

BIOGRAPHICAL INFORMATION

Joseph Hershman
Technical Lead
Miner and Miner

Specific Responsibilities

Joined Miner and Miner in April, 2001. Responsibilities have included programming customizations, requirements gathering and analysis, and helping with object modeling and problem solving for major ArcFM/Designer implementations. In addition, he has helped develop, implement and integrate application software with oracle databases and work management systems.

Past Experience

Previous work experience includes development and support of a comprehensive environmental data management system and managing the implementation of GIS and automated mapping at an environmental consulting firm

Educational Information

B.S. – Engineering Physics, University of Illinois, Urbana-Champaign
M.S. – Civil Engineering, Colorado State University, Fort Collins

Professional Membership

Professional Engineer, State of Colorado

Dave Gallelli
Technical Lead
Miner and Miner

Specific Responsibilities

Mr. Gallelli joined Miner and Miner in March 1996 as an Application Analyst, designing and developing custom applications for several electric utilities. As a Project Manager he has lead the design, implementation and deployment of integrated M&M applications for domestic and foreign electric, gas and water utilities. Current responsibilities as Technical Lead include business process modeling, requirements analysis, interface design, data modeling, software configuration, general implementation guidance and rollout support.

Past Experience

Prior to joining Miner & Miner, Mr. Gallelli served as GIS Analyst for PAR Government Systems Corporation, participating in the development of an Environmental Monitoring and Management System, including design, software and hardware specifications, He also developed automated geo-referencing methods for digital aerial photography using GPS.

Previously, he served as a Research Physicist at the Army Research Laboratories, developing computer models of microwave radar systems.

Educational Information

B.S. – Physics, Drexel University, Philadelphia, PA

M.S. – Environmental and Resources Engineering, State University of New York, Syracuse, NY

USING WEB SERVICES TO INTEGRATE GIS INTO THE ENTERPRISE

Joseph Hershman
Dave Gallelli
Miner and Miner
4701 Royal Vista Circle
Fort Collins, CO 80528

ABSTRACT

Integrating geospatial applications with other enterprise systems allow organizations to realize greater returns on their IT investments. The difficulty is finding the means to facilitate this integration. Web Services answer these difficult challenges by allowing previously incompatible applications to interoperate regardless of language, platform, or operating system. The key ingredients, including XML, SOAP, and WSDL have been adopted across the entire software industry. A case study integrating a JAVA work management system and a COM GIS is described.

INTRODUCTION

One of the most difficult challenges facing businesses today is how to adapt to the rapid changes in technology. Further escalating the problem is that, not only do the individual systems require upkeep; it is often imperative that they be able to communicate with one another. Included in this list of systems is the enterprise GIS. One example of a business requirement that calls for seamless integration is between a workflow management system and a GIS based design and work sketch system. Various solutions to this integration problem will be reviewed. The advantages of the use of Web Services for enterprise application integration are described. The paper concludes with a real world implementation example.

BUSINESS PROBLEM

The modern corporation utilizes numerous information systems to accomplish its business objectives. A GIS, Customer Information System (CIS), accounting system, and workflow management system are just a few examples of the systems that a single corporation may maintain. Business processes typically require sharing of data between these various systems. Because these systems generally are not designed to communicate with one another, the business workflow tends to be cumbersome and inefficient. In order to streamline and automate these

workflows, the component systems should be integrated. The enterprise therefore requires a cost-effective and robust approach for integrating its various information systems.

Integration Problems

The integration of different software systems is complicated by numerous factors. It is not unusual that a single company maintain Unix/Linux systems and Windows based systems. Even within single operating systems there can be problems with integration. For example, a Windows system could be a Win32 application, a COM application, or a .NET application. Similarly a Unix/Linux system could be running native C++ applications along with Java applications. Even when the systems are based on the same technology they may not expose an application-programming interface (API) that allows simple system integration. Other complications include the varied databases and programming languages on which software systems are based.

Even after an enterprise has successfully integrated multiple systems the problem is not resolved. At some point in time each individual system will be upgraded. Installing these upgrades will often break the interfaces. Another problem to contend with is that a division of the company may find another software package that addresses their business needs better. These system migrations are sure to break any tightly coupled system integrations.

