Adapting legacy applications as web services

Dietmar Kuebler
Wolfgang Eibach

January 01, 2002

With the advent of dynamic e-business and its open standards-based supporting technologies, valuable legacy applications that support essential business processes in enterprises can join this new area of distributed computing. In this paper, we provide an overview of how legacy applications have been enabled for applying web services technology. An example then illustrates the use of the conceptual architecture for the legacy application.

Introduction

Web services are the next stage of evolution for e-business. They view everything as a service that can be dynamically discovered and orchestrated, using messaging on the network. Enterprises can dynamically sell their services to others by publishing their web services.

The web services architecture (WSA) is an ideal technology to incorporate enterprise legacy applications into this new area. This is the next step in creating a Web-browser-based user interface that interacts with an enterprise application in order to enable program-to-program communication. This implies doing the same business in a more automated fashion and with a more efficient approach.

Since the beginnings of e-commerce, many applications have been developed to support the growing demand of online shopping systems. Many of the systems use proprietary implementations, often because of the lack of open standards at the time of development. However, the business processes these earlier-generation proprietary applications support will continue to be necessary in the future. This can be credited to the significant investments that have been made into these e-business applications.

These legacy applications are more complex than you might expect in the first place. For example, a value added tax (VAT) calculation -- similar to sales tax calculation in the US -- becomes much more complex if it supports trade situations around the globe with all the different conditions that have to be handled.

In order to enable legacy applications to participate in dynamic e-business, we need to apply web service technologies which will allow the services to be defined so as to hide some of the complexity of the legacy application interface.
Once the applications have been technically adapted, the business processes are likely to evolve to a more automated business process with reduced human intervention.

**Revitalizing legacy applications**

The concept of web services, together with the service-oriented architecture (SOA) approach, creates an opportunity for legacy applications to be made available for the Web. In this article, we will show how a conceptual architecture can be applied to many of today’s legacy applications running on mainframes and other servers. This includes applications that under the control of mainframe transactions managers, that is, the Customer Information Control System Transaction Server (CICS TS) or Information Management System (IMS). (See Related topics.)

This concept has many advantages:

- It can make proven production level applications available as web services. This allows for a rapid deployment of stable web services to customers of a web service provider.
- The high investment in large legacy applications is preserved for users of the web service, while still available unchanged as a server application on the mainframe host.
- It can leverage the advantages of running mainframe-based applications to the web users, such as:
  - **High scalability**: The number of potential users for a web service can often not be determined when the service is started. A mainframe such as the zSeries Parallel Sysplex cluster-based service is dynamically managed by the Workload Manager (WLM) and can scale to a very high degree. (See Related topics.)
  - **High availability**: Web services must be accessible for business-to-consumer (B2C) and business-to-business (B2B) users with a 99.99% availability at 24x7x365. A mainframe cluster can be setup with no single point of failure that ensures this level of availability.
  - **Continuous operation**: Mainframes can allow for scheduled system outages for software or hardware upgrades without any interruption to the applications running on the cluster. Additional systems can be brought in without any interruption to the operation.

**Conceptual architecture**

Our paper first shows a conceptual architecture that can be used as a template to make legacy applications that are already in production available as web services. Figure 1 shows the structure of this architecture, and the following description explains the components in detail.

Later in this paper we will show an example of a real implementation of this concept.
The conceptual architecture shown in Figure 1 covers the three components of a web service: service provider, service request, and service broker. The service requestor is the consumer of a web service, and is most likely a program running on the server of a business customer. The service provider program gets the information about available services from a UDDI repository, the service broker. The available web services are described in the Web Services Description Language (WSDL) to support platform-neutral communications.

When the service requestor has selected a service, it will use the WSDL description to find out how to access the service. Once found, the WSDL description is used to generate a SOAP request message that is sent to the application server, which acts as the service provider.

This SOAP request is issued as a HTTP POST request in above example. Using HTTP has the advantage that it can pass the normally available firewalls on web servers. If a requirement exists for more reliable messaging as provided by HTTP, you can use a reliable message queue system such as MQSeries Internet Passthrough (MQ IPT), or Oracle Application Server. MQ IPT also runs on top of the HTTP protocol and can therefore pass through firewalls (see Related topics). However, it also provides all the advantages which MQ messaging brings to applications.

