Model Problems in Technologies for Interoperability: Web Services

Grace A. Lewis
Lutz Wrage

June 2006

Integration of Software-Intensive Systems Initiative

Technical Note
CMU/SEI-2006-TN-021

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

Abstract ..... **vii**

1 Introduction **1**

2 Model Problem Process Applied to an Evaluation of Web Services for Interoperability **3**
2.1 Identify Model Problem Context 3
2.2 Develop Hypotheses 4
2.3 Develop Criteria for Hypotheses 4
2.4 Design and Implement Model Solution 5

3 Evaluation **9**
3.1 Results for Hypothesis 1 9
3.1.1 Web Services Standards are Emerging 10
3.1.2 The Differences Between Development and Deployment Environments Can Be Problematic for Web Services 11
3.1.3 Making Wrapper Code Available Simplifies Development on the Same Platform 12
3.2 Results for Hypothesis 2 13
3.2.1 WSDL Is Not Enough to Describe All Aspects of Services 14
3.2.2 Service Granularity Can Have a Large Impact on Performance 17
3.2.3 There Is Not Yet a Market for Services 17
3.3 Results for Hypothesis 3 19
3.3.1 Interoperability Testing is Important 19
3.3.2 Tool Interoperability is Crucial 20
3.3.3 Adding Greater Expectations to Web Services Makes It Harder to Achieve Interoperability 21

4 Conclusions and Request for Feedback **23**

References **24**

List of Figures

Figure 1: Model Problem Process for Technology Evaluation 3

Figure 2: Deployment View for the Complete Set of Model Solutions 7

Figure 3: Class Diagram for the Business and Data Management Logic of the Extended HR System 8

Figure 4: Simplified Representation of the Reassignment Service 13

Figure 5: Static Binding to Web Service 16

Figure 6: Dynamic Binding to a Web Service 16

Figure 7: Web Services Protocol Stack 21

List of Tables

Table 1: Hypotheses and Their Criteria for the Web Services Investigation 4

Table 2: Scenario, Model Solution, and C&C View for Hypothesis 1 5

Table 3: Scenario, Model Solution, and C&C View for Hypothesis 2 6

Table 4: Scenario, Model Solution, and C&C View for Hypothesis 3 6

Abstract

Web service technologies (or Web services) are experiencing a growing popularity in U.S. Department of Defense, industry, and non-defense government organizations due to their potential to enable interoperability between applications implemented on different platforms. This potential stems from Web services being based on standards that have been widely accepted and implemented, such as the Simple Object Access Protocol and the Web Services Description Language. The large number of products and tools created to facilitate the development of Web services has also contributed to their popularity. This technical note presents the results of applying the model problem approach in an initial investigation of the potential of Web services to enable interoperability.

1 Introduction

The Integration of Software-Intensive Systems (ISIS) team at the Carnegie Mellon® Software Engineering Institute (SEI) is examining technologies and approaches for the construction of systems that are required to interoperate with other systems, with the purpose of identifying gaps between what these technologies and approaches offer and what users expect of them. The end goal of this research is to provide users with information about what can be expected from the current state of technology and to provide technology suppliers with information about user expectations.

From a technology perspective, there are many current approaches to building systems for which there are interoperability requirements. Each approach has particular advantages and disadvantages with respect to interoperability, and each works well in some circumstances but not in others [Lewis 04]. In this report we investigate Web services, one of many technologies for accomplishing interoperability.

Web services have been defined by the World Wide Web Consortium (W3C) as follows:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically [Web services description language] WSDL). Other systems interact with the Web service in a manner prescribed by its description using [simple object access protocol] SOAP-messages, typically conveyed using [hypertext transfer protocol] HTTP with an [extensible markup language] XML serialization in conjunction with other Web-related standards [W3C 04b].

Web services are an approach to implementing a service-oriented architecture (SOA), where all of the following apply.

- Service interfaces are described using WSDL [W3C 06].
- Message payload (i.e., content) is transmitted using SOAP over HTTP [W3C 03, W3C 04a].
- Universal Description Discovery and Integration (UDDI) is used for service registration and discovery [OASIS 05b].[1]

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

1 The use of a service registry is optional.

Other combinations of technologies can be used to implement SOAs, but using Web services is by far the most common approach. For this reason, the acronym SOA is often used in a way that implies the use of Web services as the implementation technology.

Web services are experiencing a growing popularity due to a number of factors, including those below.

- Systems can interact with one another via standard Web technologies.
- Services can be built once and reused many times.
- Services can be implemented in any programming language and on any platform.
- Systems can advertise capabilities as services for other systems to use.
- There is tremendous vendor support for Web service technology.

Given these benefits, it is no surprise that there is great interest from U.S. Department of Defense (DoD), industry, and government non-DoD organizations in using Web services for interoperability between systems.

In addition to those benefits, there are claims found in literature, experience reports, and vendor documentation that contribute to this interest.

- The implementation of Web services is very easy.
- Web services are the solution to integration between systems on different platforms.
- There are numerous public repositories and Web sites offering Web services that can be easily integrated into applications.

To verify these claims about Web services, we chose the model problem approach [Wallnau 01, Lewis 05b]. The model problem approach involves (1) formulating hypotheses about the technology and (2) examining these hypotheses against very specific criteria through experimentation. In this way, the hypotheses are either sustained or refuted. The model problem approach has the advantage of producing very efficient and representative experiments that not only evaluate technologies within the context of their future use but also generate hands-on competence with the technologies.

Section 2 presents the definition of the model problems. Results and details about the experience with Web services are presented in Section 3. Finally, Section 4 presents our conclusions.

2 Model Problem Process Applied to an Evaluation of Web Services for Interoperability

Model problems[2] are part of a larger process for context-based technology evaluation that includes steps to establish the context for the model problem and to capture user expectations on the evaluated technology [Lewis 05a]. A graphical representation of the model problem process is presented in Figure 1.

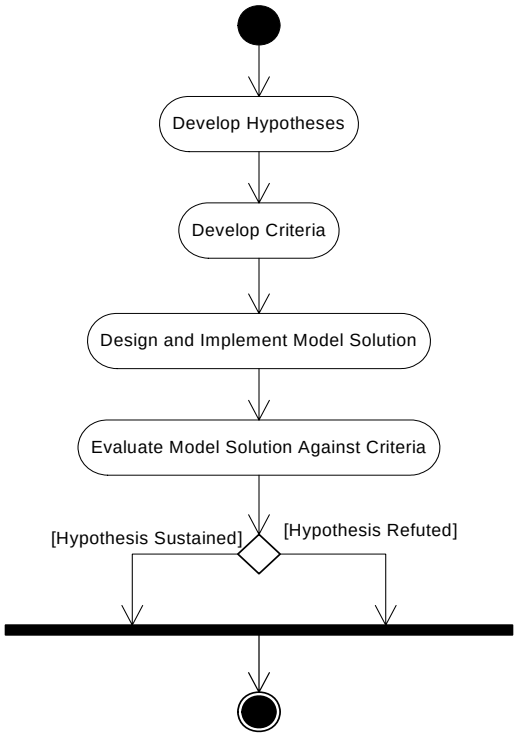


Figure 1: Model Problem Process for Technology Evaluation

2.1 Identify Model Problem Context

The context for the model problems in this report is a military human resources (HR) system. Military personnel are often reassigned to new locations, which triggers additional processes such as the actual reassignment, payroll adjustment, and flight booking. In this context, it is common for different systems on potentially different platforms to perform each of those processes.

[2] The model problem technique was initially created to evaluate commercially available software and system components [Wallnau 01]. For our purpose, the original model problem process has been slightly modified so that it extends to the evaluation of technology in general.

