INTEGRATING CONTACT MANAGEMENT USING
SERVICE-ORIENTED ARCHITECTURE

A Project
Presented
to the Faculty of
California State University, Chico

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Sheeba Jose
Spring 2009
INTEGRATING CONTACT MANAGEMENT USING
SERVICE-ORIENTED ARCHITECTURE

A Project
by
Sheeba Jose

Spring 2009

APPROVED BY THE DEAN OF THE SCHOOL OF
GRADUATE, INTERNATIONAL, AND INTERDISCIPLINARY STUDIES:

_________________________________
Susan E. Place, Ph.D.

APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

Abdel-Moaty M. Fayek
Graduate Coordinator

Anne M. Keuneke, Ph.D., Chair

Seung-Bae Im, Ph.D.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>Abstract</td>
<td>vi</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>I. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Objectives</td>
<td>4</td>
</tr>
<tr>
<td>Scope</td>
<td>4</td>
</tr>
<tr>
<td>An Outline of the Report</td>
<td>5</td>
</tr>
<tr>
<td>II. Background</td>
<td>6</td>
</tr>
<tr>
<td>Problem Domain</td>
<td>6</td>
</tr>
<tr>
<td>Integration Strategies</td>
<td>7</td>
</tr>
<tr>
<td>III. Web Service Implementation</td>
<td>22</td>
</tr>
<tr>
<td>Introduction</td>
<td>22</td>
</tr>
<tr>
<td>CMS Web Services</td>
<td>23</td>
</tr>
<tr>
<td>Web Service Implementation and Deployment</td>
<td>24</td>
</tr>
<tr>
<td>Web Service Access by Contact Client</td>
<td>31</td>
</tr>
<tr>
<td>IV. Conclusions</td>
<td>41</td>
</tr>
<tr>
<td>Summary</td>
<td>41</td>
</tr>
<tr>
<td>Roadblocks</td>
<td>43</td>
</tr>
<tr>
<td>Enhancements</td>
<td>45</td>
</tr>
<tr>
<td>References</td>
<td>47</td>
</tr>
</tbody>
</table>
Appendices

A. Use Cases for Contact Management System ............................................ 53
B. Project Source Code ................................................................................. 78
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Point-to-Point Architecture</td>
<td>11</td>
</tr>
<tr>
<td>2. Hub-and-Spoke Integration Architecture</td>
<td>12</td>
</tr>
<tr>
<td>3. Enterprise Service Bus Architecture</td>
<td>15</td>
</tr>
<tr>
<td>4. GetContactService</td>
<td>26</td>
</tr>
<tr>
<td>5. Web.xml Entry</td>
<td>27</td>
</tr>
<tr>
<td>6. WSDL Generation</td>
<td>29</td>
</tr>
<tr>
<td>7. Property File Entries</td>
<td>31</td>
</tr>
<tr>
<td>8. Search Contact</td>
<td>33</td>
</tr>
<tr>
<td>9. Display Name</td>
<td>34</td>
</tr>
<tr>
<td>10. PerformAction Method</td>
<td>35</td>
</tr>
<tr>
<td>11. GetContactByID</td>
<td>36</td>
</tr>
<tr>
<td>12. CC – CMS Interaction</td>
<td>38</td>
</tr>
<tr>
<td>13. Get Contact Name</td>
<td>39</td>
</tr>
<tr>
<td>14. Display Name</td>
<td>40</td>
</tr>
</tbody>
</table>
ABSTRACT

INTEGRATING CONTACT MANAGEMENT USING SERVICE-ORIENTED ARCHITECTURE

by

Sheeba Jose

Master of Science in Computer Science
California State University, Chico
Spring 2009

Enterprise application integration can be defined as a set of technologies and the use of software to integrate applications. This project uses open standards and the principles of Service-Oriented Architecture (SOA) to demonstrate application integration.

Applications in an enterprise are designed to work in silos. Each application provides capabilities and functions to satisfy specific requirements. But, the information within an enterprise needs to be made available across these silos—hence the need for loosely coupled, less invasive integration strategies. This project demonstrates how SOA can be used to integrate two disparate applications.

The project consists of two applications: a Contact Management system (CMS), which is a Web application developed in J2EE, and a Contact Client (CC)
application developed using Java Swing. The CMS system manages contact information of an enterprise. This system also provides services as Web services that applications can use to retrieve and create contact information. The client application uses these Web services to retrieve a contact by a unique contact identifier and to create new contacts.
CHAPTER I

INTRODUCTION

Employees and customers are two important entities of any organization. They play a vital role in the day-to-day business activities and operations of an organization. Employees need to interact with each other for fulfilling their job responsibilities and also for managing various operations within the organization. They also need to be in constant touch with clients and customers in order to transact business. They primarily focus on availing the products or services to the customers in a timely and efficient manner. This case study focused on a small subset of data that is widely used within an organization – contact information for both employees and customers. This includes names, addresses, email addresses, and telephone numbers of contacts. The process of organizing and storing the contact information plays a significant role in managing the organization’s operations successfully as this information is required frequently for carrying out various operations of the organization.

The focus of this case study is to demonstrate how organizations can maintain a “single source of truth” for master data – data common throughout the enterprise. The advantage of having a single source of truth is as follows:

- A single documented source for information and data.
- Accurate and reliable data.
• Avoids duplication of data and the need for different systems to maintain the same information.

• Avoids the need to cleanse and validate data.

• Ownership of data within an organization. This enables governance and controls to be put in place so that the quality of data can be maintained and enforced.

In most organizations, various departments have their own software systems or applications which store the contact information for the customers locally, but not in a centralized manner. Let us discuss this with an example. John Smith works for a bookstore called BooksVault, Inc. BooksVault sells fiction and comedy and has a presence on the internet. Customers can either submit orders online or purchase books from the store. About sixty percent of all orders are submitted online. The process of buying a book is simple for the customers visiting the store. The customer pays for a copy of the book, if it is available. Otherwise, he/she can place an order online and has a choice to either pick up the book later or have the book shipped to him/her to an address of his choice.

John Smith manages the Internet store. He tracks customer orders and ensures that the shipping department ships the orders to the customers as soon as the books are available. Customer information is needed by other systems used at BooksVault, including the CRM (Customer Relationship Management) [1] system. The online storefront is not integrated with the CRM system as BooksVault, Inc. does not have an integration infrastructure. The storefront application uses Microsoft’s .NET [2] technology and the CRM application runs on a PowerBuilder-Sybase [3] client-server environment. Both applications allow data to be exported to flat files, but have no inbuilt
integration capabilities and were bought from two different vendors. From an application vendor’s perspective, these applications are designed to work in their individual silos. Extracts from the online system are generated on a weekly basis and then uploaded to the CRM system.

John reviewed the various systems with the IT (Information Technology) department and identified four different systems that maintained contact information. He has asked the IT team to propose a strategy to store customer information centrally and to have all the systems integrated, so that different systems can share the customer information. This approach will definitely help them in reducing the cost of operations and also in managing the operations and resources more effectively.

In an organization, duplication of contact data can occur, if multiple systems store this information separately. Human Resources would own and manage employee records using a system that is most suited to their needs. The payroll department would have their own application for payroll processing. When a new employee joins the firm, both the human resources department and the payroll department will need to update their systems separately. This process involves a lot of manual efforts which can be avoided by storing the data once and integrating them across systems.

