Chapter 1

Introduction

Service-Oriented Architecture (SOA) is an architectural paradigm for building information systems based on loosely coupled components (services) and dynamic binding. SOA achieves loose coupling by defining the interfaces of the system with a universally available simple and ubiquitous interface that defines generic semantics for the system. Services are thus described in a uniform way and can be discovered and consumed by arbitrary clients. Generally speaking, a service is a well-defined and self-contained functionality encapsulated in a form that is readily consumable and understandable by clients. The service interface is independent of the underlying technology of the service provider and the service consumer.

The Internet can be seen as a large-scale distributed information system with numerous information providers and users. From the view-point of information systems engineering, the Internet has led the evolution from static contents to *web services*. Web services are a realization of the abstract SOA architectural principles. Specifically, web services are distributed software components which can be accessed over the Internet using well established web mechanisms and XML (eXtensible Markup Language)-based open standards and transport protocols, such as SOAP (Simple Object Access Protocol) and HTTP (HyperText Transfer Protocol). Public interfaces of web services are defined and described using the W3C (World Wide Web Consortium) based standard, Web Service Description Language (WSDL).

Web services have a wide range of applications and are primarily used for enterprise integration. The biggest advantage of web services lies

on their simplicity in expression, communication, and servicing. The componentized architecture of web services also makes them reusable, thus reducing development time and costs [Gottschalk et al., 2002].

1.1 Services Hosted on Mobile Devices

Recently, the processing powers and memory capabilities of high-end mobile phones and PDAs (Personal Digital Assistant) have increased significantly. Mobile devices like smartphones are becoming increasingly pervasive and are being used in a wide range of applications like location based services, mobile banking services, ubiquitous computing etc. The higher data transmission rates achieved with Third Generation (3G) and Fourth Generation (4G) wireless technologies, and the fast creeping all-IP based mobile broadband networks are boosting this growth in the cellular market. These developments are bringing out a large scope and demand for software applications on such high-end smartphones.

Currently, mobile devices are used to access web services, where mobile device applications request a service that is available on the Internet (e.g. a stock quote). However, little work has been done in hosting web services from mobile devices, where an external application requests a web service provided by the mobile device. A mobile hosted service has to reside on a mobile device, operating over a wireless network. Services are requested by an external client (mobile or stationary) that initiates the request on demand. This opens up a new set of applications and opportunities in different domains like mobile community support, collaborative learning, social systems, disaster management, etc.

The main limiting factor in providing mobile hosted services is associated with the addressability of the device. To provide services, the mobile devices must be addressable through the same methods used by today's web service implementations. This means that the device needs to be addressable through an IP address or URL (Uniform Resource

Locator) and be able to receive incoming HTTP (HyperText Transfer Protocol) requests.

Mobile devices are generally connected to the Internet either through a GPRS (General Packet Radio Service) network or through a WLAN (Wireless Local Area Network). The router at the edge of the network employs Network Address Translation (NAT) to provide access to the internal addresses, and all responses from within the network appear to originate from the router [Wikman, 2006]. Furthermore, mobile IP addresses change frequently as the device roams between cells and networks, and every time obtains a new IP address.

A solution to the addressability problem has been proposed by Nokia [Wikman, 2006]. The solution is based on an intermediary gateway maintaining a connection to the mobile device. The gateway then acts as an HTTP router and forwards incoming HTTP request to the appropriate mobile device. The solution has been realized by running a mobile web server on the mobile device, addressable through a public URL, like `https://username.mymobilesite.net/`. Further, the problem of addressing is also resolved by Mobile IP version 6 (*Mobile IPv6*) [Johnson et al., 2004]. The key benefit of Mobile IPv6 is that the existing connections through which mobile devices communicate are still maintained, even if they change locations.

1.2 Motivation

To meet the demand of the cellular domain and to reap the benefits of the fast developing web services standards, the scope of mobile terminals as both service clients and service providers is promising. *Mobile web services* enable communication via XML interfaces and standardized protocols on the radio link, where today proprietary and application specific interfaces are required. Several organizations are working on supporting mobile web services to be applicable. The Open Mobile Alliance (OMA) and Liberty Alliance (LA) are working on developing the

specifications. SUN and IBM toolkits are available on the development front for mobile web services; and some data service applications such as OTA (over-the-air provisioning) and application handover are in progress on the commercial front. Thus, it can be anticipated that mobile web services will create many opportunities in the near future to mobile operators, wireless equipment vendors, third-party application developers, and end users.

The paradigm shift of mobile devices from the role of service consumers to service providers is a step towards the practical realization of various computing domains; such as pervasive computing, ubiquitous computing, ambient computing, and context-aware computing. For example, the applications hosted on a mobile device provide information about the associated user (e.g. location, agenda) as well as the surrounding environment (e.g. signal strength, bandwidth). Mobile devices also support multiple integrated devices (e.g. cameras) and auxiliary devices (e.g. GPS (Global Positioning System) receivers, printers). The hosted services provide a gateway to make their functionality available to the outside world (e.g. providing paramedic assistance). In the absence of such provisioning functionality, the mobile user has to regularly update the contents with the device's state to a standard stationary server.

Hosting services on mobile devices can create a lucrative market. There are a number of applications where hosting services on mobile devices is important and can improve a business process. For example, a mobile device installed on a vehicle or carried by a skilled person can expose a service to track its exact location at any time. This type of service can be used by the freight forwarding or shipping companies to track the delivery and estimate the arrival time directly.

Services hosted on mobile devices can also find their application in supply chain management systems. A person running a small business and using a laptop or a handheld device in the field can be part of a supply chain system used by an enterprise. The services offered by such a

person in the field can be available through hosted services on his mobile device. For example, a technician in the field can expose his progress of work through a service to the manager if he is using a device to log the events and milestones while completing their tasks. The reason for hosting these services on his mobile device is that the data to be used in these services needs to be updated frequently. Since he is always updating the data while working in the field, it makes sense to host the service on his mobile device.

In an emergency or disaster situation, especially skilled people like doctors, nurses, and rescue teams can be located using their mobile services for help and support. In addition, the people needing help from the rescue teams can provide their location automatically by hosting a simple tracking service on their smartphones. A number of other applications, such as a wallet service and a parcel tracking system are also possible for customers, which are presented in [Srirama et al., 2006a; McFaddin et al., 2003].

A Motivating Scenario

Let us consider the following scenario in a real-world environment.

Alex is a journalist in a news broadcasting company and has a busy life travelling on sites for collecting news. He needs to coordinate with other journalists and organizations at different locations, covering events like sports, major events in cities, emergency situations, political demonstrations, accidents, etc. Alex uses his smartphone as a web service provider and can simply host the contents he gathers and publishes on the mobile device. Alex collaborates with different journalists and makes the collected contents available to his editor. The editor can browse through the data at any instance using a client application which collects and synchronizes the information available from the journalists' mobile service providers. Other journalists in the team can also look at the

contents of Alex's smartphone and can better synchronize their activities (e.g. covering some major distributed event).

In the traditional solutions, journalists upload their contents to a static server held by the editor or a third party. The key difference now is that Alex's service provider allows parallel access by both the team members and the editor without requiring him to upload the contents. Besides, Alex's smartphone is equipped with an integrated digital camera. To publish the photographs using currently available technologies, Alex has to upload the files to a stationary web server. In that case Alex has to bear the cost of data transfer between his smartphone and the web server. But with a web service provider deployed on his smartphone, the requesting clients of the service become responsible for the cost of data transfer for the pictures.

Meanwhile, Alex also commutes frequently between suburbs and city centers on public transports (e.g. buses, trains, ferries). Commuters on these transports are usually in very close proximity and most people carry handheld smart devices with one or more network interfaces (e.g. WLAN, Bluetooth, GSM). These smart devices are diversely equipped and a wide variety of services could be shared among people in a peer to peer fashion. For example, location information can be collected by a device with a GPS receiver and served to others peers. Someone able to access the Internet could forward news headlines or stock market levels to others. Commuters can download software components or services from other peers. People travelling in different directions can also share information about traffic and delays using their devices.

Let us assume that Alex gets on a busy train and starts the *Travel Planner* application, causing his smartphone to broadcast a request for location information (Figure 1.1). Any device that receives this request and is able to provide a matching service can potentially respond to it. Let us also assume that P2 and P5 are able to provide the service using GPS receivers. So they send back a response to Alex, containing all related attributes of their service (e.g. accuracy, time).

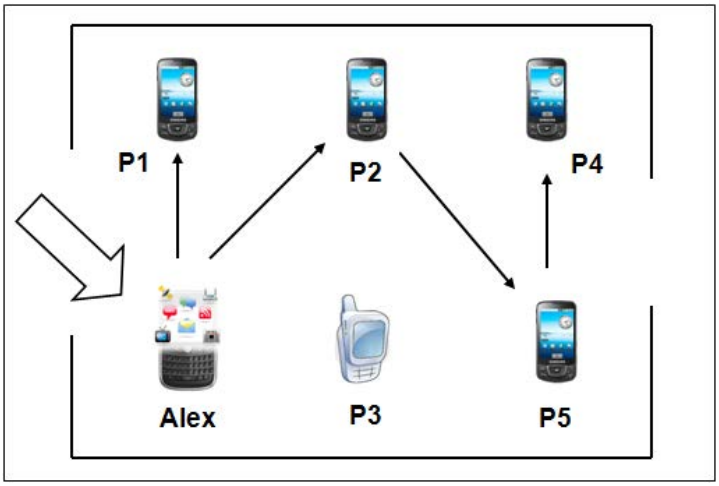

Figure 1.1: Train travel scenario.

Moreover, each device maintains information about the user's mobility pattern, based on his/her past journeys. This information is also sent to Alex, so that his mobile device may estimate how long the service provider will be co-located with him. After receiving responses from the co-located devices, Alex can then select the best result according to his requirements.

Such a scenario clearly shows the potential of mobile hosted service provisioning and motivates the research on its development.

1.3 Challenging Issues

The development and deployment of mobile hosted services faces a number of challenges. Mobile hosted services impose additional requirements to the service architecture, which are not necessarily supported by other architectures like enterprise and business services. The loosely-coupled nature and independence of mobile hosted services make it difficult to achieve real-time assurance of service performance. Movement of service providers between networks or movement of services across domains makes mobile hosted services difficult and complex to control. In addition, the huge number of services possible with mobile

service hosts makes the discovery of these services quite complex in wireless networks. Proper discovery mechanisms are required for successful adoption of mobile hosted services into commercial environments.

Most of the mobile devices have low resource capabilities. Therefore, provisioning services from such devices is challenging, due to the constraints of wireless bandwidth, limited processing power, and limited memory capacity of the mobile devices. While service delivery and management from mobile devices were observed to be technically feasible by previous works, the ability to provide services with a reasonable scalability appears to be quite challenging. Running complex and heavy-duty services on a mobile device can consume most of its resources and may obstruct the device to perform its core functions (e.g. telephony).

In terms of scalability, the mobile service host has to process a reasonable number of clients, over long durations, and without seriously impeding normal functioning of the device for its user. The execution environment of mobile hosted services should be able to make use of the resources available in its vicinity and leverage the capability to run heavy-duty services. The architectural design to provide real-time assurance of services needs to take such factors into consideration. The challenge is to design a flexible, scalable, and distributed execution environment, which can exploit resources from available networks based on the dynamic condition of the mobile service provider.

Moreover, integration with legacy systems further complicates the design and deployment of the services to be hosted on mobile devices. The service provisioning architecture may need to provide means to integrate existing legacy services which do not necessarily conform to mobile hosted services (e.g. authentication or authorization). Clearly, deployment of services on mobile devices comes with a different set of requirements and challenges. This thesis addresses the issues.

1.4 Research Goals

Even though service delivery from mobile devices was shown to be feasible technically, the capability to provide complex and heavy-duty services with adequate Quality of Service (QoS) appears to be very difficult. A major problem is that, in mobile networks nodes can join and leave at any time. This can result in difficulties keeping an up-to-date view of the available services and the service providers. Therefore, proper mechanisms are required for discovering the services, managing the mobility of service providers, and updating service registries of the mobile hosted services in commercial environments. Peer to peer (P2P) is a distributed computing model where nodes can act both as a server and client. One promising way of utilizing the functionalities of mobile web services could potentially be in P2P environments. Deploying mobile web services in P2P networks can open up numerous scopes for both P2P and mobile web services.

One of the goals of this research is to a construct a framework to facilitate service provisioning from mobile service providers in P2P environments. The next goal is to design the architecture of the mobile service provider to host heavy-duty services. Then the subsequent goals are to formulate a novel solution for partitioning complex applications, and developing an adaptive platform that offloads the partitions to improve performance of the mobile hosted service provisioning. The most important goals addressed in the thesis are briefly presented below.

- ***Service provisioning in mobile P2P environments:*** In recent years, P2P technology has been used in vast application domains like entertainment systems, ubiquitous computing, pervasive computing, collaborative systems, etc. P2P architecture has gained popularity as a low-cost individual computing technology. Combining mobile based service provisioning to P2P environments can lead to scenarios where each mobile device can act both as

service provider and consumer, or create composite services collaboratively. The merging can also provide better options for service discovery out of the huge number of possible web services provided by mobile service providers. One of the main goals of the thesis is to realize mobile service provisioning in P2P networks and to build a framework which incorporates effective service discovery mechanisms, manages the dynamic service registries, and handles the mobility of the service providers.

- ***Hosting complex services on mobile devices:*** The increasing processing power and storage of mobile devices and their support of multiple network interfaces allow them to host services in distributed environments. Some recent efforts have already attempted to facilitate provisioning of mobile hosted services. However, these efforts have not addressed the issue about how to host heavy-duty services on mobile devices with limited computing resources (e.g. processing power, memory). Another principal goal of the thesis is trying to build a framework which partitions the workload of complex mobile services in a distributed environment, and keeps the web service interfaces on mobile devices. The aim is to use the mobile device as the integration point with the support of backend nodes and other web services. The functions requiring the resources or participation of the mobile device need to be executed locally. The target is to support hosting services involving complex business processes on the mobile device, by partitioning the tasks and transparently delegating the heavy-duty tasks to remote servers.

- ***Developing efficient service partitioning algorithm:*** Offloading is the mechanism to leverage effectively the capability of mobile devices, by delegating some computing load to nearby resource-rich surrogates. It is still lacking of a generic solution to

leverage the capability of mobile devices in hosting services, by efficiently partitioning the tasks and offloading the resource-demanding tasks according to the runtime resource context of the mobile device. One primary objective of the thesis is to establish an efficient algorithm for service application partitioning, which considers the dynamic status of the mobile device in terms of the available processing power, memory space, and communication bandwidth.

- ***Designing offloading platform for pervasive services:*** In pervasive computing environment, resource availability and user mobility are highly dynamic. To ensure efficient service execution on mobile devices, runtime offloading needs an intelligent decision-making module to adapt to the dynamic changes in the pervasive environment. An adaptive offloading system is required which can trigger offloading at the right time and offload the right tasks to achieve low offloading overhead and efficient service execution. For example, when the wireless connection is excellent, the offloading system should decide to offload a large amount of application execution to avoid additional offloading in the near future. Another key objective of the thesis is to design an adaptive offloading system that can efficiently select the right partitions to offload and beneficially trigger task offloading to surrogates at the right time.

There is increasingly broad availability of tethered computing, storage, and communication resources being spare on commercial clouds, or at wireless hotspots, or at user's personal computers. Hence, the goal is to enhance a mobile device's run-time capacity to dynamically and transparently establish a distributed execution platform with the other computing resources in its environment.

1.5 Research Contributions

The contributions of the thesis lie in the design of a light-weight and efficient framework for hosting services involving complex business processes on mobile devices. The framework provides support for executing functions locally which require the resources of the mobile device and offloading the resource demanding tasks to backend servers. The offloading framework considers both the interaction properties and the resource consumptions while performing execution partitioning and offloading. In brief, the contributions of the thesis are highlighted below.

- ***Scalability and integration of mobile hosted services:*** In terms of scalability, the communication using verbose XML based SOAP messages introduces message overhead to the mobile service host, consuming its resources extensively. However, compression of SOAP messages comes with the trade-off of extra processing cost and causes further performance latencies. The performance penalties of different compression techniques are studied in detail, and the observations identified that BinXML is the most suitable compression option. This binary encoding technique was adapted for the mobile service invocation cycle. In terms of integration, the Enterprise Service Bus (ESB) was studied in detail and found to be the appropriate platform for realizing the integration framework.

- ***Mobile service provider in P2P environments:*** The research acknowledged that P2P environments offer a large scope for many applications where mobile service providers can be adapted. The P2P architecture offers several technical advantages to the mobile host with improved mobile service discovery mechanisms. The thesis realized mobile service provisioning in P2P environments using the JXTA (Juxtapose) network. The framework is designed with a mobile services gateway based on the ESB architecture,

which incorporates effective service discovery mechanisms, manages the dynamic service registries, and handles the mobility of the service providers. The framework solves the integration and interoperability problems with traditional web services platform.

