Chapter 7

Extension of EAI Framework to Services Oriented Architectures

7.1 Introduction

The concept of Web services and the components Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI) definitely provide an environment within which it is possible to design a flexible approach for EAI. Present Enterprise Application Integration solutions are usually tightly coupled, proprietary, and are not easily amenable to changes. In compression with this, an easy and scalable EAI approach can be woven around the concept of web services and its components.

In chapter 6 we have seen the flexibility and scalability offered by Services Oriented Architectures (SOA) and its components for EAI approach. A complete architecture as per our proposal for EAI is in place already as was shown in fig. 5.3. We propose a strategy to extend this architecture around the concepts of web services for EAI. Apart from what is shown in fig. 5.3, due to web services, the Legacy System, through central server (Registry), is made available to outside world in this proposed strategy.

7.2 Present Scenario

Fig. 7.1 shows existing scenario of legacy systems (LS) belonging to an enterprise. These LS are self-sufficient and are able to cater local needs. These are complete monolithic softwares working on different platforms, which include hardware (h/w), Operating System (OS), language for user interface, and DBMS. A software development team comprising of programmers and the legacy site management contributed for the software development.
Depending upon language expertise of programmers and interests of the management a decision was taken about h/w, OS, language and DBMS. The decision of software development team of a legacy system was typically completely independent of decision of another software development team of another legacy system belonging to the same enterprise. This has resulted in a heterogeneous environment as far as the enterprise is considered. The advent of Internet and new requirements from existing top level management like collective reports comprising enterprise wide data, distributed transactions etc. necessitates information / data exchange across legacy sites in absence of human interference. This necessitates direct communication between legacy systems for data exchange, which is absent currently.

7.3 System Architecture

Fig. 2 shows System Architecture. Apart from pre-existing software components viz. Legacy User Interface and Legacy DB, the architecture introduces four new components viz. Local Interface & Transaction Manager (LITM), Message Formatter/ Converter (MFC), SOAP Interface and Central Server. The LITM is divided logically in three sub-parts. First part of LITM is specific to Legacy System and varies from legacy system to system. Second part works as interface with other components of architecture.
Third part of LITM is trusted with transaction management activities for distributed transactions and communicating the successful local as well as distributed transactions to Data Warehouse.
The MFC uses WSDL and UDDI. This is responsible for forming message body part as per WSDL specification and searching necessary services for distributed transactions using UDDI.

SOAP interface component is responsible for creating SOAP envelop, Header part for the message body received from MFC for outgoing messages and detaching message body part from incoming messages for use by MFC.

The Central Server does three functions. Firstly it maintains registry of web services offered by different legacy sites within enterprise for internal use and web services which can be offered to outside world. Secondly it helps SOAP messages to hop. Finally it helps in creating/ maintaining data warehouse for the enterprise by posting successfully completed local/ remote/ distributed transactions to Data warehouse. These three parts viz. Registry, Intermediary and DWH are grouped together conceptually, however they may reside on a single server or a set of servers with dedicated functionality.

The first part of LITM is non-portable, specific to the local legacy system and varies from legacy system to system. All other components are completely portable and hence need to be developed only once for an enterprise.

7.4 Implementation Details

Implementation of the strategy is two folds. One, bring a remote legacy system in the network for remote/ distributed transactions. Second, the functionality of various components shown in the fig. 7.2 of system architecture.

I. Bring Legacy system in Network:

To bring a legacy system in network following steps are needed.

Step 1: Service creation
The system is already in place for local services. The legacy system software is modified to certain extent so as to generate triggers/events to initiate first part of LITM on occurrence of local (legacy) database updates. Identify services that need to be part of EAI. This may involve rewriting code to certain extent or creating new services, e.g. a newly added service for electronic exchange of money from one branch to another branch in case of a banking application. Also identify services which can be offered to external world.

**Step 2: Generate WSDL documents:**

WSDL documents are created for the services that have been identified in step 1. Software tools and Integrated Development Environment (IDEs) may be used in generating WSDL document.

**Step 3: Register Services in the Registry:**

With the help of a software tool, known as Registrar, register the services in the registry. This is used to publish the service information. The registrar is typically a newly developed application that is deployed to take care of the service publication tasks. To avoid creation of metadata repository, containing information of data structures of all legacy systems, it is recommended that all legacy systems offer services through RPC only.

**II Functionality of Various components:**

A legacy system is taking part in two ways, one, It requests for data/service from another legacy system (working as a client) and two, It provides data/service to another legacy system (working as a service provider). The roles of components shown in system architecture, thus, are in two ways. Also mode of communication between these components may be either synchronous or asynchronous. In case of asynchronous communication, logs can be maintained between every two components.

**A> LITM:**

**Client:** The local interface part, i.e. first part of LITM accepts data/service request from legacy system and initializes second part of LITM. Remaining part of LITM makes an entry into its transaction management log and then hands
over the request to MFC for further processing, either directly (in case of synchronous communication) or through a separate log (in case of asynchronous communication).

In response scenario, it accepts response for a request made earlier, makes appropriate entry against earlier entry and hands over data to legacy system.

**Service Provider:** It receives a request for data/service from remote legacy system. Through its first part hands over the request to local legacy system for processing, makes appropriate log entry.

In response scenario, when processing is complete and data is received from local legacy system, it hands over response to MFC. While doing so, it makes appropriate log entries in its transaction management log for follow-up inquiries.

**B> MFC:**

**Client:** Receives a request from LITM, finds the legacy system which can provide this service, finds the RPC and parameters required for the RPC using UDDI. Converts this RPC call into body part of message sticking to XSD standards agreed enterprise wide. Finally hands over this XML file to SOAP Interface.

In response scenario, it receives XML format message body part from SOAP interface. Separates the data part from message body and hands over to LITM.

**Service Provider:** Receives XML format message body part from SOAP Interface. Analyses it and separates the required RPC and its parameter values. Hands over them to LITM.

In response scenario it receives data from LITM, converts it into body part of message and hands over to SOAP Interface.
C> SOAP Interface:

Client: Receives request from MFC in the form of message body part. Converts it into appropriate SOAP message by adding envelop and header, binds it to transport protocol and sends it over wire.

In the response scenario, it receives a SOAP message in response to a request, detaches message body part and hands over to MFC.

Service Provider: Receives a SOAP message from remote Legacy System, detaches message body part and hands over to MFC.

In response scenario, receives data in message body part from MFC, adds envelop, header and makes it a SOAP message and sends over the wire.

D> Central Server:
The central server provides three functionality viz. Registry, Intermediary, and DWH.

(a) Registry:

It is repository of services being offered by various legacy systems of the enterprise. As shown in the architecture, it is deploys on central server. The deployment of the WSDL files of various legacy systems will contain information about services offered by them in the form of name(s) of procedures, which can be called remotely, alongwith necessary parameters.

(b) Intermediary:

It is an application that enables SOAP messages to hop. Unlike usual intermediary, who do not act on SOAP message, this intermediary is expected to act on few SOAP messages. The SOAP messages that are acted upon by this intermediary are the ones, which contain information of successfully completed local/remote/distributed transactions. Such messages are sent to DWH part of the central server.

(c) DWH:

This part of central server receives SOAP messages from intermediary. The successfully completed local/remote/distributed transactions are posted to data warehouse database by this application.
Apart from above functions, the central server registry also contains information about services offered by various legacy systems of the enterprise to the outside world. Any external client may access these services once an agreement is reached between the client and the enterprise.

7.5 Summary of Work

In this chapter we have proposed a web services based approach for EAI. This approach presents an easy, scalable and loosely coupled EAI, as against current approaches that are tightly coupled, proprietary, and not so amenable to changes. This approach makes good use of strengths of web services components viz. SOAP, WSDL, and UDDI. The approach, apart from EAI gives opportunity for enterprise to offer its services globally. Also data warehouse, which is created on central server, and frequently updated, provide good reporting facility to the top management to find status of enterprise business, performance-related issues and other necessary statistical information.

A paper based on this work titled “Web Services based Strategy for Enterprise Application Integration” has been submitted to IADIS International Conference WWW/Internet 2005, Lisbon, Portugal, 19-22 October 2005.
Chapter 8

Conclusions

The problem of Integrating Applications across an Enterprise has raised many serious issues that researchers have been tackling in the recent years. In this effort, legacy applications pose a very major roadblock. On one hand, these applications are running today and are carrying out important business functions. On the other hand, these are based on some very old technologies, are poorly documented, and are difficult to maintain.

This thesis has attempted to look at the problem of reengineering these legacy applications in the perspective of Enterprise Application Integration (EAI). After carrying out an exhaustive survey of available papers in this area, we have proposed a new framework, which requires making minimal changes to the legacy application so as to just expose the main interfaces of that application to the modern integration tools based on persistent messaging. Thus in our proposed reengineering scheme, these interfaces, traditionally known as APIs, are created by identifying the main transactions of the application, as also the access point invocations. These APIs are wrapped within a suitable object wrapper, and the wrapper software is then exposed to the EAI framework.

The salient features of our approach are as follows:

(i) Minimal changes are required to existing legacy application.

(ii) The object wrapper software, together with the intrinsic data management of the application, provide the necessary properties of data integrity and persistence.
(iii) True distributed transactions can be created using the proposed framework.

(iv) Once the reengineering is completed, the legacy application becomes a part of the Integrated System and yet can be independently maintained as long as the wrapper APIs are preserved.

In support of our proposed framework, we have carried out exhaustive experimentation. For this purpose, we have built a comprehensive experimental setup with four different servers supporting J2EE application server, a FoxPro legacy application, a DB2 application and a data warehouse server using Oracle, respectively. This work is reported in Chapter 5. As can be seen, we have been able to demonstrate the seamless interaction between legacy applications and new applications within the J2EE framework.

Services Oriented Architectures (SOA) is fast emerging as the new integration standard. Recognizing this, we have studied the main features of this architecture, and then we have further extended our framework to comply with this architecture. This work is reported in Chapter 7. We have shown that this extension permits our earlier reengineering approach to further enhance the capability of legacy applications to publish their transactional and data access interfaces to the outside world. This, in our opinion, significantly enhances the value of these legacy applications, as even systems external to the enterprise can now invoke these through the Publish/Subscribe model and the discovery mechanism inherent in the emerging SOA scenarios.
We have published one paper in an international conference [56] and have already submitted another based on our work [57].

Scope for Future Work

While we believe that our work provides a significant contribution in the field of EAI vis-à-vis the inclusion of legacy applications, we are aware of the potential for further work especially in the context of emerging standards under SOA framework. Very specifically, there is a need to provide an intrinsic SOAP messaging capability directly within the legacy applications. There would also be issues of performance that need to be resolved if these legacy applications are required to perform on-line operations triggered by a user on an internet client.

We propose to continue our research in future to address some of these issues.