The challenge within an organization is to collect and gather all the entities that an organization interacts with, in a centralized manner. The purpose of Contact Management System is to enhance the process of collecting contact information for both customers and employees into a central repository and integrate the data with other systems. This is necessary to avoid the effort involved in re-keying the same data in different systems. This helps in saving time, improving data accuracy, and reducing the
cost of operations. This approach saves both manual and material resources and helps in better management of operations and business processes of an organization. The ultimate goal of this system is to enhance the overall productivity of the organization.

Objectives

One of the goals of this project work was to demonstrate how Service-Oriented Architecture (SOA) [4] can play a vital role in integrating diverse applications. This project is an attempt to take a very common business problem domain, provide a solution for the problem, and demonstrate how applications within an enterprise can be morphed into becoming service providers and consumers.

The Contact Management System provides a service to an enterprise – it manages contacts. This system is a consumer, as users who interact with the system have the ability to maintain contacts within the system. The system also acts as a provider by providing contact information to other systems that need it, through an integration platform – Web Services [5].

To recap, the main objectives of this project work are as follows:

1. Provide a solution to a common business problem within an enterprise.
2. Make the solution available to other applications within the enterprise.
3. To demonstrate how SOA can provide an integration platform using industry-based standards and protocols [5], [6], [7], [8].

Scope

Maintaining contacts is a common business requirement in most organizations. An example of this requirement would be the need to maintain a list of all
the employees within an organization for the Payroll department. A typical organization will have multiple applications. Many of these applications will use data from other systems. This project provides an approach to consolidate contact information within an organization and provide an integration platform to provide access to this centralized data to other applications that need it.

The complexity of managing contacts within an enterprise varies depending on size and nature of the business transacted by the organization. The complexity arises from the range of relationships and the volume of data to be maintained. This project work assumes a contact management system that is not very complex and has relatively low volume of data. The application designed had not been subjected to any performance benchmarks or load tests. The application is designed using prevailing industry standards and best practices [4], [5], [6], [7], [8], [9].

An Outline of the Report

Chapter II contains information regarding the research and study conducted for this project. Also, this chapter provides information regarding some of the similar integration technologies. Chapter III gives an overview of the Web service implementation and the interaction between the integrated applications. Chapter IV provides a summary of accomplishments and roadblocks occurred during the course of design and development of this project. This chapter also lists future improvements and enhancements.
CHAPTER II

BACKGROUND

The research and study conducted for this project is based on the need for integration of applications within an enterprise, by providing a concrete example of managing contact information of business entities in a centralized manner. This chapter is organized by setting the stage with a problem domain, followed by a review of various integration strategies that could be used as potential solutions. This chapter also compares and contrasts the merits and demerits of the identified solutions and finally describes SOA as an integration strategy that organizations can leverage to build a scalable and agile integration strategy.

Problem Domain

Contact management is one among the most common functionalities used in any business organization. Mostly, this is a part of Customer Relationship Management systems (CRM) [1]. The primary objective of a CRM system is to retain existing customers and quantify the value of services provided to customers by an organization. The subject of this study was to identify the potential reuse of business entities (contacts) within an organization. The challenge within an organization is to manage all the entities that an organization interacts with, in a central location, and integrate these data with other systems that need this information. The topic selected for this project is integrating
contact information for both employees and customers as this is probably the most common information that businesses need across various systems and applications. In less mature organizations, this kind of information is maintained manually in spreadsheets and in mature organizations; these data are maintained in CRM systems. For this project, a custom Web application (Contact Management System) is used to maintain contact information. This application demonstrates the ability to maintain data in a centralized manner and to integrate with other systems. To demonstrate the integration capability a Java swing-based application (Contact Client) is used.

Integration Strategies

Let us review the importance of information exchange and data sharing among applications. Applications, in general, are designed to perform very specific business functions or capabilities. An organization would have many divisions and business units that cater to different business functions. The HR department would manage the human resources organization, the Sales division would manage inventory and sales, the accounting department would manage the financials and so on. For one single application to manage all of these functions would be difficult because of the enormous scope and breadth of the functionality that is required. This is the reason why Enterprise Resource Planning (ERP) [10] solutions are expensive and cost prohibitive to implement for small businesses. Thus a single application is unlikely to provide all the varying business needs within an organization, and hence the need for multiple applications. Off-the-shelf applications are designed to meet generic requirements so that
the vendor can sell their product to multiple customers, and may not address very specific business requirements within an organization.

Applications may retrieve information from other systems for delivering capabilities they are designed to perform. Applications need to share data and information for managing the business operations within an organization. Without integration, each system would be independent and would need to capture data and information separately in silos, even if the same information exists in other systems. This would result in duplication of efforts to capture the same information which in turn, would increase the overall cost of operations. It can also lead to inconsistent data across systems. The primary benefit of application integration is reuse – reuse of data or services. Thus, improved business agility and increased productivity can be achieved by enabling systems to share information. Application integration can be divided into the following high-level categories:

- Information exchange or data replication. In this type of integration, data is usually replicated from one system to another. A large part of integration efforts would fall into this type.

- Service or functionality. In this type of integration, one application takes advantage of the functionality provided by another application, as a service or capability.

- Process integration. In this type of integration, an EAI (Enterprise Application Integration) [11] tool is used to combine multiple services from multiple applications to control and to initiate services in the desired sequence.

The focus of this case study is on information exchange which is the most common type of integration. One of the important features of a good integration strategy
is that the systems or applications involved in an integrated environment should be loosely coupled to each other. In order to facilitate loose coupling, the applications should not be affected by the changes in technology, processing time, data representation or processes of other applications [12], as mentioned below:

- **Technology.** Applications should not be required to know about the technologies used by other applications. Applications should be able to change its technology without affecting others.

- **Processes.** Applications should be independent of processes taking place in other systems. Applications should be allowed to change their processes without affecting others.

- **Data.** An application should not depend on data representation of other applications and its changes should not affect other applications.

The loosely coupled integrated systems would ease the upgrade and maintenance of systems and reduce the cost of operations. There are different scenarios or architectures that can be used for data transfer among different applications. Different integration strategies and their features [13], [14], [15] are described below.

**Point-to-Point Architecture**

Traditionally, point-to-point architecture [13], [14], [15], [16], [17], [18], [19] is the most commonly used architecture for integration of applications. Legacy systems implemented this type of integration using file transfers. Shared databases, FTP, journals etc. can be used to accomplish integration while using this architecture.

In this type of integration, applications taking part in exchange of information are directly linked to each other. The data elements and formats are defined such that the
The consuming system understands definitions. The information flows directly from one application to the other.

The communication between two applications is facilitated through APIs (Application Programming Interface). Those APIs are custom developed for accomplishing the data exchange between the applications. Any changes to API definitions might require changes to both the provider and the consumer. Usually, an interface which was originally built for facilitating the communication with one particular application cannot be used for handling the communication with a different application without significant modifications. This would result in tightly coupled integrated systems, though the initial development and deployment of such interfaces are relatively cheaper and easier.

Each new system requires new interfaces and connection links. This can get expensive as there will be an exponential increase in the number of interfaces and connection links, when more and more applications need to be integrated.