- ***Partitioning mobile hosted service execution:*** Compared to stationary nodes, mobile devices have limited memory capacity and slower processing speed. Therefore, hosting complex services on mobile devices demands a flexible and scalable execution environment, which can exploit backend servers to delegate heavy-duty executions. The thesis constructs the framework which partitions the workload of complex mobile services and keeps the web service interfaces on mobile devices. Light-weight and open source kXML and kSOAP packages are used for developing the partitioned mobile service provider architecture. The architecture facilitates the use of application partitioning techniques to execute the service application in a distributed manner.

- ***Efficient partitioning approach:*** An ideal partitioning solution must consider the dynamic conditions of the resources, to partition the tasks between a mobile device and its backend nodes. The thesis proposes an efficient $(k+1)$ partitioning algorithm that considers a combination of the cost values of memory, processing speed, and bandwidth resources of the mobile device. The partitioning approach also considers the interaction properties and dependencies between the tasks while performing execution partitioning. The proposed partitioning approach thus significantly increases the efficiency of mobile hosted services in dynamic execution environments.

- ***Adaptive offloading system:*** To accomplish runtime offloading beneficially on mobile devices, distributed service execution needs

an adaptive offloading system which can trigger offloading at the right time and offload the right tasks to achieve low offloading overheads. The thesis designs an adaptive offloading system that uses the Fuzzy Control model to make effective offloading decisions and to achieve adaptability in triggering execution offloading. A composite cost metric which represents the interaction properties between task executions is used by the adaptive offloading system for selecting execution partitioning efficiently on the runtime.

1.6 Thesis Structure

The remainder of this thesis is organized as follows.

Chapter 2 discusses the state of the art for the research addressed by the thesis. The chapter first introduces the web services technology along with associated standards and protocols. Later, the chapter introduces the developments in mobile technology domain in terms of device and transmission capabilities, enabling opportunities for mobile web services. Then the supported platforms, standardization efforts, and SOAP transmission mechanisms for mobile web services are discussed in detail. The chapter concludes with a discussion about the currently open issues and some related works in realizing mobile web services.

Chapter 3 explains the research with mobile hosted service provisioning and provides the motivation and related technologies for the upcoming chapters. First, the chapter introduces the concept of mobile hosted service provisioning, along with the feasibility, performance, and application analysis of the mobile service provider. Then some alternative technologies for hosting services on mobile devices are discussed. The chapter later addresses the scalability related standards for mobile service providers. Finally, the integration aspects are discussed for mobile hosted service provisioning, by introducing the ESB technology, the Java Business Integration (JBI) specification, and the open source ServiceMix tool in realizing the integration framework.

Chapter 4 describes the details of mobile web service provisioning in P2P networks. First, the chapter introduces the convergence of web services and P2P technologies, the basics of P2P technologies and JXTA platform, and the related projects addressing this issue. The chapter later introduces a P2P framework, based on the JXTA network and ESB based mobile services gateway. Then the chapter discusses how distributed service registries handle mobility of the service providers in P2P networks. Later, the discovery mechanisms are discussed for mobile web services. And finally, the process of service discovery and invocation in the JXME network (JXTA for J2ME (Java 2 Platform, Micro Edition)) proposed in the thesis, are explained.

Chapter 5 presents the framework for provisioning heavy-duty services from resource constrained mobile devices, by partitioning the execution of service applications. The chapter first introduces the service partitioning techniques along with the design guidelines. Then the chapter discusses different service partitioning schemes applicable in hosting partitioned mobile services. Next, the proposed framework for partitioned mobile service provider is presented. The chapter concludes with the performance model for partitioned mobile services and the evaluation of performance with partitioning.

Chapter 6 proposes an efficient service partitioning approach that considers a combination of the dynamic cost values of memory, processing speed, and bandwidth resources of the mobile device. First, the chapter explains the concept of resource consideration in offloading pervasive services. Then the cost values and implications for dynamic offloading are discussed in detail. The chapter then presents the efficient $(k+1)$ partitioning algorithm that considers the available memory, processing power, and bandwidth of the mobile device in making partitioning decision of the tasks. Finally, the chapter concludes with experimental evaluations that demonstrate the efficiency of the proposed approach.

Chapter 7 presents an adaptive offloading system for pervasive services in cloud environments, which makes triggering decisions by

using an offloading inference engine. The chapter first identifies the decision-making problems for adaptive offloading. Then the chapter introduces a distributed offloading platform and the approach in realizing the platform using Java virtual machines (JVM). Next, the adaptive offloading inference engine is introduced, which uses a Fuzzy Control model to trigger offloading and considers class granularity and a composite metric for selecting efficient partitioning. The chapter concludes with the evaluation of different performance metrics and other factors affecting the performance.

Finally, **Chapter 8** concludes the findings of the thesis and discusses the possible directions for future work associated with this research.

Chapter 2

Background

The Internet can be seen as a large-scale distributed information system with numerous information users and providers. Web services are widely used to realize loose coupling and provide distributed access between different information systems. The usual convention today is that an enterprise or a central service provider uses web services to provide access to its infrastructure. From the viewpoint of information systems engineering, this has influenced three major trends. First trend is the evolution from static contents to web services. The next trend is the evolution from client-server systems to peer-to-peer and pervasive computing systems. Finally, with the novel developments in wireless communication technology, the latest trend is the shift from stationary to mobile distributed information management. This chapter introduces these developments from the literature and points out the research goals for the thesis in provisioning services from resource constrained mobile devices.

2.1 Web Services Technologies

Service-Oriented Architecture (SOA) is a trend in information systems engineering and the software industry's response to the problem of managing large monolithic applications [Burbeck, 2000], [Endrei et al., 2004]. SOA is a component model that delivers application functionality as services to end-user applications and other services, bringing the benefits of loose coupling and encapsulation to the enterprise application integration. Services encapsulate reusable business function and are

defined by explicit, platform independent interfaces. Services are invoked through communication protocols that facilitate location transparency and interoperability. SOA defines participating roles as, *service provider*, *service client*, and *service registry*. Figure 2.1 shows the SOA collaborations. The operations *publish*, *find*, *bind* and the artifacts *services* and *service descriptions* are all shown in the figure. SOA is not a new notion and many technologies like CORBA (Common Object Request Broker Architecture) [OMG, 2004] and DCOM (Distributed Component Object Model) [Brown and Kindel, 1998] at least partly represent this idea. Using web services for SOA provides certain advantages over other technologies. Specifically, web services are based on a set of still evolving, well-defined W3C standards, which allow lot more than just defining interfaces.

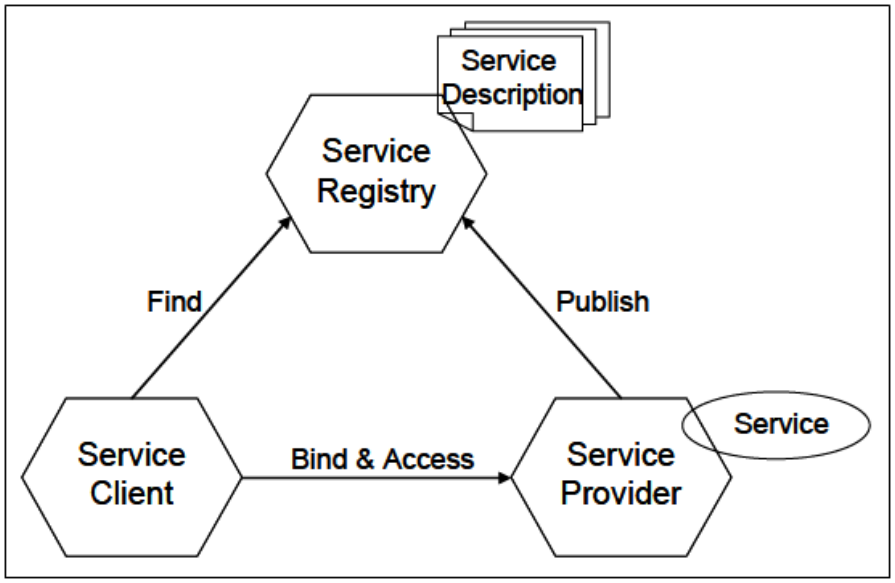

Figure 2.1: SOA collaborations [Gottschalk et al., 2002].

Web services are self-contained modular applications, which can be described, published, located, and invoked over a network, generally the World Wide Web (WWW). They are changing the Internet from business-to-consumer interactions to business-to-business interactions. Web services technology and its protocol stack are based on open standards and are widely accepted in the Internet community. Web services have

wide range of applications and can range from simple stock quotes to pervasive applications with context-awareness like weather forecasts, map services etc. The main advantage of web service technology lies in its simplicity in expression, communication, and servicing. The componentized architecture of web services also makes them reusable, thereby reducing the development time and costs [Alonso et al., 2004].

2.1.1 Web Services Architecture

Web services architecture is defined by the Web Services Activity (WSA) [Booth et al., 2004] and enables application-to-application communication over the Internet. The goal of the Web Services Activity is to develop a set of technologies in order to lead web services to their full potential [Gottschalk et al., 2002]. Public interfaces of web services are defined and described using Web Services Description Language (WSDL). Web services allow access to software components through standard Web technologies and protocols like SOAP and HTTP [Fielding et al., 1999], regardless of their platforms, implementation details. By following the same architecture as SOA, a service provider develops and deploys the web service and publishes its description and binding/access details (WSDL) to a public registry, generally a UDDI based registry. Any potential client queries the UDDI registry (Find) and retrieves the service description. After the WSDL has been retrieved, the service requester binds to the service provider by invoking the service through SOAP. The communication between the client and UDDI registry is also based on SOAP [Curbera et al., 2002]. The following subsections introduce web services protocols, standards, and their industrial adaptability efforts.

2.1.2 Simple Object Access Protocol (SOAP)

SOAP is a lightweight XML based protocol for exchanging structured information between peers in a decentralized and distributed environment [Gudgin et al., 2007]. It provides messages to communicate

between applications running on different operating systems, with different technologies, and programming languages. A SOAP message is an XML document that consists of a mandatory *SOAP Envelope*, which contains an optional *SOAP Header* and a mandatory *SOAP Body*. The SOAP message structure is shown in Figure 2.2.

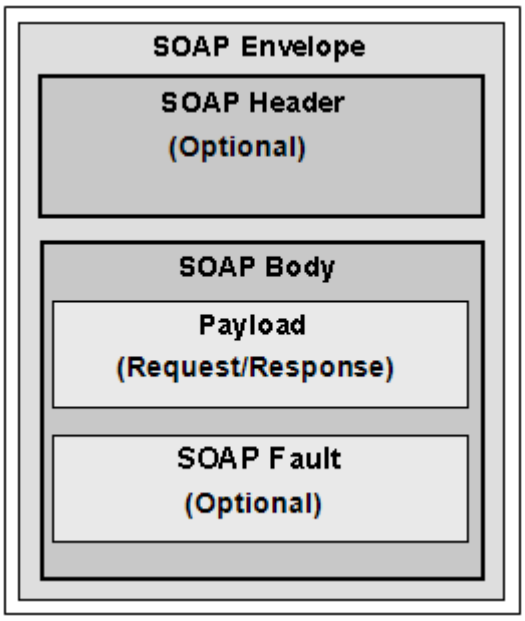

Figure 2.2: SOAP message structure (based on [Gudgin et al., 2007]).

Envelope is the root element of the SOAP message. It specifies two things: an XML namespace and an encoding style. The *namespace* declaration gives a clue for the used SOAP version. SOAP versions 1.1 and 1.2 are almost the same; the complete set of differences between them can be viewed at [Mitra and Lafon, 2007]. The *encodingStyle* attribute indicates the serialization rules used in the message, which can be explicitly overridden in the child elements of the Envelope.

Header is an optional sub-element of Envelope, which is a flexible mechanism for extending the SOAP message in a decentralized manner without prior agreement between the communicating parties. The header entries contain information such as, authentication information, digital signatures, transaction management details, payment details etc. There are a couple of defined attributes for the header entries. Such as *actor*,

which indicates the recipient of the header element; and *mustUnderstand*, which indicates whether successful processing of the header entry is mandatory by the recipient.

Body is a mandatory sub-element of Envelope, which contains the message *Payload* intended for the recipient of the SOAP message. Generally, it contains marshaled RPC (Remote Procedure Call) calls or error reports. In the case of a request message, the payload of the message is typically a request to perform some service, and optionally, to return some results. In the case of a response message, the payload is typically the results of the request, or a fault. The optional **Fault** sub-element of the SOAP body specifies error information. An error could be generated at any intermediary along the message path while processing SOAP message.

2.1.3 Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is an XML based specification that defines the means of describing web services [Christensen et al., 2001]. It specifies the Interface information describing all available public functions, data type information for all message requests and message responses, binding information about the transport protocol to be used, address information for locating the specified web service etc. Using WSDL, a client can locate a web service and invoke any of its publicly available functions. The process can also be automated, enabling applications to easily integrate with new services with little or no manual interaction. A WSDL document describes a web service as a collection of abstract items called *ports* or *endpoints*. The WSDL specification uses the following main elements in the definition of services: *definitions*, *types*, *message*, *portType*, *binding*, and *service*. The WSDL 2.0 version uses the terms *interface* and *endpoint*, instead of *portType* and *port* respectively used by WSDL 1.1. The structure of WSDL is shown in Figure 2.3.

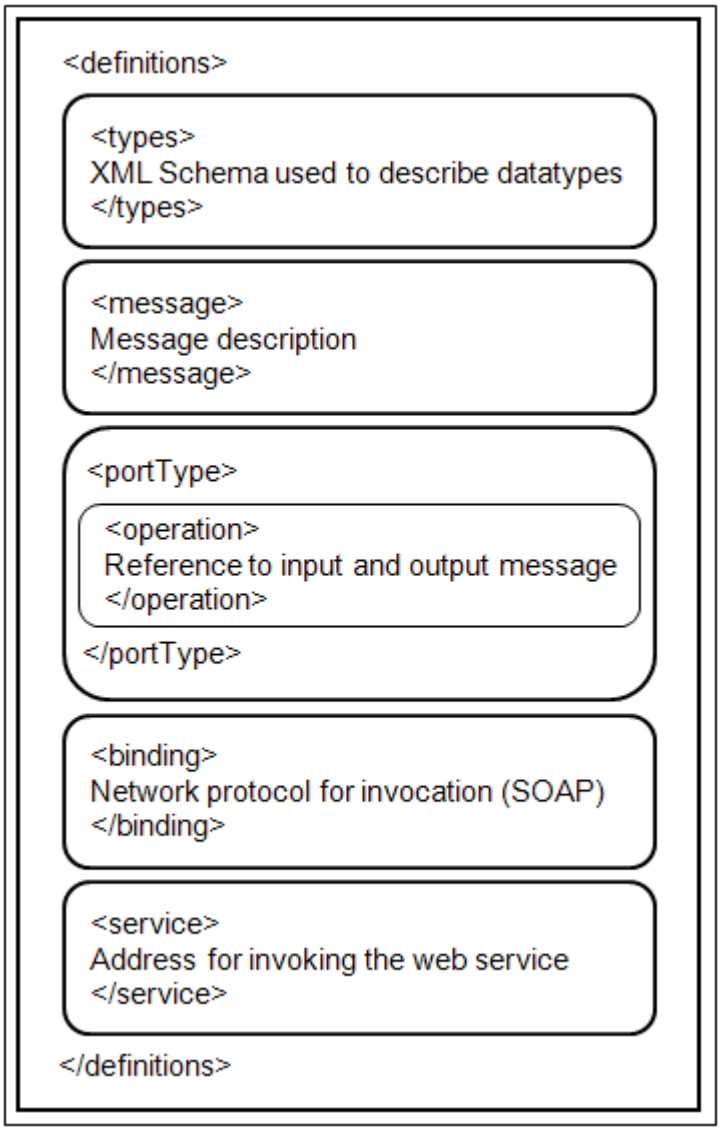

Figure 2.3: Structure of WSDL document [Christensen et al., 2001].

Definitions is the root element of every WSDL document. It defines the name of the web service, declares multiple namespaces used throughout the document, and contains all the sub-elements described below.

Types is a sub-element of Definitions and the container of all data type definitions. The element describes all the data types exchanged between the web service consumer and the web service provider. The complex data types and the corresponding user defined data types are represented using some type system, such as XSD (XML Schema

Definition). If the service uses only XML schema with built-in simple types such as strings and integers, then the Types element is not required.

Message is an abstract definition of the data being communicated. It defines all the input and output parameters of the publicly available functions of a service. The element specifies the name of the message and contains zero or more message part elements.

PortType is an abstract set of operations, supported by one or more ports. It is the most important element of the WSDL document, as it defines the web service. The element specifies the operations that can be performed, and the messages that are involved for the service. **Operation** is a sub-element of PortType and gives an abstract description of an action supported by the service.

Binding element describes the concrete specifics of how the service will be implemented on the wire. WSDL includes built-in extensions for defining SOAP services, and hence the SOAP specific information is incorporated in this element.

Service is a collection of related ports. It defines the address for invoking the specified web service. **Port** is a sub-element of Service, which is a single endpoint, defined as a combination of a binding and a network address.

2.1.4 Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovery and Integration (UDDI) is a cross-industry effort, driven by all major platform and software providers, such as Dell, Fujitsu, HP, IBM, Intel, Microsoft, SAP, Oracle, Sun, and Hitachi. UDDI is the name of a group of web-based registries, which expose information about an entity (be it a business entity or another entity), and its technical interfaces, or APIs (Application Programming Interface). These registries are run by multiple operator sites, whose basic services can be accessed by anyone free of charge [Bellwood, 2002].