ENTERPRISE INTEGRATION ARCHITECTURE

Integration architectures can be categorized by the extent to which the integrated systems are dependent on (or coupled with) on the specific architecture or data structures of the other system(s) with which it is integrated. For simplicity we here divide these approaches into the broad categories of “tightly coupled” and “loosely coupled” architectures

Tightly Coupled Integration

Examples of tightly coupled integration architectures would include any approach that employs direct system-to-system API calls without the aid of intermediary middleware (such as an Object Request Broker - ORB). Also included would be any approach that involves one system directly accessing the data structures (database objects or data files) of another system. While the use of database links, gateways, and intermediate interface tables may reduce the strength of coupling, the systems are still strongly interdependent since (1) they depend on the existence of database objects peculiar to the individual system, and (2) they require that the remote database instance be running and accepting connections at all times.

The main advantage to the tightly coupled approaches is that they generally exhibit faster execution and better user response times than more loosely couple approaches.

The disadvantages are numerous:

- The use of direct API calls generally requires either that both APIs were developed with compatible software (e.g., both COM or both JAVA), or that some bridging technique be used to get the systems to communicate (e.g., java-com bridge). Development costs can be high depending on the extent of functionality to be shared between the systems. This is especially true if the API's are not directly compatible.
- The integration is highly sensitive to internal design changes in either system. Such changes could result from system customizations, system upgrades to new releases, or even data model changes.
- The combination of the above factors result in relatively high costs for maintaining the integrated systems.

Loosely Coupled Integration

Loosely coupled approaches to integration would include any approach that employs middleware to mediate communication between the systems. Two of the most commonly used types of middleware today are Message Oriented Middleware (or MOM) and Object Request Broker (ORB).

MOMs range from the simplest message queue to the more advanced message brokers. Message brokers are used to transform messages from the output format of the message sender to the expected format of the message receiver. Message brokers thus provide both routing and translation services. Popular off-the-shelf MOMs include IBM's MQ Series (recently incorporated into their WebSphere product offerings) and Oracle's Advanced Queuing and Message Broker products.

An ORB is middleware that transparently brokers client requests for services from distributed objects. They allow client programs to obtain services from remote servers in a language-, platform-, and location-independent manner. Some ORBs have implemented a Publish-Subscribe model, whereby objects publish events that other remote objects can subscribe to. The ORB handles event notification transparently. ORBs are more sophisticated than message brokers in that they can provide many additional services, such as object naming, transaction management, security, etc.

The advantages to the use of such middleware are many:

- Middleware reduces dependencies between systems; the loose coupling better facilitates upgrades/replacements/migrations of individual systems. When a component system of such an integration architecture is replaced the remaining systems participating in the integration need not be modified (at least in principle) to accommodate the newcomer. (Of course, the middleware components will have to be reconfigured, and the new system will still need to be configured or customized to fit into the existing integration architecture.)
- Since foreign APIs are not directly used by individual systems, there are no language dependencies – for example, COM apps can easily converse with JAVA apps. Also, depending on the middleware used, there may be fewer platform/OS dependencies as well.

- These loosely coupled architectures are more easily extended than their tightly coupled counterparts because new systems can be incorporated into the architecture with no (or minimal) changes to the existing systems.
- Maintenance costs tend to be less because the same infrastructure can often be used to integrate many systems. Development costs also tend to be cheaper since off-the-shelf products are used for data transformation and transportation.

Enterprise Integration Environments

Most enterprises still find themselves in an environment where many individual systems participate in one or more point-to-point integrations with other systems. Note that the individual integrations might be either loosely or tightly coupled, and any of the above architectures might be in use for a particular integration. The integration environment tends to be heterogeneous in nature, with many varying interfaces to be supported. Maintenance costs are subsequently high.

A few enterprises have established integration environments where a centralized integration bus serves as the hub for integration of all participating systems. Such “hub-and-spoke” environments might be found in new start-up companies recently bringing their systems on-line, or those established enterprises with IT budgets large enough to finance the migration of their legacy point-to-point interfaces to a homogenous and centralized integration environment. This arrangement necessarily requires a loosely coupled approach to the integrations, generally employing sophisticated commercial middleware as the core of this architecture. A less expensive approach to providing a very loosely coupled integration between multiple systems in a hub and spoke type architecture would use a Web Service as the central hub. The Web Services offers much simpler implementation the existing products along with complete language and platform neutrality.