For example, in a real military HR system that we are familiar with, the data of personnel that are due for reassignment is transferred in batch mode to a separate reassignment system where it is processed and transferred (again in batch mode) back to the HR system. On a similar note, payroll data is processed in a system outside of the HR system with events transferred in batch mode from the HR system to the payroll system, so that pay is properly calculated every period.

2.2 Develop Hypotheses

For Web services, we defined the following initial hypotheses based on claims found in literature, experience reports, and vendor Web sites:

1. It is fairly easy for developers to connect applications developed for the same platform using Web services.
2. There are a large number of public, easily discoverable, and high-quality Web services that can be used in applications. (High-quality Web services are those for which the interfaces are well documented and straightforward to use.)
3. There are no problems regarding data types if Web services are used to connect applications on different platforms (i.e., Java 2 Enterprise Edition [J2EE] and .NET).

2.3 Develop Criteria for Hypotheses

These are the defined criteria for the hypotheses stated in Section 2.2.

Table 1: Hypotheses and Their Criteria for the Web Services Investigation

Hypothesis	Criteria
1. It is fairly easy for developers to connect applications developed for the same platform using Web services.	• Documentation is available on how to implement and access Web services in the selected platform. • Tools and libraries are available to implement Web services in the selected platform. • Tools and libraries are available to generate code in the selected platform to access a Web-based service from the associated WSDL document that describes the service. • Two applications can connect using Web services.
2. There are a large number of public, easily discoverable, and high-quality Web services that can be used in applications. (High-quality Web services are those for which the interfaces are well documented and straightforward to use.)	• Developers are able to locate Web services for use in their application by using public UDDI repositories or searching on the Internet. • The Web services are well documented, and there is guidance on how to use them.
3. There are no problems regarding data types if Web services are used to connect applications on different platforms (i.e., J2EE and .NET).	• The two applications can exchange complex, date, and floating point data types with no data inconsistencies between the two platforms. • This exchange can be done using default mechanisms provided with the Web services tools and libraries.

2.4 Design and Implement Model Solution

There was no product evaluation for the design and implementation of the model solution; we used either open-source products or products for which there were existing licenses. Nonetheless, as will be seen in the scenario descriptions, those products are commonly found in organizations, and they are readily available. To sustain or refute the above hypotheses, we defined the following scenarios and technical solutions.

Table 2: Scenario, Model Solution, and C&C View for Hypothesis 1

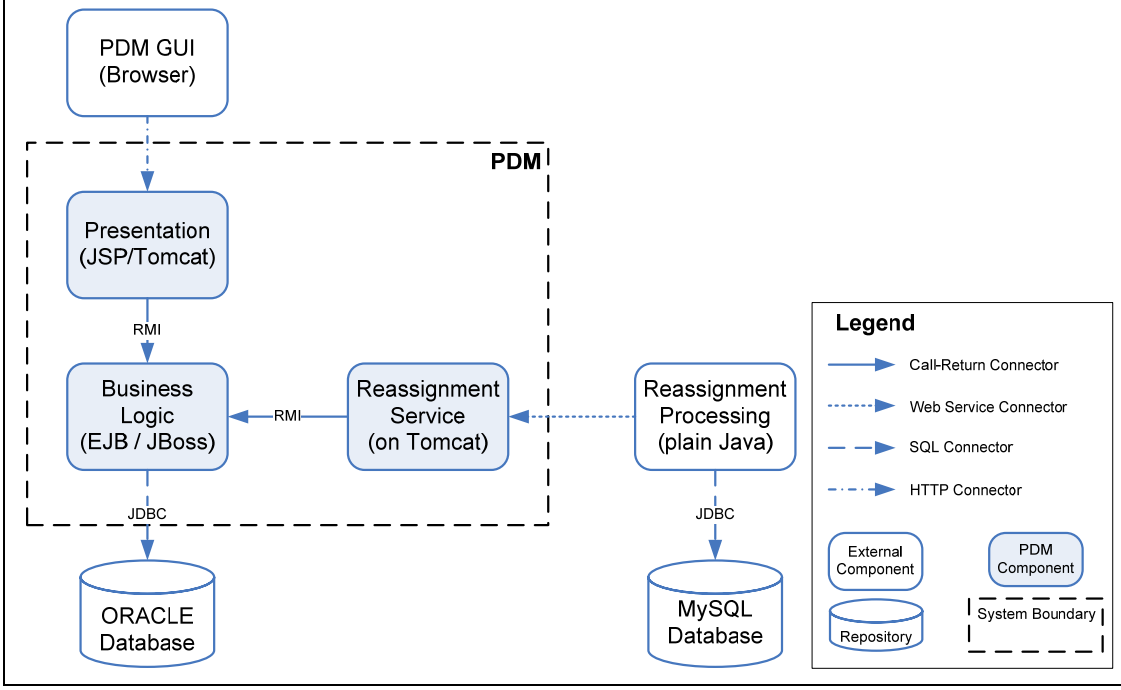
Hypothesis 1	It is fairly easy for developers to connect applications developed for the same platform using Web services.
Scenario	It is expected that a person is given a new assignment every year. Personnel reassignment is processed by a separate system. Every so often, the reassignment system queries the HR system for employees soon to be reassigned. Employee data is transferred in batch mode to the reassignment system, which processes the reassignments and sends the data back to the HR system in batch mode as well.
Model Solution	The existing personnel data management (PDM) HR system is a J2EE application that uses a number of Enterprise Java Beans (EJBs) to perform basic Create-Retrieve-Update-Delete (CRUD) operations and validations on personnel data. The user interface is implemented as a set of Java Server Pages (JSPs) that are accessed through a browser [Sun 06]. Apache Tomcat is used as the servlet container [Apache 06]. The J2EE application server used is JBoss Application Server [JBoss 05]. Data is stored in an Oracle database [Oracle 05]. A Web service with two operations will be added to the HR system: one to download pending reassignments and the other to upload processed reassignments. The reassignment processing system is created as a Java application that invokes the Web service to download and process the data and then invoke the Web service again to upload the processed data. The reassignment system data is stored in a MySQL database [MySQL 06]. A component and connector (C&C) view of the model solution follows [Clements 02].
	

