in diverse places, for example an order inventory process which not only verifies the execution of that order, but also renews the quantity of the item ordered and produces a billing arrangement and a shipment order. To extract the code for processing an order, it would then be essential to recognize all of the data objects formed as an outcome of that dispensing. The next step after recognizing the code is to mine that code and to rebuild it as a split module with its own interface. This is done by copying the effective code blocks into a common structure and by keeping all of the data objects they relate to into a familiar data interface.

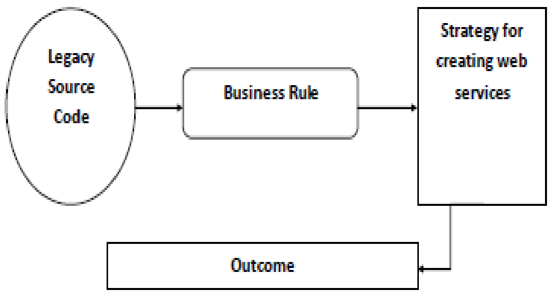

Fig 3. Layout for migrating Source Code

In C, the interfaces are arguments of the type structure and whereas in COBOL the objects are level 1 item in the linkage section and in PL/I the objects are related data structures with the pointers to them as arguments to the main process. The final outcome will be a subroutine with a call interface. Thus, the business logic code will be rearranged into a self-contained subprogram. This is a prerequisite to packaging it.

A practical consequence of this code reengineering procedure is a citation of the accessible business rules. For every data outcome of a particular use case, the conditions, assignments, computations and I/O procedure is vital. Also, to determine that outcome an outline of a data flow tree is established. The concluding outcome builds the root node of the tree. The other nodes are the parameters and transitional variables which flow into that ending outcome. The branches of the tree correspond to the state transitions, which are activated by conditional statements such as if, switch case and loop statements.

Thus, a business procedure is a connection of control and data flow, it is indispensable to portray both view point. With the help of this illustration, it becomes feasible for the user to choose whether an accessible rule, realized within a legacy system is worth reclaiming it as a general function in a service-oriented architecture arena.

Packaging the recovered code

Once a business regulation has been found, acceptable and creditable of reclaiming, the next step is to package it. The objective of the packaging phenomenon is to provide the constituent extracted from the legacy code with a WDSL interface. The method used is to alter every entry into a process and to change each argument into an XML data constituent. The data structures will become multifaceted rudiments with one or more sub-elements.

The processes will have their parameters and outcomes as indication to the data element descriptions. Both the technique and the argument will be developed into an XML representation with a SOAP framework. With the help of a certain tool the automation procedure is being conducted for transformation in relation to the various programming languages.

Besides, creating the WSDL interface description, it also improves the packaged component with two extra modules. One module is for parsing the inward message and extracting the data from it. This data is then assigned to the equivalent parameters in the packaged module. The other module is for forming the return message from the outcome generated by the packaged element. In this way, the legacy code can be reprocessed as a web service without having to transform the code.

The two generated subroutines act as a link between the WSDL interface and the call interface of the original code. This is a prerequisite to packaging it [2]. The idea is to evade manual exploitation of the legacy code, since manual intrusion is not only expensive, but also error prone. To be effective packaging must be automated.

This simple fact has been acknowledged for offering packaging solutions for entire programs and databases. Nevertheless, the logic they contain can be reused, but only if it is extracted from the novel context and converted into another web compatible one format.

As of now, service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Assembling the Code

The last step is assembling the code as web service, so as to link the web service to the overlying business procedures and protocols. This is made possible by means of a proxy component. The business process work flow actually calls upon the proxy which is accessible in the same address space as the process definition.

On the application server, there is a scheduler, which receives the SOAP message & concludes which web-service is to be performed and forwards the WSDL contents to that particular service, in fact into the packaged legacy code. The packaging of the code parses the XML input data and moves the data to the suitable addresses in the packaged component.

Once the packaged component has been implemented, its outcome is converted by the package into an XML output data structure, which goes back to the scheduler to be transmitted back to the web client. Thus, in this way the business process can be processed on any client anywhere and still is able to access the legacy functions on the application server.

4. ISSUES RELATED TO REUSABILITY

High performance is an issue with web services. There is a significant time penalty to be paid for in interpreting the business process language and in sending messages to and from the application server. Parsing the WSDL requests and responses is only one of the many time consuming bottle necks. The conversion of the WSDL interface to the local language interface and back is a minor problem compared to the total performance issue.

Relative to interpreting the BPEL, marshalling and dispatching the messages and parsing the WSDL interfaces, it is insignificant. It must be seen that the dynamic binding of business processes to distributed web services is in itself a resource consuming process with or without packaging [4]. Thus, it can be concluded that using packaged web services is more efficient than using non-packaged web services [3].

5. CONCLUSION

Web Services offered within the framework of a service-oriented architecture promise to make applications more flexible, easier to compose and cheaper to develop [5]. In this paper it has been shown and discussed, how legacy code can be reused to help develop web services. It would be unwise to ignore the vast amount of proven legacy software available within corporations and public administrations, when migrating to a service-oriented architecture is a feasible solution. There are three ways of doing this.

One is to reverse engineer the code and to re-implement the algorithm in another language. Another is to package the executable code and to access it via the existing interface. The third alternative is to transform the source code as has been described here. This third alternative is to be recommended when the code is sufficiently independent of its environment and the cost of reimplementation is also not too high. The technology for doing so is available. Doing so avoids the cost and risks of having to develop them from scratch. The savings is the difference between the cost of recovering and packaging the legacy functions as opposed to the cost of designing, coding and testing. It promises to be significant. The main problem has turned out to be reentrancy. The state of the data contained within a packaged web service is that of the last caller. Thus, if different processes are using the same service, their data will be mixed.

One solution is to store the internal data state in a temporary database under the identity of that user. The other solution is to have a scheduler. Both solutions have advantages and disadvantages. However, this is not a problem specific to packaged legacy code, but to all web services. It has to be solved in order for this technology to be accepted widely.

6. REFERENCES

[1] Lavery,J./Boldyreff,B./Ling,B./Allison,C.: Modelling the evolution of legacy systems to Web-based systems , Journal of Software Maintenance and Evolution,Vol.16, Nr. 1,2004, p.5.

[2] Sneed, H.: Extracting Business Logic from existing COBOL Programs as a Basis for Reuse , Proc. of 9th IWPC-2001, IEEE Computer Society, Toronto, May, 2001, p. 167.

[3] Manoj, T./Redmond, R./Yoon, V./Singh, R.: A semantic approach to monitor Business Process Performance , Comm. of ACM, Vol. 48, No. 12, Dec. 2005, p. 55.

[4] Zou, Y./ Lau, T./ Kontogiannis, K.: Model Driven Business Process Recovery , Proc. of 11th WCRE-2004, IEEE Computer Society Press, Delft, N.L. Nov. 2004, p. 224.

[5] Jones, S.: Towards an acceptable Definition of Web Services , IEEE Software, May 2005, p. 87.