WEB SERVICE INTEGRATION

Services

A service is a contractually defined behavior. Service oriented programming differs from standard programming in that it relies on schemas and contracts rather than structure and behavior. A schema defines what a message looks like (i.e., its structure). A contract defines the exposed methods the service presents.

Web Services

A Web Service is a service that is accessed via the Internet (or Intranet). Web Services communicate using ubiquitous Web protocols and data formats such as HTTP and XML. Any system supporting these Web standards will be able to support Web Services

A Web Service contract describes the services provided in terms of the messages the Web Service accepts and generates rather than how the service is implemented. By focusing solely on

messages, the Web Services model is completely language, platform, and object-model agnostic. This allows a Web Service to be implemented using the full feature set of the programming language, object model, and platform best suited to solving a particular business problem. The key to Web Service interoperability is its reliance solely on Web standards.

However, simply agreeing that Web Services should be accessed through standard Web protocols is not sufficient to make these services easy to use. Web Services become easy to use when the service provider and consumer rely on standard ways to represent messages and contracts. The Simple Object Access Protocol (SOAP) is an industry standard for using XML to represent data and commands in an extensible way. A Web Service can choose to use SOAP to specify its message formats. XML is also the enabling technology for the Web Service contracts. Web Services Description Language (WSDL) is an XML grammar for documenting Web Service contracts. The document describes a Web Service by specifying the location of the service and the operations (or methods) the service exposes.

CASE STUDY

Project Background

The client is a moderately sized electrical utility in the southeastern US. The production environment included an existing point-to-point integration of Logica's Work Management Information System (WMIS) with Miner & Miner's Job Planning and Design Environment (JPDE). The integrated system had been used in production for about 4 years and was approaching obsolescence.

Both WMIS and JPDE were using Oracle 7.x as the back-end database. There were separate database instances for each system. The existing, tightly coupled integration between the systems was accomplished using PL/SQL calls and DB links across database instances. Custom integration code had been written for both systems.

In 2002 Oracle dropped support for the 7.x versions of their database server. Thus the client was forced to either migrate to a higher version of Oracle or lose vendor support. Since the client was also several versions behind the latest releases of both WMIS and JPDE, the decision was made to upgrade to the latest releases of both systems.

Project Objectives

The project objectives were to:

- Upgrade Oracle to maintain Oracle support
- Take advantage of new functionality of WMIS 2.7 while maintaining integration with electric facility database (GIS)
- Take advantage of new functionality of Designer 8.3 (an object-oriented re-write of JPDE) while maintaining integration with WMIS

- Minimize customization of core WMIS and Designer products (to better facilitate future upgrades of either system)

System Responsibilities

Logica's WMIS 2.7 was to be used to satisfy the core business requirements for job creation, job cost estimation, design review and approval, and as-built updating. It was also to be responsible for managing and controlling the workflow of both systems.

Miner & Miner's Designer 8.3 would be used for graphic design creation, work sketch creation, electrical design calculations and analysis, and maintenance of the electric distribution facility system of record (the GIS).

Integration Requirements

The integration requirements were to:

- Share design information between systems
- Share As-Built information between systems
- Synchronize definitions of construction units in both systems
- Synchronize workflow status in both systems
- Enforce the workflow across both systems

Design Goals

The primary design goals for the integration architecture were:

- An open framework that would allow systems of various types (languages and platforms) to communicate. This would allow us to re-use the integration architecture on future projects.
- An architecture that would be extensible so new components could be added without changing the core system. Extensibility was important because we needed the ability to augment the business processing components if necessary.
- An inexpensive and straightforward implementation.

Hardware and Software Environment

The GIS maintained by Designer includes an ArcSDE database running on top of Oracle 9.2.0.3.0. The data server is an enterprise class E4500 SUN server. Designer itself runs in a Windows environment and is an extension of ArcFM, which is in turn an extension of ArcGIS.

The WMIS database is running under Oracle 8.1.7.4.0 and resides on a separate SUN Enterprise class data server. The WMIS client application runs in a windows environment.

Since the end-users are distributed across multiple remote offices, both WMIS and Designer client applications are accessed via WAN connections to Citrix application servers.

Vendor support for Integration

WMIS 2.7 incorporates a fairly robust and configurable integration framework. This framework provides both Java and PL/AQL APIs. It also provides support for sending and receiving XML messages via either IBM MQ Series or Oracle Advanced Queuing message queues. Further, WMIS supports both synchronous (request/response) and asynchronous (send and forget) modes of messaging.

Designer 8.3 incorporates a COM API with interfaces for importing and exporting Designs and Construction Unit libraries as XML documents or as DOM (Document Object Model) objects. However Designer provides no interfaces for interacting with message queues.

Solutions Considered

We briefly considered using a Java-COM bridge to invoke the WMIS Java API from custom COM code. However the client had no budget for WMIS customizations, and we were concerned with the potential cost and limitations of this bridging approach. In addition there were all the caveats already discussed regarding such strongly coupled integrations.

We also considered making calls to the PL/SQL API from our COM code. However, this would have required a connection to the remote (WMS) database instance, raising administration and security concerns. Finally, the client had experienced some pain with the PL/SQL API in the previous JPDE/WMIS integration.

For these reasons we limited our options to those employing the out of the box (OOTB) messaging support of WMIS. We further eliminated the use of MQ Series as the messaging service since the client had no existing MQ license (or budget for one), while Oracle AQ is included in the client's Enterprise Edition license of Oracle Server.

We rejected the use of synchronous (request/response) messaging for two reasons: (1) there was the potential for slow system response and increased user wait time, and (2) the request/response mode is dependent on the WMIS system being available at all times. System down time could result in messaging errors, thus interrupting the workflow in Designer.

We therefore chose to implement asynchronous messaging, whereby outbound messages are triggered by workflow events in one or the other system. In this loosely coupled messaging mode the sending system does not wait for a response from the target system. It merely places the message on an outbound queue and continues with its internal processing, regardless of the status of the receiving system.

Selected Integration Architecture

The integration architecture finally selected was that of a simplified message broker. The entire integration is depicted in Figure 1, while the architecture of the message broker is summarized in Figure 2. The message broker consists of:

1. A message router web service, responsible for routing messages to appropriate, message processors and translators

2. A message monitor windows service, responsible for retrieving messages from the inbound (from WMIS) message queue and passing them to the message router

In this solution, XSLT is used to translate messages from the message protocol of the origin system to the message protocol of the destination system. The actual transport of messages into and out of the WMIS message queues is accomplished using Oracle Message Propagation. Messages transported out of Designer use a Microsoft provided component designed to allow COM applications to access Web Services. Messages transported into Designer perform direct updates of the GIS database via stored procedures.

For this particular implementation, seven individual messages were configured to handle all interface points included in the design workflow. A custom message processor (described below) was built to handle each message. The message processor is responsible for providing all the business logic required by a specific message type. Examples of business logic include: (1) transforming messages using an xslt transformation and places the message on the inbound WMIS queue, (2) comparisons of final construction design and as-built designs and processing various discrepancies based on client specific rules.

Scalability of the Architecture

The architecture depicted in Figure X is highly scalable. Additional web services, message monitors, and message queues can be easily added and configured to accommodate increased transaction loads.

Extensibility of the Architecture

The message-processing model was designed to take advantages of features in the .net framework. The model uses a component known as the message processor to act upon each message. Every message has a unique message processor. The business logic contained within a processor could be as simple as sending the message to another Web Service or it could apply complex business rules and database updates. The .net framework allows a simple method to create these processor components based on information read from a text file. This ability makes the application extensible because new message processors can be added at any time without modifying the existing system. All that is required is that the configuration file be updated to make the Web Service aware of the new message processor.

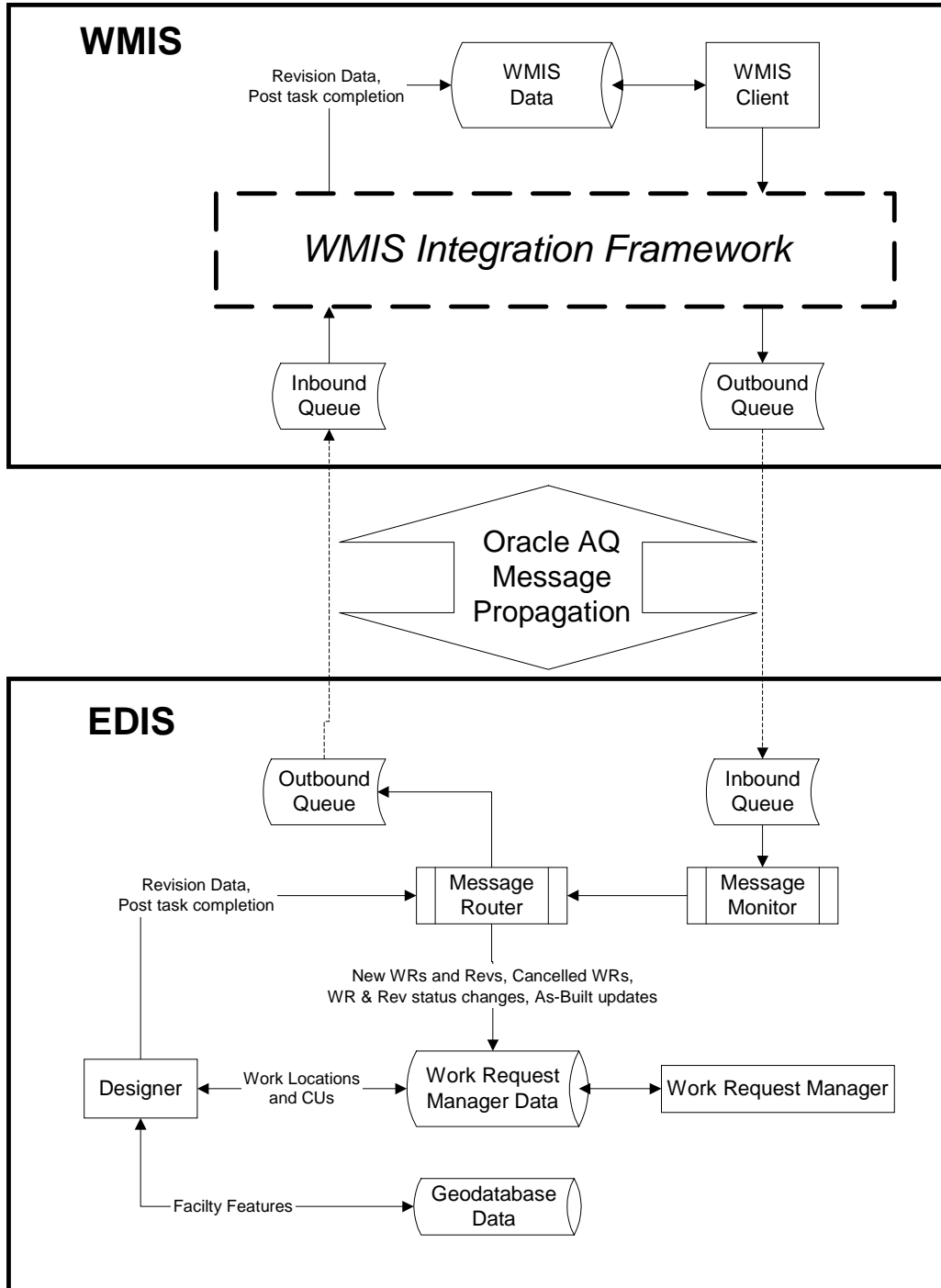


Figure 1. Architecture of the entire integrated system.