Table 3: Scenario, Model Solution, and C&C View for Hypothesis 2

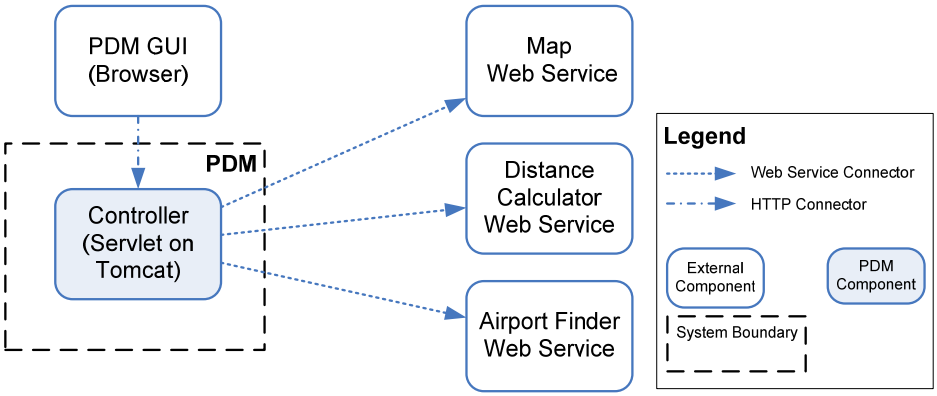
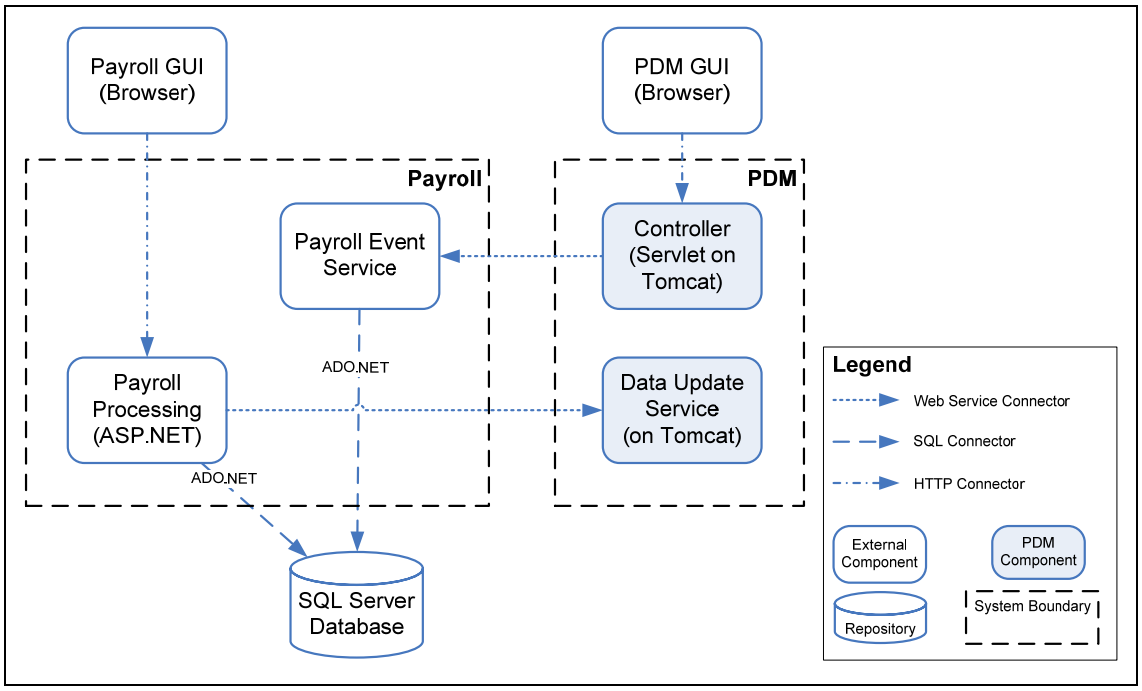
Hypothesis 2	There are a large number of public, easily discoverable, and high-quality Web services that can be used in applications. (High-quality Web services are those for which the interfaces are well documented and straightforward to use.)
Scenario	The PDM HR system is also used to initiate travel arrangements for reassigned personnel. It uses external Web services to find travel options to reach the location of the new assignment.[3]
Model Solution	The HR System will be extended with functionality that uses an external Web service or a collection of Web services to look for travel options given the person's current location and the location of the new assignment. A C&C view of the model solution is presented below.
	

Table 4: Scenario, Model Solution, and C&C View for Hypothesis 3

Hypothesis 3	There are no problems regarding data types if Web services are used to connect applications on different platforms (i.e., J2EE and .NET).
Scenario	A person who is reassigned is given a salary adjustment and a relocation bonus. The payroll system has to be informed of the reassignment so that the adjustment and bonus can be reflected in the next paycheck. As a confirmation mechanism, once the adjustment and bonus are processed, the payroll system notifies the HR system.
Model Solution	The payroll system is implemented as a .NET application. This payroll system will contain a Web service with an operation for receiving salary adjustments and bonuses. The HR system will be extended with a Web service to perform an operation for notification of payroll events. This will allow testing of the Web services in both directions (J2EE to .NET and .NET to J2EE). The payroll and HR systems will interact as follows: • Each time a person's pay changes in the HR system, HR calls the salary adjustment service in the payroll application. • The payroll application stores the received adjustment information in a database for processing by a payroll specialist. • Each time a payroll specialist processes a salary adjustment, the payroll system calls the new Web service in the HR system with the information that the processing has been completed. The parameters of Web service operations will include complex (such as a record), date, and floating point data types, which were expected to cause problems between these two platforms. A C&C view of the model solution follows.

[3] In Section 3.2, we describe why we had to modify this scenario.

Table 4: Scenario, Model Solution, and C&C View for Hypothesis 3 (cont.)



A deployment view for the complete set of model solutions is presented in Figure 2.

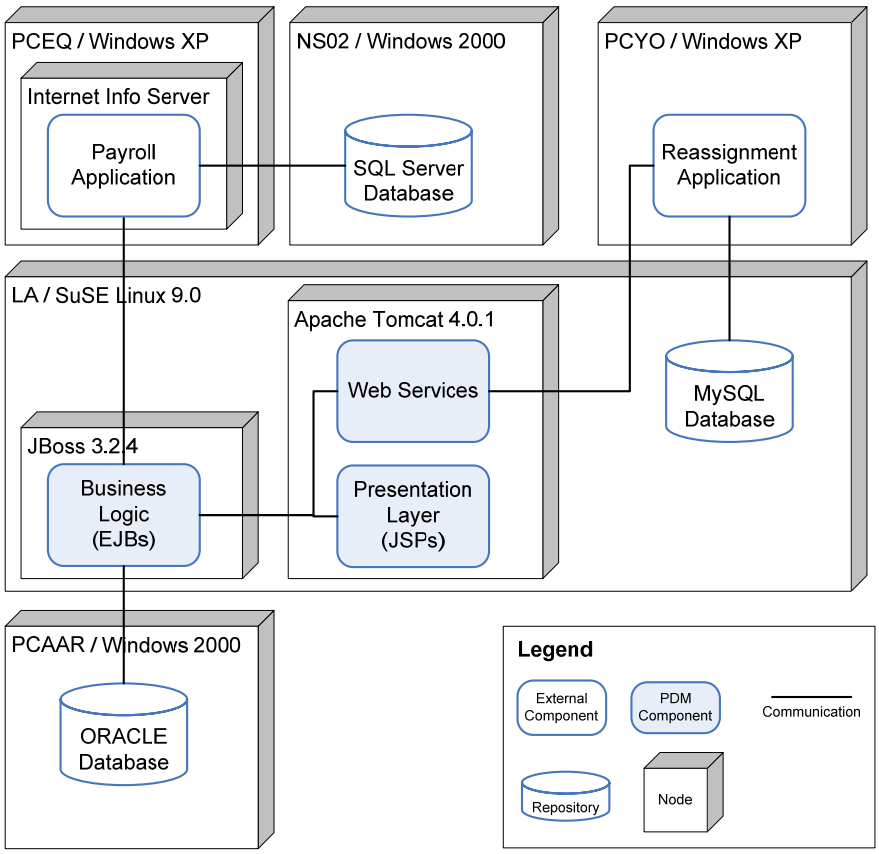


Figure 2: Deployment View for the Complete Set of Model Solutions

A class diagram for the extended HR system is presented in Figure 3. In Section 3, we present the evaluation of the model solution against the model problem criteria described in this section.