Communication channels are needed between each pair of applications. As new systems are integrated, new communication links need to be established and this leads to complicated “spaghetti” [13] architecture. This makes the maintenance of systems difficult. The number of connections needed by a set of n applications connected thru point-to-point architecture is given by n(n -1) as shown in Figure 1.

Maintenance and upgrade of systems often cause some difficulty in deciding which system needs to be modified first as the changes can potentially cascade across multiple systems. This might lead to inconsistent design and unpredictable failures. Troubleshooting will be difficult when systems fail since point-to-point integration
strategies often lack a centralized monitoring capability. With point-to-point architecture, the applications should have the features to support fault tolerance, availability, and scalability. This architecture is suitable for organizations with one or two applications and simple integration requirements. This approach also provides a quick and easy solution in the initial phase of integration.

Hub-and-Spoke Architecture

Hub-and-spoke architecture [13], [14], [15], [16], [17], [19], [20] is also known as “message broker” [20]. This approach tried to fix the issues related to connection management and scalability, associated with the point-to-point architecture.

As shown in Figure 2, this type of integration consists of a central hub with integrating applications connected to the hub as spoke systems. The connected applications do not communicate directly with each other, but only via the hub. The hub in this case acts as a router and routes messages from one system to the other.
Fig. 2. Hub-and-spoke integration architecture.

The centralized hub or broker would be a middleware solution and there are many products in the market from major vendors. A few examples are IBM’s WebSphere MQ [21], Java 2 Enterprise Edition Java Message Service [22], Microsoft’s MSMQ [23] etc. Lightweight connectors, which are developed and deployed on top of these applications, act as connecting links between the hub and the spokes.
While point-to-point integration leads to an exponential increase in the number of interfaces and connection links, hub-and-spoke approach reduces those to a linear increase as this uses a middle-ware for facilitating the communication. In this architecture, data transfer occurs through the central hub. The hub decides where to send the data or if it needs to be re-formatted. This helps in integrating applications with different formats of data. This architecture does not require the existing applications to be changed to facilitate the integration. The central hub would have the capabilities required for tasks like validation, routing, data transformation, and asynchronous messaging. It provides a central place for data transfer and thus avoids the need for direct communication of applications with each other. This helps in reducing the number of connection links. The hub acts as a layer of separation and hence applications can be changed without disturbing other applications. This helps in providing a loosely coupled architecture compared to the point-to-point architecture. This model is cost-effective due to the easy provision for connection management. The workings of the hub can be monitored through a central console.

One of the disadvantages of this model is that the failure of the central hub can result in the single-point failure for the entire system. The applications would not be able to communicate to each other if the hub undergoes any failure. Enterprise Application Integration tools based on this architecture are monolithic and are based on proprietary standards. The emergence of SOA, which is based on open standards leads to the evolution of Enterprise Service Bus.
Enterprise Service Bus Architecture

The next evolution in this area is Enterprise Service Bus (ESB) [13], [14], [20], [24], [25]. This is mainly based on SOA concepts of loosely coupled applications and open standards. ESB is a framework of middleware tools that enables reuse of business processes and promotes the reuse of capabilities or services provided by different applications across an enterprise. This framework is usually built on open standards and thus promotes application integration of disparate applications. The ‘bus’ acts as the “message broker” and applications talk to each other via the bus. The ESB architecture is shown in Figure 3.

The salient features of an ESB include:

- Business agility – Business can react and adapt to changing business needs.
- Supports open standards and hence easier to integrate applications.
- Provides the infrastructure needed to promote distributed services.
- Services are managed and monitored centrally.
- Provides a robust messaging gateway with built in message confirmation and re-routing algorithms even when systems are off-line.
- Built in access control and security.
- Based on SOA and promotes loose coupling of integrated applications.

ESB and SOA are closely linked as an ESB provides the framework and infrastructure required to implement reusable services [26].

Service-Oriented Architecture

The central theme of this project work is the use of SOA as a viable integration strategy to facilitate the integration of contact management in an enterprise.
As defined by OASIS [27], “Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” [4]. SOA is an organization of business processes, connected in a loosely coupled manner, with each process offering concrete business capability or functionality called a service. These business capabilities or services are then made available to others using standard communication protocols, to promote reuse. This reuse
of business processes along with the underlying technologies is collectively called
Service-Oriented Architecture.

The significance of this architecture is explained in the following excerpt,

The reality in IT enterprises is that infrastructure is heterogeneous across operating
systems, applications, system software, and application infrastructure. Some
existing applications are used to run current business processes, so starting from
scratch to build new infrastructure isn’t an option. Enterprises should quickly
respond to business changes with agility; leverage existing investments in
applications and application infrastructure to address newer business requirements;
support new channels of interactions with customers, partners, and suppliers; and
feature an architecture that supports organic business. SOA with its loosely coupled
nature allows enterprises to plug in new services or upgrade existing services in a
granular fashion to address the new business requirements, provides the option to
make the services consumable across different channels, and exposes the existing
enterprise and legacy applications as services, thereby safeguarding existing IT
infrastructure investments. [28]

More information regarding different aspects of SOA can be found in [4],
[29], [30], [31].

**SOA Concepts**

The following contains a discussion about some of the important concepts of
SOA [4]. The principle concepts of SOA are services, policies, contracts, and execution
contest.

**Services.** A service is a reusable business process that accomplishes concrete
business functionality. It is the ability to perform a unit of work. Services are provided by
service providers and users of a service are called consumers. Consumers access a service
through a service interface. This usually defines the overall access mechanism and the
exchange of information, required to utilize the capability offered by the service provider.
A service needs to be visible or the service provider and the consumer must be able to see
each other and interact with each other. This is true for any integration, as the calling
function or method must be able to access the functions and methods of the provider. Visibility is further dependent on the following conditions:

i. Awareness. The consumer should be aware of the existence of service producers and the capabilities provided by the producers.

ii. Willingness. The provider and consumer should be willing to participate in an interaction. Once the consumer is aware of a service provider, the consumer can initiate the interaction by calling the service interface of the provider. The provider has the following options:

   - Participate in the interaction.
   - Ignore the request from the consumer.
   - Respond and deny participating in the interaction.

iii. Reachability. For a service to be visible, the service needs to be reachable. Even if the consumer is aware of the provider, unless the consumer can communicate with provider, no interaction can take place.

   **Policies and Contracts.** Services conform to a set policies and contracts, which define the service definition. A contract is an agreement between two or more entities and defines the interaction between the consumer and the producer. A policy refers to any condition or constraint enforced upon the use or deployment of a service.

   **Execution Context.** Execution Context of a service consists of all the infrastructural elements that are needed for actual service interaction to take place. It is the actual connection or path from one service to another. The service contract or the service description contains information about the execution context including the preferred protocol, assumptions, policies, security requirements, input, and output.
parameters. A consumer of a service will need to abide by the guidelines and polices referenced by the service definition. All of the above guidelines and requirements are collectively called the execution context of a service.