The SOAP request message after passing the firewall is handled by the HTTP server. This server analyses the HTTP header information, and finds as part of the Uniform Resource Name (URN), the name of the SOAP router component. The request message is passed to the SOAP router specified. The SOAP router analyses the HTTP header, and finds the location of a web service adapter. The router will then pass the request to the requested adapter.

**Web service adapter**

The web service adapter has to be developed for each service to enable them for services access. It is typically a Java application program that connects to the backend server. This connection can be any communication link supported by the backend server, that is, it can be a native MQSeries channel running on a Transmission Control Protocol/Internet Protocol (TCP/IP) or on an Advanced
Program to Program Communication (APPC) protocol connection. Other options to connect to the backend system are to use the CICS TS External Call Interface (ECI) if the backend system uses the CICS TS transaction system.

The web service adapter may call one single backend application per SOAP service request. It is also possible to generate complex web services where one SOAP request leads to multiple requests to one or more backends server systems. The combined results of the backend requests are composed into one SOAP response, which is then passed back to the service requestor.

The web service adapter hides the complexity of calling a backend function, and requires only for you to specify the subset of parameters absolutely necessary to execute a service.

**Transaction management**

A complex business component contains multiple transactions to be performed. These transaction requests may be performed to one or more backend systems. It is an important business requirement that all of these backend requests must belong to one main transaction, known as a logical unit of work (LUW).

An LUW must conform to the requirements defined for transaction systems, commonly known as the ACID paradigm:

- **A** for atomicity: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **C** for consistency: A correct execution of the LUW must take all involved resources from one consistent state to another.
- **I** for isolation: All updates made inside an LUW should not be made visible to other transactions until the LUW is committed.
- **D** for durability: Once the updates are made inside a LUW are committed, these changes must never be lost because of subsequent failures.
- Finally, rollback is also considered an important property that can undo all previous executed calls.

Control of the execution of an LUW can take place on different locations of a distributed system. One possibility is to place it at the client location, which would be the service requester in case of the web services architecture. This is a problem today, because no solutions to control an LUW via an Internet connection are available. In the proposed conceptual architecture presented in this paper, the control of the LUW is in the web service adapter. This solution makes use of the transaction control extensions of the Enterprise JavaBeans (EJB), and uses the backend system transaction managers to control the changes made to resources belonging to those systems.

**Figure 2** shows a variation of the conceptual architecture which makes use of Java 2 Enterprise Edition (J2EE) and Enterprise JavaBeans (EJB). In addition, a transaction resource manager is added on the backend server. Typical transaction managers for mainframe backend servers are the CICS TS system or IMS.
Making use of the transaction coordination capabilities of EJBs, the control of many single requests to transactions on the backend server or servers can be grouped together into an LUW.

When the web service adapter, if implemented as EJB session bean, receives a SOAP request for a complex service, it starts a unit of work. Then, the transactions on the backend systems are invoked by sending middleware messages. These messages may be issued to one ore more backend systems. If any response from an invoked backend transaction ends with an error, the session JavaBean will initiate a rollback for all transactions started on the different systems. This guarantees either a complete execution of the SOAP service request, or the state is as it was before the request was issued. This avoids any inconsistent states on the databases of the backend legacy systems.

**A sample implementation**

To verify the architecture described here, a sample has been developed. It uses an already available backend host application running under IMS transaction manager. This system has various services offered today via a proprietary interface to subscribers.

For an example we implemented the VAT calculation used in Europe implemented as a service. This implementation uses a SOAP server on the host location side. The SOAP client accesses the service via an application service, this case a WebSphere Application Server running on Windows 2000. Communication between client and host Access layer uses SOAP-enabled MQSeries, using HTTP as the underlying transport mechanism. The MQ IPT package is required to allow the messages to pass through the host firewall. **Figure 3** shows an overview picture of the sample implementation for a SOAP-enabled VAT service.
Figure 3: SOAP enabled sample for a legacy application

VAT SOAP service interface in WSDL

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="VatSOAPService-interface"
    targetNamespace="http://www.boeblingen.de.ibm.com/webservices/vat/vatsoapservice-interface"
    xmlns="http://schemas.xmlsoap.org/wsdl/
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
    xmlns:tns="http://www.boeblingen.de.ibm.com/webservices/vat/vatsoapservice"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <message name="IndoSimpleCalculationRequest">
    <part name="city" type="xsd:string"/>
    <part name="zipCode" type="xsd:string"/>
    <part name="county" type="xsd:string"/>
    <part name="state" type="xsd:string"/>
    <part name="country" type="xsd:string"/>
    <part name="currency" type="xsd:string"/>
    <part name="amountExponent" type="xsd:int"/>
    <part name="quantity" type="xsd:int"/>
    <part name="unitPrice" type="xsd:long"/>
    <part name="vatClass" type="xsd:string"/>
  </message>

  <message name="OutdoSimpleCalculationResponse">
    <part name="return" type="xsd:long"/>
  </message>

  <message name="IndoComplexCalculationRequest">
    <part name="city" type="xsd:string"/>
    <part name="zipCode" type="xsd:string"/>
    <part name="county" type="xsd:string"/>
    <part name="state" type="xsd:string"/>
    <part name="country" type="xsd:string"/>
    <part name="currency" type="xsd:string"/>
    <part name="amountExponent" type="xsd:int"/>
    <part name="vssis" type="ArrayOfItems"/>
  </message>

  <message name="OutdoComplexCalculationResponse">
    <part name="return" type="ArrayOfVatSOAPServiceItem"/>
  </message>

  <types>
    <xsd:schema targetNamespace="urn:webservices-vat-service"
                xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <xsd:complexType name="item">
        <xsd:element name="quantity" type="xsd:int"/>
      </xsd:complexType>
    </xsd:schema>
  </types>
</definitions>
```
<xsd:element name="unitPrice" type="xsd:long"/>
<xsd:element name="net" type="xsd:long"/>
<xsd:element name="gross" type="xsd:long"/>
<xsd:element name="vatTaxAmount" type="xsd:long"/>
<xsd:element name="vatTaxPercent" type="xsd:long"/>
<xsd:element name="vatClass" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="ArrayOfItems">
<xsd:complexContent mixed="false">
<xsd:restriction base="soapenc:Array">
    <xsd:attribute n1:arrayType="item[]" ref="soapenc:arrayType" xmlns:n1="http://schemas.xmlsoap.org/wsdl/"/>
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
</xsd:schema>
</types>

<portType name="VatSOAPService">
    <operation name="doSimpleCalculation">
        <input message="IndoSimpleCalculationRequest"/>
        <output message="OutdoSimpleCalculationResponse"/>
    </operation>
    <operation name="doComplexCalculation">
        <input message="IndoComplexCalculationRequest"/>
        <output message="OutdoComplexCalculationResponse"/>
    </operation>
</portType>

<binding name="VatSOAPServiceBinding" type="VatSOAPService">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="doSimpleCalculation">
        <soap:operation soapAction="urn:webservices-vat-service"/>
        <input>
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:webservices-vat-service"
                use="encoded"/>
        </input>
        <output>
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:webservices-vat-service"
                use="encoded"/>
        </output>
    </operation>
    <operation name="doComplexCalculation">
        <soap:operation soapAction="urn:webservices-vat-service"/>
        <input>
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:webservices-vat-service"
                use="encoded"/>
        </input>
        <output>
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:webservices-vat-service"
                use="encoded"/>
        </output>
    </operation>
</binding>
</definitions>
Conclusion

Investment in application development is always an issue for the enterprise. As an open standard networked world begins to dominate the IT world, the preservation of these investments is likely to change. Today many enterprises are trying to change their legacy applications into component-oriented systems. A new option to facilitate this change can be the use of web services technology. Elementary services from legacy applications can be wrapped as web services and orchestrated to higher value services.

In this article, we have described how the web service technology can be applied to mature legacy applications. We have shown how it is possible to change the proprietary interface into those based on open standards as needed for dynamic e-business.

As an example, we have shown how to accomplish this effort with a "real-life" problem by using a legacy application from the e-commerce field. Providers of e-marketplaces, or the like, have the need for essential services especially for payment and billing purposes. They also need the stability, scalability, and high availability provided by large mainframe host servers. We have also described how transaction management can be handled, without changing the existing system used by the legacy application.
Related topic

- For more information about mainframe hardware and applications you should look into IBM's zSeries, Parallel Sysplex for zOS, IBM CICS Transaction Server, IMS, and IBM Workload Manager WLM.