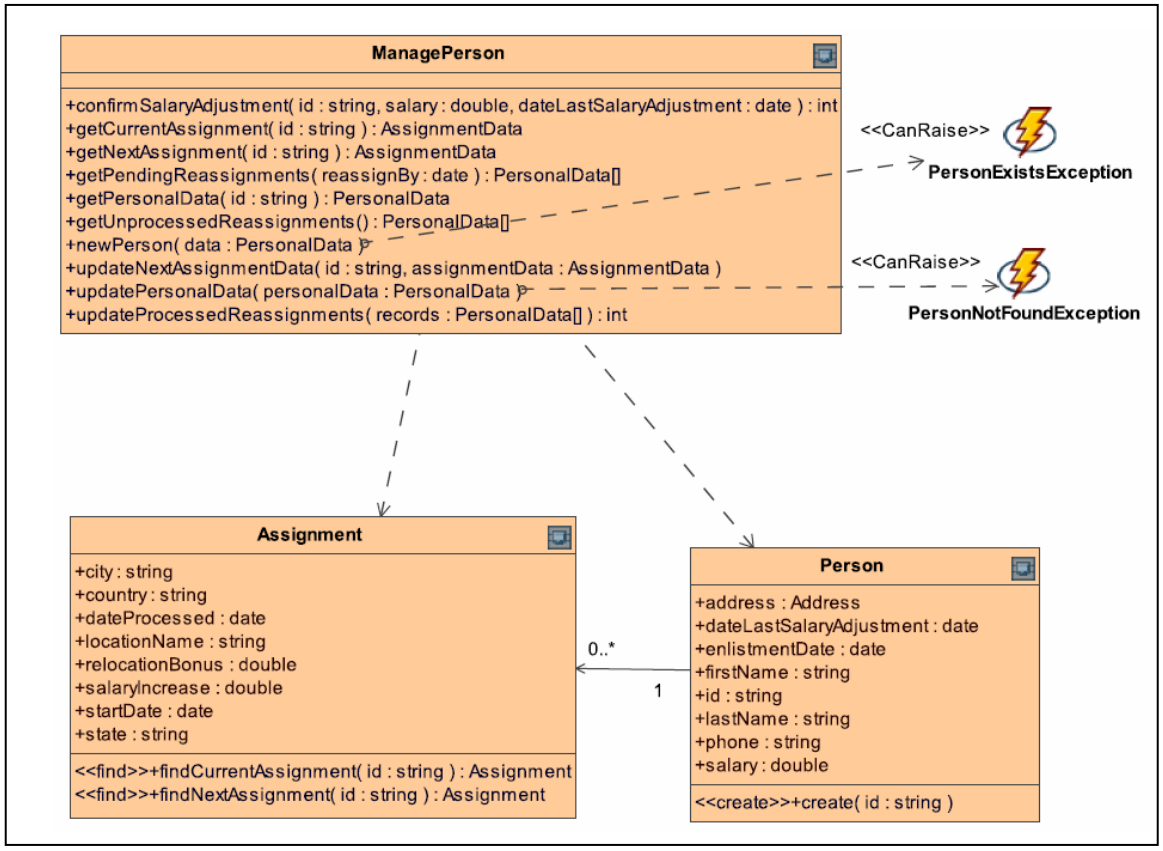


Figure 3: Class Diagram for the Business and Data Management Logic of the Extended HR System

3 Evaluation

In this section, we provide the results of evaluating the model solutions against the criteria in order to determine whether the hypotheses are sustained or refuted.

3.1 Results for Hypothesis 1

Our first hypothesis, "It is fairly easy for developers to connect applications developed for the same platform using Web services," was sustained. It was easy to connect the reassignment application developed as a Java client to the HR system developed as a J2EE-based application.

The Web services interface for the HR system was generated using the same MDA-based[4] development tool that was used initially to generate the J2EE infrastructure code and Web-based user interface for the HR system. This work is described in *Model Problems in Technologies for Interoperability: Model-Driven Architecture* [Lewis 05b]. The development tool generated

- the WSDL file describing the Web services interface
- the code that needs to be present on the server side to handle the calls to the Web service and the SOAP messages within these calls
- the code that needs to be present on the client to invoke the Web service
- a small test client stub

From conversations with colleagues we've seen that this result is common in integrated development environments (IDEs) that have functionality for Web services development. The tool also provided a way to easily visualize the messages being exchanged, which was very useful for troubleshooting.

There were some difficulties associated with the MDA tool and in the actual deployment of the Web service. Once we overcame those obstacles, however, the deployment of subsequent services was very simple. A more detailed description of the findings follows.

[4] MDA stands for model-driven architecture and is the implementation of model-driven development maintained by the Object Management Group. The main idea behind an MDA-based development tool is to separate functionality from infrastructure by means of platform-independent models that are transformed into platform-specific models through the application of transformation rules. The end goal for most current MDA-based development tools is to generate code for specific platforms from models. For more information, go to http://www.sei.cmu.edu/isis/guide/technologies/mda.htm.

3.1.1 Web Services Standards are Emerging

To understand Web services, it is useful to understand WSDL and SOAP. The primers on SOAP 1.2 and WSDL 2.0 that are part of the standard documents for Web services were a good way to learn about the latest versions of these two specifications. However, the MDA tool generated Web services conforming to SOAP 1.1 and WSDL 1.1, as do most such tools in the market. There are major differences between versions. For example, just in additional or changed syntax there are 15 differences between the SOAP 1.1 and SOAP 1.2. While these changes are beneficial because they make the SOAP 1.2 clearer and more robust, they can make valid SOAP 1.1 messages become invalid under SOAP 1.2. As a result, once tools and libraries start supporting SOAP 1.2, existing applications and services will have to follow in order to continue operation.

The Web Services Interoperability Organization (WS-I) attempts to provide guidance on the use of Web services standards. Established in early 2002, WS-I is an open industry effort chartered to promote Web services interoperability across platforms, applications, and programming languages. This organization brings together a diverse community of Web services leaders to respond to customer needs by providing resources and recommended practices for developing interoperable Web services [WS-I 06a]. One of its deliverables is the Basic Profile, which is a set of nonproprietary Web services specifications that promote interoperability by providing clarifications, refinements, interpretations, and amplifications in areas of the standards that are subject to multiple interpretations [WS-I 06b].

As a test, we ran the generated WSDL file against the WS-I Analyzer tool to verify the conformance of the generated WSDL file to the WS-I Basic Profile. The tool found several issues related to namespaces and data types. For example, although the tool generated a valid WSDL document, it used Java types within the WSDL document instead of XML Schema types. This difference was not a problem during testing with the generated test client because both service and test client were generated by the same tool, but we suspected that this would cause problems outside of that setting, which it eventually did as described in Section 3.3. After we changed the Java types to XML Schema types and fixed other small problems with date/time data types and invalid `portType`[5] names, the WSDL document passed the conformance test.

The standards related to other cross-cutting aspects of Web services, such as security or transactions, are even more undefined and emerging, which adds even more value to the effort that organizations such as the WS-I are making.

[5] The `portType` element of a WSDL document includes the set of operations supported by the Web service. Each operation includes the input and the output messages of the operation (e.g., request message and response message for a certain operation).

3.1.2 The Differences Between Development and Deployment Environments Can Be Problematic for Web Services

Testing the model solution involved a good deal of troubleshooting; the most problematic aspect of which was the difference between the development and deployment environments.

The simplest way to explain how Web services work in the model solution is the following sequence:

1. The client generates a SOAP message that is sent within an HTTP request to an HTTP server.
2. The HTTP server recognizes this as a SOAP message and sends it to a SOAP servlet running within a servlet engine such as Tomcat.
3. The SOAP servlet decodes the SOAP message and calls the code associated with the requested Web services operation.
4. The SOAP servlet receives the return data from the called code and sends a SOAP message back in an HTTP response.

The generated test client worked with the Apache SOAP servlet running locally within the Tomcat 3.0 servlet engine that was included in the MDA tool. The model solution called for the servlet engine to reside on the server instead of the client. JBoss, the application server for the HR application, comes with Tomcat 4.0. Initially, the difference in versions was not considered to be a problem because most products are backwards-compatible. Unfortunately, when the Web services code was moved over to the Tomcat instance on the server, the model solution did not work. The Apache SOAP servlet included with the MDA tool conformed to the XML Schema 1999 specification, and the Apache SOAP servlet included with JBoss conformed to the XML Schema 2001 specification.[6]

After extensive debugging and troubleshooting, we traced the cause of the problem to a portion of the code generated by the MDA tool. This code assigned a default message serializer and deserializer for basic data types and worked without any problems with the older Apache SOAP implementation included with that tool. After this portion of code was removed, the model solution worked. Our assumption is that the serializer/deserializer assigned for basic data types by the MDA tool was different from the one assigned by default by the newer Apache SOAP implementation. Removing the code portion made the newer Apache SOAP implementation use its default serializer/deserializer instead of the one wrongly assigned by the MDA tool. Although the solution to the problem was trivial, we expended a significant amount of effort to find it. Moreover, this problem is not something that should be discovered towards the end of a project, especially in a real-world implementation.