Benefits of SOA

Integration of applications using SOA offers many benefits [4], [32, pp 59-64], [33, pp.119-120] over other traditional methods of integration. SOA is based on open standards and hence makes the integration of disparate applications easier and cost-effective. The single biggest advantage of SAO is that it helps to build a scalable infrastructure that is based on component assembly rather than building isolated individual capabilities that cannot be reused. The management and monitoring of these components can be centralized and thus making this much more cost effective than traditional integration approaches. Some of the benefits of SOA are listed below:

- Individual SOA components offer concrete capabilities making these components easier to deploy and debug compared to traditional monolithic applications.
- Reduced complexity improves business agility. Changes in business processes are easier to implement and hence reducing the time to market services. This helps in improving the overall performance of the organization.
- Loose coupling between systems and applications improves reliability due to reduced interdependency.
- Better governance and controls help streamline business processes.
- Easier to integrate and deploy new services or applications.
- Reduces overall cost of operations.
- Enhanced monitoring of systems helps in detecting and fixing the issues faster.
SOA Promotes Loose Coupling

Let us see how SOA promotes loose coupling among applications in an integrated environment [34]. In SOA, services are offered and described by providers. Consumers use service descriptions offered by providers to interact with providers and to gain access to the functionalities provided by the service providers. All the information required by the consumer about a provider is available through the contracts or service descriptions. The consumer does not need to know anything about how a service is implemented by the provider. So the consumer can easily make use of service from another provider without the need to change any interfaces. This feature allows the applications to be loosely coupled to each other. In a similar manner, the provider also does not need to be concerned about the changes or technologies of the consumer.

SOA and Web Services

SOA can be implemented using different service-based technologies such as Web services, WCF, SOAP, RPC, DCOM, CORBA, or JINI [35]. In SOA design, services communicate via a standard communications protocol. Web services can be used to facilitate this communication framework [36]. Web services [37] can be described as programs designed to support interoperability among machines spread over the network. The goal of web services is to make platform-independent services that can be accessed using common Internet protocols. The services operations are described using WSDL (Web Services Description Language) [38], which is a machine-readable language.

Service provider, service broker, and service requestor are the three major components or participants of SOA [39]. Publish, find, and bind are the basic operations performed by these participants.
Service provider. A Web service is created by a service provider. The service registry will contain the information related to this Web service.

Service Broker. Service brokers allow service registration, publication, and discovery of services. The service provider publishes the interface and access information of a Web service to the service registry. The service requestor gets this information from the service registry. Service brokers give provisions for publishing and discovering information regarding Web services.

Service Requestor. The service requestor gets information regarding the providers from the broker registry using find operation and binds to the service providers to access the Web services.

SOA Governance

The following are the minimum set of guidelines for SOA implementation and adoption [4):

- Identify business needs or capabilities which can be classified as services.
- Define how consumers and producers should communicate with each other. Consumers need to call services provided by producers and hence producers need to be “visible” to consumers.
- Ability to understand how the interaction among different systems is established.
- Provision for understanding effects of utilizing services.
- Services should have descriptions.
- Execution context required to facilitate the interaction between systems should be identifiable.
• Handling of policies and enforcement of contracts should be clearly defined.

• Execution context required to facilitate the interaction between systems should be identifiable.

This project demonstrates integration capabilities using two applications—Contact Management System (CMS) and Contact Client (CC). Chapter III describes the Web services implementation and the interaction between CMS and CC applications.
CHAPTER III

WEB SERVICE IMPLEMENTATION

Introduction

This chapter describes how Service-Oriented Architecture can be used to integrate applications by demonstrating the integration of data between two applications: Contact Management System (CMS) and Contact Client (CC). For this project, the CMS application was designed and implemented to provide services using Web services technology. This tool can be used by the CC application (and potentially other client applications) for integrating contact data.

Contact Management System

Contact Management System (CMS) is a web-based J2EE (Java 2 Enterprise Edition) [40] application for managing contact information of an enterprise. This application stores contact information (e.g., names, addresses, phone numbers, etc.) in a relational database. The use cases for this application are in Appendix A. The CMS application also provides two Web services (“create contacts” and “to access contacts”) that can be used by other (consumer) applications. Thus, the CMS application serves the role of a producer in the SOA design paradigm.
Contact Client

Contact Client (CC) is a Java Swing [41] application. This application plays the role of a consumer in the SOA design paradigm as it utilizes the Web services provided by CMS application for data integration. This application does not have a database for storing or retrieving contact data. The CC application solely relies on the services provided by the CMS application and does not manipulate the data directly.

CMS Web Services

The following excerpt taken from Microsoft’s website provides a definition for Web services:

Web services are applications, available over the Internet, that provide some kind of service, either programmatic or informative, to other applications. Web services differ from Web applications in that they generally involve application-to-application communication, and are not intended to be accessed via a Web browser. Instead, clients can be written in any language that supports HTTP and SOAP. A client transmits a message or remote method call to a Web service, which processes the message and returns a response to the client. Web services do not usually have any sort of user interface built in, and it is generally up to the client to process input and display output. [42]

CMS Web services provide application-to-application communication between CMS and Contact Client applications. The two Web services provided by the CMS application are CreateContactService and GetContactService.

CreateContactService

Applications can use this Web service to create new contacts by providing the contact information.
GetContactService

Applications such as Contact Client can use GetContactService provided by CMS to retrieve a contact’s first name and last name by providing a contact_id. The contact_id is a unique identifier which had been assigned to a contact when the contact was created by the CMS application.

This chapter includes a detailed explanation of the different steps involved in the implementation of the GetContactService Web service, the setup required by the consumer (CC) application to access this Web service, and the interaction between CMS and CC applications when CC application uses GetContactService to retrieve the contact information. This gives an insight into how Web services were implemented as a part of the CMS application and also how they can be accessed by the consumer (CC) application.

The following contains the steps required by the CMS (producer) application to implement and deploy the GetContactService Web service. This is followed by a description of how the CC (consumer) application accesses this service to retrieve a contact’s first name and last name by providing a contact_id.

Web Service Implementation and Deployment

Since Web services technology is a widely accepted industry standard, application development platforms (J2EE, .NET, etc.) support Web services. Java supports Web services through its application development environment (e.g., J2EE). This means one can use J2EE to build and deploy Web services. CMS is a Web application developed using J2EE and hence it can support Web services.
The GetContactService is a Web service provided by the CMS application. By using this service, the CC application (or other applications) can retrieve the first name and last name of a contact by providing a contact_id. The following contains a description of how the GetContactService Web service is defined.

**Defining the Web Service**

First, one needs to define the GetContactService Web service. A Web service is implemented by creating a Java class with Web service annotation tags [43]. Annotation tags are special commands that usually start with the character ‘@’ and the Java compiler knows how to process these commands. The annotations themselves implement most of the service definitions.

The Java class GetContactServiceImpl.java contains the implementation for GetContactService. This class belongs to the CMS application. The code snippet shown in Figure 4 is the beginning of the class definition of GetContactServiceImpl.java.

The annotations (@WebService, @WebMethod, @WebParam, @WebResult, etc.) used for developing the GetContactService are also noted in Figure 4 and are defined as follows:

- @WebService defines the Java class GetContactServiceImpl as implementing a Web service.
- @WebMethod identifies the method in the class that provides the Web service operation. In the GetContactServiceImpl class, the method locateCustomer provides the implementation for the GetContactService. This method takes the string “contactId” as the input parameter and returns the Contact object.
Fig. 4. GetContactService.

- @WebParam defines the input (the contactId parameter of locateCustomer) required for calling this Web service.
- @WebResult defines the output or return value of the Web service. The return value of GetContactService is “CustomerRecord” as shown in Figure 4.