From a business developer's point of view, UDDI is similar to an Internet search engine, which can be used to browse UDDI registries to view different businesses that expose web services and specifications of those services. Software developers can use the UDDI Programmers API to query the registry for discovering services matching different criteria. Both business developers and software developers can publish new business entities and services to the UDDI registry. Conceptually, a business can register three types of information into a UDDI registry.

- *White pages* with basic contact information and identifiers about a company, including business name, address, contact information and unique identifiers. This information allows clients to discover web services based on the business identification.
- *Yellow pages* with information that describes a web service using different categorizations (taxonomies).
- *Green pages* with technical information that describes the behaviors and supported functions of a web service. This information includes pointers to the grouping information of web services and the locations of web services.

The information that makes up a registration consists of several data structure types. These data structures are shown in Figure 2.4, and are briefly described below.

Business Entity represents basic information about a business, including contact data, categorization, identifiers, descriptions, etc. Publisher Assertion is used to establish public relationships between two Business Entity structures. Such a relationship is visible to the public only if both companies have created the same Publisher Assertion documents separately, i.e., business relationships cannot be one-sided.

Business Service represents a single, logical service classification, including information about how to bind to a service, what

type of service it is, etc. A Business Entity can contain one or more Business Service structures.

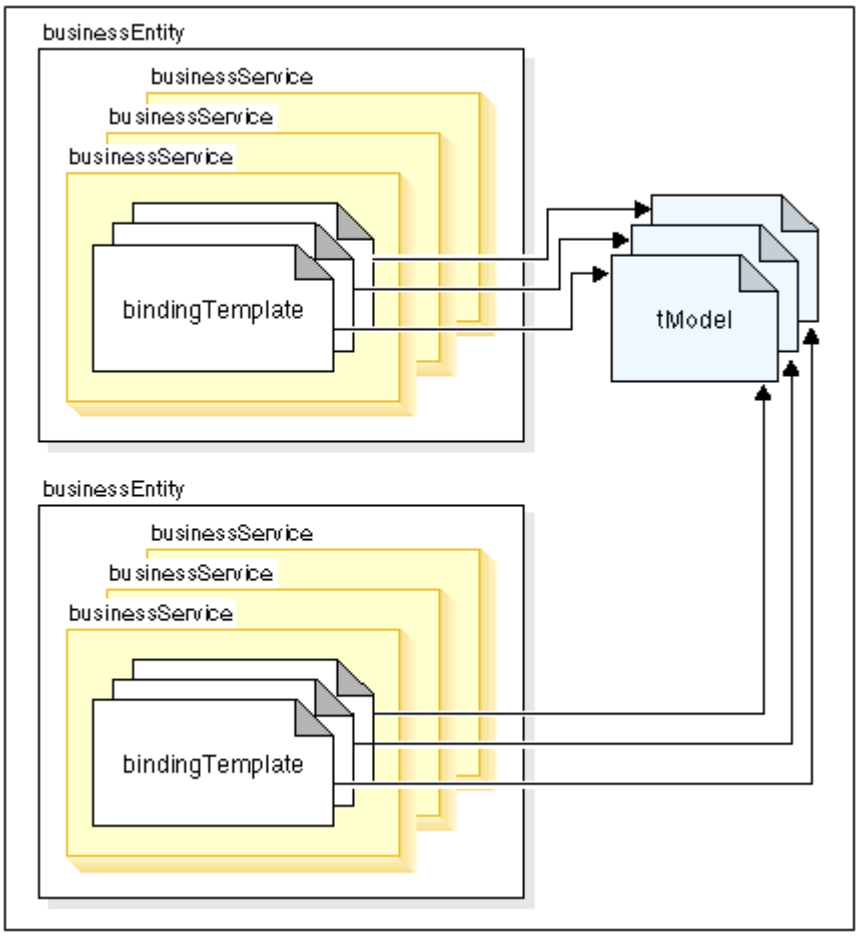

Figure 2.4: Schematic view of a UDDI registry (based on [Bellwood, 2002]).

Binding Template contains the technical descriptions of the web services, represented by the Business Service structure. It also contains the access point URL (Uniform Resource Locator) of the web service, but does not contain the service specification details. A Business Service can contain one or more Binding Templates. It is similar to the `<service>` element of the WSDL described above.

TModel is an abstract description of a particular specification or behavior, to which the web service adheres. For example, a TModel can be defined to represent a portType defined by the WSDL. Then a business service, implementing the portType can be specified by associating the TModel with one of the binding templates of the business service.

2.1.5 Web Services Interoperability

In order to facilitate the development of truly interoperable web services, the Web Services Interoperability (WS-I) organization [WS-I, 2004] was formed in February 2002. WS-I is an open, industry consortium of about 150 companies from diverse industries like automotive, consumer packaged goods, finance, government, insurance, media, telecommunications, travel, and computer industries. As a standards integrator WS-I supports the relationships with standards bodies who own specifications and fosters communication and cooperation with industry consortia and other organizations. The main goal of the consortium is to encourage web service adoption and accelerating interoperable web service development by providing guidance, best practices, and other resources.

WS-I delivers a collection of profiles, which are descriptions of conventions and practices for the use of specific combination of web services through which systems can interact. These profiles support technical requirements and specifications to achieve interoperable web services. The consortium is also responsible for providing use cases, usage scenarios, sample applications, and testing tools. WS-I has finalized the *Basic Profile*, *Attachments Profile*, *Simple SOAP Binding Profile*, and *Basic Security Profile* for web services. WS-I Basic Profile Version 1.2 [Ballinger et al., 2007] supports the specifications like SOAP 1.1, WSDL 1.1, UDDI 2.0, WS-Addressing etc. The profile supports three usage scenarios. A usage scenario is a design pattern of interacting entities like actors, roles and message exchange patterns [Werden et al., 2003]. The scenarios are, **(1)** *One-way* usage scenario, which is to be used when loss of information can be tolerated. **(2)** *Synchronous request/response* usage scenario. **(3)** *Basic callback* usage scenario simulating an asynchronous operation using synchronous operations. The Attachment profile complements the Basic Profile 1.1 to add support for SOAP with Attachments (SwA), with SOAP messages. Simple SOAP Binding Profile

is derived from Basic Profile requirements related to serialization of an envelope, and its representation in the message.

2.2 Developments in Mobile Technology Domain

Alongside to the SOA developments, the capabilities of today's high-end mobile phones and PDAs have increased significantly, both in terms of processing powers and memory capabilities. Smartphones are becoming pervasive and are being used in wide range of applications like location based services, mobile banking services, ubiquitous computing etc. The main driving force for the rapid acceptance of such small mobile devices is the capability to run applications at anytime and anywhere, especially while on the move [Helal et al., 1999].

The experience from Japanese market shows that the most important factor in this development is that the terminals are permanently carried around, and thus people can use the devices all the time for various things [Ichikawa, 2002]. The telecom industry estimated that there are around 5.3 billion mobile users by October 2010 (that is 77 percent of the world population) [ITU, 2010]. According to their analysis over 85 percent of the mobile handsets will be Internet-enabled by 2011 and the tendency will be growing [ITU, 2010]. The market capture of such smartphones became evident when 12.1 million PDA-sized devices were sold in 2003, including all PDA-phones and smartphones. The number of Java enabled mobile phones sold during the same time, has outnumbered the number of PCs (Personal Computers) sold [Rollman and Schneider, 2004]. In 2005, smartphones outsold PDAs by a factor of 3.4 to 1. Then during the first half of 2006, 42.1 million smartphones and PDAs were sold in combination. According to [CIA, 2006], Internet-enabled smartphone sales will rapidly grow and will surpass PDA sales by an 11:1 margin in 2011.

2.2.1 Device Capabilities

Traditionally, the hand-held devices have many resource limitations like low computation capabilities, limited storage capacities, and small display screens that could only display few lines of text with poor rendering quality. The new smartphones have larger graphics-oriented screens with support for colors, which enhance the wireless experience and entice the cellular users to exploit them for different services. They are also being provided with built-in cameras, infrared ports that can be used to control home devices, and even fingerprint sensors for secure transactions [NTT DoCoMo, 2005].

From the hardware aspects of smartphones, most of these mobile devices are using CPUs (Central Processing Unit) which are based on *ARM (Advanced RISC Machine) architecture*. The ARM architecture is a 16/32-bit *RISC (Reduced Instruction Set Computing)* processor architecture which is being widely used in embedded designs. ARM processors offer combination of advanced logic, robust functionality, and energy efficiency at low cost and simpler designs enabling easy integration. Smartphones are specifically using ARM9 series processors. ARM9 [ARM, 2005] processor can deliver up to 220MIPS (Million instructions per second) at 200MHz (Megahertz) on a 0.18µm process.

These higher speeds enable mobile devices to swiftly operate more complex data and thus tremendously increase their computability. All ARM9 family processors feature the *Thumb* compressed instruction set and *EmbeddedICE* JTAG-based (Joint Test Action Group) software debug logic. Apart from smartphones ARM9 processors are used in automotive control, instrumentation, safety systems, set-top boxes, high-end printers, PDAs, and multimedia formats such as MP3 (MPEG-1 Audio Layer 3) audio and MPEG4 (Moving Picture Experts Group) video.

There are also breakthroughs in the memory capabilities of smartphones, like the NAND flash memory. Traditionally SDRAM (Synchronous Dynamic Random Access Memory) and NOR-based flash

memory are used in smartphones. The transition to NAND flash memory has huge advantage in performance. NAND is 60 times faster and 80 times less energy consuming than NOR-based memory [Greenberg, 2005]. All these improvements expand the utility of mobile devices and fulfill users' demand in terms of device performance, and thus expanding their user base.

2.2.2 Transmission Capabilities

Concurrent to the device capabilities, the data transmission rates across the wireless network also have increased significantly. Traditionally the second-generation (2G) GSM (Global System for Mobile communications) networks delivered high quality and secure mobile voice and data services like SMS (Short Message Service), circuit switched Internet access, etc. with full roaming capabilities across the world. The GSM platform is a widely successful wireless technology. But, with the advent of the interim-generation (2.5G) technologies [ETSI, 1997] like GPRS (General Packet Radio Service) [Rysavy, 1998] and EDGE (Enhanced Data rates for GSM Evolution) [Ericsson, 2003a], and third-generation (3G) technologies [3GPP, 2007] like UMTS (Universal Mobile Telecommunications System) [Umtsworld, 2002], still higher data transmission rates (in the order of few hundred KBs to 2MBs) are achieved in the wireless domain.

The advent of Fourth Generation Wireless Services (4G) technologies and their deployment in south Asian countries suggests that mobile data transmission at the rate of few GB (Gigabyte) is also possible [4GPress, 2005]. 4G technologies will provide an end-to-end IP solution where voice, data, and streamed multimedia can be served to users on an "Anytime, Anywhere" basis. 4G will be capable of providing 100 Mbps and 1 Gbps, in outdoor and indoor environments respectively, with end-to-end quality of service and high security [Kim and Prasad, 2006]. NTT DoCoMo had already achieved 2.5Gbps packet transmission in the

downlink with 4G, while moving at 20km/h [4GPress, 2005]. These higher data transmission rates achieved with 3G and 4G technologies, along with the fast growing all-IP broadband based mobile networks boosting the rapid growth in the cellular market.

2.2.3 Nomadic Mobile Services

The developments in device capabilities and data transmission rates brought out a large scope and demand for software applications for smartphones in high-end wireless networks. Many software markets have evolved like NTT DoCoMo [NTT DoCoMo, 2007c] capturing the demand of this large mobile user base. Many nomadic services were provided to the mobile phone users. For example, NTT DoCoMo provides phone, video phone, *i-mode* (proprietary mobile internet platform) [NTT DoCoMo, 2007b], and mail (i-mode mail, Short Mail, and SMS) services. With i-mode, mobile phone users can get easy access to thousands of Internet sites, as well as specialized services such as e-mail, online shopping, mobile banking, ticket reservations, and restaurant reviews. Likewise, a free mapping, search, and navigation application for mobile phones is being provided by LocatioNet Systems [LocatioNet, 2007]. The company's free service called *Amaze* is like a hybrid between the popular *TomTom GPS (Global Positioning System) system* [TomTom, 2007] and *Google Maps* [Google, 2007].

Similarly, Google provides its local search tool *wireless search service* [Google Mobile, 2007], designed specifically for mobile device users, in particular travelers. Travel tools like the wireless email device *BlackBerry*, provides the ability to be permanently online with instant access to email. *PayPal* provides a mobile payment service, that lets mobile users send and receive funds on their cell phone via text message. Apart from these services many Location Based Services (LBS) have been developed in improving the general tourism experience. Further with the start of initial 4G services to consumers, a range of new choices can be

offered, including even higher speed in mobile Internet access, video and TV in an on-the-go, real time, on-demand format, and new options for networking and entertainment for home. Ultimately, 4G will allow consumers to select a single provider to deliver their entire home, mobile, and entertainment services [ETC, 2007].

These nomadic services bring benefits to all the participants of the mobile Web. The *mobile users* benefit as their mobile phone becomes the network computer and wallet PC (Personal Computer). The *enterprises* can benefit as they can support technologies and services that allow for anywhere and anytime connectivity with the business information sources. The *mobile operator networks* can increase their revenues with "open" business models. For example, NTT DoCoMo has proved their success with the *i-mode portal*, where the operator provides a framework and environment in which third party content developers can deploy their services [NTT DoCoMo, 2007a]. The *content providers* can in turn get incentives from these open models.

From the analysis of these nomadic mobile services, each operator provided some set of services applicable to specific group, over specific platforms. But most of the approaches were proprietary and followed specific protocols. This makes the services un-interoperable and the integration of services becomes highly challenging. Therefore, to overcome the interoperability issues and to reap the benefits of the SOA domain, the scope of the mobile terminals as both service clients and service providers is being observed.

2.3 Mobile Web Services

In the mobile web services domain, the resource constrained mobile devices are used as both web service clients and providers. Web services have a broad range of service distributions, and on the other hand cellular phones have large and swiftly expanding user base. Combining these two domains brings us a new trend and leads to manifold

opportunities to mobile operators, wireless equipment vendors, third-party application developers, and end users. By following the basic web services architecture, mobile web services enable communication via open XML interfaces and standardized protocols over radio links. Until recently, still the proprietary, application-specific, and terminal-specific interfaces are required for communication over radio links.

To support mobile web services, several organizations are working on the specifications front, such as, OMA [OMA, 2004, 2006a] and LA [Tourzan and Koga, 2006]. Some practical data service applications are available on the commercial front, such as, over-the-air provisioning (OTA), application handover etc. On the development front, SUN [Sun Microsystems, 2007e] and IBM toolkits [IBM Corporation, 2007a] are available for developing mobile web services. Thus, having these developments in early stages, we can safely assume that mobile web services are the road ahead. Figure 2.5 shows the deployment scenario of mobile web services, where mobile devices are used both as service providers and clients. This section introduces some of the mobile web services, the platforms, and the APIs supporting mobile web service development for the smartphones.

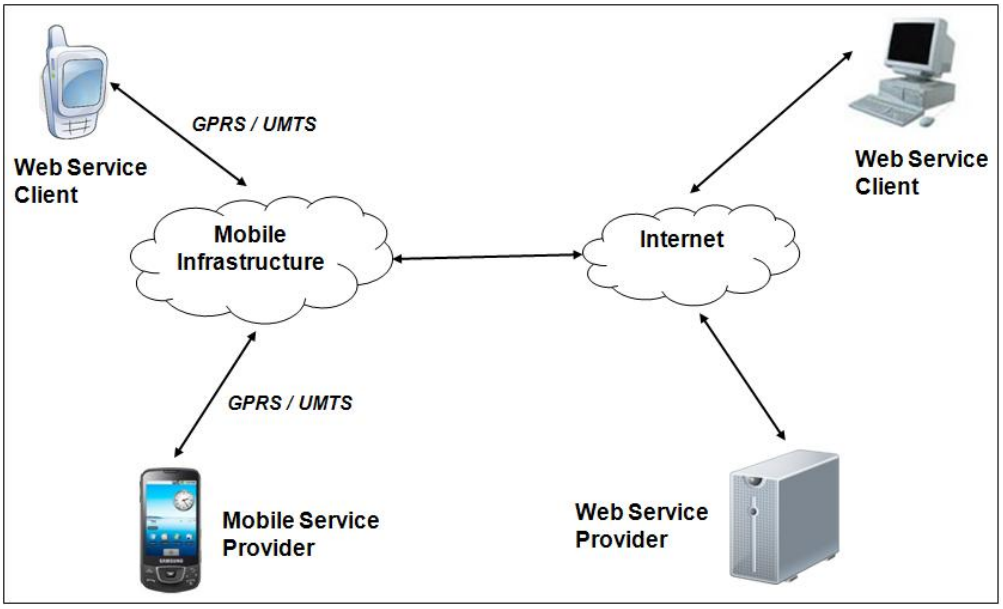

Figure 2.5: Mobile devices as service providers and clients.