The lesson learned from this experience has to do with IDEs in general, as well as with the emerging characteristics of standards related to Web services. Most modern IDEs come with

[6] To complicate matters even further, the reassignment application was developed using Eclipse, which can also run its own instance of Tomcat.

embedded infrastructure software and a “safe environment” for testing. Because of this, initial testing happens on a single machine and within this safe environment. Given the potential for having a large number of components as part of a Web services solution (i.e., HTTP server, servlet engine, SOAP servlet, application server, and database), the development environment should resemble the deployment environment as much as possible. Also, a change in one component of the solution requires the re-evaluation of the appropriateness of the rest of the components of the solution.

3.1.3 Making Wrapper Code Available Simplifies Development on the Same Platform

A Web service with two operations was added to the HR system, as indicated in Table 2 on page 5.

1. Retrieve pending reassignments.
   This operation retrieves personnel records from the HR system on the people due for reassignment by a given date. It receives a date as input and returns an array of data of the type `PersonalData`.
2. Upload processed reassignments.
   This operation uploads to the HR system the processed personnel records with the new assignment data. It receives an array of data of the type `PersonalData` as input and returns the number of records retrieved in the first operation.

The data exchanged was defined as a complex type in the SOAP message and matched the `PersonalData` data type (Java class) used inside the HR system (as illustrated in Figure 3 on page 8). This practice is common. Figure 4 shows a simplified class diagram representation of the reassignment Web service.

The code we had to write for the reassignment system was extremely simple because we reused the same `PersonalData` data type, as well as the same client-side wrapper code that was generated by the MDA tool. On another platform, this code would have to have been created based on the WSDL file. This situation is not really a surprise; it is similar to what happens when two applications communicate using other middleware technologies such as a common object request broker architecture. Organizations that wish to facilitate the development of client applications can make wrapper and data type code available for download.

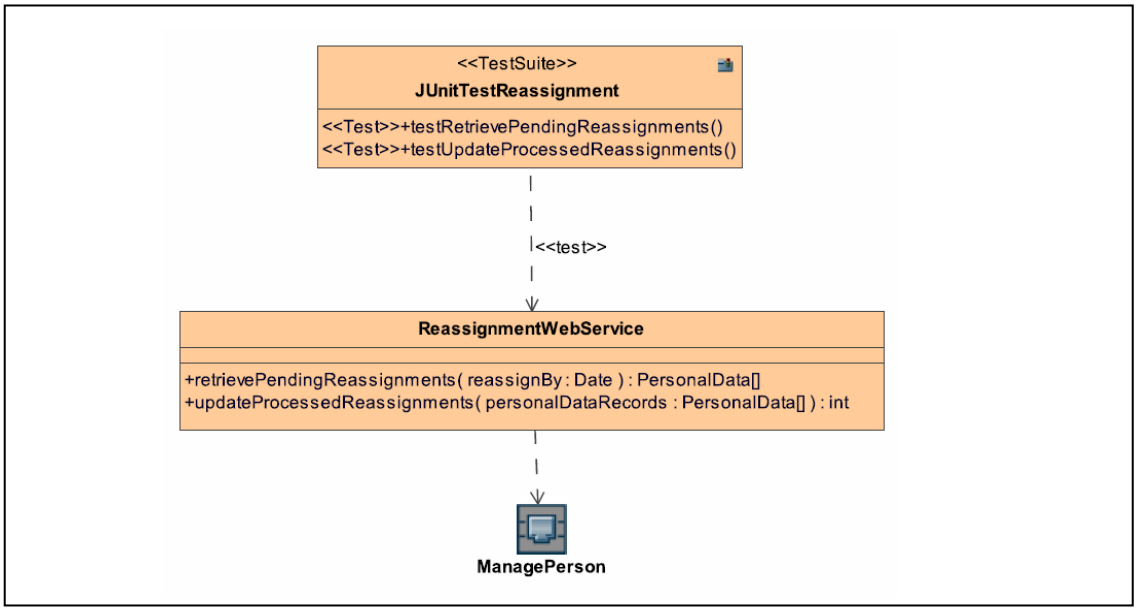


Figure 4: Simplified Representation of the Reassignment Service

3.2 Results for Hypothesis 2

The hypothesis "There are a large number of public, easily discoverable, and high quality Web services that can be used in applications" was refuted.

The goal of this aspect of our research was to locate public Web services that, given an origin and a destination, would provide a list of travel options (similar to the response provided on an airline travel agency Web site). The first step was to search public UDDI repositories such as IBM's UDDI Business Registry[7] or Web services portals such as WSindex.[8] The result of those searches was quite disappointing. Most entries pointed to nonexistent Web services or "toy" Web services created by people learning about UDDI. The next step was to use a search engine to find appropriate Web services. That experience was equally disappointing, because the Web services found were not related to travel. Given these results, we changed the scenario for hypothesis 2 to "The HR System uses external Web services to locate the airport that is closest to the location of the new assignment."

We found three Web services that, in combination, could find the closest airport. Here is the process that we devised:

1. Locate airport information using a Web service from WebserviceX.NET.[9] We located all airports in the destination country using the `GetAirportInformationByCountry` operation in this Web service. The

[7] The IBM UDDI Business Registry, the IBM UDDI Test Registry, and the IBM UDDI Beta Test Registry Web sites are no longer available.

[8] For more information, go to http://www.wsindex.org/

[9] For more information, go to http://www.webservicex.net/WS/WSDetails.aspx?WSID=20&CATID=7

operation returns a list of airports in a given country, along with their latitude and longitude values.

2. Find latitude and longitude of the destination city using Microsoft's TerraService Web Service.[10] The `ConvertPlaceToLatLon` operation in this Web service takes city, state, and country input and returns the corresponding latitude and longitude.
3. Use the *Calculate Distance between Two Coordinates* Web service from InnerGears.[11] We used this service to calculate the distance between airports and the destination city. The `CalcDistance2Coords` operation takes two coordinates expressed as latitude/longitude and returns the distance between them.
4. Select the airport that has the shortest distance to the destination.

A WSDL2Java tool from Axis was used to generate all the necessary infrastructure code to connect to these Web services [Apache 05]. This new scenario for hypothesis 2 also allowed for the exploration of Web services composition, another interesting topic. A more detailed description of the findings follows.

3.2.1 WSDL Is Not Enough to Describe All Aspects of Services

Interoperability is much more than the capability to exchange data between systems. It also requires a shared understanding of that information and how to act upon it.

> Interoperability is the ability of a collection of communicating entities to (a) share specified information and (b) operate on that information according to an agreed operational semantics [Brownsword 04].

The ability to exchange data is called *syntactic interoperability,* and the ability to operate on that data according to agreed-upon semantics is termed *semantic interoperability.* Both varieties are necessary prerequisites to achieve interoperability.

In its simplest form, a WSDL document is an XML-based document that describes what a Web service can do, where it resides, and how to invoke it. A WSDL document does not convey semantics. WSDL deals with data formats and representations; it does not deal with meaning and interpretation of data and operations. A simple example of the importance of shared semantic meaning of data involves price quotes from online vendors. Both the requesting customer and quoting vendor may share a common understanding of the raw value of the figure quoted (e.g., $199.99). However, there must also be a deeper understanding of the meaning of that value. For example, does the quoted price include sales tax? Or is a shipping charge included? The way semantics are shared can range from information shared at design time using English text to formal approaches using ontological service descriptions. All these approaches are outside the scope of WSDL, and unfortunately current technology

[10] For more information, visit http://terraserver-usa.com/webservices.aspx.
[11] For more information, visit http://www.innergears.com.