In the GetContactServiceImpl class, the method call to contactManager.findById(contactId) returns the Contact object for the method argument (contactId), which is a unique identifier for a contact in the Contact table. The contactManager is a Java class which handles the retrieval of the Contact data from the database. This class belongs to the CMS application.

Now that the GetContactService Web service has been defined, the following describes the steps required to deploy it.
Deployment of Web Service

Once the Web service has been defined, the next step is to deploy the service on an application server. Before deploying the Web service, a configuration entry needs to be made to “web.xml” in order to access the GetContactService Web service over the Internet. “web.xml” is a standard configuration file for J2EE Web applications, and it defines the configuration information for the application components.

The code snippet taken from “web.xml” of the CMS application (shown in Figure 5) associates the Java file GetContactServiceImpl.java from the package

```xml
<servlet>
  <servlet-name>GetContactServiceImpl</servlet-name>
  <servlet-class>com.ms.services.GetContactServiceImpl</servlet-class>
</servlet>
```

Fig. 5. Web.xml entry.

com.ms.services to a name—GetContactServiceImpl. This allows the CMS Web service, GetContactService, to be accessed over the Internet using the URL http://ServerIP:port/contact/GetContactServiceImpl, where “contact” is the name of the CMS application and “ServerIP” and “port” refers to the IP and port of the server where GetContactService is deployed.

JBoss [44] Server is the application server used for the CMS application. Once the configuration entries are made in “web.xml” [45], the deployment of Web
services is done by starting the JBoss Server. If the JBoss Server is already running, it will need to be restarted in order to deploy the new Web service. At this point, the GetContact Web service has been defined and deployed.

A service contract is an agreement between the service provider and a consumer and is one of the important characteristics of Service-Oriented Architecture. A Web service uses a contract which defines the interaction between the producer and the consumer. The contracts for the Web services provided by the CMS application are defined using WSDL (Web Services Description Language). It is an XML document and contains the basic information required by a consumer to use the Web service. The following contains a description of WSDL generation for the GetContactService Web service.

**WSDL Generation**

For the CMS application, the WSDL is automatically generated by J2EE from the Web service implementation Java file once the Web service is deployed. The implementation file for the GetContactService Web service is GetContactServiceImpl.java. So the WSDL for GetContactService is generated using GetContactServiceImpl.java when this class is deployed on the server. No additional task is required to generate the WSDL, and J2EE takes care of creating the WSDL. The WSDL for the GetContactServiceImpl Web service can be found in Appendix B.

Figure 6 shows the relationship between the Java code that defines the GetContactServiceImpl.java Web service and the generated WSDL file. The Java file specifies the Web service elements like input parameter (contactId), the output parameter (CutomerRecord), and the Web service implementation method (locateCustomer). The
lines in Figure 6 link the elements from the Web service Java file, GetContactServiceImpl.java, to the corresponding elements of the WSDL for the GetContactService Web service.

Line 1 shows the output produced by the Web service and is identified by an attribute named “CustomerRecord.” Line 2 shows the input parameter for the method “locateCustomer,” (contact_id), and the data type of contact_id is String. Line 3 shows
that the operation or method name that implements the service is called “locateCustomer.” Thus, the WSDL contains the operations that can be performed, the input parameters that are required to call the service, and the resulting service output. So, WSDL for GetContactService contains a description of the GetContactService Web service and consumer applications can use this WSDL to access the Web service.

WSDL contains the service location or endpoint address. For example, 

```xml
<soap:address location="http://72.211.220.2:8090/contact/GetContactServiceImpl" />
```

entry in the WSDL (refer to Figure 6) specifies the location of the GetContactService. This URL (http://72.211.220.2:8090/contact/GetContactServiceImpl) for the location of the service includes the host name (72.211.220.2) and port (8090) information of the server where the Web service resides. WSDL is in XML and since XML is an open standard, consumer applications in any programming language can use this WSDL to access the Web services.

WSDL for a Web service can be accessed once the Web service is deployed. So once GetContactService is deployed, its WSDL can be accessed by using the URL, http://ServerIP:port/contact/GetContactServiceImpl?wsdl.

The following describes the setup required by the Contact Client (consumer) in order to access the GetContactService Web service and how the Contact Client application accesses this Web service. Contact Client is a Java swing application and contains the user interface screens needed for user entry and to display the output.
Web Service Access by Contact Client

Setup required by the CC Application

The following contains a description of the setup required by the CC (Contact Client) application for accessing the GetContactService Web service: 1) Configuration entries in the Client.properties file, and 2) Client stub generation.

Configuration Entries in the Client.properties File. A properties file, Client.properties, is used for the CC application to set properties for the components of the application. The CC application retrieves the name and path of the WSDL for the GetContactService Web service, the IP, and the port of the server where this Web service resides from the Client.properties file; hence, this information needs to be entered into the Client.properties file. Figure 7 shows the entries made for GetContactService in the Client.properties file.

```
1. WebService.URL.GetContactById = /contact/GetContactServiceImpl?wsdl
2. WebService.HostName = 72.211.220.2
3. WebService.Port = 8090
```

Fig 7. Property file entries.

Line 1 of Figure 7 specifies the WSDL (/contact/GetContactServiceImpl?wsdl) for the GetContactService Web service. Lines 2 and 3 specify the IP (72.211.220.2) and port (8090) of the server where the Web service resides.

Client Stub Generation. The Contact Client (or the consumer application) needs to have a Java file (called a stub file) for each Web service it accesses. This stub file is generated by a command line utility called wsconsume [46] that is provided by the
JBoss Server. This tool uses the WSDL of the Web service to generate the client stub file. The command,

```
wsconsume –k http://servername:port/contact/GetContactServiceImpl?wsdl
```
generates the Java source file, GetContactServiceImplService.java, which is the client stub for GetContactService. In the above command, /contact/GetContactServiceImpl?wsdl, represents the WSDL for GetContactService. The tool (wsconsume) uses this WSDL to create the stub file. The “servername” and “port” of the above command specifies the hostname (IP) and port of the server where the GetContactService Web service is deployed. This client stub is then added to the Contact Client source code.

To better explain the use of this stub file and how Contact Client uses this file to access the Web service, the following contains a use case where a user employs the “SearchById” screen of the Contact Client application to retrieve a contact’s first name and last name by providing the contact_id.

**Use Case: CC Application-Search by ID**

The user enters a value for “Contact ID” in the “Search Contact by ID” screen of CC application as shown in Figure 8.

In the “Contact ID” field, the user enters the value 7, which refers to the contact_id of the contact whose first name and last name need to be retrieved. Then the user clicks the “Ok” button to retrieve the data. Using the GetContactService Web service provided by the CMS application, the CC application retrieves the first name and last name of contact_id 7. The contact information is stored in the relational database of the CMS application. The retrieved data is displayed as shown in Figure 9.
Figure 9 displays the first name (John) and last name (Smith) of the contact with contact_id 7. The following contains an overview of the overall interaction between the CMS and CC applications once the user clicks on the “Ok” button after entering the contact_id information.

Step1. User Clicks “OK” Button. The ContactIdSearchDemo class represents the user interface screen shown in Figure 8. When the user clicks the “Ok” button, the performAction method of ContactIdSearchDemo is called. The code snippet from performAction method is shown in Figure 10.