Mobile terminals accessing web services are common these days and they cater for anytime and anywhere access to services [Balani, 2003a], [Forum Nokia, 2004], [Ellis and Young, 2003], [Benatallah and Maamar, 2003]. Some interesting mobile web service applications are the services like e-mail, information search, language translation, company news, etc. for employees who travel regularly. There are also public web services accessible from smartphones like the weather forecast, stock quotes etc.

Apart from the applications, there are some research efforts on efficient access of mobile web services with minimum loads on the mobile devices. For instance [Yang et al., 2003], propose an infrastructure for organizing and efficiently accessing mobile web services in broadcast environments. The idea is that, since sending data from a wireless device is the most power consuming process, therefore broadcasting can save power consumption by avoiding the costly uplink transmissions [Imielinski et al., 1994]. The approach defines a multi-channel model to carry information about mobile web services. The *UDDI channel* includes registry information about m-services (mobile web services); the *m-service channel* contains the description and executable code of each service; and the *data channel* contains the actual data needed for executing the mobile web service. The mobile service platforms take care of the resource issues for the services. The service platforms are discussed in the following subsection.

The usage of mobile terminals as web service client is also significant in the geospatial and location based services. Geographic Information Systems (GIS) make accessing geographical information service at anywhere and anytime feasible. Open Geospatial Consortium (OGC) is designing and providing GIS service over Internet with web service standards [Whiteside, 2007]. OGC Web Services (OWS) represent evolutionary, standards-based frameworks that enable seamless integration of a variety of online geo-processing services, and facilitate accessing them from smartphones and PDAs [Brisaboa et al., 2007].

Regarding industrial applications, there is a need for cross-organizational secure provisioning of services by experts who are usually on the move and only have their handheld devices or laptops with them [Pulkkinen et al., 2007]. The maintenance experts use mobile phones, PDAs, laptops to access traditional web services that provide functionality for the condition monitoring, billing, maintenance, faults analysis, repair management, and other activities. On the other hand, there is a need to collect data from their mobile devices. These data includes activity logs, collected information on customers' sites (text notes, photos, videos, audio recordings, etc.), current locations of experts, their availability, etc. Mobile hosted service provisioning (Chapter 3) may facilitate and generalize the solutions for accessing the data that reside on experts' mobile devices.

2.3.1 Platforms Supporting Mobile Web Services

Unlike the normal desktop applications, mobile applications are particularly restricted by the runtime environment of the devices. Usually a mobile application can only run on certain models of devices. The limitations come from different aspects: device operating system, the programming language and platforms used, device capabilities, and the size of device storage. Therefore, many factors have to be considered for developing applications for mobile devices. This subsection discusses some of the application development platforms available for smartphones.

Symbian OS (Operating System) C++

Symbian is an operating system derived from the Epoc operating system. Epoc was developed by Psion for their handhelds in the 80's [Harrison, 2003]. The C++ based Symbian OS (Operating System) provides a secure, reliable operating system for mobile information devices. It is specifically designed with low power consumption and small memory footprint suitable for mobile devices, and provides a stable platform for the

telecommunication technologies such as GPRS, Bluetooth, SyncML, and ultimately 3G. Symbian OS is not only an operating system but actually a full software and communication platform. Symbian Inc. develops the base operating system and licenses it out to phone manufacturers. Vendors then build a user interface on top of the base operating system. The vendors can also customize the operating system for specific purposes.

Symbian OS phones available on the market are based on three user interfaces open to C++ programmers: *Nokia Series 80 Platform* (Nokia 9200 series communicator) [Forum Nokia, 2007c], *Nokia Series 60 Platform* (Nokia 7650, Nokia 3650) [Forum Nokia, 2007b], and *UIQ (User Interface Quartz) technology* (SonyEricsson P800/P900/P910i/P990 smartphones) [UIQ, 2007]. Apart from C++ support, all these designs are also open to Java programming. CodeWarrior for Symbian OS from Metrowerks, C++Builder Mobile Set from Borland, and Visual Studio from Microsoft provide tool support for Symbian C++ programmers.

PersonalJava

PersonalJava [Sun Microsystems, 2007d] is a Java programming environment targeted at developing applications for resource-constrained devices like smartphones, PDAs, and many other embedded devices. It was the first attempt by Sun to produce a Java application environment, Java Virtual Machine (JVM) for mobile devices. PersonalJava specifies a reduced set of class libraries compared to the Java desktop environment. Over the years, the Symbian OS port of PersonalJava [Symbian, 2007] has undergone substantial optimizations that in combination with the hardware performance enhancements make PersonalJava a powerful alternative for the development of mobile applications. In addition, the memory footprint of PersonalJava application environment has been optimized to run in resource-limited environments while providing near desktop web-fidelity [Frank, 2004].

The PersonalJava profile is based on the JDK1.1 (Java Development Kit) [Sun Microsystems, 2008], but makes a number of packages, classes, and methods optional. It gives the capability to create Web applets and other mobile phone applications. Currently PersonalJava is going through *Sun End of Life (EOL)* process as J2ME (Java 2 Platform, Micro Edition) is preferred ahead of PersonalJava. Further support to this platform will slowly be removed, though some of today's smartphones use it. PersonalJava profile supported smartphones include Nokia Communicator 9200 series phones/PDAs (9210, 9290 and 9210i) and SonyEricsson P800/P900/P910i smartphones [Sun Microsystems, 2007b].

Java ME

Java™ Platform, Micro Edition (Java ME) [Sun Microsystems, 2007f] is a subset of the *Java 2 Platform, Standard Edition (J2SE)*, and is the most ubiquitous Java application platform for mobile devices such as mobile phones, PDAs, TV set-top boxes, printers, in-vehicle telematics systems, and a broad range of other embedded devices. The Java ME platform includes flexible user interfaces, a robust security model, a broad range of built-in network protocols, and extensive support for networked and offline applications that can be downloaded dynamically. Java ME is very successful and every major manufacturer is embedding Java ME on some of their phones. All Java ME implementations provide support for the HTTP protocol [Fielding et al., 1999]. This guarantees the availability of HTTP as a transport mechanism for web services.

Java ME technology was originally created in order to deal with the constraints associated with building Java applications for small devices with limited memory, display and power capacity. Java ME platform is a collection of technologies and specifications that can be combined to construct a complete Java runtime environment specifically to fit the requirements of a particular device or market. The Java ME

technology is based on three elements; *configuration* as the most basic set of libraries and virtual machine capabilities for a broad range of devices, *profile* as a set of APIs that support a narrower range of devices, and an *optional package* as a set of technology-specific APIs. Over time the Java ME platform has been divided into two configurations; the *Connected Limited Device Configuration (CLDC)* [Sun Microsystems, 2000] and the *Connected Device Configuration (CDC)* [Sun Microsystems, 2005].

CLDC is specifically designed to cater for the devices with very limited memory, processing power and graphical capabilities. The development environment for Java ME on the CLDC devices is the *Mobile Information Device Profile (MIDP)*. Combined with CLDC and its *Kilo Virtual Machine (kVM)* [Sun Microsystems, 2000], this profile provides a complete Java Runtime Environment (JRE) for mobile phones and devices with similar capabilities.

CDC comes with three different profiles: the *Foundation Profile* (JSR 219), the *Personal Basis Profile* (JSR 217) and the *Personal Profile* (JSR 216). The Personal Profile is aimed at devices that require full Graphical User Interface (GUI) or Internet applet support, such as high-end PDAs, smartphones, and game consoles. Personal Profile replaces PersonalJava technology, and provides PersonalJava applications a clear migration path to the Java ME platform.

Combining various optional packages can further extend the Java ME platform. These optional packages offer standard APIs to support both existing and emerging technologies like Bluetooth, web services, wireless messaging, multimedia, database connectivity, etc.

.NET Compact Framework

Microsoft .NET Framework [MSDN, 2007b] is a software component added to the Microsoft Windows operating system. The .NET Compact Framework [MSDN, 2007a] is a subset of .NET Framework and provides a robust environment for developing mobile applications. The framework

is intended to be used by applications created for the Windows platform. The .NET Compact Framework's managed code and web services enable the development of secure, downloadable applications on devices such as PDAs, mobile phones, and set-top boxes. The framework uses some of the class libraries of the .NET Framework and a few libraries designed specifically for mobile devices.

2.3.2 SOAP Implementations for Resource Constrained Environments

To act as web service clients, the mobile devices should be able to consume web service messages. To request a web service, a smartphone should create the web service request messages (SOAP requests), send it to the web service provider, and be able to process the response messages received from the provider. To achieve this, the smartphone should support a SOAP parser for processing the web service messages. Many SOAP parsers like kSOAP2, WSOAP, etc. exist specifically for the resource constrained devices. This subsection discusses some of these platforms and SOAP processors.

gSOAP

The gSOAP toolkit is a platform-independent development environment for C and C++ based web services [Engelen and Gallivan, 2002]. gSOAP provides a transparent SOAP API through the use of compiler technology that hides irrelevant SOAP-specific details from the user. The compiler automatically maps native and user-defined C and C++ data types to semantically equivalent SOAP data types, and vice-versa. As a result, full SOAP interoperability is achieved with a simple API. The gSOAP toolkit is a mature and fast toolkit and is available as open source. The toolkit supports many platforms including embedded systems, and follows the WS-I Basic Profile 1.0a compliance recommendations.

eSOAP

Embedded SOAP (eSOAP) [Silva, 2001; Silva, 2007] is a small lightweight implementation of the SOAP 1.1 specification and is exclusively designed for embedded systems. The eSOAP toolkit has C++ and Java libraries that provide a SOAP processing engine for the embedded system. eSOAP achieves easy interoperability for networked embedded systems and the library is compact with a memory footprint less than 150KB. The toolkit is also portable as the core engine of eSOAP is written totally in ANSI (American National Standards Institute) C++. It uses C++ Standard Template Library (STL) whenever possible, and where STL is not available it provides a container library. The toolkit also provides a Java library with interface and classes for web service client development.

WSOAP

The Wireless SOAP (WSOAP) [Apte et al., 2005] aims to provide static encoding based on SOAP schema, leverages WSDL service description to create adaptive encoding for web service interfaces. WSOAP is actually a set of optimization techniques. The approach concentrates on functional message equivalence (also called Name Space Equivalency) rather than exactness. This protocol can be extremely useful between mobile devices and gateways where the resources are very limited, as WSOAP can reduce SOAP message size by 3-12 times.

Wingfoot SOAP

Wingfoot SOAP [Wingfoot, 2007b] is a lightweight client implementation of SOAP 1.1. It is specifically targeted for the MIDP/CLDC platform, but can also be used in PersonalJava, J2SE, and J2EE (Java 2 Platform, Enterprise Edition) environments. Wingfoot SOAP provides two different binaries. The first version `kvmwsoap_1.06.jar` targets the CLDC/MIDP platforms. The binary package includes a lightweight XML parser and is

37K in size. The XML parser is based on kXML. The second version `j2sewsoap_1.06.jar` targets the CDC/PersonalJava, J2SE, and J2EE platforms. It also includes the lightweight XML parser and is 34.5K in size [Wingfoot, 2007a]. Wingfoot SOAP provides its own mechanism of sending the SOAP messages over HTTP. SOAP over alternative transport protocols like SMTP (Simple Mail Transfer Protocol), UDP (User Datagram Protocol) etc. can be realized using Wingfoot SOAP, by implementing the Transport interface.

Java Specification Request 172 (JSR 172)

The Java Specification Request 172 (JSR 172) [Ellis and Young, 2003], defines the J2ME web services specification and a set of API for accessing web services from the J2ME environment. Sun also has provided a reference implementation of this API with the J2ME Web Services APIs (WSA) [Sun Microsystems, 2007a]. WSA enables J2ME devices to be web services clients, providing a programming model that is consistent with the standard web services platform. The API facilitates web service access from both CDC and CLDC configurations. JSR 172 also provides a light weight XML parser for J2ME platform, and is the first attempt towards the standardization of mobile web services.

kSOAP2

kSOAP is an open source API for SOAP parsing [kSOAP, 2007]. It is based on kXML parser, a lightweight and open source XML parser. kXML uses *XML pull parser* mechanism, a slight modification of *DOM (Document Object Model)* parser [Balani, 2003b]. Using the pull parser, the application is in control of when and where it asks the parser for the next event. kSOAP provides a SOAP parser with special type of mapping and marshalling mechanisms. Both kSOAP and kXML are thin, easy to use, and well documented; and hence can be used for resource constrained devices like mobile phones. The kSOAP parser automatically converts the

SOAP messages to Java data objects, similar to SOAP parsers. kSOAP is completely redesigned to kSOAP2 [kSOAP2, 2007], which has improved support for literal encoding and made *SOAP Serialization* support to be optional. kSOAP2 uses kXML2, the updated version of the kXML.

2.3.3 Different Transportation Protocols

SOAP messages can be carried on top of any underlying protocols such as HTTP, TCP (Transmission Control Protocol), UDP, BEEP (Block Extensible Exchange Protocol), and SMTP. Therefore, a binding framework has been defined for SOAP instead of a fixed binding. Specifically, the *SOAP binding framework specification* [Gudgin et al., 2003] provides a high level of flexibility in terms of how SOAP messages are transmitted. The following subsections introduce SOAP over different protocols, and address the ongoing research in this domain.

SOAP over HTTP

HTTP protocol is the most widely used transport mechanism for SOAP binding among the web services community. SOAP over HTTP is also the only concrete binding specification defined in the SOAP binding framework proposal. There are many reasons why HTTP is an attractive binding option. HTTP is already widely used on the Internet and universally supported by web servers. This provides a strong base for the adoption of web services with SOAP over HTTP binding. Moreover HTTP uses TCP/IP (Transmission Control Protocol/Internet Protocol) as its underlying transport, which ensures reliability of delivered packets, and most firewalls allow HTTP packets to pass through them.

SOAP over HTTP message can be transported by encapsulating the SOAP request into the HTTP GET or POST message body. Similarly, a SOAP response can be encapsulated into the body of the HTTP response. The message formats of the HTTP request and response are described in the Appendix. Listing 2.1 and 2.3 show the sample SOAP request and

response messages transmitted over the HTTP protocol. The service being invoked (`searchArticle`) is to find articles written on a specific date by a journalist on his mobile device (Listing 2.2).

Listing 2.1: SOAP request message over HTTP protocol.

```
POST / HTTP / 1.1
SOAPAction: searchArticle
Content-Type: text/xml
Content-Length: 1022
User-Agent: kSOAP/2.0
Host: 192.168.82.94:6666

<soap-env:Envelope
xmlns:xsi="http://www.w3.org/2003/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2003/XMLSchema"
xmlns:soap-enc="http://www.w3.org/2003/05/soap-
encoding"
xmlns:soap-env="http://www.w3.org/2003/05/soap-envelope">
   <soap-env:Header/>
   <soap-env:Body>
      <n0:searchArticle>
       … … …
      </n0:searchArticle>
   </soap-env:Body>
</soap-env:Envelope>
```

Listing 2.2: Sample web service `searchArticle`.

```
<webservice>
      <uri>http://mobilews.com/searchArticle</uri>
      <class>webservices.searchArticle</class>
      <operation>searchArticleByDate</operation>
</webservice>
```

Listing 2.3: SOAP response message over HTTP protocol.

```
HTTP/1.0 200 OK
Server: MobileServiceProvider
Content-Length: 585
Content-type: text/xml
Request-ID: 3
Connection: close

<soap-env:Envelope
xmlns:xsi="http://www.w3.org/2003/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2003/XMLSchema"
xmlns:soap-enc="http://www.w3.org/2003/05/soap-encoding"
xmlns:soap-env="http://www.w3.org/2003/05/soap-envelope">
   <soap-env:Header/>
   <soap-env:Body>
      <n0:searchArticleResponse>
       … … …
      </n0:searchArticleResponse>
   </soap-env:Body>
</soap-env:Envelope>
```

SOAP over Alternative Transportation Protocols

Instead of transmitting SOAP over HTTP, the message can directly be transported using TCP as the underlying protocol. There is not yet an official specification for *SOAP binding with TCP*; however, Apache Axis2 [Apache Software Foundation, 2007a] and Microsoft WSE (Web Service Enhancement) 2.0 [MSDN, 2007c] already include APIs that enable sending SOAP messages via TCP channel.

Axis2 also supports *SOAP over JMS* (Java Message Service). JMS offers a common way to create, send, receive, and read enterprise messages. JMS is appropriate transport protocol for SOAP when there is a requirement for web services to communicate asynchronously and reliably. Asynchronous communication ensures the sender of a message not to wait for a reply, and reliability assures the sender that the message will be delivered. But the problem with JMS mechanism is that the communicating parties are required to use the same infrastructure of

JMS, such as IBM WebSphere MQ [IBM Corporation, 2008]. Considering this issue, WS-ReliableMessaging [Bilorusets et al., 2005] standard draft has been developed to provide a framework for interoperability between different reliable transport infrastructures.

SOAP over SMTP is also possible and is presented in the W3C specification. In this transport mechanism, SOAP messages are encapsulated in the bodies of emails. SOAP over SMTP only allows asynchronous message exchange between web services [Cunnings et al., 2001]. *SOAP over BEEP* is also possible and is studied by [Kefali, 2004]. *SOAP over JXTA*, a peer to peer (P2P) technology, is addressed in [JXTA Community, 2007a]. In this mechanism, the SOAP messages are exchanged over the JXTA pipes.

Apart from these protocols, *SOAP over UDP* specification [Gudgin et al., 2004] defines the means of encapsulating SOAP messages into UDP packets. With this specification the SOAP message must be small enough to fit in one UDP packet. So the maximum size of a message should be 65,536 bytes (2^{16}). The specification supports four messaging patters: *Unicast one-way*, *Multicast one-way*, *Unicast request, unicast response* and *Multicast request, unicast response*. Using a set of extensive experiments, [Lai et al., 2005] show that the throughput of SOAP over UDP is 10 times higher than SOAP over HTTP in Wi-Fi environments. However, UDP is highly unreliable and packets delivered by UDP may be duplicated, arrive out of sequence, or even not reach their destinations. But UDP can be used reliably to transport messages by applying an Automatic Repeat Request (ARQ) protocol over UDP. [Gehlen et al., 2006] presents a reliable UDP SOAP binding for a mobile web service based middleware.

Considering the issues discussed above and not having proper standards and specifications, the thesis proceeded with SOAP over HTTP for mobile hosted service provisioning. Moreover, in mobile application development the J2ME standard mandates that all MIDP implementations must provide support for the HTTP protocol.

2.3.4 Standardization Efforts for Mobile Web Services

Most of the platforms and implementations supporting the development of mobile web services are proprietary, and have evolved concurrently. Hence, there are some standardization efforts from different groups to achieve some uniformity among these developments. This subsection discusses some of the prominent standardization efforts.

JSR 172

Before the emergence of JSR 172, there was no standardized support for web services in mobile environments. JSR 172 is the first attempt towards the standardization for mobile web services [Ellis and Young, 2003]. JSR 172 defines a J2ME web services specification and thus extends the web services platform to include Java ME client devices. J2ME web services specification defines two new optional packages: *XML Processing APIs* and *RPC-based access to web services*.

With JSR 172, the web services applications are portable and the specification allows generated stubs to be independent of the implementations. Therefore, these applications can be dynamically provisioned to any J2ME supported platform. With JSR 172, services and clients no longer have to be implemented on the same platform or by the same organization [Sun Microsystems, 2007c].

LA

Liberty Alliance (LA) project is the global body that is working to define and provide technology, knowledge, and certifications to build *identity* into the foundations of mobile and web service communication. The members of the Liberty Alliance envision a networked world, across which individuals and businesses can engage in virtually any transaction without compromising the privacy and security of the identity information [LA, 2007].

The Liberty Alliance project proposes the use of federated network identity to solve network identity problems, due to the lack of connectivity between identities in the wireless applications, especially in mobile networks. The *Liberty Identity Web Services Framework (ID-WSF)* [Tourzan and Koga, 2006] builds the federated identity foundation, and provides a framework for identity-based web services in a federated network environment. Nokia has developed *Nokia Mobile Web Services Framework* for its Series 60 and Series 80 phones, based on the Liberty ID-WSF specification [Forum Nokia, 2007a].

OMA

Open Mobile Alliance (OMA) group is directed at defining a unique specification/framework for mobile data services to achieve interoperability. OMA was formed in June 2002 by nearly 200 companies including the world's leading mobile operators, device and network suppliers, information technology companies, and content and service providers [OMA, 2004].

The current possible applications have a number of drawbacks. Firstly, the applications have to be created through tightly-coupled, costly, and close alliances between value-added service providers. Secondly, the applications have to be created based on mostly propriety models and disparate standards such as Wireless Application Protocol (WAP), Location, Presence, Identity etc. Thirdly, most of these standards have been devised specifically for the mobile environment from the ground up.

The *OMA Web Services Enabler specification* [OMA, 2006b] and *OMA Mobile Web Services Requirements specification* [OMA, 2006a] are destined to cover the drawbacks and envisioned to support the four types of mobile web service interactions: *server-to-server*, *server-to-mobile terminal*, *mobile terminal-to-server*, and *mobile terminal-to-mobile terminal (peer-to-peer)*.

2.4 Open Issues and Challenges

The next generation devices such as smartphones and PDAs are enabled to conduct tasks almost like personal computers and bring endless possibilities for wireless communication. Meanwhile, web services technology is designed to support interoperable machine to machine interaction over the network so that applications could communicate with each other directly in order to exchange data or conduct a task. Recent developments of 3G and 4G technologies have significantly increased the wireless data transmission rates and due to their popularity a great amount of applications for mobile based services are available and more are under development.

Mobility is the primary benefit provided by smartphones [Fox and Box, 2004]. But no effective mechanism is available for publishing and discovering mobile based services from smartphones with reasonable performance. Therefore, an efficient discovery mechanism is necessary and is critical to reduce bottlenecks in mobile service provisioning with success. The frequently used centralized registry UDDI for web services is designed for stable networks and cannot cater for the dynamic and spontaneous nature of mobile nodes. Therefore, an appropriate and effective mechanism for mobile service discovery is of urgent need.

Lack of interoperability is a major barrier of mobile based service provisioning. The mobile phone environment is characterized by different devices, platforms, and APIs [Teder, 2006]. This diversity hinders the deployment of mobile based services. Firstly, it increases the number of applications need to be developed and deployed for individual platforms in the smartphone market. Secondly, the applications must be customized for every mobile configuration with which it interfaces. The large number of existing configurations makes this complex and time consuming [OMA, 2006b].

Cellular networks provide security for the carried messages only up to the extent of mobile networks; whereas, mobile web service

messages may travel beyond this range [Moyo et al., 2006]. As mobile computing differs from traditional computing paradigms, therefore it would be inappropriate to implement WS-Security [Nadalin et al., 2006a] on smartphone platforms in the same manner as on traditional platforms [Nadalin, 2003]. The current processing power and memory of smartphones render the implementation of cryptographic operations on smartphones [Zheng and Ni, 2006].

The growth of data resulting from WS-Security is problematic because mobile subscribers are usually charged by the amount of data they transfer [Tian et al., 2004], and the battery power is consumed more by data transfer operations than CPU processing [Kangasharju, 2007]. Solutions exist to mitigate the additional size of SOAP messages; for example, data compression and binary XML encodings [Tian et al., 2004]. On the other hand, compressing and encrypting XML data results in interoperability problems. XML Encryption does not provide a mechanism to instruct a decrypting entity that it must decompress data after decrypting. Therefore, binary XML encodings are preferred to reduce the size of secured XML elements [Kangasharju et al., 2006].

Mobile web services operate in a networked environment characterized by limited bandwidth and high latency [Hirsch et al., 2006]. The requirement that the devices need to continue processing in the absence of a network connection is also an important fact. The mobile network constraints of high latency and disconnected operation may be handled programmatically through the use of asynchronous method calls [Kangasharju et al., 2007]. An asynchronous method does not block after finishing execution, and continues to perform other tasks in the meanwhile [Hirsch et al., 2006]. This allows processing to take place while a response is pending. On the arrival of the response, a callback method is invoked allowing the response to be handled.

With advances in mobile devices and wireless communications, the demand for mobile devices to run heavy-duty applications is increasing. But most of the available mobile devices have low resource capabilities.

In comparison to desktop nodes, mobile devices usually come with limited memory capacity, slower processing speed, and limited communication bandwidth. Meanwhile, in real-life environments stationary computing resources are normally rich. For instance, in offices or cafes, some stationary computers may remain idle while mobile devices are busy. As such, it makes sense for these resource constrained mobile devices to make use of the resources available in their vicinity and leverage their capability to run heavy-duty applications.

Mobile devices can provide services involving complex business processes by partitioning the tasks and delegating the heavy-duty tasks to remote servers. An ideal partitioning solution should consider memory, CPU and bandwidth simultaneously. Therefore, to successfully host complex services on mobile devices, the demand for a flexible, light-weight, and scalable execution environment is crucial, which can exploit resources from available networks or the Cloud as necessary.

2.5 Related Work

There have been some research works on facilitating mobile devices in hosting services. However, the existing work in the area is only feasible for simple services that demand very less resources. One of the earliest works in this area was proposed by IBM. They developed a prototype for a shopper-kiosk application running on a PDA where a shopper can use his/her wallet services to pay bills [McFaddin et al., 2003]. But this effort was specific to a particular scenario using Bluetooth and therefore not suitable for generalized application scenarios.

[Srirama et al., 2006a] have investigated web service hosting on mobile devices in detail. Later, this work is further extended to provide a secure communication and an access control for mobile web service provisioning [Srirama and Naumenko, 2007; Srirama et al., 2007b]. The authors presented a distributed semantics-based authorization mechanism for accessing mobile web services. [Pham and Gehlen, 2005]

also proposed a light-weight SOAP server architecture for mobile devices and provided an implementation with J2ME platform. The proposed SOAP server is useful for its light-weight architecture in providing access to web services via HTTP protocol.

Riva et al. proposed a mobile service framework, which can reflect dynamic context changes in an ad-hoc network [Riva et al., 2010]. This method monitors the context of a service requester and forwards the requester to a relevant host. However, the proposed framework is designed for an ad-hoc environment and requires the client to monitor the context. Hemmati et al. also proposed a framework, which supports the migration of service codes and execution states [Hemmati et al., 2005]. The proposed framework decides a relevant target host based on their context information, which is collected by a context manager. However, this method does not support the web based mobile services.

Hao et al. proposed a cost model and a web-let infrastructure, which supports the migration and execution of web services to improve the performance of real-time applications [Hao et al., 2006]. Specifically, they use an algorithm for making the decision about a target host. However, their method needs a user's intervention for taking the decision of migration. Pratistha et al. also proposed a cost model, which is used to decide about a target host for service migration [Pratistha et al., 2005a; Pratistha et al., 2005b]. However, this method is not based on any mobile environment and requires selecting a cost model manually for the service provider.

In another work, a light-weight infrastructure referred to as Micro-Services was proposed, which is capable of hosting web services from mobile devices [Pratistha et al., 2003]. It is limited to performing simple and short operations and does not consider the intermittent bandwidth characteristic of the wireless medium. Gu et al. also proposed an adaptive infrastructure for Java application offloading [Gu et al., 2003], which adopted the MINCUT heuristic algorithm [Stoer and Wagner, 1997] to dynamically partition an application. Their effort only considers memory

requirement for partitioning decisions. However, most of the mobile applications are equally sensitive to bandwidth and CPU usage constraints.

Most of the existing efforts only consider how to host simple services on mobile devices. The work presented in this thesis, provides a complete framework for hosting complex and heavy-weight services on mobile devices [Hassan et al., 2010]. The proposed mechanisms and algorithms for partitioning service execution are briefly illustrated and justified with prototype experiments. In particular, the framework takes into account the available memory, CPU, and bandwidth resources of the mobile device to dynamically partition the work load of the hosted services. Another work presented in the thesis, realizes mobile service provisioning in P2P environments [Hassan, 2009]. The proposed approach exploits the JXTA architecture to provide P2P based mobile services, which supports integration with traditional web service platforms.

2.6 Summary

This chapter discussed the background study for the research addressed by the thesis. The chapter first introduced the web services technology along with associated standards and protocols. Later, the developments in mobile technology domain are discussed in terms of device and transmission capabilities for mobile web services. The supported platforms, standardization efforts, and SOAP transmission mechanisms for mobile web services were introduced. The chapter concluded with a discussion on the currently open issues and some related works in realizing mobile web services.

Chapter 3

Mobile Hosted Service Provisioning

While mobile service clients are common these days and many software tools already exist in the market, the research with providing services from mobile devices is still scarce. However, a mobile device in the role of a service provider enables entirely new scenarios and end-user services. This chapter explains the research with mobile hosted service provisioning and tries to provide motivation and state of the art for the upcoming chapters.

3.1 Mobile Devices as Service Providers

In recent time speed of wireless data transmission has increased significantly with the introduction of 3G and beyond mobile technologies like UMTS, GPRS, and EDGE [GSM1 2009]. Also processing power and capabilities of mobile devices have increased drastically, enabling better usage of mobile devices in different application domains. This enables communication via XML-based service interfaces and standardized protocols also on the radio links. Mobile hosted services lead to many opportunities to mobile operators, third-party application developers, and end users. It is easy to imagine that in future mobile based service applications will generate a large percentage of all service requests.

The paradigm shift of mobile devices from the role of service consumer to service provider is a step towards the realization of various computing paradigms such as pervasive computing, ubiquitous computing, ambient computing and context-aware computing. For example, the applications hosted on a mobile device provide information

about the associated user (e.g. location, agenda) as well as the surrounding environment (e.g. signal strength, bandwidth). Mobile devices also support multiple integrated devices (e.g. camera) and auxiliary devices (e.g. GPS receivers, printers). For the hosted services, it provides a gateway to make available its functionality to the outside world (e.g. providing paramedics assistance). These developments have resulted in a service research paradigm which is referred to as *Nomadic Mobile Services* [Halteren and Pawar, 2006]. A Nomadic Mobile Service (NMS) is hosted on a mobile device such as a handheld device, smartphone, or an embedded device capable of connecting to the Internet using a wireless network. The mobile device roams from one network to another which gives nomadic characteristics to the services it hosts.

In general, mobile hosted services have several potential advantages in addition to the current web service solutions. As a service provider, the mobile terminal becomes a multi-user device where the carrier of the device can work in parallel with the service clients. From a business viewpoint, this is profitable to the service owner. Traditionally the service owner subscribes and pays to upload the services to a stationary server and then clients pay to access the information. In the mobile service provisioning scheme, the service owner hosts services locally and the payment responsibility shifts to the actual clients using the services. Another commercial aspect would be the possibility for small mobile operators to set up their own mobile based service business without having stationary structures. Thus moving the service provisioning arrangement from centralized to decentralized architectures. In such architecture, individuals can host services within proximity for business purposes; such as, delivery/pick-up services, temporary retail services, collaborative journalism, emergency services etc.

3.1.1 Feasibility of Implementing Mobile Service Provider

Previously, nomadic mobile service provisioning was attempted in several research works with different proprietary technologies. For instance, Halteren et al. proposes a proxy based middleware based on the Jini surrogate architecture [Halteren and Pawar, 2006]. Such proprietary approaches seriously affected the interoperability of the provided NMS. Later, the interoperability issues and the issues with integration of NMS were overcome in the project *mobile web service provisioning* [Srirama, 2007]. In this project, a web services based *Mobile Host* was developed on a smartphone and its performance was extensively analyzed, proving the feasibility of the concept of hosting services on mobile devices [Srirama, 2004].

Similar to the web services architecture, the basic architecture of the mobile host as a service provider can be implemented on smartphones. Though the service provider is hosted on a smartphone, standard WSDL can be used to describe the services and standard UDDI registry can be used for publishing and un-publishing the services (Figure 3.1) [Srirama et al., 2006a]. The mobile host has been developed as a web service handler built on top of a normal web server. Mobile web service messages can be exchanged using the SOAP over different transportation protocols like HTTP, BEEP, UDP, and WAP. In the implementation of mobile host, the web service requests sent by HTTP tunneling are diverted and handled by the web service handler. The key challenges addressed in the development of mobile host are threefold: to keep the mobile host fully compatible with the web service interfaces such that clients will not notice the difference; to design the mobile host with a very small footprint that is acceptable in the smartphone world; and to limit the performance overhead of the web service functionality such that neither the services themselves, nor the normal functioning of the smartphone for the user is seriously impeded.

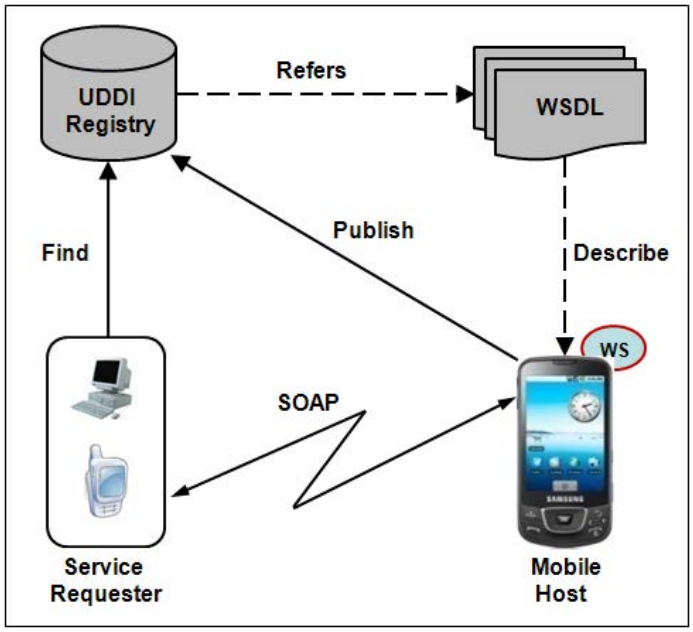

Figure 3.1: Mobile devices as service provider [Srirama et al., 2007a]).

Alternate architectures for mobile service provisioning are also possible with SOAP compliant proxy or gateway in between the mobile host and the service requester. The communication between the client and the proxy would be using SOAP, and the communication between the proxy and the mobile host can be using a protocol efficient for data transport across the mobile networks. Many such proprietary protocols and implementations have evolved like gSOAP, eSOAP etc. But while developing the mobile host, only the basic mobile web services architecture was considered. The main interest was to check the feasibility with performance, of having such a standard mobile host implemented on smartphone.

3.1.2 Sample Services Provided by Mobile Service Provider

This subsection describes some of the basic services successfully provided from the mobile host. The sample services give an idea of the services already possible from smartphones. These services were used in calculating the performance loads of the initial mobile service provider.

Mobile Photo Album Service

Generally, today's smartphones are being equipped with an integrated digital camera. The photographs taken with these smartphones can later be uploaded or transferred to PCs through cables or by using wireless methods like *Infrared* or *Bluetooth*. Using currently available technologies, if a user wants to publish the photographs taken with the mobile device to the public or individuals, he has to upload the images to a web server from where they can be accessed. The user can also send the images through *Multimedia Messaging Service (MMS)* [Novak and Svensson, 2001] or some other means of messaging to the clients.

Therefore, the mobile owner bears the payment for the communication between his smartphone and the web server or the receiver's device. With a mobile service provider implemented and deployed on the smartphone, interested people can access the mobile host using a standard service client, and can browse through the pictures they are interested in. Here the responsibility for the payment shifts to the actual clients who are accessing the pictures provided by the mobile service host. This service is comparable to any other online image album service or blog service, but implemented on the mobile terminal.

Location Information Service

This dedicated service provides the exact location information of the mobile terminal, such as *Global Positioning System (GPS)* data [NAVSTAR, 1995]. The GPS is a satellite-based radio navigation system worldwide developed by the Department of Defense (DoD). The constellation consists of 24 satellites and is fully operational since 1995. GPS provides two levels of service, *Standard Positioning Service* and the *Precise Positioning Service*. The Standard Positioning Service is a positioning and timing service which is available to all GPS users worldwide. The Precise Positioning Service is a highly accurate military

positioning, velocity, and timing service which is available worldwide to users authorized by the U.S. [NAVSTAR, 2007].

The Location Information Service uses a *Socket GPS receiver* for getting the GPS co-ordinates using Standard Positioning Service. The external device connects to the smartphone via Bluetooth. The GPS data can also be collected while taking the pictures and these two details can be mapped together, which opens up scopes for many interesting scenarios (e.g. traveler's diary). The GPS co-ordinates can always be mapped to geospatial maps.

3.1.3 Performance Evaluation of Mobile Service Provider

The evaluation of the mobile service provider was conducted using the mobile photo album service and the location information service described above [Srirama et al., 2006b]. The test setup comprised a mobile service provider deployed on a SonyEricsson P800 smartphone and a standalone Apache Axis [Apache Software Foundation, 2007d] web service client. The smartphone had an internal memory of 12 MB and a 128 MB memory stick duo card. The ARM9 processor of the device clocked at 156MHz. The Axis client invoked different services (in this context, it is assumed that the client knows the location (Uniform Resource Identifier (URI)) of the service and the service description deployed on the mobile device. The performance of the mobile service provider was observed, by taking timestamps and memory foot prints while the service provider was processing the web service request.

Performance Model of the Mobile Service Provider

Figure 3.2 shows different operations performed and time components that constitute one complete service invocation cycle, along the time axes. The client initiates the call for the web service and then the mobile host processes the request, populates the response, and sends response back to the client. The total time taken for this service invocation (T_{mwsp})

constitutes, the time taken by client for constructing valid SOAP message (T_{cc}), the time taken to transmit the SOAP request to mobile host (T_{reqt}), the time taken for de-serializing the XML based SOAP message to SOAP Envelope object (T_{sd}), the time taken by the mobile host to execute the respective business logic and to populate the response ($T_{process}$), the time taken for serializing the SOAP Envelope object back to XML data streams (T_{ss}), the time taken to transmit the SOAP response back to the client (T_{rest}), and lastly the time taken by the client to process the response (T_{cp}).

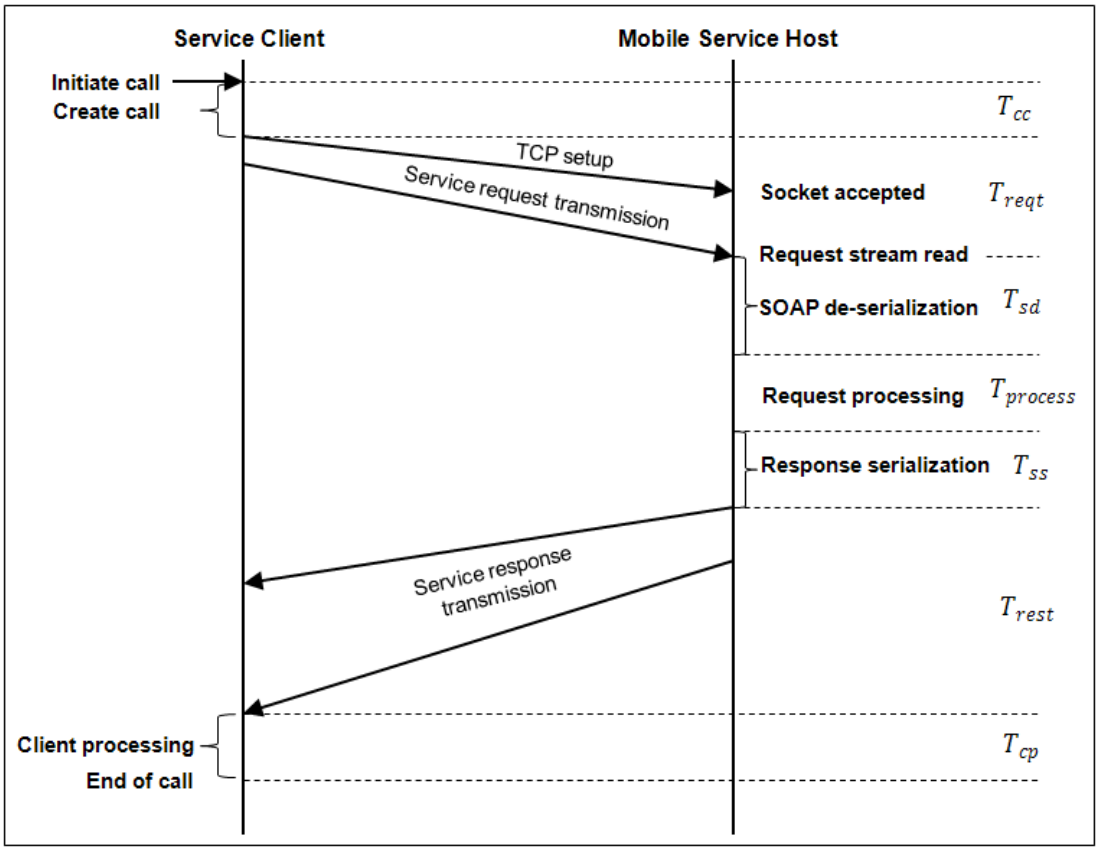

Figure 3.2: Mobile service invocation [Srirama et al., 2006b].

The invocation process is shown in Figure 3.2 and the total time taken for the service invocation is given in the following equation.

$$T_{mwsp} = T_{cc} + T_{reqt} + T_{sd} + T_{process} + T_{ss} + T_{rest} + T_{cp} \quad (3.1)$$

The request and response messages are transferred to the mobile host in the form of TCP packets. Therefore, some delay could be caused by packet loss, TCP acknowledgements, TCP congestion control etc. The delay is shown in the figure as the slanting lines for request and response transmissions. T_{tcp} represents this delay caused by the transmission protocol.

$$T_{tcp} = \delta_{reqt} + \delta_{rest} \tag{3.2}$$

where, δ_{reqt} and δ_{rest} are the respective propagation delays caused while transmitting the SOAP request and response messages. In the performance model the transmission times (T_{reqt}, T_{rest}) also include these TCP delays and an estimation of these delays is not specifically observed. Besides, T_{cc} and T_{cp} are almost negligible, as the client in this analysis is a PC. Hence the total transmission delay (T_{trans}) can be obtained by subtracting the total server processing time from the T_{mwsp}, as shown in the following equation.

$$T_{trans} \approx T_{mwsp} - T_{sd} - T_{process} - T_{ss} \tag{3.3}$$

Performance Analysis of the Mobile Host

For the analysis of the *mobile photo album service*, 8 different images were selected with memory sizes ranging from 5KB to 100KB. The service client tried to browse through these pictures. The *location information service* used an external GPS device for providing the GPS data to the client and had a response size of approximately 2KB. The services, mobile photo album service and location information service were observed to be quite appropriate for the performance evaluation. The response of the mobile picture service is comparatively large and this provided a large scope for observing the effects of different parameters like the

transmission delays, the encoding performed on response messages, the actual service delay etc. on the performance of the mobile host. The location information service returns just a small string (approximately 2KB) containing the GPS data as the response, which provided the scope for observing the behavior of the mobile host under concurrent requests from multiple clients. This enables observing the robustness of the mobile host.

In the analysis, first the mobile picture service was used for observing the SOAP processing delay of the server. The results showed a significant difference (approximately 20%) between the time taken for web service access and the normal HTTP access (Figure 3.3). The SOAP overhead and the Base64 encoding performed on the images before serialization of the response have caused the size of the response to increase by more than 50%. The actual SOAP overhead caused to the size of the response is observed to be 578 bytes. The increase in the size has increased the transmission delay and thus increasing the delay in response.

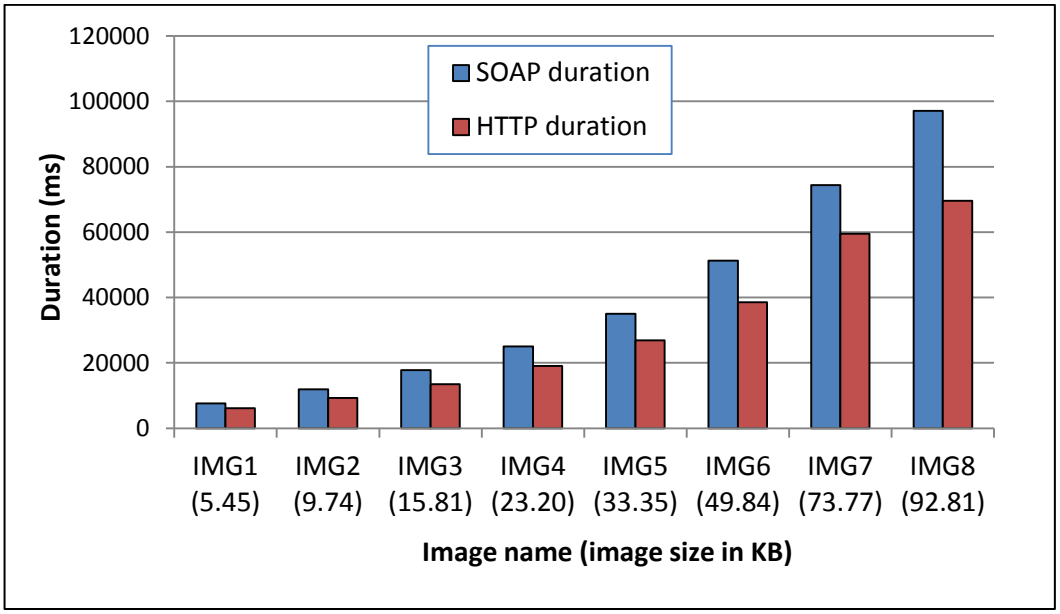

Figure 3.3: Difference between round-trip durations for SOAP and HTTP requests [Srirama et al., 2006b].

In order to identify the actual times taken for different activities on the mobile host like T_{sd}, T_{ss}, T_{trans} etc., the location information service was requested by the client and the time stamps were taken while the mobile host was processing the request. These time stamps were later processed to get the operational time delays. Figure 3.4 shows the time delays of different activities for the location information service.

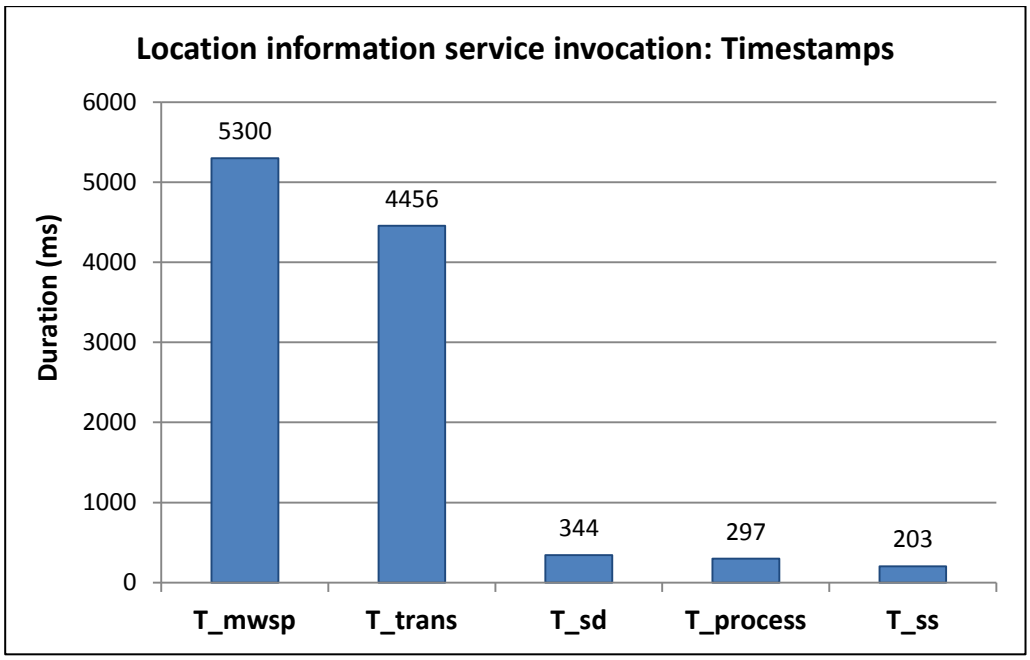

Figure 3.4: Timestamps for GPS data provisioning service [Srirama et al., 2006b].

By observing the results and the performance of the mobile host, it can be concluded that service delivery as well as service administration can be performed with reasonable ergonomic quality by normal mobile device users. As the most important result, it turns out that the total service processing time on the mobile host is only a small fraction of the total request-response invocation cycle time (<10%). Rest of the time is taken for transmission delay in the GPRS network. Therefore, increase in the transmission rates can increase the processing capability of the mobile host. This makes the performance of the mobile service provider directly proportional to the achievable higher data transmission rates.

The evolving mobile communication technologies in 3G, 3.5G, and 4G will help the mobile service provider to perform successfully in commercial environments [Srirama et al., 2006b].

In terms of performance of the mobile host, the key question was whether a reasonable number of clients could be supported without preventing the mobile user from using the smartphone in the normal fashion (e.g. use other services, or perform telephony operations). This study can define the limit for the number of concurrent participants in the collaborative application environments. Concurrent requests were generated by simulating multiple clients for the services deployed on the mobile host. The results of this scalability analysis are very encouraging and the mobile host was successful in handling up to 10 concurrent accesses for reasonable service like location information service with a response size of approximately 2KB. But the same analysis conducted for the mobile photo album service could process only 3 concurrent requests, where the response size is approximately 50KB. The main reason for not being able to process more service clients was due to the transmission delay. It was also observed that the number of concurrent access affects the ability of the mobile host to access its internal and external resources.

The study of the memory footprints revealed that memory usage was not a major problem with the mobile service provider, as most of the time the amount of free memory was at least 20% of the total memory allocated for the JVM (Java Virtual Machine). Approximately 200 data traces were observed as the experiments were repeated several times in order to have statistically valid results [Srirama, 2004].

3.1.4 Applications of Mobile Service Provider

Mobile service provider opens up a new set of applications and it can be useful in many domains like mobile community support, collaborative learning, social systems, emergency services etc. Primarily, the smartphone can act as a multi-user device without additional manual

effort on the mobile carrier's side. Thus, the mobile service provider is of significant use in any scenario that requires monitoring and tracking of the mobile carrier's activities. For example, the mobile device can be used to get the location details of an individual, which can be used in scenarios like emergency services, guided tourism etc. In a distress call, the mobile terminal could provide a geographical description of its location (as pictures) along with location details.

Another interesting scenario is with *Remote Patient Tele-Monitoring*, where the mobile device gathers patient's data collected from medical sensors attached to the patient's body and delivers this data in a real-time fashion to the healthcare professionals. Attaching medical sensors and other equipment to mobile devices has become feasible with advancements in technologies like Bluetooth.

An interesting application scenario involves the smooth co-ordination between journalists and their respective organizations [Srirama et al., 2006a]. Journalists can be at different locations across the globe, covering different events like sports, conferences etc. An editor can always keep track of the location of journalists and the content they have gathered. Standard client applications can be developed for the editor, which synchronize the information stored by editor and data delivered by the mobile service providers. Traditionally, the journalists upload their contents to a server held by the editor. But in the new scenario, parallel access to the mobile service provider by both the journalist and the editor is possible. Even the other journalists in the team can access each other's information, thus better synchronizing their activities (e.g. in covering some major distributed event).

The scope of the mobile service provider in many m-learning (mobile learning) application scenarios can be envisioned, like podcasting, mobile blogging, mobile learning, media sharing service, expertise finder service etc. As the service provider, the mobile terminal can provide access to information like pictures, audios, videos, tags, documents, location details, and other learning services [Chatti et al., 2006]. In

mobile learning, learners can share audio or video lecture recordings or go for the field study and take the pictures of the location. Peers can then browse through the pictures taken, add tags, and give their suggestions or comments. In an expertise finder, learners can look for reliable sources of resources, persons who share the same interests, and experts with the required know-how that can help achieving better results. In the m-learning aspect these experts can share the information among the other users. Examples of such use cases could be exchanging mathematical formulas and the experts validating or even correcting those [Belov et al., 2005].

Services hosted on a mobile device can also find their application in supply chain management systems. A person running a small business and using a laptop or a handheld device in the field can be a part of a supply chain system used by an enterprise. The services offered by him in the field can be available through the hosted services on his mobile device.

From commercial viewpoint, with the mobile service providers there can be a reversal of payment structures in the cellular world. Traditionally the service owners have to pay to upload their service and data to a stationary server. Then the clients have to pay again to access the information. In the mobile services scheme, the responsibility for payment can be shifted only to the actual clients who access the information or services provided by the service owners from their mobile devices. Thus mobile service provisioning renders the possibility for small mobile operators to set up their own mobile based service businesses without resorting to stationary server structures.

3.2 Related Mobile Server Approaches

Web services are not the only means of providing services from devices like smartphones and PDAs. The provisioning can also be based on any distributed communication technology like Java Remote Method

Invocation (RMI) or Jini, if the device supports the respective platform. Halteren et al. have addressed nomadic mobile service provisioning based on Jini technology. The Jini system architecture consists of three categories: *programming model*, *infrastructure*, and *services* [Sun Microsystems, 2001]. The infrastructure is the set of components that enables building a federated Jini system, while the services are the entities within the federation. The basic infrastructure consists of the discovery/join protocol and the lookup service. Discovery is the process by which a Jini-enabled device locates lookup services on the network and obtains references to them. Join is the process by which a device registers the services it offers with a lookup service. The programming model includes models for leasing, event notification, and transactions. The Jini infrastructure is built on top of the Java application environment. Figure 3.5 shows the architecture of nomadic mobile service provisioning.

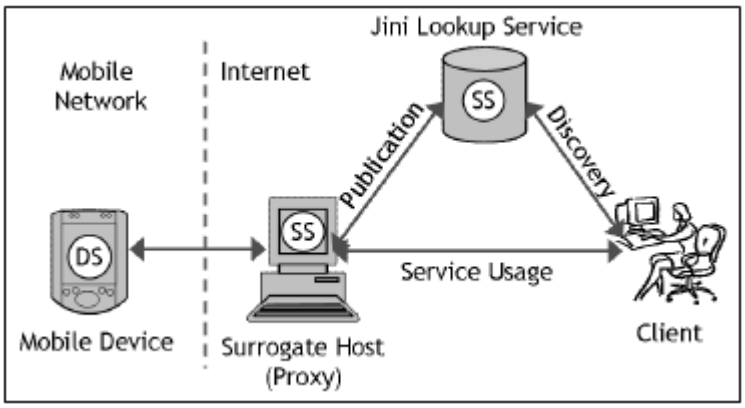

Figure 3.5: Proxy based NMS provisioning [Pawar et al., 2007].

The Jini based nomadic mobile service provisioning proposes the Mobile Service Platform (MSP) as a supporting infrastructure, which extends the SOA paradigm to the mobile device. MSP is a middleware that facilitates the development and deployment of innovative services on the mobile device for clients located anywhere on the Internet. The MSP design is based on the *Jini Surrogate Architecture Specification* which enables a device to join a Jini network (device that cannot directly participate in a Jini Network) with the aid of a third party [Sun

Microsystems, 2007g]. MSP consists of an `HTTPInterconnect` protocol to meet the specifications of the Jini Surrogate Architecture and provides a custom set of APIs for building and running services on a mobile device.

Using this architecture a service provided from a device is composed of two components: (1) a *device service*, which is a service running on the mobile device; and (2) a *surrogate service*, which is the representation of the device service in the fixed network. The surrogate service (SS) functions as a proxy for the device service (DS) and is responsible for providing the service to the clients. The MSP supports the communication between the device service and the surrogate service. Thus using mobile service platform a service hosted on a mobile device can participate as a Jini service in the Jini network [Halteren and Pawar, 2006]. Figure 3.6 shows the elements of MSP and the stages of NMS lifecycle are numbered from 1 to 6.

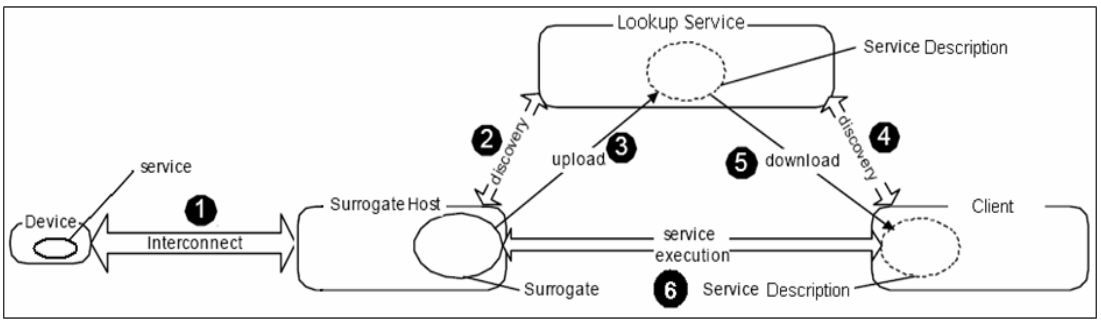

Figure 3.6: MSP and NMS lifecycle [Halteren and Pawar, 2006].

For publishing a service in the network, the surrogate contacts the Jini lookup service for the service registration. After the lookup service is discovered either through unicast or multicast discovery [Sun Microsystems, 2001], the NMS description is registered with the lookup service. The surrogate needs to periodically renew the NMS registration. In case the registration is not renewed for a certain time, the lookup service will discard it. For accessing the published service, a client downloads a service proxy which communicates with the actual service using any remote invocation protocol (e.g. RMI). To ensure that the client

receives latest data from the device service, some mechanism is necessary for the communication between the device service and surrogate, and regular synchronization of their respective data.

For this purpose the MSP uses the `HTTPInterconnect` protocol, which defines the following three types of interactions between the device service and surrogate: (1) *One-Way* messaging allows for unacknowledged message delivery; (2) *Request-Response* messaging supports reliable message delivery; (3) *Streaming* interaction supports exchange of continuous data (streams). Each message has an `operationID` and `sequenceID`. Each operation offered by the service to its surrogate has a unique `operationID`, so each message can trigger a certain operation. The body of a message contains data specific to the operation to be performed by the message.

The approach has some advantages over mobile hosted service provisioning. Since the surrogate is located in the fixed network, it serves a potentially unlimited number of clients and thus minimizes the bandwidth usage in the mobile network. However, the surrogate must be aware of the change in the state of a device service. Most serious limitation of this approach is that, it is based on a proprietary protocol. The technology (Jini) is not interoperable. So the client should be aware of Jini technology to be able to access the services.

Moreover, the services are to be developed both for the surrogate and the mobile device and changes are not propagated. Therefore, the approach tightly fixes the service provided by the mobile device to the protocol (`HTTPInterconnect`), technology (Jini), and surrogate host; thus seriously affecting the interoperability of the provided services. The only advantage with this approach is the support for unlimited number of clients, which can also be achieved with proper QoS support for the mobile hosted service provisioning using a partitioned mobile services framework [Hassan et al., 2010]. The partitioned mobile services framework is discussed in Chapter 5.

3.3 Scalability Aspects of Mobile Service Provider

Providing proper scalability is crucial in achieving suitable QoS from the mobile service provider. The pattern of web service communication introduces a lot of message overhead to the XML based SOAP messages exchanged. This consumes a lot of resources, since this additional information is to be exchanged over the radio link. Moreover, XML-based messages are larger and require more processing than other protocols such as RMI or CORBA [OMG, 2004] and the binding requires more computation [Davis and Parashar, 2002]. Hence only few concurrent requests can be handled by the mobile service provider. Thus for improving scalability, the messages are to be compressed without seriously affecting the interoperability of the mobile hosted services.

The messages can be compressed with standard compression techniques like Gzip or XML-specific compression techniques like XMill to obtain smaller message sizes. Canonical XML [Boyer, 2001] targets the logical equivalence of these compressed XML messages. There is also an effort with the Fast Web Services [Sandoz et al., 2003], Fast Infoset standard draft [Sandoz et al., 2004], Efficient XML [AgileDelta, 2007], BinXML [Ericsson, 2003b] etc. to specify a binary format for XML data, that is an efficient alternative to XML in resource constrained environments. Similarly, there is some effort with BiM (Binary Format for Metadata) standard [Heuer et al., 2002] for the binary encoding of MPEG-7 Metadata. This section introduces some of these XML data compression technologies.

3.3.1 XML Compression

XML has become a de facto standard for information transfer on the machine level and the application independent information storage. This is the reason why XML is used in service communication to send data without having to worry about the incompatibilities between data representations in computer architectures, operating systems, and

services. But XML is quite verbose and is expensive in terms of storage space, processing time and resources required for parsing. Simple text compressors such as GZip could reduce the size, but they do not adapt to the specific demands of XML language, such as duplication of names and tags that are spread throughout the document. GZip compression [Loup Gailly, 2007] is based on numerous improvements to the LZ77 compression algorithm [Ziv and Lempel, 1977]. The basic idea behind LZ77 is to try to match current data in a dictionary of previously observed data. The dictionary and the current data are implemented using a fixed size window that slides over the data. When previously observed data are encountered, it is replaced by a reference to the dictionary; thus reducing the size of the text.

There are mainly two drawbacks with the simple text compressors. Firstly, the redundant information in XML is spread over the whole document, but most text compressors work locally. For example, LZWM flushes its dictionary-based statistical table several times when working through large files. This approach is useful in text documents with few repetitions of words at the beginning and end of the files, but with XML this approach would be futile. Secondly, the compressors lack the semantic understanding of different types of XML elements.

There are also some XML aware text compressors such as XMill [Liefke and Suciu, 1999], XMLPPM etc. XMill is a user-configurable XML compressor that groups text items with similar syntax and meaning, and compresses them together. It separates structure, layout and data, and distributes data elements into separate data streams (int, char, string, base64, etc.). This distribution is user-definable. Simple text compressions like GZip, BZip2 [Seward, 2005] compress these data streams. The grouping of data increases the redundancy of the document, and therefore in many cases, XMill can achieve a higher degree of compression than simple GZip compression. XMLPPM is a compressor for XML, based on the Prediction by Partial Match (PPM) [Cleary and Teahan, 1997]. PPM predicts the upcoming character after certain

strings, builds up a prediction table, and compresses the document according to the probability of the contained strings. XMLPPM [Cheney, 2001] uses Multiplexed Hierarchical Modelling (MHM), i.e., several compressors (multiplexed) based on the XML structure and a hierarchical model of the document as context, rather than a sequence of characters. MHM is based on Encoded SAX (Simple API for XML) algorithm (ESAX). ESAX encodes the incoming SAX events into single bytes and stores the original names in a symbol table for later decoding. While these kinds of XML compressors achieve good results, but they are generally quite slow.

Apart from these compression strategies there are also some attempts to optimize the XML, thus reducing the size of the message [Rodriguez, 2002]. Some of these optimization principles include flattening the structure pattern, removing the additional comments, structuring node names and namespaces etc. These optimization mechanisms still preserve structural equivalence of XML documents. Any two XML documents within an application context are logically equivalent, if they only vary in physical representations based on syntactic changes permitted by XML and namespaces in XML. Two structurally equivalent XML documents have a byte-for-byte identical canonical XML document. Canonicalization of an XML document involves only the information that an XML processor requires to execute the application.

3.3.2 Binary XML

Apart from XML compression, there exits several attempts in creating a standardized binary representation of XML data with equivalent capabilities of expression, but more efficient in terms of storage/transfer and parsing. Fast Web Services is an initiative at Sun Microsystems aimed at the identification of performance problems in existing implementations of web services standards. It attempted at defining binary-based messages that consume less bandwidth and are faster and

require less memory to be processed. Fast Web Services makes use of Fast Infoset documents for carrying the content of a SOAP message. The Fast Infoset standard draft specifies a binary format for XML infosets that is an efficient alternative to XML.

Message Transmission Optimization Mechanism (MTOM) specification from W3C describes an abstract feature for optimizing the transmission of a SOAP message by selectively binary encoding portions of the message, while still presenting an XML Infoset to the SOAP application [Gudgin et al., 2005]. Similar to the Fast Web Services, WAP Binary XML (WBXML) [Martin and Jano, 1999] is defined by the WAP Forum and is a part of the WAP 2.0 specification. The main purpose of WBXML is to create a compact representation of XML to reduce the transmission size with no loss of structure or semantics. WBXML does not include any support for compression of data. Millau [Girardot and Sundaresan, 2000] is a version of WBXML with support for compression.

The AgileDelta's Efficient XML provides a codec that encodes and decodes XML in a binary format that dramatically reduces the size. Similarly, BiM (Binary Format for Metadata) standard is defined for the binary encoding of MPEG-7 Metadata [Heuer et al., 2002]. MPEG-7 focuses on an XML based metadata system for describing the content of multimedia. There are two ways to transmit MPEG-7 information, either in *Textual Format (TeM)* or in a *Binary Format (BiM)*. Both provide similar functionality since both are able to transmit the description trees dynamically and incrementally. The benefit with BiM is that it compresses the XML code. BiM is designed in a way that it allows fast parsing and filtering of the XML data at the binary level, without having to decompress again. But to make it work, both decoder and encoder of BiM should be aware of XML schema in advance.

[Ericsson and Levenshteyn, 2003] gives a comparison of different compression technologies for XML data (BiM is not considered) and suggests BinXML as a good option to compress web service messages, considering compression ratio, processing time and resource usage. Based

on this analysis, the study has adapted BinXML in the scalability analysis of the mobile host for compressing the mobile web service messages. BinXML is explained in the next subsection. Even though there are many benefits in using binary XML, but there is one serious limitation. With the binary encoding, the messages lose their human readable self-descriptive nature of XML, and therefore, any existing standards and tools that rely on this mechanism will be seriously impacted.

3.3.3 BinXML

BinXML is a very simple binary representation of XML data [Ericsson, 2003b]. The encoding replaces each tag and attribute with a unique byte value and replaces each end tag with 0xFF. By using a state machine and 6 special byte values including 0xFF, any XML data with circa 245 tags can be represented in this format. The reserved codes are 0x00-0x03, 0xFE, and 0xFF. The approach is specifically designed to target SOAP messages across radio links.

BinXML uses a simple SAX parser and supports a variety of compression algorithms like LZO and GZip. BinXML has no support for schemas or DTDs, and encoding/decoding is done based on the message itself. Each message contains enough information so that the receiving end can decompress it without prior knowledge of the message schema. The main version of BinXML is implemented in C. There is also an available Java port which was adapted for the J2ME version. The J2ME version uses the SAX parser provided by the Java ME Web services or JSR-172. Listing 3.1 shows a SOAP response message and its equivalent BinXML encoding is provided in Table 3.1.

Listing 3.1: An example SOAP message.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
           "http://www.w3.org/2003/05/soap-envelope/">
    <SOAP-ENV:Body>
         <searchArticleResponse>
              <status xsi:type="xsd:string">
                   The article was successfully received!
              </status>
         </searchArticleResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Table 3.1: BinXML encoding of the SOAP message in Listing 3.1.

<0x01>	New tag
<0x10>	The new tag is assigned code 0x10
SOAP-ENV:Envelope	Name of the new tag
<0x00>	End of string
<0x03>	New attribute
<0x10>	Code of the new attribute (attribute code space), 0x10 is free
xmlns:SOAP-ENV<0x00>	Name of attribute + End of string
http://www.w3.org/2003/05/soap-envelope/<0x00>	Value of the attribute + End of string
<0xFE>	Soft end of tag. Represents the > in XML
<0x01>	New tag
<0x11>	Code 0x11
SOAP-ENV:Body<0x00>	Name of the new tag + End of string
<0xFE>	Soft end of tag
<0x01>	New tag
<0x12>	Code 0x12
searchArticleResponse <0x00>	Name of the new tag + End of string

<0xFE>	Soft end of tag
<0x01>	New tag
<0x13>	Code 0x13
status<0x00>	Name of the new tag + End of string
<0x03>	New attribute
<0x11>	Code of the new attribute 0x11
xsi:type<0x00>	Name of attribute + End of string
xsd:string<0x00>	Value of the attribute + End of string
<0xFE>	Soft end of tag
<0x02>	String
The article was successfully received!<0x00>	Value of string + end of string
<0xFF>	Hard end of tag (/> in XML)
<0xFF>	Hard end of tag
<0xFF>	Hard end of tag
<0xFF>	Hard end of tag

The values, in the format <0xYZ>, shown in Table 3.1 indicate the byte value of YZ in hexadecimal form. The size of the original SOAP message is 249 bytes excluding the white spaces (271 bytes with white spaces). The size of the encoded byte stream of the message is 195 bytes. This is a significant reduction (approximately 30%) to the size of the XML message. Using BinXML the compression ratio can be very significant where the SOAP message has repeated tags and very deep structure.

3.4 Integration Aspects of Mobile Hosted Service Provisioning

For the integration of mobile hosted services to the service networks, intermediary nodes can be utilized in deployment scenarios. *Enterprise Service Bus (ESB)* technology can be the ideal platform for realizing the

intermediary nodes in the integration. This section explains the ESB technology, the *Java Business Integration (JBI)* specification, and the open source *ServiceMix* tool in realizing the integration framework.

3.4.1 Enterprise Service Bus (ESB)

Enterprise networks commonly deploy disparate applications, platforms, and business processes that need to communicate or exchange data with each other. The applications, platforms and processes generally have non-compatible data formats and non-compatible communications protocols. This leads to serious integration problems within the networks. The integration problem extends further if two or more of such enterprise networks have to communicate among themselves.

Enterprise Application Integration (EAI) was the first technology addressing these integration issues. EAI software follows the *hub and spoke model* and the software acts as a hub that translates data and messages between different applications. It uses adaptors that reformat incoming data to a canonical format that is understood by both EAI and outgoing adaptors. To implement the connection among the EAI components and to achieve internal integration, the technology uses asynchronous message broker like *Java Message Service (JMS)*. EAI products were large, inflexible, hard to manage, and were generally expensive vendor solutions [IONA Technologies, 2005].

ESBs are the next developments in the enterprise integration and a standards-based ESB solves the integration problem without the drawbacks of the EAI solutions. ESB provides a set of infrastructure capabilities, implemented by the middleware technology that enables the integration of services in an SOA [Keen et al., 2004], [Chappell, 2004]. The enterprise service bus is not a product, but an architectural best practice concept for implementing a SOA. Enterprise service bus is an architecture that exploits service applications, messaging middleware, intelligent routing, and transformation. ESBs act as a lightweight,

ubiquitous integration backbone through which software services and application components flow [Schulte, 2007].

ESB basically consists of a set of service containers that are interconnected with a reliable messaging bus. Generally in point-to-point integration solutions, each of the n components requires $n-1$ interfaces for full communication among them. But with the bus solution each component requires only a single interface to the bus for global communication. The ESB offers dedicated infrastructure providing the capability to route and transport service requests to the correct service providers.

The ESB supports multiple integration paradigms in order to fully support the variety of interaction patterns that are required in a comprehensive SOA between the service containers. Therefore, it has support for *service-oriented architectures* in which applications communicate through reusable services with well-defined interfaces, support for *message-driven architectures* in which applications send messages through the ESB to receiving applications, and support for *event-driven architectures* in which applications generate and consume messages independently of one another. Figure 3.7 shows a high-level view of the ESB.

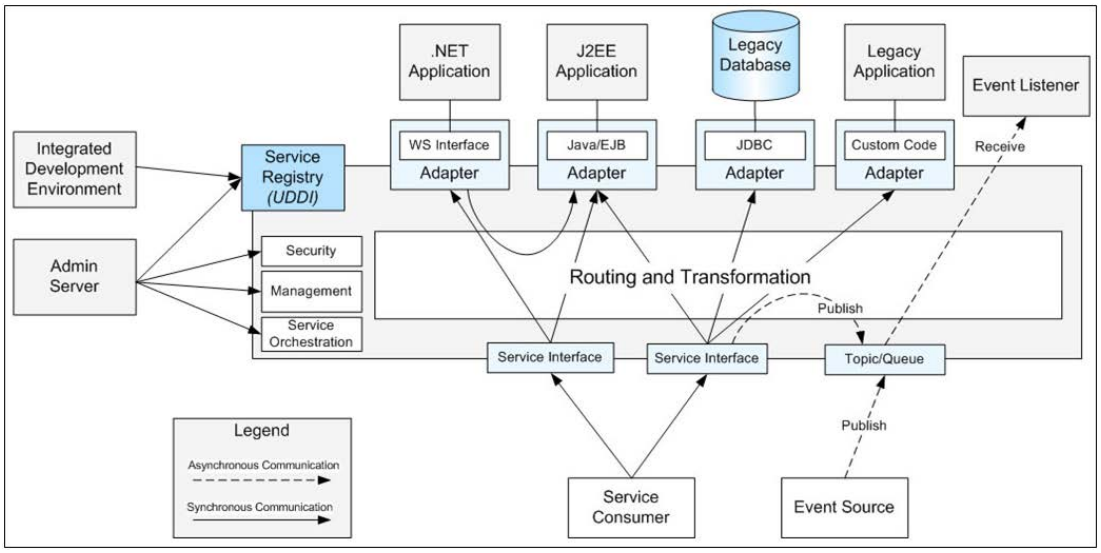

Figure 3.7: Enterprise service bus (based on [Schulte, 2007]).

ESB Features

The ESB provides a number of functionalities and features to achieve mobile hosted service integration [Thomas and Buckley, 2003], [PolarLake, 2005]. These include:

- *Transformation:* The functionality provides the ability to map one data format onto another in order to ensure inter-operability between the various systems plugged into the ESB. The data formats are generally based on XML. The transformation can be performed by the ESB itself or delegated to external components.

- *Routing:* A key prerequisite of any integration platform is to identify which data to process and where the data need to be sent. ESB supports content based routing and filtering, typically using XPATH (XML Path Language).

- *Communication:* The functionality supports the delivery of messages throughout the organization using different communication mechanisms like *synchronous/asynchronous*, *request/response*, *one-way*, *call-back* etc.

- *Support for highly distributed environments*, through which the ESB does not have to route all the messages through a central hub in order to perform routing and transformation.

- *Security:* The functionality supports secure transfer of messages among the participating service engines using mechanisms like user authentication, access controls etc.

- *Orchestration:* Orchestration is the process of automated coordination of composite application components that participate in a business process. Generally ESBs achieve service composition, choreography, and orchestration using BPEL (Business Process Execution Language) engines [Plebani et al., 2011].

- *Fault avoidance and fault tolerance* using intelligent routing and exception handling. In ESBs a fault often does not roll back all the work that has occurred at the time of a business exception. Rather it will follow some rules and execute compensating transactions and/or accept the state of the process as satisfactory at the time of the exception.

- *Transaction:* The functionality supports the reliable end-to-end delivery of messages among the ESB components.

- *Support for pluggable services:* Pluggable services can be provided by third parties and can still interoperate reliably with the bus and the remaining components.

- *Breadth of connectivity:* The functionality supports the ability to connect to different types of systems like databases, enterprise applications (e.g. SAP), and legacy systems using well established standard mechanisms and tools.

ESB Products

Many ESB products exist like the Sonic Software, Artix, Cape Clear etc., which can be adapted for realizing the middleware framework. IONA Technologies extends its legacy EAI architecture to achieve *IONA Artix* ESB. *PolarLake* and *FusionWare* take a server-centric, connector-based approach in their ESB product design. Only *Cape Clear* and *Cordys* ESB systems use a truly open and distributed SOA.

[Borck, 2005] provides a detailed survey of the ESB products and their pros and cons focusing at the achieved interoperability, management, scalability, security, supported features, and their value for money. The survey recommends the *Sonic SOA Suite* among the contemporary ESB products. ESB products like *BEA AquaLogic Service Bus* [BEA AquaLogic, 2007] and *IBM WebSphere Software* [IBM

Corporation, 2007b] were not considered in this survey. An analysis provided by [MacVittie, 2006], recommends *BEA AquaLogic Service Bus* considering the features provided by the product, without considering any performance measurements. Sun Microsystems has defined JSR 208, *Java Business Integration (JBI)* specification for enterprise application integration. ESB products like *ServiceMix* and *OpenESB* [java.net, 2008] are based on this specification.

Observing these different ESB tools, standards, and surveys, the thesis considered JBI and ServiceMix for the realization of the ESB middleware framework. The following subsections briefly introduce the JBI specification and the open source ServiceMix tool.

3.4.2 Java Business Integration (JBI)

The Java Business Integration (JBI) specification (JSR 208) [Ten-Hove and Walker, 2005] defines a platform for building enterprise-class ESBs using a pluggable and service based design. The Java based standard allows building enterprise integration systems by using plug-in components which interoperate through mediated normalized message exchanges.

Message normalization is the process of mapping context-specific data to a context-neutral abstraction in a standard format. The *normalized message* consists of the message content (also called payload), message properties or metadata, and optional message attachments referenced by the payload. JBI uses the normalized messages for interaction between consumers and providers. The basic architecture of JBI is shown in Figure 3.8. The *Normalized Message Router (NMR)* receives messages from JBI components and routes them to the appropriate components for processing. This decouples the service providers from consumers. The NMR delivers messages with varying QoS, depending on application needs and the nature of the messages being delivered.

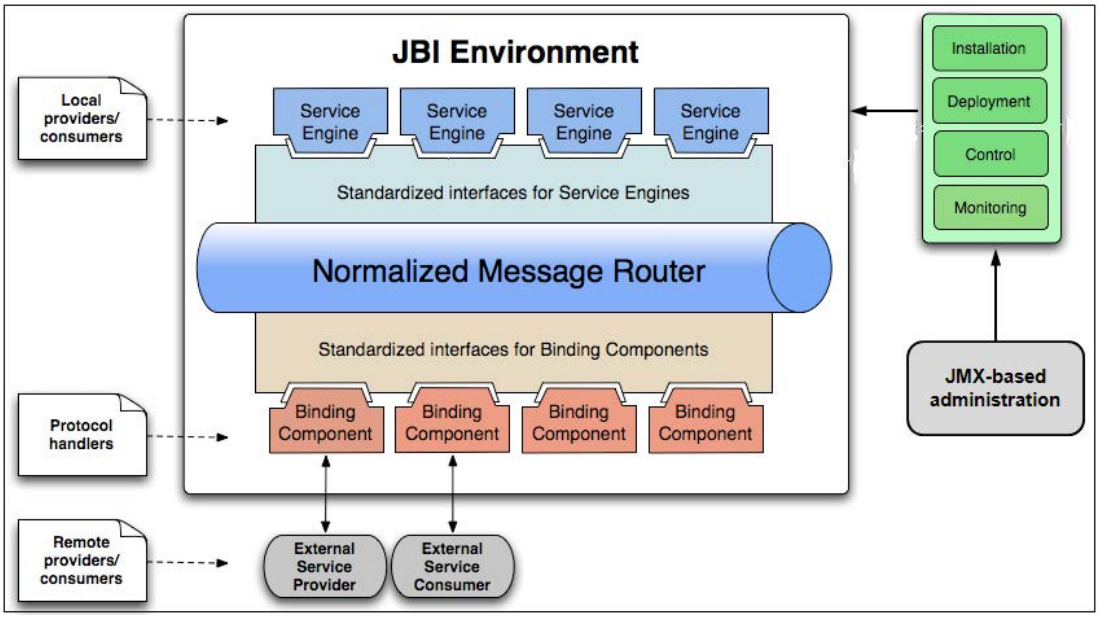

Figure 3.8: Java business integration architecture (based on [Ten-Hove and Walker, 2005]).

JBI Components

The JBI component framework provides a pluggable interface that allows components to interact with the JBI environment. The framework supports two types of components: *service engines* and *binding components*. Service engines are components responsible for implementing the business logic and they can be service providers/consumers. Service engine components generally provide support for content-based routing, orchestration, rules, data transformations etc. Binding components are used to send and receive messages across specific protocols and transports. The binding components marshall and unmarshall messages to and from protocol-specific data formats to normalized messages. Thus binding components allow the JBI environment to process only normalized messages.

In JBI, component-specific artifacts like a BPEL process deployed to a service engine or binding component are called *service units*. Service units are grouped into an aggregate deployment file called a *service*

assembly. The service assembly includes a deployment descriptor that indicates the component, into which each service unit is to be deployed. Binding components and service engines interact with the NMR via a *delivery channel*. A delivery channel is a bidirectional asynchronous communication pipe between a component and the NMR. Each component is provided with a single delivery channel, so the same channel can be used for both inbound and outbound communications. A service consumer uses its delivery channel to initiate a service invocation, while the provider uses its delivery channel to receive such invocations. A single component can act both as a service consumer and a service provider under different circumstances. The JBI environment provides features for deployment, control, and monitoring the components through JMX (Java Management Extensions) based administration tools.

Endpoint activation is the process by which a service provider informs the NMR that it provides a service. After the endpoint activation, NMR can route service invocations to that service. In brief, an endpoint refers to a specific address, accessible by a particular protocol, used to access a precise service. JBI supports two types of endpoints: *Internal endpoints* and *External endpoints*. Internal endpoints are exposed by service providers within the JBI environment, while external endpoints are endpoints outside the JBI environment that are exposed by binding components. The binding components act as service consumers and expose an internal endpoint for the use of external service consumers. In JBI, endpoints can be referred *implicitly* where the NMR selects the endpoint based on the required service type, or *explicitly* where a consumer component chooses the endpoint based on its own logic and configuration, or *dynamically* where an endpoint reference is used to provide a call-back address that the service provider should use to send further messages [Apache ServiceMix, 2007], [Vinoski, 2005].

JBI Message Exchange Patterns

Service invocation in JBI refers to an instance of an end-to-end interaction between a service consumer and a service provider, involving the swapping of *message exchanges* between the components and the NMR. The message exchange is the container for the normalized message and the state of the service invocation. A consumer component initiates the exchanges by creating a new message exchange instance. Each operation that a service provider makes available has a particular *message exchange pattern (MEP)* associated with it.

JBI supports four types of MEPs described below.

(1) *In-Only MEP*: This is a one-way messaging pattern. In this the consumer component initiates the request by creating a new message exchange instance and sending it to the NMR. The NMR queues the message exchange instance for delivery to the respective provider. The provider component accepts the instance from the NMR, processes the request in the message exchange, and completes the MEP by setting the status of the instance to *done*, and sending the instance to the NMR. The NMR routes the instance back to the consumer component, which accepts the notification.

(2) *Robust In-Only MEP*: This is almost the same as In-Only MEP, with one extension. In this MEP, the provider may respond with a fault if it fails to process the request. Here the *done* status for the message exchange is assigned by the consumer.

(3) *In-Out MEP*: The consumer issues the request to provider, with the expectation of a response. The Provider may respond with a valid response or with a fault if it fails to process the request. In any case the *done* status for the message exchange is assigned by the consumer component.

(4) *In-Optional-Out MEP*: The consumer issues a request to provider, which may result in a response (the provider may set *done*

status without responding to the consumer). This is an extension of In-Out MEP with both consumer and provider having the option of generating a fault during the interaction [Ten-Hove, 2006].

3.4.3 ServiceMix

ServiceMix is an ESB, based on JBI specification and it combines the functionalities of both the SOA and the Event Driven Architecture (EDA) to achieve an agile, enterprise ESB. Released under the Apache license, ServiceMix is an open source ESB and SOA toolkit built on JBI semantics and API [Apache Software Foundation, 2007c]. The open source and open standards-based features of ServiceMix allow for low entry cost, maximum flexibility, reuse, and investment protection. The tool supports any number of third party vendor supplied components and protocol bindings that conform to the JBI open standard specification. These components and bindings not only interoperate among each other via the ServiceMix ESB, but can easily be replaced with alternate components that provide the same or enhanced services, without affecting the final deployment scenarios of the applications.

As mentioned already, ServiceMix allows services to operate in an event driven architecture as well (i.e. the services are decoupled and the providers listen for service requests on the bus). It supports such architecture for events occurring both internal and external to the bus. In other words, JMS binding components can listen for the arrival of messages that are external to the bus, while the other components can listen for the messages on the normalized message bus itself. The bus is also responsible for quality of service (QoS) features such as message persistence, guaranteed delivery, failure handling, and transaction support.

ServiceMix is *lightweight* and easily embeddable in the enterprise network. Lightweight design patterns let the coupling between objects to be loose and integrate services without forcing code into the business logic

[Tate, 2005]. ServiceMix is lightweight with integrated *Spring support* [Johnson, 2005] and it can be run at the edge of the network (inside a client or server), as a standalone ESB provider, or as a service within another ESB.

ServiceMix can also be used in a Java Standard Edition or in a Java Enterprise Edition application server. It is completely integrated with JBoss [JBoss, 2007] and Apache Geronimo [Apache Geronimo, 2007] and lets the applications deploy JBI components and services directly into Geronimo. Some of the service components included with ServiceMix comprise rules-based routing via the Drools rule engine, timer integration via the Quartz library, transformation using Extensible Stylesheet Language Transformations (XSLT) etc. [Apache ServiceMix, 2007], [Hanson, 2005].

3.5 Summary

This chapter introduced the mobile hosted service provisioning concept and explained the details of the mobile service provider with its thorough performance and application analysis. The chapter later discussed the scalability related standards and specifications for mobile hosted services. Then the chapter discussed the integration issues for mobile hosted services and introduced the ESB technology. The JBI specification and open source ServiceMix tool in realizing the integration framework are also explained in detail.