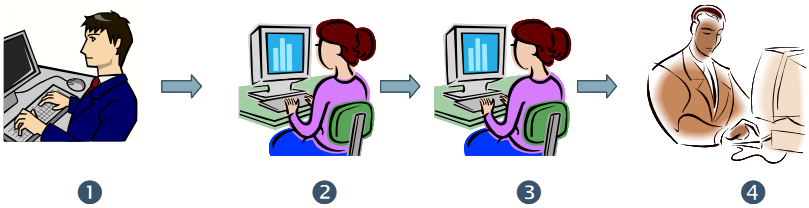
has not matured to a point where ontologies are in common use. Syntactic and semantic interoperability are areas of active research.

All this said, the types of the data being exchanged in the Web services operations are specified using XML Schema Part 2: Datatypes. The XML Schema standard provides facilities for defining data types to be used in XML Schemas as well as other XML specifications [W3C 04c]. It allows the specification of constraints such as value ranges for numerical values, sizes and valid values for strings, and patterns for dates. Nonetheless, it is rare to find a WSDL document that specifies anything other than the data type (e.g., `int`, `string`, or `double`).

One problem that we encountered was that a WSDL document can contain description elements that document details and assumptions, but we found little helpful documentation in the WSDL documents we used. As an example, there are different ways to express latitude and longitude. One way is to use degrees as the unit, with values between –90 and +90 for latitude and values between –180 and +180 for longitude. Another way is to use radians (a measurement of angle) as the unit. Another possibility is to use `string` as the data type and to represent latitude and longitude as numbers followed by N, S, W or E to indicate direction. It was necessary to execute tests before binding to the Web services to determine the correct units and notation. While it was possible to execute these tests in the context of the model problem, in a production environment this time-consuming activity would not be acceptable.

Another problem was that some aspects of data fall outside of the scope of WSDL. The reassignment system, illustrated in the figure in Table 2, obtains the reassignment location data from a `Location` table in the MySQL database. We had initially loaded the data for this table from two sites: one that contained a list of countries and capitals and a second that listed major U.S. cities. That level of detail was enough for our exercise. While preparing the data for loading, we noticed that some of the country capital city names were in their native language. Also, some of those native language names contained special characters. The airport information Web service, however, expected all country names in English. We could fix this problem by loading the data from another site, but we saw other problems such as the use of old country names instead of new country names (Burma instead of Myanmar, for example). Also, U.S. state names were abbreviated in one site and spelled out in another. A WSDL document by itself would not be able to indicate all these constraints and requirements, not even if full XML Schema data types were used.

This issue looms larger when the binding to services is to be done at runtime, not at design time as in our model problem. Binding to services at design time is referred to as static or fully grounded binding. Figure 5 shows an example of what happens during static binding. In this case, discovery, composition, and invocation of Web services are done at design time, allowing the developer to discover the semantics of a service before it is actually used. In the case of dynamic binding, illustrated in Figure 6, the binding to the Web services is done at runtime.



❶ Bob (a service provider) creates a Web service and places its description in an "accessible place" (e.g., in a UDDI registry).

❷ Alice (the application developer) obtains the destination of Bob's Web service (e.g., by searching the UDDI registry).

❸ Alice adds code to her application to invoke Bob's Web service.

❹ Joe (the end user) uses Alice's application without knowing that there is a Web service behind it.

Figure 5: Static Binding to Web Service

❶ Bob (a service provider) creates a Web service, describes it, and registers the service with a service repository.

❷ Alice (the application developer) writes code in her application that is able to query the service repository at runtime.

❸ Alice writes code in her application that selects a service from the list returned by the query.

❹ Alice writes code in her application that can invoke the selected service transparently.

❺ Joe (the end user) uses Alice's application without knowing that there is a Web service behind it.

Figure 6: Dynamic Binding to a Web Service

Dynamic binding requires detailed Web services descriptions that convey syntax as well as semantics. Using our example of finding reassignment location data, how does an application know that it has to translate the country name into English? How does an application know that it has to convert from a state's full name to its abbreviation? A developer who binds to these services at design time can recognize the discrepancy and include code to translate the names. However, to develop an application that searches for an airport location service and automatically binds to it requires enough intelligence in the application to determine that this translation is needed.

A significant area of current work and research in dynamic binding is that of Semantic Web Services (SWS), which uses a markup language that is descriptive enough for a computer to obtain automatically the information it needs to discover, compose, and invoke Web services without human intervention. SWS is usually described using concepts from an ontology to provide the shared semantics between service provider and service consumer. If an application searches for an airport information Web service that takes a `Country` as input and produces a list of `Airports` as output, and both `Country` and `Airport` are concepts within a common ontology, then the application developer and the Web service provider

would be talking about the same thing when they refer to airports and countries. The details of SWS are outside of the scope of this report. The technical note named *Model Problems in Technologies for Interoperability: OWL Web Ontology Language for Services (OWL-S)* describes the use of OWL-S for the dynamic discovery, composition, and invocation of Web services [Metcalf 06].

Fully dynamic binding, as described above, is difficult to achieve. In some situations, it is possible to involve the end user in the selection of services to use. The application can retrieve a list of appropriate Web services and obtain additional input from the end user to connect to one of the services. Examples of end user inputs include making a selection from a service list and providing mappings between application data and service inputs and outputs.

3.2.2 Service Granularity Can Have a Large Impact on Performance

An important design decision when developing Web services is choosing the right granularity for operations. Service interfaces can affect the end-to-end performance in a system because services are executed across a network as an exchange of a service request and a service response. If service interfaces are too coarse-grained, consumers receive more data than they need in their response message. If service interfaces are too fine-grained, consumers have to make multiple trips to the service to get all the data they need.

In the modified scenario for hypothesis 2, the `CalcDistance2Coords` operation in the *Calculate Distance between Two Coordinates* Web service had to be called for each airport returned by the `GetAirportInformationByCountry` operation in the airport information Web service. This was necessary because the first Web service did not have an operation that calculated the closest airport to a given set of coordinates, even though it had all necessary information to do it. (It would just have been one more call, but the service was not defined this way. The service provider did not anticipate that someone would want to do what we wanted to do.) As a result, in the model problem, finding the closest airport took between three seconds and two minutes depending on how many airports were in the country. These times are obviously not acceptable for a system from which users expect near real-time responses.

3.2.3 There Is Not Yet a Market for Services

The quality of public Web services was often poor in our experience, but there were huge differences in quality even among the Web services used in the model problem. We faced the following quality problems (or at least unexpected characteristics of the data returned by the Web services) during development of our model solution:

- The list of airports was returned as a string that contained an XML document instead of a list with all its elements defined. XML parsing code had to be created to extract the necessary information.
- There were no airports listed for some countries.

- Some tags contained typographical errors or were inconsistent.
- Duplicate records were returned.

Would these problems be present if someone were paying for the use of these services? Probably not: businesses must have incentives to provide and maintain high-quality Web services. Our reference to incentives does not necessarily imply that users must pay for services; the service provider may also receive a derived business benefit that results in a positive return on investment.