Line 2 on the Figure 10, retrieves the WSDL URL (http://72.211.220.2:8090/contact/GetContactServiceImpl?wsdl) for the Web service,
GetContactService, using the values specified in the property file for the CC application called Client.properties.

The WSDL for GetContactService contains the information required by the CC application to invoke the Web service such as input parameters and output parameters along with the data types, the location of the service, and the method to invoke.

Line 3 (Figure 10) instantiates the Java object GetContactByID. GetContactByID.java is a Java object that contains the code required to invoke the GetContactService Web service. Line 7 (Figure 10) shows that method getContact (of the
```java
public void performAction() {
    String webServiceURL =
            getSwingSet2().getWebServiceURL("WebService.URL.GetContactByID");

    // ...

    3. GetContactByID getContactByID = new GetContactByID();

    4. String contactId = getContactId();

    5. if (contactId != null)

    6. {

7.  Contact contact = getContactByID.getContact(contactId,
                                            ,webServiceURL);

8.  setFirstName(contact.getFirstName());

9.  setLastName(contact.getLastName());

    }
```

Fig. 10. PerformAction method.

getContactByID class) is called and two arguments are passed to this method. The first argument is the value of contact_id (entered by the user as shown in Figure 8) and the second is the Web service URL for GetContactService. The client stub generated is instantiated in the method getContact of the getContactByID class. The code snippet from the class, GetContactByID.Java is shown in Figure 11.

GetContactServiceImplService.java is the stub that was generated using the JBoss tool, wsconsume. The following describes how this stub generates the proxy (Java class) object that is ultimately responsible for the communication between Contact Client and the CMS Web service, GetContactservice.
1. public class GetContactByID {
2. private static GetContactServiceImpl port;
3. public Contact getContact(String contactId, String webServiceURL) {
4. URL url = null;
5. try {
6. url = new URL(webServiceURL);
7. } catch (MalformedURLException e) {
8. e.printStackTrace();
9. }
10. // *** OPTIONS B ****
11. // JAX-WS Use generated stubs from wsconsume
12. GetContactServiceImplService service = new GetContactServiceImplService(url);
13. port = service.getGetContactServiceImplPort();
14. Contact contact = port.locateCustomer(contactId);
15. if (contact != null) {
16. System.out.println("First Name " + contact.getFirstName());
17. System.out.println("Last Name " + contact.getLastName());
18. } else
19. System.out.println("Contact is null");
20. return contact;
21. }
22. }

Fig. 11. GetContactByID.

Step 2: CC Application Generates a Proxy Using WSDL. Line 3 (Figure 11) is the beginning of the implementation of the method, getContact. Line 10 (Figure 11) instantiates the client stub, GetContactServiceImplService.java. The only parameter passed during instantiation is the WSDL URL for GetContactService (http://72.211.220.2:8090/contact/GetContactServiceImpl?wsdl).

GetContactServiceImplService accesses the GetContactService Web service using the URL http://72.211.220.2:8090/contact/GetContactServiceImpl?wsdl to retrieve the WSDL for GetContactService Web service. The Contact Client uses the client stub (GetContactServiceImplService) for creating a proxy (a Java class) using the WSDL. The method call service.getGetContactServiceImplPort() on line 11 assigns the created proxy to the variable “port.” Thus, variable “port” represents the proxy object.
A proxy is a runtime object that provides the methods exposed by the Web service and proxy gets this information from the WSDL. The method exposed by the GetContactService Web service is “locateCustomer” (Figure 4) that accepts contact_id as a String parameter and proxy gets this information from the WSDL. The proxy is a client (Contact Client) component that represents the server side component that is being invoked. The proxy internally manages its communication with the server side component. The communication between the proxy and the Web service is managed by J2EE and no custom code needs to be developed to handle this communication. Invoking a function on the proxy invokes the same method on the Web service and this is described in Step 3.

**Step 3: The “locateCustomer” is Invoked.** When the Contact Client invokes the “locateCustomer” operation on the proxy (as shown in line 12 of Figure 11), it would in turn invoke the same method “locateCustomer” on the Web service GetContactService(refer to Figure 4) which returns a Contact object.

**Step 4: The Results Returned to CC.** The proxy method call “locateCustomer” returns the Contact object which is the output of the GetContactService. This Contact object is passed back to the client proxy. The Contact object contains the first name and last name of the Contact, which is the output displayed to the user.

Figure 12 is the pictorial representation of steps 1, 2, 3, and 4. The following contains a summary of the steps 1, 2, 3, and 4. As shown in Figure 12, Contact Client accesses the WSDL for GetContactService Web service using the URL http://72.211.220.2:8090/contact/GetContactServiceImpl?wsdl. The GetContactService Web service responds to the above request by providing the WSDL for
GetContactService Web service. GetContactServiceImplService uses the WSDL to create the proxy and calling “locateCustomer” method on the proxy invokes the “locateCustomer” method on the Web service GetContactService, which then returns the Contact object for the contact_id parameter that was passed as an argument to the “locateCustomer.” The results are then returned to the proxy object and thus the Contact client then displays the first name and last name of the contact from the Contact object.

Figure 13 is the code snippet taken from the Java class that represents the GUI screen, ContactIdSearchDemo.java, which displays the first name and last name of the contact derived from the Contact object.
if (contactId != null) {
    Contact contact = getContactByID.getContact(contactId, webServiceURL);
    if (contact.getFirstName().isEmpty())
        setMessage("Contact does not exist");
    setFirstName(contact.getFirstName());
    setLastName(contact.getLastName());
}

Fig. 13. Get contact name.

The Contact object contains the first name and last name and the setFirstName method (Figure 13) sets the “First Name” on the screen. Similarly, the setLastName method sets the “Last Name” on the screen.

Figure 14 displays the first name and last name of the contact with Contact ID 7. The Contact Client did not retrieve this information directly from the CMS database.

The CC application used the GetContactService Web service to retrieve this information.

This concludes the description of how Web services are used to integrate the CMS and CC applications.
if (contactId != null) {
    Contact contact = getContactById.getContact(contactId, webServiceURL);
    if (contact.getFirstName().isEmpty())
        setMessage("Contact does not exist");
    setFirstName(contact.getFirstName());
    setLastName(contact.getLastName());
}
CHAPTER IV

CONCLUSIONS

Summary

SOA is a design approach that is business focused and promotes the creation of independent and self-contained business processes that can be reused within an enterprise. This design approach enables the creation of agile business processes called services that can be modified or changed to changing business needs. Businesses that can adapt and accommodate changes in the market place are more successful compared to businesses that are slow to turn around to meet new business needs and challenges. Besides agility, the other important characteristics of SOA include platform independence to support a heterogeneous set of platforms and systems, loosely coupled interfaces that reduce impact to other systems when propagating changes to processes or systems within an organization, and the support for common standards and protocols that enable systems to exchange information.

The goal of this project was to demonstrate how the contact data maintained by the Contact Management System (CMS) could be integrated with other applications, using Services Oriented Architecture. CMS is the contact management system that provides Web services for data integration and acts as the provider. Contact Client is the
consumer and exchanges data with CMS using an interface (Web service) built using the design principles of SOA.

The CMS integration services developed using Web services are based on platform independent standards and design principles of Service-Oriented Architecture.

Web services deployed by the CMS application can be accessed over the network using HTTP which is an industry standard protocol. So Web services provided by CMS could be accessed by any client developed on any platform, making the CMS Web services platform independent.

The services supported by the CMS application uses Web services which use open standards protocols (SOAP and XML) to exchange data between the systems. So the interfaces between the CMS and CC applications are independent as the data are exchanged between the two systems using XML and SOAP.

The WSDL for the deployed Web services (provided by the CMS) contains basic information required by a consumer to use the service including location of the service, input, and output parameters. Since the WSDL is an XML file, it can be processed by applications developed on any platform supporting open standards. This capability of supporting platform-independence for integrated applications is one of the key design principles of SOA.

The CC application being a Java Swing application can run on any platform supported by Java. Moreover, irrespective of the platform, CC can access the Web services deployed by CMS as the protocols supporting Web services are platform independent and vendor agnostic.
WSDL defines the location of the Web service. A consumer calls a Web service by looking up the location from the WSDL and if the location changes, the entry in the WSDL would be updated and the consumer need not change code or configuration to reflect the change in location of the Web service. So the Web services can also be location independent as the WSDL defines the location of the Web service.

The above features of using standard protocols and open standards and having location and platform independence makes the systems, Contact Client and CMS, loosely coupled to each other.

Thus, this project demonstrates the integration capabilities between CMS and CC are based on open standards, are platform and programming language independent, are location independent and these are the basic tenants of SOA design principles.

Roadblocks

The following contains some of the major roadblocks encountered during the course of design and development of this project.

Selecting a technology to demonstrate integration capabilities required some research and reading. The original design was to use JMS (Java Message Service) [22] for application integration using message brokers. JMS designed by Sun Microsystems is a messaging API that supports both point-to-point and publish-subscribe messaging models. In order to design systems that use messaging as an integration platform, a JMS provider is required. A JMS provider is a messaging system that implements the JMS specification. A consuming application retrieves the messages from the message queue using JMS API. The limitation with JMS is that the consumer and producer would need
to be written using Java technology, where as Web services would provide platform independence and was a much better choice. Hence, Web Service was selected as the technology of choice to implement the integration between CMS and Contact Client applications.

One of the challenges was also to figure out a way to demonstrate the integration capabilities provided by the CMS application. Web services were developed and deployed as a part of CMS application. In order to demonstrate the integration capabilities, a consumer was developed as a part of the CMS application itself. This involved developing input screens similar to the screens developed for Contact Client within CMS itself. For example, to use the getContactService Web service, the user would need a screen that accepts a contact_id and on clicking a “Submit” button, the first name and last is displayed to the user by invoking the getContactService Web service. But this was not a realistic scenario as the consumer and the producer belonged to the same application, and the exchange of information within the same CMS application was not a convincing example of integration. This was solved by developing a second application, Contact Client using Java Swing.

Another major roadblock was the configuration and setup requirements for JBoss application server to support Web services. JBoss application server requires special setup and configuration to support Web Services which was not performed during the installation of the JBoss application server. Without this special configuration, the deployed Web services would error out. After some research and reading through the documentation, it became clear that additional setup and configuration was required to deploy Web services. The setup requirements to support Web services involved
downloading the JBossWS [47] module and running scripts to complete the configuration. JBossWS is the JBoss module that supports Web services implementations. Once the server was configured correctly, the deployed Web services provided by the CMS application produced the desired results. Installation information for JBoss can be found at [48].

Enhancements

An ideal enhancement would be to implement security for the Web services using WS-Security. WS-Security [49] is a set of Web service security standards jointly developed by leading technology vendors such as IBM, Sun Microsystems, and Microsoft. Since Web services use HTTP as a transport medium, the message is transported in clear text which is not secure and can be read by unauthorized individuals and can also be manipulated. Web services security needs to take into account three different aspects [50]:

1. Message Integrity - Ensures that the message has not been modified while in transport
2. Message Confidentially - Ensures that the message is not transported in clear text.
3. Message Authentication - Ensures that only valid users can invoke or call a service.

WS-security specification provides mechanisms that address these three aspects of message security.
Another enhancement would be to create a Web service for deleting contacts as a part of CMS application. This would allow integrating applications to delete a contact. This functionality was not implemented due to time constraints.

Another enhancement would be to implement a dynamic Web service lookup so that applications that need to use the services provided by CMS could automatically look up the services and invoke the required services. This would eliminate the need for Web service stubs (refer to Chapter 3-Client Stub Generation). Web services can be published to a Universal Description, Discovery and Integration (UDDI) [51] registry. A consumer can look up the registry, bind to the Web service and invoke the methods on the Web service dynamically. This was not implemented as it was beyond the scope of this project.

In technology, there is always a “step further” that one can take a project. This project proved to be complex enough to show the proof of concept, yet also to be of the right scope to allow further enhancements. This author feels confident, after achieving the project goals successfully, to be able to tackle additional technological enhancements as their need arises.
REFERENCES


USE CASES FOR CONTACT MANAGEMENT SYSTEM

Use Case 1: User Login

1. User access the CMS application by accessing the URL http://72.211.220.2:8090/contact.

2. The user is presented with a Welcome Screen as shown in Figure A-1.

3. To login to the application, the user needs to click on the link, “Click here to Login.”

4. The user is presented with a login screen as shown in the Figure A-2.

5. User enters user ID and password (admin/admin) and submits the page.
   6a. System validates the credentials and if successful and displays the home page as shown in the Figure A-3.
   6b. An error page is displayed to the use, if the credentials are invalid.
Fig. A-2. Login screen.

Fig. A-3. Home page.
Code Table Maintenance

The CMS system provides the users with the ability to maintain a code table to add address types, phone types and email types. These entity types are then made available as drop down values in the maintenance screens for the contact attributes.

Use Case 1: Code Table Maintenance

1. User selects “Add Code” from the main menu as shown in Figure A-4.

![Fig. A-4. Add Code Menu Selection.](image)

2. System displays the entry screen as shown in the Figure A-5.

![Fig. A-5. Add address page.](image)
3. User selects the appropriate code type from the drop-down menu for “Code Type.”

4. User enters a description for the selected code type as the value for “Code Desc.”

5. User enters a numerical value for sequence. This determines the sort order for the entered value, when these values are displayed on display screen.

6. User clicks “Submit” to submit the page and to persist the values entered on the screen. The user will be presented with a screen displaying all available code values as shown in Figure A-6.

![Manage your Contacts](image)

**Fig. A-6. Home address type.**

7. To edit code table values, click “Edit Code.” The user will be presented with a screen as shown in Figure A-7 for editing code table values.

![Manage your Contacts](image)

**Fig. A-7. Edit code table.**
8. “Code Type,” “Code Dec,” and “Sequence” can be edited. Edit the required fields and click “Submit.”

**Use Case 2: Create a New Contact**

Select the “Create New Contact” menu from the home page as shown in Figure A-8.

![Create contact menu](image1)

**Fig. A-8. Create contact menu.**

1. System displays the entry screen to enter first name and last name attributes. This is shown in Figure A-9.

![Create contact screen](image2)

**Fig. A-9. Create contact screen.**
2. User enters first name and last name of the contact and submits the page. The first name and last name of the contact will be saved and the user will be presented with the screen as shown in Figure A-10, where the user can enter address, phone, and email information of the contact by selecting the appropriate link.

Fig. A-10. Contact details entry.

**Use Case 3: Edit Contact**

1. Search for a Contact by selecting the ‘Search Contact’ menu as shown in Figure A-11.

2. The user will be presented with a search screen as shown in Figure A-12.
3. Search for a contact using first name and/or last name on the search screen as shown in Figure A-12. Enter first name, last name and/or both, and click “Submit.” The search result screen will be presented as shown in Figure A-13.
4. On the search result screen, click “Edit Contact.”

5. The system displays the edit page as shown in Figure A-14.

Fig. A-13. Search results screen.

Fig. A-14. Edit page.
6. User changes the first name and/or last name.

7. Click “Submit” to submit the page.

**Use Case 4: Delete Contact**

1. Search for a Contact using the “Search Contact” menu.

2. From the Search Results screen, click “Delete Contact” as shown in Figure A-15.

3. The system displays a confirmation dialog box as shown in Figure A-16.

4. Click “OK” to confirm delete.

**Use Case 5: Create Address**

1. Search for a contact using the “Search Contact” menu.

2. From the Search Results screen, click “View Contact” as shown in Figure A-17.
Fig. A-16. Delete confirmation screen.

Fig. A-17. View contact link.
3. Click “Add Address” as shown in Figure A-18. The system displays the address create page.

![Add Address](image)

**Fig. A-18. Add address link.**

4. Select the appropriate address type and enter the address information on address create page as shown in Figure A-19.

5. Click “Submit” to submit the page. The user will be displayed with a screen displaying the address as shown in Figure A-20.

**Use Case 6: Edit Address**

1. Search for a contact using the “Search Contact” menu.

2. From the search results screen click “View Contact.”

3. Click “Edit Address” as shown in Figure A-21.

4. The system displays the address edit page as shown in Figure A-22. Select the appropriate address type and edit the address information.
Fig. A-19. Create address.

Fig. A-20. Display address.
Fig. A-21. Edit address link.

Fig. A-22. Edit address page.
5. The value for the field Address Line 2 has been edited to “Apt.10.” Click “Submit” to submit the page. The edited address will be saved and displayed on the screen as shown in Figure A-23.

![Manage your Contacts](image)

Fig. A-23. Display edited address.

**Use Case 7: Delete Address**

1. Search for a Contact using the “Search Contact” menu.
2. From the Search Results screen, click “View Contact.”
3. Click “Delete Address.” The system displays a confirmation dialog box as shown in Figure A-24.
4. Click “OK” to confirm delete.
Fig. A-24. Delete address confirmation.

Use Case 8: Create Email

1. Search for a Contact using the “Search Contact” menu.

2. From the search results screen, click “View Contact.”

3. Click “Add Email” as shown in Figure A-25.

4. The system displays the email entry page as shown in Figure A-26. Select the appropriate email type and email information. Click “Submit” to submit the page and the user will be presented with a page as shown in Figure A-27.
Fig. A-25. Add email link.

Fig. A-26. Email entry page.
### Use Case 9: Edit Email

1. Search for a Contact using the “Search Contact” menu.
2. From the search results screen, click “View Contact.”
3. Click “Edit Email” as shown in Figure A-28.
4. The system displays the email edit page as shown in Figure A-29. Select the appropriate email type and edit the email information. Change the email to `jsmith@xyz.com` as in Figure A-29.
5. Click “Submit” to submit the page.
Fig. A-28. Edit email link.

Fig. A-29. Edit email.
Use Case 10: Delete Email

1. Search for a Contact using the “Search Contact” menu.

2. From the search results screen, click “View Contact.”

3. Click “Delete Email” as in Figure A-30.

4. The system displays a confirmation dialog box as in Figure A-31.

5. Click “OK” to confirm delete.

Fig. A-30. Delete email link.
Use Case 11: Create Phone

1. Search for a Contact using the “Search Contact” menu.
2. From the search results screen, click “View Contact.”
3. Click “Add Phone” as in Figure A-32.
4. The system displays the phone entry page as in Figure A-33. Select the appropriate phone type and enter phone information.
5. Click “Submit” to submit the page.
Fig. A-32. Add phone link.

Fig. A-33. Phone entry page.
Use Case 12: Edit Phone

1. Search for a Contact using the “Search Contact” menu.

2. From the search results screen, click “View Contact.”

3. Click “Edit Phone” as in Figure A-34.

4. The system displays the email phone entry page as in Figure A-35.

5. Select the appropriate phone type and enter the phone information. Click “Submit” to submit the page.
Fig. A-35. Phone entry page.

**Use Case 13: Delete Phone**

1. Search for a Contact using the “Search Contact” menu.

2. From the search results screen, click “View Contact.”

3. Click “Delete Phone” as in Figure A-36.

Fig. A-36. Delete phone link.
4. The system displays a confirmation dialog box as in Figure A-37.

5. Click “OK” to confirm deletion of phone record.

Fig. A-37. Delete phone confirmation.
<definitions name="GetContactServiceImplService"
	targetNamespace="http://services.ms.com/"
	xmlns="http://schemas.xmlsoap.org/wsdl/
	xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
	xmlns:tns="http://services.ms.com/"
	xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<types>
<xs:schema targetNamespace="http://services.ms.com/" version="1.0"
	xmlns:tns="http://services.ms.com/"
	xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="contact">
<xs:complexContent>
<xs:extension base="tns:abstractContact">
<xs:sequence />
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType abstract="true" name="abstractContact">
<xs:complexContent>
<xs:extension base="tns:baseModel">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="addresses" nillable="true" type="xs:anyType" />
<xs:element minOccurs="0" name="contactId" type="xs:long" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="emails" nillable="true" type="xs:anyType" />
<xs:element minOccurs="0" name="firstName" type="xs:string" />
<xs:element minOccurs="0" name="lastName" type="xs:string" />
<xs:element minOccurs="0" name="middleName" type="xs:string" />
<xs:element maxOccurs="0" minOccurs="0" name="phones" nillable="true" type="xs:anyType" />
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="baseModel">
</xs:complexType>
</xs:schema>
</types>
</definitions>
<message name="GetContactServiceImpl_locateCustomer">
  <part name="contactId" type="xsd:string" />
</message>

<message name="GetContactServiceImpl_locateCustomerResponse">
  <part name="CustomerRecord" type="tns:contact" />
</message>

<portType name="GetContactServiceImpl">
  <operation name="locateCustomer" parameterOrder="contactId">
    <input message="tns:GetContactServiceImpl_locateCustomer" />
    <output message="tns:GetContactServiceImpl_locateCustomerResponse" />
  </operation>
</portType>

<binding name="GetContactServiceImplBinding" type="tns:GetContactServiceImpl">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="locateCustomer">
    <soap:operation soapAction="" />
    <input>
      <soap:body namespace="http://services.ms.com/" use="literal" />  
    </input>
    <output>
      <soap:body namespace="http://services.ms.com/" use="literal" />  
    </output>
  </operation>
</binding>

<service name="GetContactServiceImplService">
  <port binding="tns:GetContactServiceImplBinding" name="GetContactServiceImplPort">
    <soap:address location="http://72.211.220.2:8090/contact/GetContactServiceImpl" />
  </port>
</service>

</definitions>