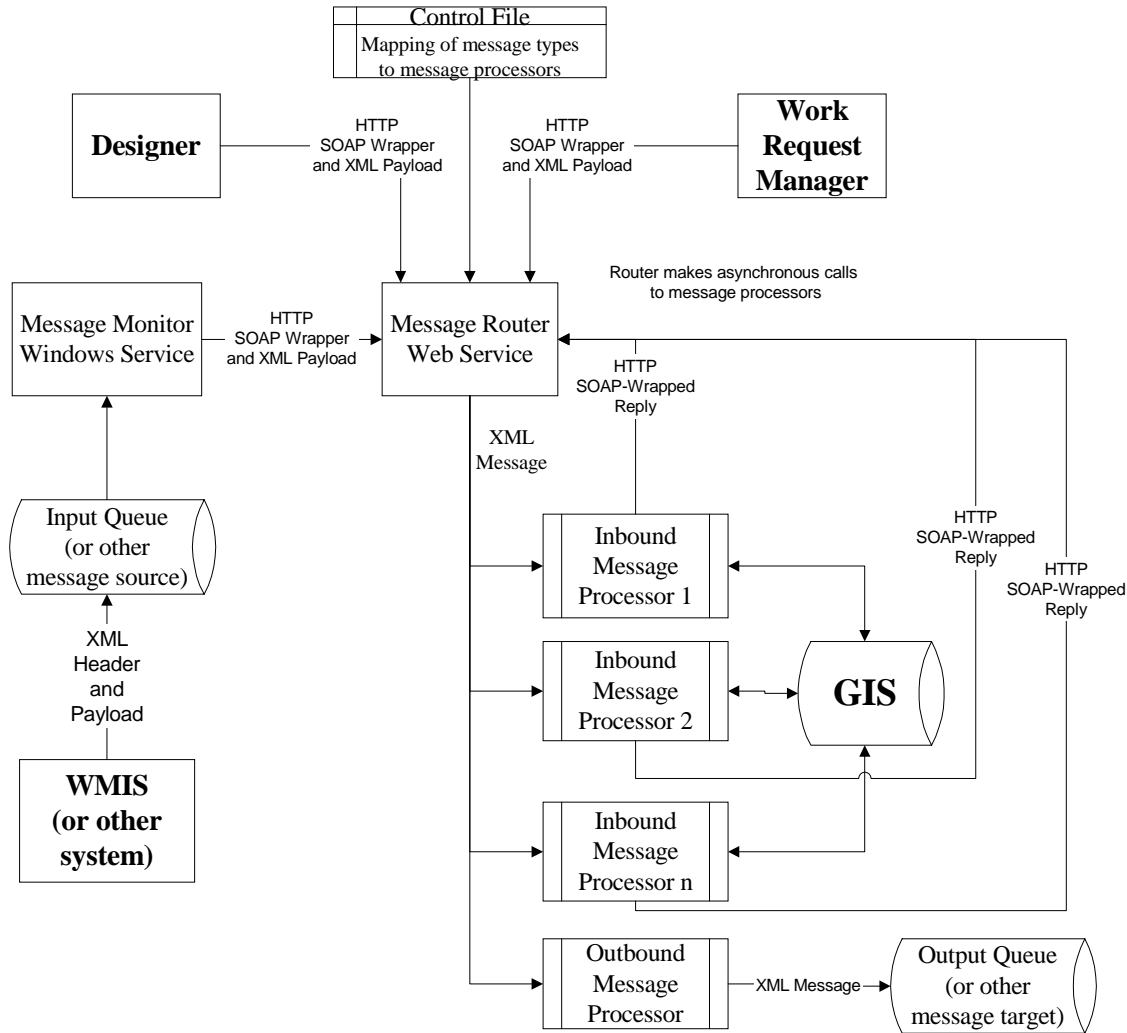


Figure 2. Architecture of the message broker

CONCLUSION

The paper presents a methodology for using a Web Service to simplify the integration of two disparate systems into a seamless workflow. This architecture could be used to support integration between a GIS and other systems such as Customer Information, Outage Management, Accounting, etc. The framework could also be extended to support complex workflows that involve many different systems.

It is not required that the additional systems support third-party messaging middleware, since these other applications can make direct requests to the web service via SOAP/HTTP. The web service would only need to be extended with additional methods, message processors and revised

service contract (as specified in the WSDL) to support the new requests. Integration with external systems (Business to Business) is also possible.

Finally, this architecture facilitates the incorporation of the systems into future Enterprise Application Integration architectures. In this scenario the existing interface components would not have to be rewritten. The systems could connect to the integration bus via the existing web service. The translation services of the bus could then be configured to handle the messages currently supplied by the web service.

REFERENCES

W3C Note, 15 March 2001, Web Services Description Language (WSDL) 1.1

W3C Note, 08 May 2000, Simple Object Access Protocol (SOAP) 1.1

Kirtland M., 2000, The Programmable Web: Web Services Provides Building Blocks for the Microsoft .NET Framework: MSDN Magazine,

Oracle, November 2002, Database Web Services An Oracle White Paper.

Linthicum, D.S., 1999, Enterprise Application Integration.

searchdatabase.techtarget.com

searchwebservices.techtarget.com