Companies are just starting to explore the concept of a service market that requires a business model in which consumers pay for services only when used. This model also imposes requirements on service providers, consumers, and brokers as follows:

- public service repositories where businesses can advertise and place their services
  The problem faced by public (or even private) service repositories is similar to that faced by component repositories—maintenance. A service repository is even more difficult to maintain than a component repository because all it has are links to services (rather than a physical component). Services can disappear or become unavailable for many reasons. Is a repository provider responsible for notifying users of service failures or outages? Services also have to be described in such a way that service consumers can find what they are looking for.
- new cost and contracting schemes
  Service brokers and providers have to determine what to charge consumers for services. Are consumers charged per use? Is there a monthly or annual fee? Do brokers work on commission? What type of contract exists between consumer and broker? How about between consumer and provider? All these questions have to be answered.
- robust and reliable services that consumers can trust as part of their applications and business processes
  As we said in Section 3.2, we found the quantity and quality of Web services for our purpose that are publicly available to be disappointing. The situation may be different with respect to private service repositories. For public repositories, there is a tradeoff between openness and how much trust customers can place in the information about services of interest to them. If the repository is open (i.e., anyone can enter a service), there is no easy way for a customer to determine the status and seriousness of the service offering. If, however, the information in the repository is carefully screened and even certified by a gatekeeper organization, the cost would be quite expensive. A viable model between these extreme views has not yet been developed.

3.3 Results for Hypothesis 3

Our third hypothesis was this: “There are no problems regarding data types if Web services are used to connect applications on different platforms (i.e., J2EE and .NET).” This hypothesis was sustained with regard to our test data exchanges. We were able to use WSDL documents to generate code to invoke a Web service implemented on .NET from a client on J2EE and vice versa. However, to make that process work, we had to edit a WSDL file manually and switch the SOAP implementation on the Java side to a newer version. A more detailed description of our findings follows.

3.3.1 Interoperability Testing is Important

Although we successfully invoked Web services across platforms, we encountered many obstacles along the way. All of the problems we saw were related not to Web services but to defects in tools, differences in supported Web service standard versions, and differences in how the standards were used.

Standards that define the basic Web service infrastructure include XML Schema, SOAP, and WSDL. Several intermediate versions of these standards have been defined before the final versions, and there are some tools and libraries still in use that implement one of the intermediate versions instead on the final one.

We chose to use an MDA tool to generate a skeleton implementation for our Web services, as explained in Section 3.1.1. When we tried to use the generated WSDL file in Visual Studio .NET to develop the payroll application, we discovered that the WSDL file contained a number of defects that made it almost completely unusable. These problems were irrelevant for our purpose as long as we used a client on the same platform, because the Java client proxies were generated correctly. The MDA tool did not use the WSDL file internally to generate any code, but Visual Studio did. Using the WS-I Analyzer tool, we repaired the WSDL file and used it to create part of the payroll application. To create the WSDL for the payroll service, we used built-in capabilities of Visual Studio, which gave us a document that passed through the WS-I Analyzer tool without any errors.

Having created the correct WSDL service descriptions and the corresponding service implementations, we tested the interoperation of the J2EE and .NET applications. The first call to the .NET payroll application failed with an exception. It turned out that the parameters were in the wrong order in the SOAP message generated by the Apache SOAP implementation. Parameters were listed in alphabetical order instead of in the order given in the WSDL service description. Because we could not fix this behavior in the Apache SOAP implementation, we replaced it with the newer Apache Axis. This change required that we also replace the MDA tool-generated Java stubs and proxies. Apache Axis provides a tool, WSDL2Java, to generate these from a WSDL document [Apache 05]. After we regenerated the Java stubs and proxies, the service interaction worked without any problems. We could exchange simple data types—including floating point values, records, and date/time values.

(There were some other potential interoperability issues that we did not explore in detail, such as the handling of empty lists, null values, high precision floating point numbers, very big numbers, and timestamps across time zones. However, these issues have been documented elsewhere.[12])

As a result of our experiments, we learned that almost all the problems we faced were caused by the MDA tool's use of an outdated Web service library. Had we implemented the same services manually, we would have been successful much earlier. Because our model problem contained only a few services and service calls, it would have been easy to create the necessary code. Bigger projects, however, can benefit much more from tool support because of the potential savings in development time. Also, in a manual approach it is likely that the code will contain errors, whereas a good tool can produce error-free code. Overall, it was our impression that achieving basic data exchange and service invocation between J2EE and .NET using Web services was much easier than with any other technology we have used.

3.3.2 Tool Interoperability is Crucial

As mentioned above, we had problems with the MDA tool because we used an experimental code generation capability that was clearly not adequate for this part of our model problem. Also, different standard versions and interpretations used on the two platforms made development more complex than necessary. To resolve the problems we saw, we had to inspect SOAP messages exchanged between the systems to find the root causes. Also, for debugging, we had to learn more about the implementation of SOAP processing than we had anticipated. Those experiences show that tool interoperability is an important factor in Web service development.

This familiar lesson was reconfirmed by our experiments: tools and components must implement the same versions of the standards and use them in the same way to be useful in developing interoperable applications based on Web services. Consider, for example, our choice between *RPC/encoded* and *document/literal* styles for data encoding in the WSDL document [Butek 05]. The *RPC/encoded* style supports a programming model familiar to developers who have used remote procedure call mechanisms such as Java RMI and .NET Remoting. Apache SOAP defaults to *RPC/encoded*. Recently, *document/literal* has become a more popular style because it is closer to the paradigm that Web services interact by asynchronous exchange of XML documents. The advantage of *document/literal* encoding is that the WSDL service description contains the complete XML Schema for the exchanged messages, which allows for message validation using standard XML processing tools. The default encoding for .NET Web Services is *document/literal*. In our experiments, we first tried to use *RPC/encoded* on both sides, but we switched to *document/literal* after we had replaced Apache SOAP with Apache Axis. The encoding style can pose an interoperability issue if older Web services that have been developed using the *RPC/encoded* style need to be integrated.

[12] Relevant information is easy to find on the Web. A search using Google (http://www.google.com/), for instance, lists more than 400,000 Web pages for the search term "Web services interoperability."

3.3.3 Adding Greater Expectations to Web Services Makes It Harder to Achieve Interoperability

During our learning process, we came across two apparently contradictory articles: "Web Services Are Not Distributed Objects" [Vogels 03] and "Like It or Not, Web Services Are Distributed Objects" [Birman 04]. After reading the articles, we saw that both authors came to the same (and fundamental) conclusion: most people fail to understand that Web services offer no more than a document exchange using standard Internet protocols.

Both Vogels and Birman blame vendors for this misconception. Birman argues, for instance, that vendors have been selling Web services technologies as if they offer the same stability, reliability, and trustworthiness of distributed objects [Birman 04]. The truth is that Web services technology *currently* cannot offer all that distributed objects can.

In our model problem, we dealt with elements of Web services that are part of the base stack of Web services protocols, as illustrated in Figure 7. The standards employed in those elements are widely accepted and are supported by a large number of vendors. As people have placed more requirements on Web services, other standards have begun to emerge in the element of orchestration and composition, as well as in areas that have to be addressed in all layers of a solution, such as security, quality of service, transactions, and management.

We suspect that our positive answer with respect to the satisfaction of Hypotheses 1 and 3 would have been different if we were dealing with standards in the orchestration and composition elements or any of those layers, as these standards are mostly emerging and even competing. For example, in the area of security alone, there are three potential specifications: (1) Security Assertion Markup Language (SAML), (2) WS-Security, and (3) Web services Security (based on WS-Security) [OASIS 05a, IBM 02, OASIS 04].

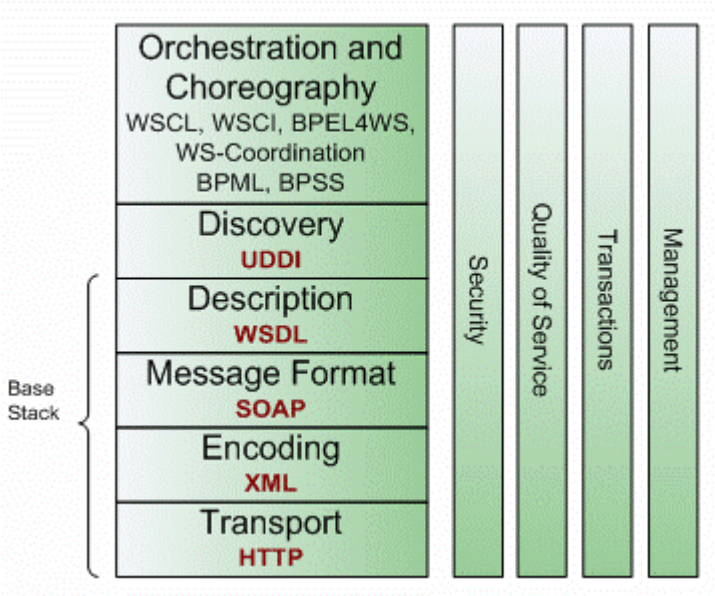


Figure 7: Web Services Protocol Stack[13]

[13] This figure is adapted from *XML and Web Services Unleashed* [Bloomberg 02].

We can conclude that Web services today can be used successfully in simple cases such as those described in this report. Any other requirements placed on a Web services solution will be troubled by partially defined standards, which will have a negative impact on interoperability.

4 Conclusions and Request for Feedback

Through our exploration of Web services using a model problem approach, we have gained competence in several technologies while examining various claims about Web services and the circumstances under which they prove true or false.

The model problem experience shows that it is fairly easy to implement Web services and connect applications developed on different platforms using Web services. The relative ease of implementing and using this technology is possible because Web service elements (i.e., discovery, description, message format, encoding, and transport) are based on widely accepted standards that are supported by a large number of vendors. Nonetheless, the standards behind Web services are still maturing, and most of the problems that we encountered, as described in this report, were due to incompatibilities between standard versions implemented by the different tools.

The model problem experience has also shown that too few public Web services are available. Most of those available are poorly documented and of poor quality. We believe the state of publicly available Web services is not going to change until there is a market for Web services that would force (or motivate) service providers to provide quality services.

The ISIS team that is investigating Web services and other technologies using the model problem approach is interested in feedback from and collaboration with the communities that are considering technologies for interoperability. In addition to Web services, OWL-S and MDA, the ISIS team is investigating Open Grid Service Architecture (OGSA), Web Services Modeling Ontology (WSMO), and other standards and technologies. Write to the ISIS team at isis-sei@sei.cmu.edu.

References

URLs are valid as of the publication date of this document.

[Apache 05] The Apache Software Foundation. *WebServices – Axis*. http://ws.apache.org/axis/java /user-guide.html#WSDL2JavaBuildingStubsSkeletonsAnd DataTypesFromWSDL (2005).

[Apache 06] The Apache Software Foundation. *Apache Tomcat.* http://tomcat.apache.org/ (2006).

[Birman 04] Birman, Kenneth P. "Like It or Not, Web Services Are Distributed Objects." *Communications of the ACM, 47,* 12 (December 2004): 60–62.

[Bloomberg 02] Bloomberg, Jason; et al. *XML and Web Services Unleashed*. Indianapolis, IN: Sams Publishing, 2002.

[Brownsword 04] Brownsword, Lisa L.; et al. *Current Perspectives on Interoperability* (CMU/SEI-2004-TR-009, ADA443493). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents /04.reports/04tr009.html.

[Butek 05] Butek, Russell. *Which style of WSDL should I use?* IBM Corporation, 2005. http://www-128.ibm.com/developerworks /webservices/library/ws-whichwsdl/.

[Clements 02] Clements, Paul; et al. *Documenting Software Architectures: Views and Beyond.* Boston, MA: Addison-Wesley, 2002.

[JBoss 05] JBoss, Inc. *JBoss Application Server.* http://www.jboss.org/products/jbossas (2005).

[Lewis 04] Lewis, Grace A. & Wrage, Lutz. *Approaches to Constructive Interoperability* (CMU/SEI-2004-TR-020, ADA431067). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents /04.reports/04tr020.html.

[Lewis 05a] Lewis, Grace A. & Wrage, Lutz. *A Process for Context-Based Technology Evaluation* (CMU/SEI-2005-TN-025, ADA441251). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn025.html.

[Lewis 05b] Lewis, Grace A. & Wrage, Lutz. *Model Problems in Technologies for Interoperability: Model-Driven Architecture* (CMU/SEI-2005-TN-022, ADA441294). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn022.html.

[Metcalf 06] Metcalf, Chris & Lewis, Grace A. *Model Problems in Technologies for Interoperability: OWL Web Ontology Language for Services (OWL-S)* (CMU/SEI-2006-TN-018). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006. http://www.sei.cmu.edu/publications/documents/06.reports/06tn018.html.

[MySQL 06] MySQL AB. *About MySQL*. http://www.mysql.com/company/ (2006).

[OASIS 04] Organization for the Advancement of Structured Information Standards. *Web Services Security v1.0 (WS-Security 2004).* March 2004. http://www.oasis-open.org/specs/index.php#wssv1.0.

[OASIS 05a] Organization for the Advancement of Structured Information Standards. *Security Assertion Markup Language (SAML) v2.0*. March 2005. http://www.oasis-open.org/specs/index.php#samlv2.0.

[OASIS 05b] Organization for the Advancement of Structured Information Standards. *OASIS UDDI*. http://www.uddi.org/ (2005).

[Oracle 05] Oracle Corporation. *Oracle Database: The First Database Designed for Grid Computing.* http://www.oracle.com/database/ (2005).

[Sun 06] Sun Microsystems. *Java EE At a Glance.* http://java.sun.com/j2ee/ (2006).

[Vogels 03] Vogels, Werner. “Web Services Are Not Distributed Objects.” *IEEE Internet Computing, 7*, 6 (November–December 2003): 59–66.

[W3C 03] World Wide Web Consortium. *HTTP - Hypertext Transfer Protocol*. http://www.w3.org/Protocols/ (2003).

[W3C 04a] World Wide Web Consortium. *Latest SOAP Versions*. http://www.w3.org/TR/soap/ (2004).

[W3C 04b] World Wide Web Consortium. *Web Services Architecture: W3C Working Group Note 11 February 2004*. http://www.w3.org/TR/ws-arch/#whatis (2004).

[W3C 04c] World Wide Web Consortium. *XML Schema Part 2: Datatypes Second Edition (W3C Recommendation. October 2004)*. http://www.w3.org/TR/xmlschema-2/ (2004).

[W3C 06] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language (W3C Candidate Recommendation 27 March 2006)*. http://www.w3.org/TR/wsdl20/ (2006).

[Wallnau 01] Wallnau, Kurt; Hissam, Scott; & Seacord, Robert. *Building Systems from Commercial Components*. New York, NY: Addison-Wesley, 2001.

[WS-I 06a] Web Services Interoperability Organization. *About WS-I*. http://www.ws-i.org/about/Default.aspx (2006).

[WS-I 06b] Web Services Interoperability Organization. *Basic Profile Version 1.1 (Final Material 2006–04–10)*. http://www.ws-i.org/Profiles/BasicProfile-1.1.html (2006).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	June 2006	

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Model Problems in Technologies for Interoperability: Web Services	

6. AUTHOR(S)
Grace A. Lewis and Lutz Wrage

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
	CMU/SEI-2006-TN-021

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Web service technologies (or Web services) are experiencing a growing popularity in U.S. Department of Defense, industry, and non-defense government organizations due to their potential to enable interoperability between applications implemented on different platforms. This potential stems from Web services being based on standards that have been widely accepted and implemented, such as the Simple Object Access Protocol and the Web Services Description Language. The large number of products and tools created to facilitate the development of Web services has also contributed to their popularity. This technical note presents the results of applying the model problem approach in an initial investigation of the potential of Web services to enable interoperability.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Web services, services, service-oriented architecture, SOA, model problem, interoperability	36

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL