Chapter 4

Mobile Service Provider in P2P Environments

In recent years, Peer to Peer (P2P) technology is being used in vast application domains like entertainment systems, ubiquitous computing, pervasive computing, collaborative systems, etc. P2P architecture enhances the adoption of mobile service provider in commercial environments. Mobile service providers can share resources of individual peers to respond to client requests. P2P offers a large scope for many applications with mobile service provider. Apart from the application scopes, P2P architecture also offers many technical advantages to mobile web service provisioning and enhances the service discovery of huge number of services possible with mobile service providers. Most recently, ad-hoc networks of mobile terminals are also participating in such P2P applications.

4.1 P2P Systems and Mobile Web Services

P2P systems are generally designed to enable loosely coupled distributed systems. The concept of services and similarities of description stack make P2P and web services comparable [Schneider, 2001]. Web services are provided by well-known hosts, and are based on a centralized model primarily focused on standard messaging formats and communication protocols. On the other hand, P2P systems are based on a decentralized model, and are less focused on the semantics of messaging formats and communication protocols. This section introduces the P2P technologies, their convergence with web services, and the projects working on the

convergence of P2P and web services technologies. The section also introduces Lucene, a tool used in advanced matching/filtering of mobile web services.

4.1.1 P2P Technologies

P2P is a set of distributed computing models used to perform a critical function in a decentralized manner. P2P networks are typically used for connecting nodes largely via ad-hoc connections. Peers are autonomous, as they are not wholly controlled by each other or by some authority. Peers depend on each other for getting information, computing resources, and forwarding requests. In P2P communication models each party has the same capabilities and can initiate a communication session. Thus in its pure form, each peer can act as both server and client. P2P takes advantage of resources of individual peers like storage space, processing power, content and achieves scalability, cost sharing, and anonymity. Thus, enables ad-hoc communication and collaboration with many data and computing intensive applications. Peers can collaborate through firewalls, NATs, and proxies to connect to other peers.

P2P technology is gaining popularity as low-cost individual computing technology. Over time, three main categories of applications have emerged in the P2P environment. First category includes the *Content and file management P2P applications*, like Napster [Carlsson and Gustavsson, 2001] and Gnutella [Ripeanu et al., 2002]. Second category includes the *Parallelizable P2P applications* that split large tasks into smaller chunks to execute in parallel over autonomous peers, like SETI@Home [Anderson et al., 2002] and Avaki [Grimshaw et al., 1997]. Third category includes the *Collaborative P2P applications* that allow users to collaborate with each other without the help of central servers to collect and relay information, like Groove [Ozzie and Burton, 2003] and Skype [Skype, 2009]. Ad-hoc networks of mobile terminals are

now also participating in enterprise P2P networks and applications, like Magi Enterprise [Bolcer et al., 2000; Milojicic et al., 2003].

P2P systems have evolved across time and provide a stable platform for many data and computation intensive applications. The applications can be categorized among three generations of P2P systems. The *first generation P2P systems* like Napster used centralized servers for maintaining an index of the connected peers and their resources. The indexes can later be queried by the peers, and the resources are downloaded from the providers using IP networks. But these centralized systems have single points of failure, produce giant communication traffic, and occupy large storage on server; thereby resulting bottlenecks.

The drawbacks of the first generation P2P systems lead to the development of *second generation P2P systems* like Gnutella and Freenet [Clarke et al., 2002], which used a complete decentralized network. Unlike Napster, Gnutella would connect users directly to a group of other users, and so on. The users on the system act as gateways to other users to find the data they need. For connecting to users, Gnutella uses a pre-existing extendable list of possible working peers, whose addresses are embedded inside the application code. But these decentralized networks formed isolated clusters in the P2P network, and their search functions were unreliable and not always capable to query the entire network. Moreover, the second generation P2P systems suffered from major overhead generated by the binding messages and queries propagating around the network, leading to bottlenecks. For example, in Gnutella the amount of broadcast messages increase exponentially with a linear increase in the depth of the search [Kwok et al., 2003].

The *third generation P2P systems* like Skype [Skype, 2009], Kazaa [Kazaa, 2009], Bit Torrent [Pouwelse et al., 2005], etc. are a hybrid of the previous two generation technologies, and have improved ability to deal with large number of peers using concepts like *Super peers* and *Edge peers*. Super peers have higher resource capabilities comparing to edge peers, and act as relays for other edge peers and super peers. Super peers

are organized dynamically and have abilities to traverse NAT and firewall. Only super peers participate in peer and resource discovery, which significantly decreases the stress to the network. Moreover, several optimization methods are used for decreasing the message overheads [Harjula et al., 2004]. Figure 4.1 categorizes some of the well-known P2P systems according to their characteristics and provided application types.

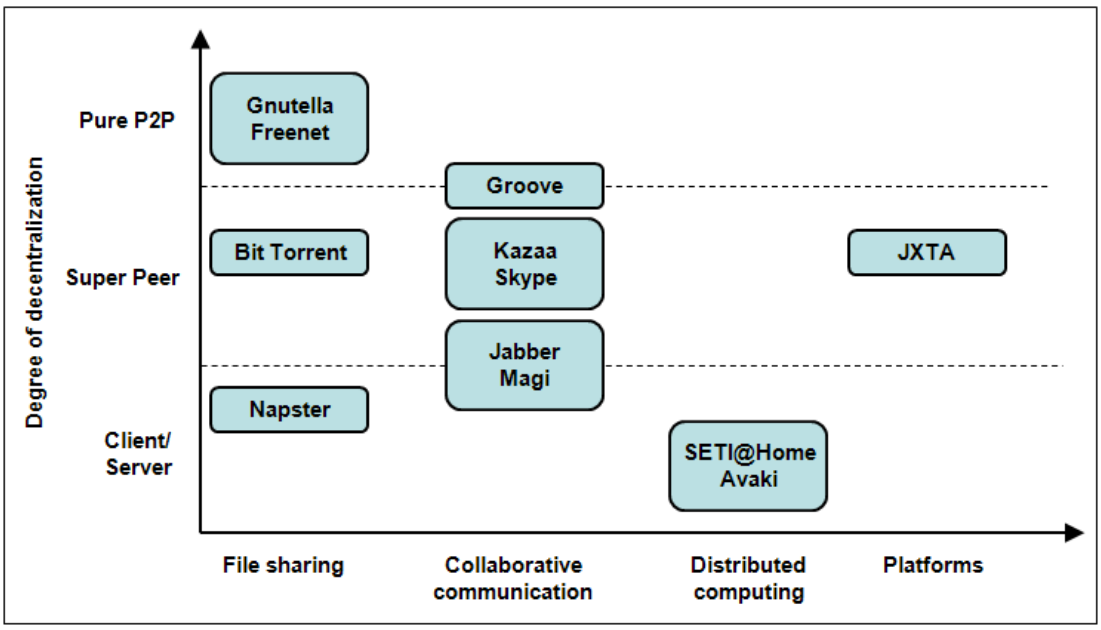

Figure 4.1: Classification of P2P technologies.

4.1.2 Convergence of P2P Systems and Web Services

Analogous to web services, P2P systems can also leverage SOA and are also designed to enable loosely coupled systems. P2P services can also be described using XML schema and WSDL. The concept of services and the similarities of description stack used in both P2P and web services make them comparable [Schneider, 2001]. Both of the technologies have a heavy emphasis on distributed computing and target reliable delivery of services. To achieve this, web services use centralized servers to create highly available systems, while P2P systems use distributed peers running the same service in a redundant manner to respond to the requests. The major difference between the technologies is that, web

services are provided from well-known hosts, based on a centralized model, and primarily focused on standardized messaging formats and communication protocols. On the other hand, P2P systems are based on a decentralized model and primarily focused on contributing processing power, content, or applications to peers in a distributed manner. P2P is less focused on the semantics of messaging formats and communication protocols.

Apart from these differences, the discovery mechanisms of the technologies are also significantly different. Web services discovery is based on centralized UDDI registries. P2P systems do not rely on centralized mechanism and distribute the information using advertisements. Peers perform service discovery in a variety of ways including multicasts, inquiring to other peers, and using hubs (known as *rendezvous*, to act as a meeting place for peers with similar interests). Considering the benefits of decentralized discovery over centralized registries, a P2P based *mobile web service discovery* mechanism is developed using the JXTA technology. Most of the current P2P technologies like Gnutella, Napster, and Magi are proprietary and generally targeted to specific applications. Only project JXTA offers a language agnostic and platform neutral system for P2P computing [JXTA Community, 2007b].

4.1.3 JXTA

JXTA technology (also known as *Juxtapose*), is a set of open protocols that allow any connected device on the network ranging from cellular phones or PDAs to PCs or servers, to communicate and collaborate in a P2P manner [JXTA, 2008]. JXTA enables these devices running on various platforms to share data and functions of their respective peers. JXTA technology is based on proven standards such as, HTTP and TCP/IP protocols. JXTA is independent of programming language, networking platform, or system platform and can work with any combination of these.

JXTA peers use XML as standard messaging format and create a virtual P2P network of devices connected over different networks.

Figure 4.2 shows the basic architecture of the JXTA. The *Core layer* encapsulates minimal and essential primitives that are common to P2P networking. It incorporates building blocks like discovery, transport with firewall handling, creation of peers and peer groups, and associated security primitives. The *Services layer* includes network services that are common in P2P environment but may not be mandatory for the operation of P2P network. For example, searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication services, etc. The *Applications layer* includes implementation of integrated applications, like P2P instant messaging, entertainment content management and delivery, document and resource sharing, distributed auction systems, P2P Email systems etc. [Wilson, 2002].

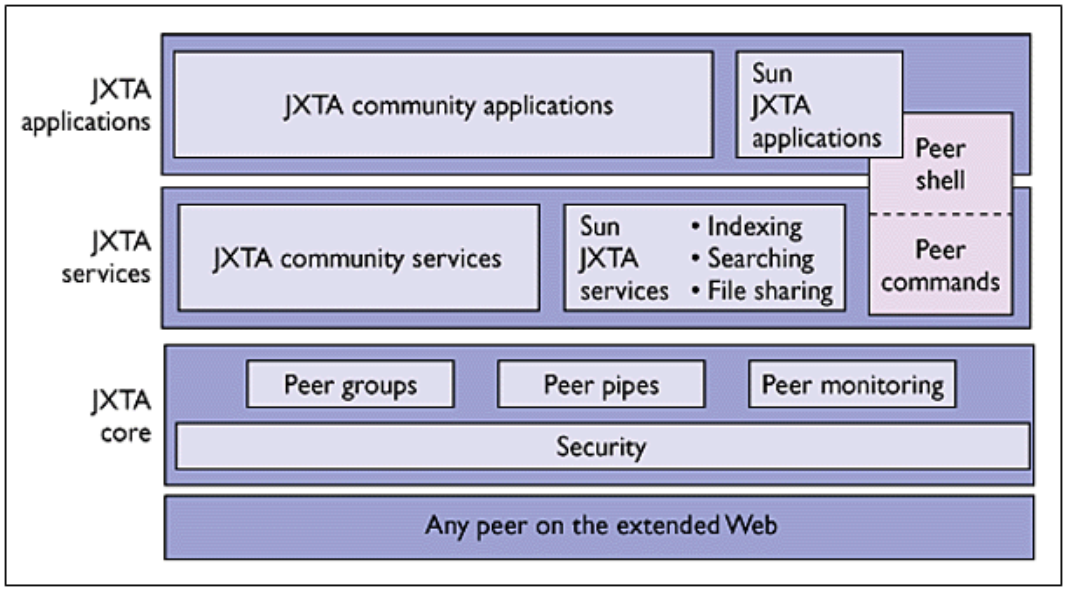

Figure 4.2: JXTA architecture.

Peers and Peer Groups

Any device like PC, PDA, or smartphone that implements one or more of the JXTA protocols could be a peer. Within JXTA network, each peer is

uniquely identified by a static *Peer ID*, which allows the peer to be addressed independently of its physical IP address. This peer ID remains permanent for the device, though it supports multiple network interfaces like Ethernet, WiFi, GPRS, etc. for connecting to the P2P network. The peers can communicate with each other using the network interface best supported by the devices. Moreover, JXTA dynamically uses either TCP or HTTP protocol to traverse network barriers like NATs and firewalls.

JXTA network supports different types of peers to be connected to the network. The general peers are called *full-featured edge peers*. A *minimal edge peer* can send and receive messages just like full-featured edge peer, but does not cache advertisements or route messages for other peers. An edge peer registers itself with a *rendezvous peer* to connect to the JXTA network. Rendezvous peers cache and maintain an index of advertisements published by other peers in the P2P network. Rendezvous peers also participate in forwarding the discovery requests across the P2P network. A *relay peer* maintains route information and routes messages to peers behind the firewalls. A *super peer* has the functionality of both relay and rendezvous peers. Peers can self-organize into *peer groups*.

The peers within a peer group agree upon a common set of services. Not all services within a peer group must be implemented by each peer. A peer just needs to implement services which are useful for it and use the implementation of services provided by the default *Net Peer Group*. Such services are Membership Service, Discovery Service, Pipe Service, etc. Each peer in JXTA network by default belongs to the Net Peer Group.

Pipes

Pipes are virtual communication channels that are used to send messages by JXTA peers. They support the transfer of any object such as binary code, data strings, and Java based objects. Pipes are bound to specific endpoints, such as TCP port and associated IP address. The endpoints of a pipe are referred to as the *input pipe* and the *output pipe*. JXTA pipes

offer two basic modes of communication, *point to point* and *propagate*. A point to point pipe connects exactly two pipe ends (an output pipe and an input pipe). The pipe is asynchronous and unidirectional. A propagate pipe sends messages from one output pipe to multiple input pipes. The broadcast is performed using IP multicast. IP Multicast is a one-to-many messaging protocol used to send one copy of data to a group of recipients who are configured to receive it. Messages transferred across pipes are simple XML documents whose envelope contains routing, digest, and credential information.

Advertisements

In JXTA the decentralization is achieved with the *advertisements*. All resources like peers, peer groups, and the services provided by peers in JXTA network are described using advertisements. Advertisements are language-neutral metadata structures represented as XML documents. Peers discover each other, the resources available over the network, and the services provided by other peers and peer groups, by searching for their corresponding advertisements. Peers may cache any of the discovered advertisements locally. Every advertisement exists with a lifetime that specifies the availability of that resource. Lifetimes provide the opportunity to control out of date resources without the need for any centralized control mechanism. To extend the lifetime, the advertisements are to be republished.

Modules

JXTA specifies *Modules* as a generic abstraction that allows peers to describe and instantiate any type of behavior in the JXTA world. Therefore, the mobile web services are published as JXTA modules in the P2P network. The module abstraction includes a *module class*, *module specification*, and *module implementation*. The module class is primarily used to advertise the existence of a behavior. Each module class contains

one or more module specifications, which contain all the information necessary to access or invoke the module. The module implementation is the implementation of a given specification. There can be more than one implementation for a given specification across different platforms.

Protocols

JXTA platform supports a set of basic protocols. Peers use these protocols to discover each other, advertise or discover network resources, and communicate or route messages. The protocols are implemented in Java and C [Traversat et al., 2003]; they are listed below.

- *Peer Discovery Protocol* enables peers to discover peer services on the network. The protocol can find peers, peer groups, and all other published advertisements.
- *Peer Resolver Protocol* allows peers to send and process generic requests. Queries can be directed to all peers in a peer group or to specific peers within the group.
- *Rendezvous Protocol* handles the details of propagating messages between peers. The rendezvous protocol is used by the Peer Resolver Protocol and the Pipe Binding Protocol to propagate messages.
- *Peer Membership Protocol* allows a peer to join or leave a peer group. The protocol supports the authentication and authorization of peers into peer groups.
- *Peer Information Protocol* provides peers a way to obtain status information from other peers on the network.
- *Pipe Binding Protocol* provides a mechanism to bind a virtual communication channel to a peer endpoint.

- *Endpoint Routing Protocol* provides a mechanism used for routing messages from a source peer to a destination peer.

JXTA for J2ME (JXME)

JXTA was initially targeted for the standalone systems. Several requirements in JXTA protocols made it hard to implement JXTA on MIDP devices; such as, caching and transfer of XML based advertisements, provisioning socket and datagram connections. The JXTA community has developed a light version of JXTA for mobile devices, called *JXME (JXTA for J2ME)* [Knudsen, 2002]. The goal of the JXME project is to bring JXTA functionality to MIDP supporting devices like smart phones. JXME simplified Mobile Host's entry into P2P domain. JXME has two versions: *proxyless* and *proxied*. The proxyless JXME version works similar to native JXTA, whereas the proxied JXME version needs a native JXTA peer to be set up as its proxy. The proxied version is lighter of the two versions and peers using this version participate in HTTP based binary communication with their proxies.

4.1.4 Related Projects Converging Web Services and JXTA

Several projects have taken steps to combine web services and JXTA domains. The project *JXTA-SOAP* [JXTA Community, 2007a] has implemented a transport mechanism for SOAP over JXTA. As discussed earlier, SOAP messages can be transmitted over any transport protocol, like HTTP, BEEP etc. JXTA-SOAP is a transport for Apache Axis, an open-source web services platform for Java. JXTA-SOAP is a transport plug-in allowing SOAP to work over JXTA. When JXTA-SOAP is used, the request/response messages are put in JXTA messages and sent through JXTA pipes, rather than putting in HTTP messages and sending over TCP/IP. The JXTA-SOAP project is being maintained by the Distributed Systems Group of University of Parma, Italy.

[Hajamohideen, 2003] proposed a *Proxy Model* for the invocation of web services in JXTA network (Figure 4.3). The model specifies how services can be published using the module advertisements and can be discovered using the discovery service. But the model does not mention how the services are invoked. The current JXTA model can use a service class implemented as local C++ or Java modules or as binary executable to invoke methods on the local class implementation. The proxy model adapts this feature of JXTA and uses the WSDL document to dynamically generate a proxy class. Systinet WASP (Web Applications and Services Platform) [Allan, 2004] toolkit was used to invoke the remote web services using the proxy class. The proxy model creates an interface to access the web service in an implementation independent manner. The proxy class thus becomes a JXTA service that can be advertised in the JXTA network. The JXTA clients do not require prior knowledge of a web service or any specific web services toolkit.

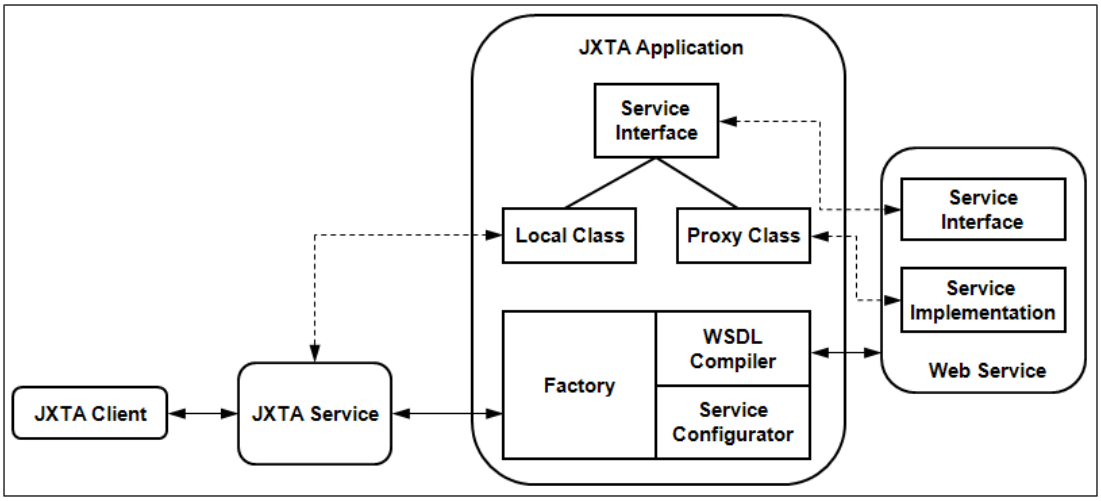

Figure 4.3: Proxy model architecture [Hajamohideen, 2003].

[Qu and Nejdl, 2004] present the mechanism of exposing existing JXTA services as web services with their *Edutella* product. Edutella also integrates web service enabled content providers into JXTA using the proxy model. Edutella is a P2P semantic web application, which is aimed to accommodate heterogeneous resource metadata repositories in a P2P

manner. Edutella facilitates the exchange of resource metadata based on RDF (Resource Description Framework).

Some similar approaches of integrating the web services, semantic web standards, and JXTA P2P technology are adopted by *WS-Talk Project Consortium* [WS-Talk, 2007]. WS-Talk uses WSDL for web service description, OWL (Web Ontology Language) for ontology description, SOAP for access and transfer; all based on JXTA pipes or sockets as a transport layer. Figure 4.4 shows the mechanism of WS-Talk in integrating web services into the JXTA environment.

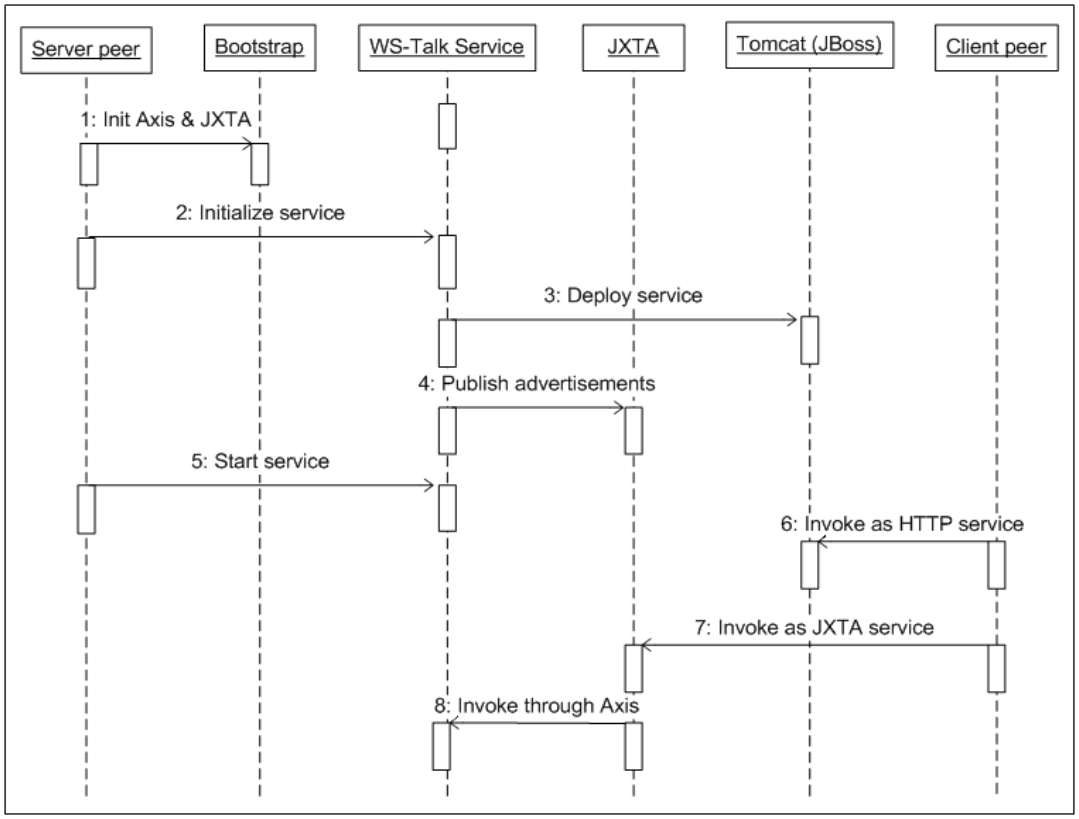

Figure 4.4: JXTA / Web services integration [WS-Talk, 2007].

4.1.5 Lucene

Lucene is the tool used in advanced matching/filtering of services for mobile web service discovery in P2P networks. It is an open source project hosted by Apache and provides a Java based high-performance, full-featured text search engine library [Cutting, 2007]. To search large

amounts of text quickly, one must first index the text and convert it to a format that can be searched rapidly, eliminating the slow sequential scanning of each file for the given word or phrase.

Lucene allows adding indexing and searching capabilities to user applications, which can index and search any data that can be converted to a textual format. Lucene can be used to search and index information kept in JXTA advertisements, webpages on remote web servers, documents stored in local file systems, etc. Lucene supports simple text files, Microsoft Word documents, HTML (HyperText Markup Language), PDF (Portable Document Format), or any other format from which textual information can be extracted.

Java Lucene is a rapid and reliable tool and the product is being used by many well-known websites like *Wikipedia*, as well as in many Java applications. To build an Index, Lucene uses different types of analyzers like *StandardAnalyzer*, *WhitespaceAnalyzer*, *StopAnalyzer*, *SnowballAnalyzer*, etc. The analyzer breaks text fields into index-able tokens. For example, StandardAnalyzer is a sophisticated general-purpose analyzer; WhitespaceAnalyzer is a simple analyzer that separates tokens using white spaces; StopAnalyzer removes common English words which are not usually useful for indexing [Gospodnetic and Hatcher, 2007].

4.2 Mobile Web Services in JXTA Network

JXTA peers use XML as standard messaging format and create a virtual P2P network across the devices connected over different networks. JXME provides a perfect platform for mobile service hosts in P2P domain, as JXTA eliminates many of the low level details of the P2P systems, like transport details. Peers can communicate with each other using the best of available network interfaces supported by the devices, like Bluetooth, WiFi, GPRS, etc.

Considering the advantages and features of JXTA, the mobile service provider is adapted in the JXTA network to check its feasibility in P2P networks. The architecture of deployment scenario in the JXME network is shown in Figure 4.5 [Hassan, 2009]. The *virtual P2P network* (also called *mobile P2P network*) is established under a mobile operator network, where at least one of the nodes belongs to the operator proprietary network and acts as a JXTA super peer. As explained previously, JXTA network supports different types of peers to be connected to the network. The general peers are called *edge peers*. The edge peers register themselves with a *rendezvous peer* to connect to the JXTA network. Rendezvous peers cache and maintain an index of advertisements published by other peers, and participate in forwarding the discovery requests across the P2P network. The *relay peers* maintain route information and route messages to peers behind the firewalls. The *super peer* has the functionality of both relay and rendezvous peers. The super peer can exist at a Base Transceiver Station (BTS) and can be connected to other base stations, extending the JXTA network.

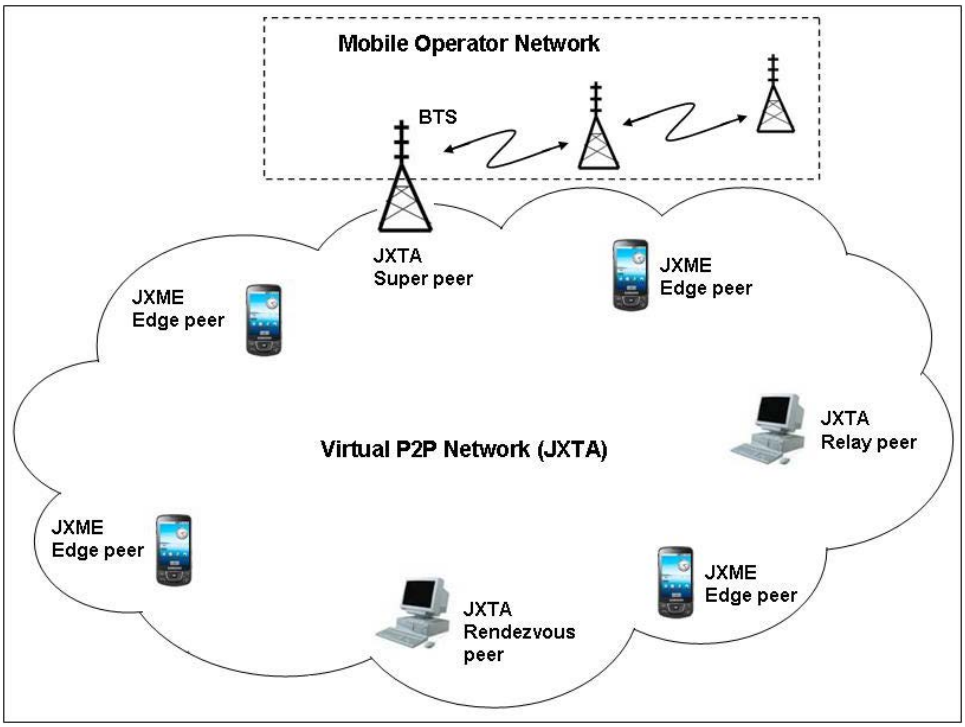

Figure 4.5: The deployment scenario in JXME network.

Any mobile service host or mobile service client can join the P2P network using the super peer at base station as the rendezvous peer. The super peer can also relay requests to and from JXTA network to smartphones. If the operator network supports, stationary peers can also join this network as both rendezvous and relay peers, which further extends the mobile P2P network.

Mobile service provider in JXME network offers many advantages in domains like collaborative learning, image sharing, and location based services, by taking advantage of individual peers' resources like storage space or processing power. Moreover, the general mobile phone users are interested in applications rather than individual components. The applications might use one or more web services at the backend and can be provided as installable applications. In this setting, the P2P network can offer easy means of storing and sharing the installable applications for the participating peers.

Within the JXTA network, each peer is uniquely identified by a static peer ID, which allows a peer to be addressed independently of its physical IP address. This peer ID remains permanent for the device, though it can support multiple network interfaces (e.g. Ethernet, WiFi) to connect to the P2P network. Using the peer ID, mobile service provider can remain always visible to the web service clients regardless of the operator networks. Mapping the peer ID to its IP address is taken care of by the JXTA system, thus eliminating the need for static public IP.

Moreover, the web services deployed on the mobile service hosts can be advertised as JXTA modules in the P2P network [JXTA, 2008]. The module advertisements are searchable, therefore the web services can be found as standard JXTA services. The module advertisements together can represent a combination of UDDI and WSDL; as they enable publishing and finding service description, and defining transport binding to the service [Schneider, 2001]. The SOAP messages are transmitted across JXTA pipes for transformation.

4.3 P2P Framework based on Mobile Services Gateway

To achieve interoperability of web services at mobile service hosts, messages need to be transformed at some intermediary level. Such transformation of messages may also be required to achieve proper scalability for mobile service hosts. This demands the necessity for some intermediate gateway in the mobile web service infrastructure. Therefore, the proposed architecture of the system is based on a *Mobile Services Gateway (MSG)*, which maintains individual user profiles and personalization information (Figure 4.6).

The mobile services gateway is established as an intermediary between the service clients and the mobile service providers in JXTA network. The virtual P2P network is established within a mobile operator network having at least one node in the proprietary network, acting as a super peer. The mobile services gateway supports super peer functionality, and leads the JXTA network to mobile operator's proprietary network. The web service clients can access the services deployed across the JXTA network and MSG. External service clients can also access the services deployed on mobile service hosts through the *Gateway Service Registry (GSR)* of MSG. Thus the MSG also acts as a gateway from the Web to the mobile P2P network.

As hand-held devices have many resource limitations like low computation power, limited storage, and limited battery life, so to conserve resources mobile service hosts can work as *'turn-on only on client request'*. The MSG can identify which request is for particular service host using their profiles and can send message to activate the mobile service provider. Besides, as the mobile service providers are in a JXTA virtual P2P network, therefore MSG supports super peers for the JXME edge peers. At least one super peer maintains a *Local Service Registry (LSR)* for the peer group, providing fast service registration and invocation within the peer group environment.

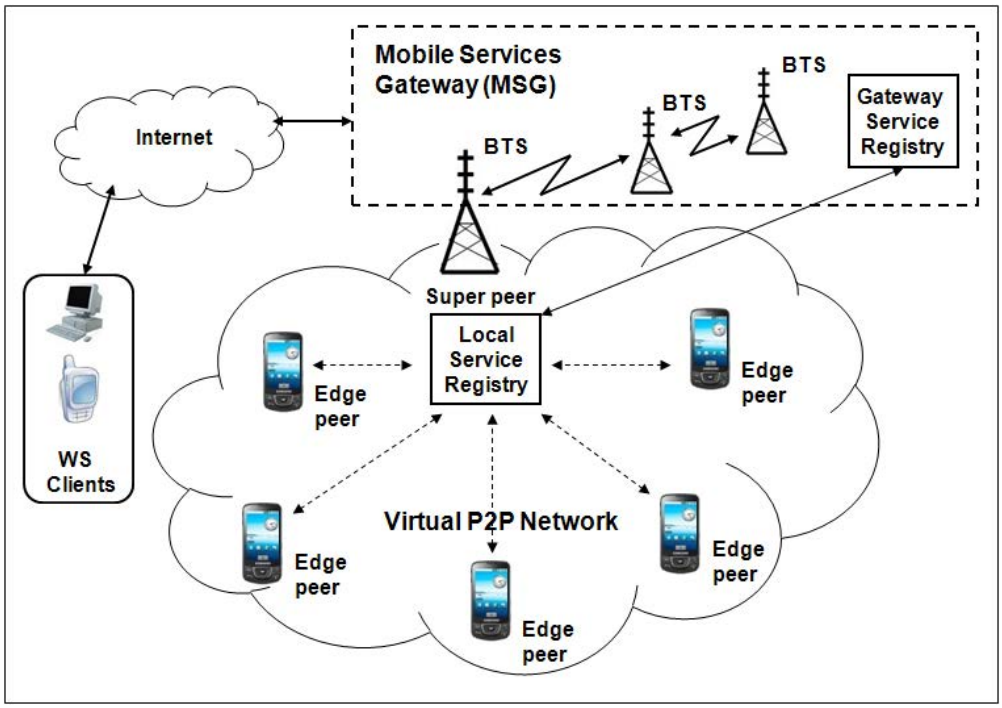

Figure 4.6: Mobile Services Gateway bridging P2P and Web environments.

Though web services deployed on mobile devices are advertised as JXTA modules and identified with peer ID, the devices can still have public IP. This enables the external clients to search the GSR registry at MSG and access the mobile web services directly. Therefore, MSG supports accessing the services over both P2P and standard web services protocols.

4.3.1 Enterprise Service Bus for Mobile Services Gateway

Enterprise Service Bus (ESB) [Borck, 2005] is the preferred choice for realizing MSG in this architecture. ESB provides a dedicated infrastructure that supports messaging, web services, data transformation, and intelligent routing for loosely coupled and highly scalable integration networks. The infrastructure is established based on web services specifications and standards. ESB exploits a *message oriented middleware*, and provides *content based routing* and *transformation*.

The architecture of the ESB is built on a *bus*, which provides message delivery, transformation, and routing of service requests; and handles the communication, integration, and service interaction. The bus is typically designed for high-throughput and guaranteed message delivery, for a variety of service providers and consumers.

When making a request, a client simply refers to the WSDL interface of a service and the service bus selects one of the available implementations of this interface to process client's request [Leymann, 2005]. The transformation functionality maps a data format onto another (usually XML) to ensure interoperability between various systems plugged onto the ESB. The bus also can perform content based routing. In this mechanism, the message is routed based on its content by evaluating the statement in the header. ESB enables services to interact with each other based on the QoS requirements of individual transactions.

The mobile services gateway is realized based on the ESB platform [Hassan, 2009]. Therefore, MSG supports transforming and routing the web service messages. The routing is based on message contents. This provides better QoS for the mobile service hosts and the clients. MSG can also be responsible to handle the security issues regarding proper authentication and authorization. To support scalability, MSG performs message transformation between web services specifications and other specifications feasible for mobile communications.

4.3.2 Categorization of Mobile P2P Services

The services provided by different mobile peers can be combined together to achieve high-level of service reusability by creating *mash-ups* [Schroth and Janner, 2007] of mobile peer services at the MSG. Based on this, mobile services can be categorized as *node services* and *compound services* (Figure 4.7).

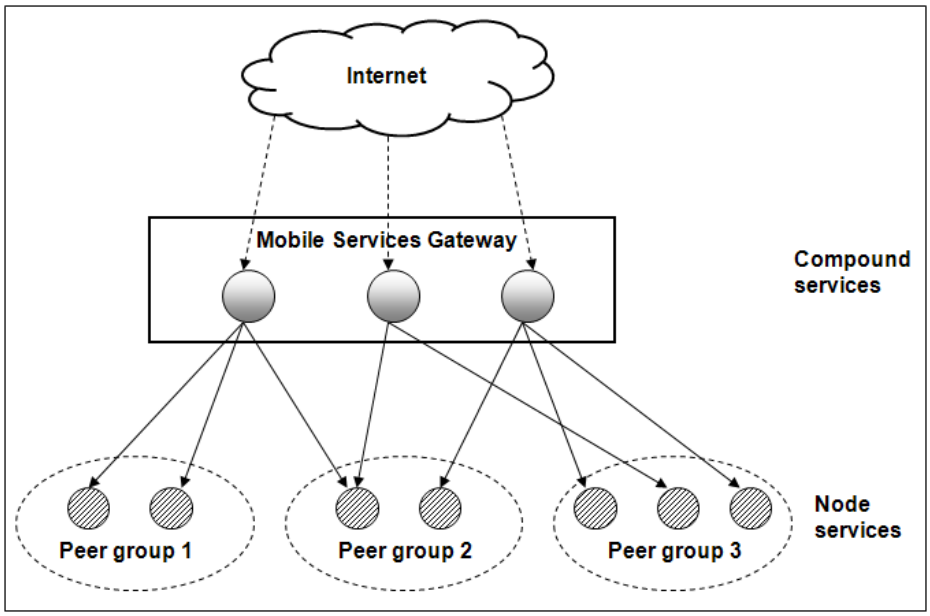

Figure 4.7: Categorization of mobile P2P services.

Node services are the fundamental services provided by the mobile nodes in the P2P network. Node services can be used as building blocks to compose new compound services to meet complex requests of clients. As some of the node services are private within their peer group, therefore the node services which are provided also for other peer groups to access can only participate in forming compound services.

On the other hand, compound services are comprised of several node services, which may also encapsulate features from other services available on the Internet. Compound services are published in the MSG so that they can be discovered by all the nodes in the network.

4.3.3 Distributed Service Registries for Mobile P2P Service Providers

The proposed architecture is based on the integration of different P2P networks and the MSG with SOAP connectivity. In this setting, mobile peers residing as the neighbors on the same P2P network are pulled together to form a peer group [Yang and Garcia, 2003]. Usually, the peers in same peer group have similar interests or come from the same

organizations. There is at least one super peer in each peer group. Super peers are responsible for mediating the other peers and providing security services in their own peer group (Figure 4.8).

A Local Service Registry (LSR) is maintained by the super peer, providing fast service registration and invocation within the peer group. A Gateway Service Registry (GSR) is maintained at the MSG, which is directly accessed by clients across the Internet. The GSR provides access to different physical P2P networks by interconnecting super peers using different transport protocols, and maintains the mapping of service descriptions between GSR and LSRs.

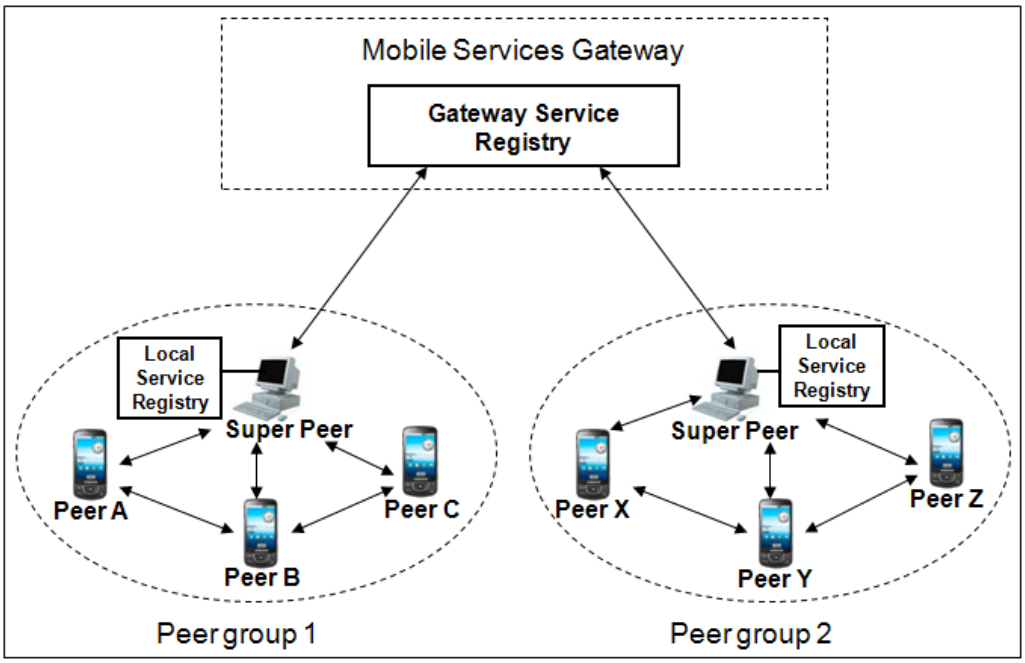

Figure 4.8: Service registries in mobile P2P networks.

When a mobile peer wants to publish its services, it registers the services to the LSR on super peer in its peer group. Some peers may only want to share their services within the peer group. In that case, registering the services onto GSR is not needed for these peers. If the peers want to provide their services to other peer groups or to any client outside their own peer group, then the super peer registers the services

onto the GSR, so that any client can discover the services and connect to them.

When a mobile peer wants to find a certain service, the request will be first directed to the super peer in its own peer group. The super peer will look up the LSR on its own site. If a matching service exists, the service description will be returned back to the requestor. Otherwise, the request will be forwarded to the GSR and the mapping maintained by GSR will indicate the service provider's details. The algorithm of the process is shown in Listing 4.1. After getting the service description from GSR, the requestor will connect and bind directly to access the services.

Listing 4.1: Finding a service in the virtual P2P network.

```
DiscoverServiceAdvertisement(){
    while(the number of service request ≠ Ø){
        Lookup Advertisements in local service cache;
        if(Search result does not match the request){
           Send service request to LSR;
           if(Search result matches the request)
              Save result in local service cache;
           else{
              Send service request to GSR;
              if(Search result matches the request)
                  Save result in local service cache;
              else
                  Return result of no matches;
                }
              }
           Return result of service discovery;
         }
      }
```

An important advantage of this architecture is the flexibility of registering new services by using super peers and LSRs. This reduces the heavy load on GSR when the number of service providers and services in the environment are very large. The other advantage is that the service requests are not flooded across the whole networked systems.

4.4 Supporting Mobility of Mobile P2P Service Providers

The deployment of mobile web services faces a significant challenge due to the mobility factor of the mobile service provider. Nodes can join or leave network at any time and can switch from one network to another. Movements of devices and services between networks make mobile web services different and complex to control.

To understand the mobility issues let us consider the GSM telephony system [Heine, 1999] to see how it allows a mobile device to roam. Mobile services in GSM consist of components both located on the mobile device and in the network. Figure 4.9 shows the simplified architecture of roaming in GSM system. The *Mobile Terminal (MT)* consists of two components. The *Mobile Equipment (ME)* is the phone itself and the *Subscriber Identity Module (SIM)* is the Smart Card. For mobile telephony, the components ME and SIM interact with the components on the *Mobile Switching Centre (MSC)*. These components are *Home Location Register (HLR)* and *Visitor Location Register (VLR)*. While a mobile device is allocated to a unique HLR, it communicates with different MSCs and VLRs according to its location.

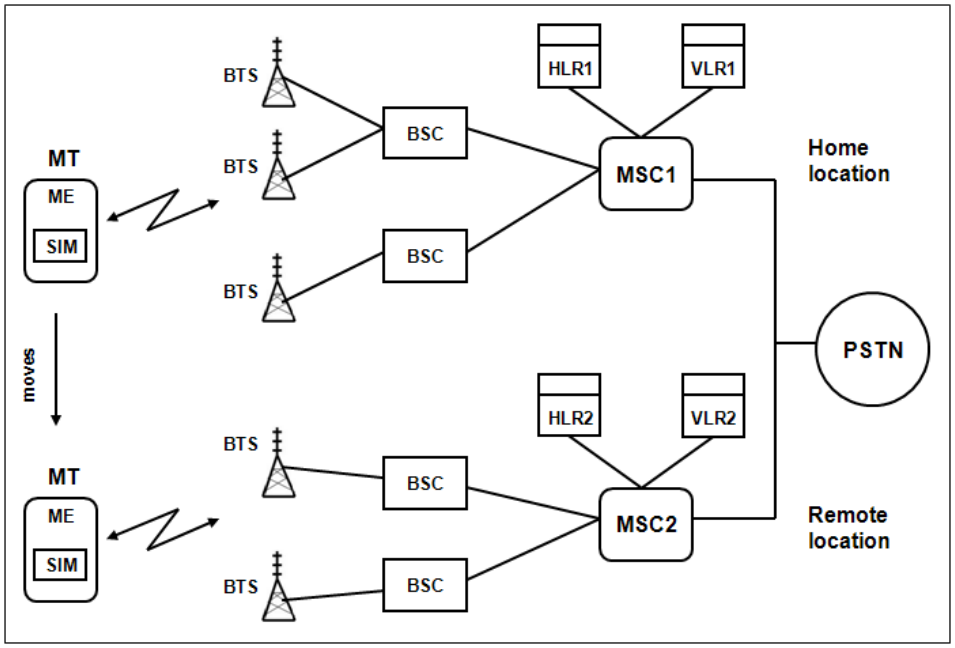

Figure 4.9: Roaming of a mobile terminal across networks.

Figure 4.9 illustrates how the system allows a mobile device to move between two different networks. The MT physically moves to a remote location and needs to access another network which is reachable in the new location. The MT now accesses the new radio network through different *Base Transceiver Stations (BTS)*. These BTSs are controlled by different *Base Station Controllers (BSC)* and a different mobile switching centre (MSC2). Therefore, the device is also registered in another visitor location register (VLR2) in the new network. However, it still communicates with the HLR of its original network. As shown in the figure, as the mobile device moves, the components ME and SIM will switch from interacting with MSC1 and VLR1, to MSC2 and VLR2.

This system is compatible with the proposed P2P architecture as the mobile device still communicates with its HLR through its original MSC. In the proposed P2P architecture, the gateway service registry maintains some data (Figure 4.10) of each registered service; namely, *Service Advertisement*, *Peer ID*, *Home Peer Group*, and *Remote Peer Group*. Service advertisement is the actual description of the advertised service. Peer ID is the unique identifier of the mobile peer service provider. Home peer group is the original peer group the mobile peer belongs to; and remote peer group is the new peer group the mobile peer moved into as a visitor. Normally, when a peer is within its home peer group, there is no value assigned to its *remote peer group* item. Having a value for the item *remote peer group* indicates the GSR that, the provider of the service currently belongs to a remote peer group.

Gateway Service Registry (GSR)	
Service advertisement	Service advertisement .....
Peer ID	Peer ID .....
Home peer group	Home peer group .....
Remote peer group	Remote peer group .....

Figure 4.10: Data maintained at GSR for service advertisements.

Three different scenarios are identified when a mobile service provider physically moves from its peer group to a new peer network. Figure 4.11 shows the node movement across peer groups; where *peer C* moves from *peer group 1 (PG1)* to *peer group 2 (PG2)*. The possible scenarios are listed below and the strategies to handle them are discussed in the next subsections.

- The mobile service provider moves to a remote peer group due to node movement and still wants to provide the same services and stay as a peer in the original peer group.
- The mobile service provider provides additional new services and wants to join a new peer group of similar interests.
- The mobile service provider stops current services and leaves its original peer group; and after moving to a remote location, joins a new peer group.

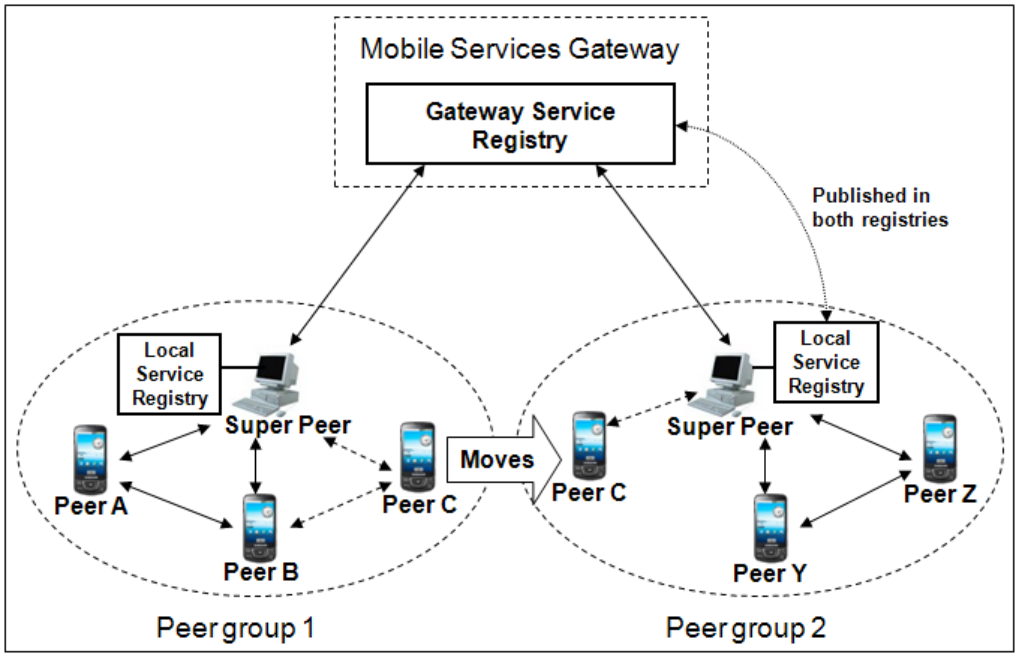

Figure 4.11: Mobile service provider moving across peer groups.

4.4.1 Peer Moving to Remote Peer Group

In the first scenario, when the mobile service provider (i.e. *peer C*) moves to a remote peer group, the mobile peer tries to register its services to the LSR2 in the new peer group (Figure 4.12). The super peer in PG2 will register the new service advertisements to its LSR2 and as well as to the GSR at the mobile services gateway. The GSR will add/update the details of the service advertisements provided by *peer C*. The reason for registering the services of *peer C* to both local and gateway service registries is that, *peer C* is in a remote network and still wants to maintain relationship with its original peer group. Besides, *peer C* has similar interests of services to its original peer group (PG1). Therefore, to maintain availability to PG1, *peer C* must communicate through the MSG.

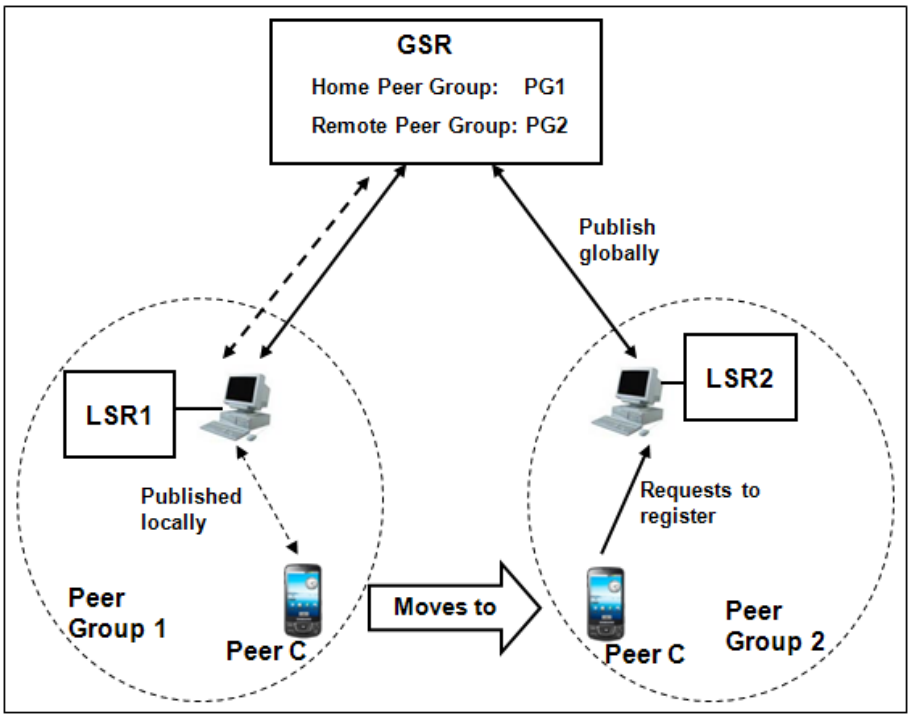

Figure 4.12: Mobile service provider moves to remote peer group.

To publish the service advertisements of *peer C* from PG2, the GSR will assign the value *PG2* to its item *remote peer group*. But the LSR1 will still keep the advertisements of *peer C*. When an external client requests

for a service provided by *peer C*, the GSR will find and locate *peer C* in PG2 directly. But if any peer in PG1 searches for a service provided by *peer C*, the super peer will be unable to locate the services and will look up onto the GSR and will be able to find the services in remote peer group PG2.

4.4.2 Peer Joining Multiple Peer Groups

In the second scenario, the mobile service provider (i.e. *peer C*) wants to provide additional new services and wants to join new peer group (PG2) of similar interests within reach (Figure 4.13).

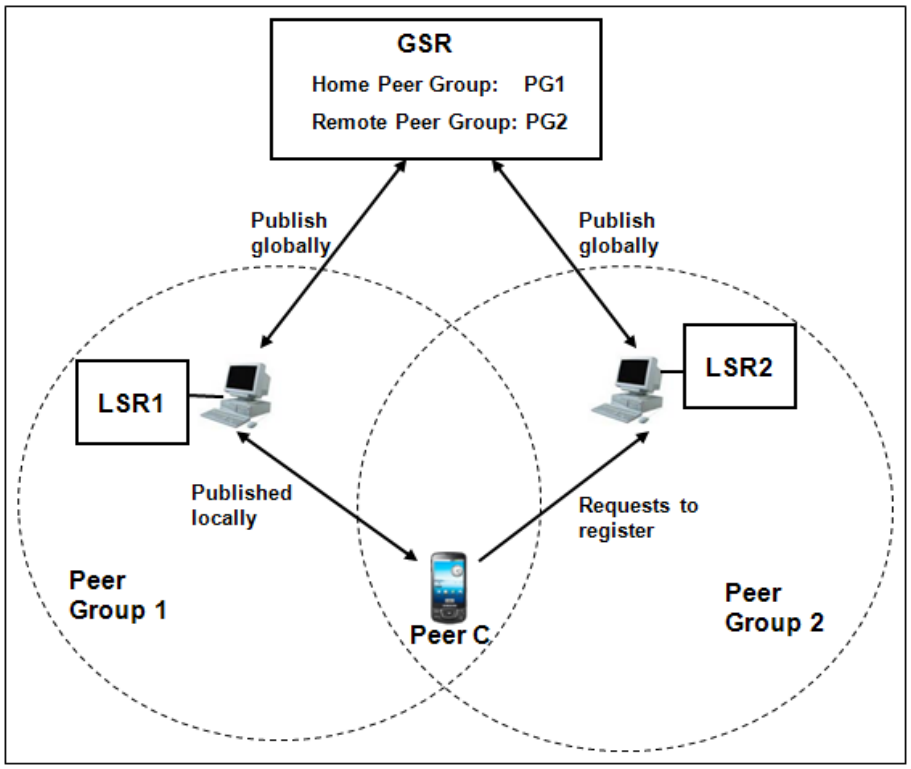

Figure 4.13: Mobile service provider joins two peer groups.

In this case, *peer C* tries to register its new services to the LSR2 in PG2, while continuing the previous services in PG1. *Peer C* has some services registered with PG1 and some services registered with PG2. If *peer C* wants to provide some or all of its services globally, then the corresponding super peer will register that service to the GSR. The GSR

will update the entry by assigning the value *PG2* to the item *remote peer group* for the services registered at LSR2 of PG2. When an external client requests for a service provided by *peer C*, the GSR will find and locate *peer C* directly in PG1 or PG2, according to the LSR where it has advertised.

4.4.3 Peer Migrating to Remote Location

In the third scenario, the mobile service provider (i.e. *peer C*) wants to stop current services and leave PG1. Later on, *peer C* moves to a remote location and joins PG2 (Figure 4.14).

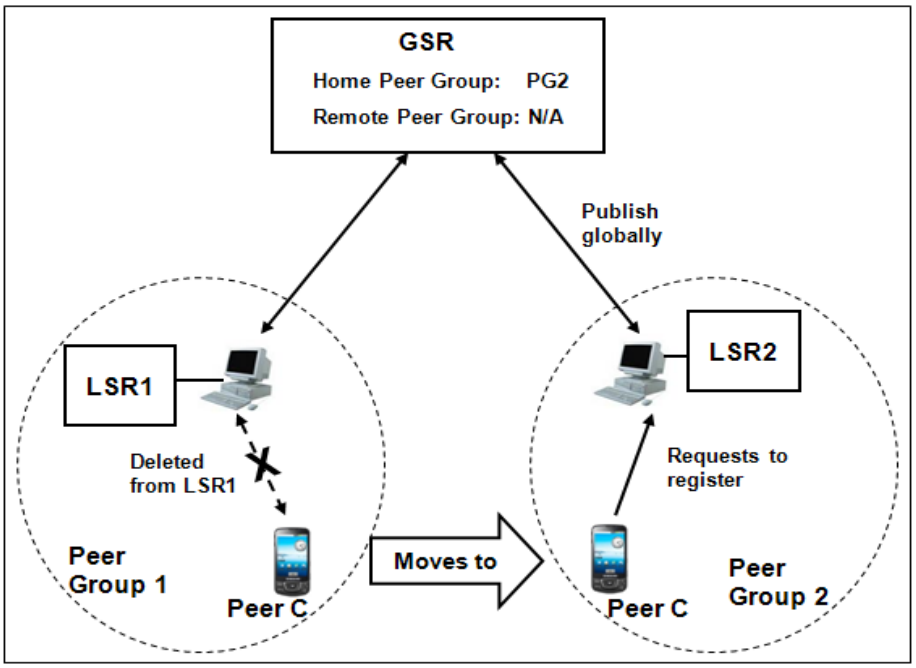

Figure 4.14: Mobile service provider migrates to remote location.

At first, *peer C* notifies the super peer of PG1 about its suspension of services. The super peer deletes the services from its LSR1 and also notifies the GSR to do so (if GSR contains any entries). After moving to a new location when *peer C* joins the new peer group (PG2), it will try to register its service advertisements to the LSR2 of PG2. If *peer C* wants to provide services globally, it will request the super peer accordingly and the advertisements will also be published on GSR. In this case, there is

no value assigned to its *remote peer group* item; as *peer C* has no attachment with its former peer group (PG1), and PG2 has become its current *home peer group*.

In this manner, the movements of the mobile P2P service providers are handled across peer groups in different networks.

4.5 Mobile Web Services Discovery

Service discovery mechanism is very crucial and the first step to achieve in deploying applications in mobile environments. Although discovery systems like DNS (Domain Name System) [Gulbrandsen et al., 2000], LDAP (Lightweight Directory Access Protocol) [Hodges and Morgan, 2002], and UDDI are available, these systems are designed for statically configured environments, and are not specifically suitable for mobile computing. The mobile computing model requires more dynamic and spontaneous discovery features, which are resilient to faults and require minimal administration [Lee et al., 2003]. The following subsections address the discovery aspects of mobile web services and propose the JXTA based service discovery mechanism.

4.5.1 Discovery Aspects of Mobile Web Services

Typically, web services are built for static networks. Once a service provider deploys the service, the service is published to a UDDI registry. The registry maintains reference of the WSDL document. The WSDL document describes the service, along with the location of service (binding information) and the operations (methods) the service exposes. Any potential web service client searches for the service in the public registry, gets the description of the service and tries to access the service.

Since mobile web services are developed mostly by using the basic web services architecture, the standard WSDL and UDDI can theoretically be used to describe and publish services. But in commercial

environments with each mobile host being able to providing some services, the bulk of services expected to be published could be quite high. In such a condition, a centralized solution can create bottlenecks and cause single points of failure.

Besides, mobile networks are quite dynamic due to the node movements. Nodes can join or leave network at any time and can switch from one operator to another. This will cause the binding information in the WSDL documents to be inaccurate. Hence the services are to be republished every time the mobile host changes its binding information. This creates real difficulty in keeping up to date information of the published mobile web services in the centralized registries.

4.5.2 Dynamic Service Discovery Mechanisms

Dynamic service discovery is an extensively explored research area. Most of the available service discovery protocols are based on the *announce-listen model*. In this model, *periodic multicast mechanism* is used for service announcement and discovery. A service advertisement is generally associated with a lifetime during which the service is expected to remain available. Using these advertisements the services are discovered dynamically. One of the frontrunners using this mechanism is the RMI based Jini technology. The Jini discovery mechanism is explained in detail for developing mobile service host, in section 3.2.

Similarly, [Yang et al., 2003] proposes an infrastructure for efficiently accessing mobile web services in broadcast environments that defines a *multi-channel model* to carry information about mobile web services. But this mechanism assumes a service proxy object acting as the registry and is available always. But services distributed over the dynamic ad-hoc networks must be discovered without a centralized registry. Specifically, the discovery protocols should be able to support spontaneous peer to peer connectivity to facilitate ad-hoc collaboration.

[Dustdar and Treiber, 2006] proposes a *distributed peer to peer web service registry* solution based on lightweight web service profiles. They have developed *VISR (View based Integration of Web Service Registries)* as a peer to peer architecture for distributed web service registry. VISR's distributed registry model allows VISR peers to operate on a common data structure and provides a common vocabulary. The idea behind the approach is that the concept of a dedicated web service registry decouples web service providers from storing the actual web service description. Therefore with VISR approach, each node has all the features of a SOA (i.e. service requestor, provider, and broker).

Similarly, *Konark service discovery protocol* was designed for discovery and delivery of device independent services in ad-hoc networks [Lee et al., 2003]. In this approach, a node multicasts the differences between services that the node knows and the ones others seem to know. In other words, each participating node gossips its knowledge about services minus the network's knowledge. So the individual nodes complement one another's limited knowledge to attain the global view of the entire network. For achieving this, Konark is designed based on a P2P model with each participating device having the capabilities to host and deliver services, and query the network for available services offered by others.

Beside these developments, Universal Plug and Play (UPnP) [UPnP Forum, 2003] discovery protocol allows devices to advertise their services to *control points* (i.e. clients) on the network. The goals of UPnP are to allow devices to connect seamlessly and to simplify the implementation of networks. Unlike VISR and Konark where each node maintains its own service registry, UPnP does not cache service information at all. Services multicast their advertisements periodically. UPnP control points discover services either by passively listening to the advertisements or by actively multicasting service discovery request messages.

Considering these developments and the need for distributed registry and dynamic discovery, the thesis studied alternative means of discovering mobile web services. The approach is based on JXTA P2P network and is explained in detail in the following subsection. The approach is conceptually similar to Konark but is truly scalable by following the open standards of web services technology and JXTA P2P protocols. The service descriptions are stored only at nodes with higher capabilities (*super peers*) and the discovery process still preserves the SOA. This approach of mobile web service discovery significantly differs from VISR where all nodes are registries and UPnP where no node is a registry. The service descriptions also benefit from widely accepted web services standards.

4.5.3 Discovery of Mobile Web Services in JXTA Network

In JXTA the decentralization is achieved with the *advertisements*. All resources like peers, peer groups, and the services provided by peers in JXTA network are described using advertisements. Advertisements are language-neutral metadata structures represented as XML documents. Peers discover each other, the resources available in the network, and the services provided by peers and peer groups, by searching for their corresponding advertisements. Peers may cache any of the discovered advertisements locally. Every advertisement exists with a *lifetime* that specifies the availability of that resource. Lifetime gives the opportunity to control out of date resources without the need for any centralized control mechanism. To extend the lifetime of services, the advertisements are to be republished.

Thus the services deployed on mobile host peers in JXTA network are to be published as JXTA advertisements, so that they can be identified as JXTA services among other peers. JXTA specifies *modules* as a generic abstraction that allows peers to describe any type of behavior in the JXTA environment. So the mobile web services are published as

JXTA modules in the P2P network. The module abstraction includes a *module class*, *module specification*, and *module implementation*. The module class is primarily used to advertise the existence of behavior. Each module class contains one or more module specifications, which contain the necessary information to invoke the module. The module implementation is the implementation of a given specification.

Figure 4.15 shows the mapping between JXTA modules and web services. The collection of module abstractions represent as UDDI in terms of publishing and finding service description, and as WSDL in terms of defining transport binding to the service.

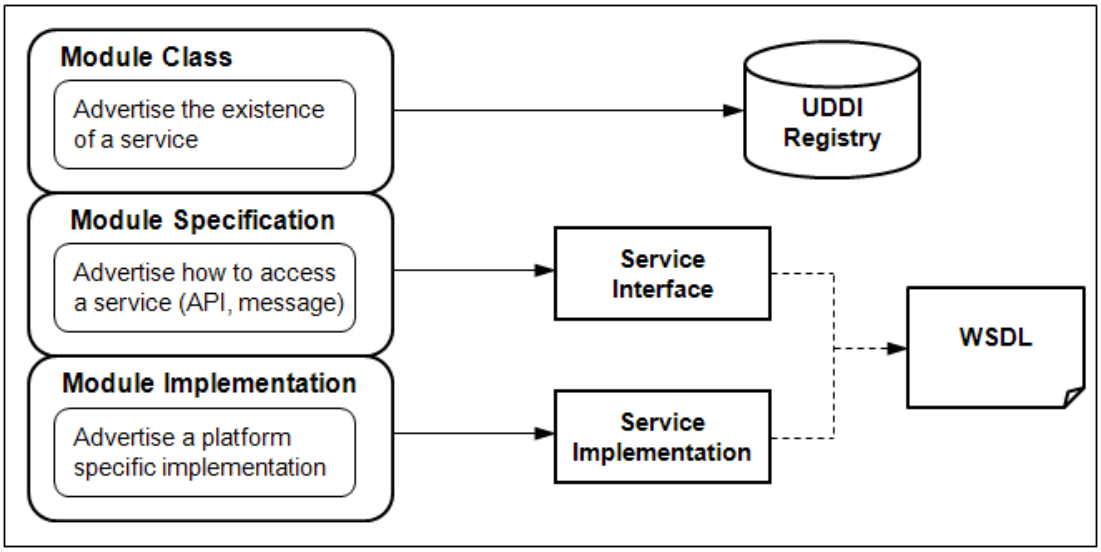

Figure 4.15: Mapping between JXTA modules and web services.

To publish mobile web services in JXTA network, a standard *Module Class Advertisement (MCA)* is published in the P2P network, declaring the availability of the web service definitions. Once new web services are developed for a mobile host, the WSDL descriptions of the services are incorporated into the *Module Specification Advertisements (MSA)* and are published in the P2P network. The MSAs are published in JXME network with an approximate lifetime that specifies the duration of time the mobile host wants to provide the service. The MSAs are cached at rendezvous peers or any other peers with sufficient resource

capabilities. Once the lifetime expires the MSAs are automatically deleted from the P2P network, thus avoiding the invalid advertisements. If the mobile host wants to extend lifetime of the provided service, the particular MSA can be republished. The structure of the MSA is shown in Listing 4.2.

Listing 4.2: Structure of MSA advertising a mobile web service.

```
<?xml version="1.0" encoding="UTF-8" ?>
<jxta:MSA>
<MSID> … … … </MSID>
<Name> … … … </Name>
<Crtr> … … … </Crtr>
<SURI> … … … </SURI>
<Vers> … … … </Vers>
<Desc> … … … </Desc>
<Parm>
    <WSDL>
        <definitions … >
                <message … > … … … </message>
                <portType … > … … … </portType>
                  … … …
        </definitions>
    </WSDL>
</Parm>
<jxta:PipeAdvertisement> … … … </jxta:PipeAdvertisement>
<Proxy> … … … </Proxy>
<Auth> … … … </Auth>
</jxta:MSA>
```

The MSA contains unique identifier <MSID> that also includes the static module class ID, which identifies the web services module class advertisement. The other elements of <MSID> include *name*, *creator*, *specification*, and *description* of the advertisement. The optional element <Parm> contains description of the web service being advertised. The <PipeAdvertisement> contains the advertisement of the pipe which can be used to connect to the web service deployed on the mobile service host. The receiving endpoint of the pipe can be addressed with a Peer ID of the respective peer. Thus if the invocation of mobile web service is

within JXTA network, using pipes the need for public IP is eliminated. Once the web services are discovered, the communication between the mobile service provider and the service client is still done using SOAP over HTTP. The remaining two elements <Proxy> and <Auth> carry the proxy module and the authentication information of the web service module.

The MSAs carrying the web service descriptions can be searched by name and description parameters. The JXTA discovery model uses a *Query Routing Protocol (QRP)* which is text based (i.e. XML-based) [Waterhouse et al., 2002]. Therefore, JXTA search implementations are platform and language independent. The JXTA API provides a simple keyword search on the name and description elements of the modules advertised in mobile P2P network.

In fact, the basic parameters are not sufficient to find out exact search results out of huge numbers of possible mobile web services. Some studies tried to extend the search criteria to WSDL level. This means that search parameters are not restricted to MSA details only, and will also include the WSDL tags and information. Such an approach is taken in the *UDDI Explorer* tool, developed for searching standalone web services [Dao, 2005]. Index searching tools can be used to perform advanced discovery of mobile web services in P2P, to match the best suited services.

4.5.4 Mobile Web Services Discovery Process

The complete mobile web service discovery process is shown as an activity diagram in Figure 4.16. The mobile service host declares its services by publishing the respective MSAs in their respective peer groups by joining them. The client initiates the query for services at the mobile services gateway. The MSG searches for the matching module advertisements in the JXTA network. The module advertisements are then filtered by weight of the keywords. The advanced matching is done using the profile and context of the client (if available). The matching results are then

forwarded back to client. The client scrolls through the list of services and selects the relevant mobile web service. The client can then access the web service from the corresponding mobile service host.

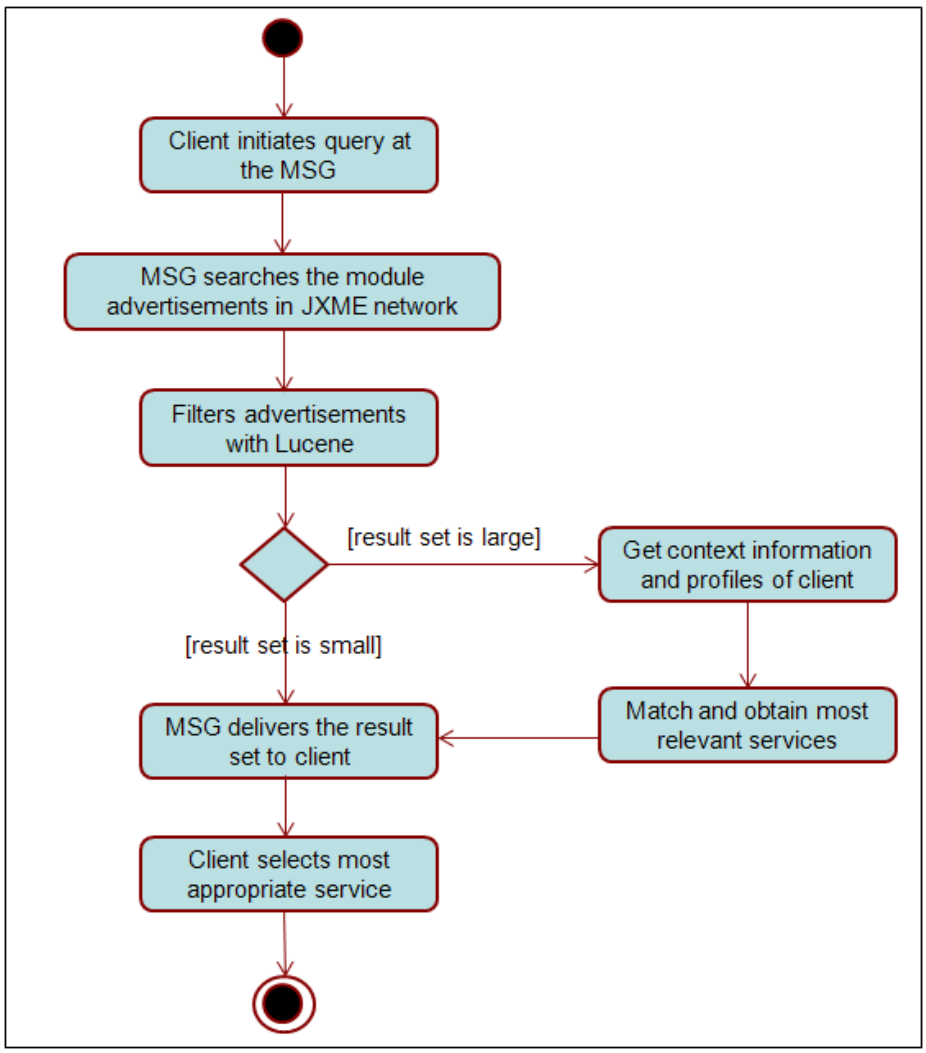

Figure 4.16: Complete mobile web service discovery process.

4.6 Mobile Web Services Invocation in JXTA Network

Once the services are discovered, they can be accessed from Internet using the IP feature. Accessing the services directly from the mobile host is also possible in JXTA network without the need for public IP. As already discussed in section 4.1.3, a pipe is to be created for receiving incoming messages over the JXTA network. The mobile host creates a

pipe for the services and adds the pipe advertisement along with its peer ID to the MSA of the services. As explained earlier, MSA has the `<jxta:PipeAdvertisement>` parameter, which can be used to hold the respective pipe advertisement of the service.

After the service discovery process, a client gets access to the pipe along with the service description using the pipe advertisement. Using this pipe the client can send the service requests to the mobile service host. In JXTA the pipes are unidirectional. So the client should create its own pipe for receiving the response and should advertise it along with the web service request.

Once the request is initiated, the web service message is received at the mobile host on default JXTA port (9700). But as the mobile host follows SOAP over HTTP, it can only receive a web service message over HTTP protocol on port 80. Therefore, a JXTA application is to be run on the mobile host which receives the web service message, extracts the SOAP contents, and initiates SOAP over HTTP request on port 80. The response from the mobile service host also follows the same port forwarding and the results are sent back to the client.

4.7 Evaluation of Prototype Mobile Services Gateway

A prototype of the mobile services gateway was designed using ServiceMix ESB, to test the MSG extensively for its performance and scalability using load test principles. A large number of clients were generated for the prototype MSG, simulating real-time mobile operator network load. The `searchArticle` service (explained in section 2.3.3) is considered for this performance analysis of the MSG.

4.7.1 Test Setup

The ServiceMix based MSG is established on a Toshiba Satellite A100 laptop equipped with a 1GB RAM and 1.83 GHz Intel processor. A Java

based server was developed and run on the same laptop on an arbitrary port (4444), mocking the mobile service provider. The server receives service request for `searchArticle` from the client and populates standard response. The response is then encoded in BinXML and the compressed response is sent back to the client in HTTP response message format. By considering this simple server, we can eliminate the pure performance delays of the mobile host and the transmission delays of the radio link; thus getting the actual performance analysis of the MSG.

For the load generation, a Java clone of the ApacheBench load generator was used from WSO2 ESB [Apache Software Foundation, 2007b; WSO2, 2007]. The load generator can initiate a large number of concurrent web service invocations simulating multiple parallel clients. The command line executable `benchmark.jar` provides a detailed statistics of the invocations, like the number of concurrent requests, successful transactions per second, mean of the client invocation times, etc. The `benchmark.jar` and `commons-cli-1.0.jar` are downloaded to a working directory and are used to simulate a large number of concurrent requests. The following sample command simulates 200 concurrent clients for the `searchArticle` service, with each client generating 10 requests for the same service.

```
java -jar benchmark.jar -p searchArticleRequest.xml -n 10
-c 200 –H "SOAPAction: urn:searchArticle" –T "text/xml;
charset=UTF-8" http://localhost:8912/soap/Service
```

4.7.2 Test Results

Figure 4.17 shows the time taken for handling a client request under multiple concurrent requests generated for the MSG. The MSG was successful in handling up to 210 concurrent requests without any connection refusals. Above this, higher numbers of concurrent requests were generating connection refusals; as because, the ServiceMix ESB can handle only as many concurrent requests as the number of threads

configured in the system [Perera, 2007]. Figure 4.17 also shows a steady increase in the average time taken for handling a client request with the increase in number of concurrent requests. After 240 concurrent requests, there is a sharp decline in the time taken to handle a client. The decline is because of a large number of failed requests at this concurrency level.

In this observation, more than 300 concurrent requests are not considered, as at this high concurrency level the number of failed requests is already more than 50% of the total requests. The mean duration of handling a single request remains mostly constant in this observation. The mean is calculated by observing the performance of MSG over long durations. The mean value lies in the range 100-150 milliseconds, and improves slightly with the increase in concurrency levels. This shows that the performance of MSG is actually improving when there are large numbers of clients to handle.

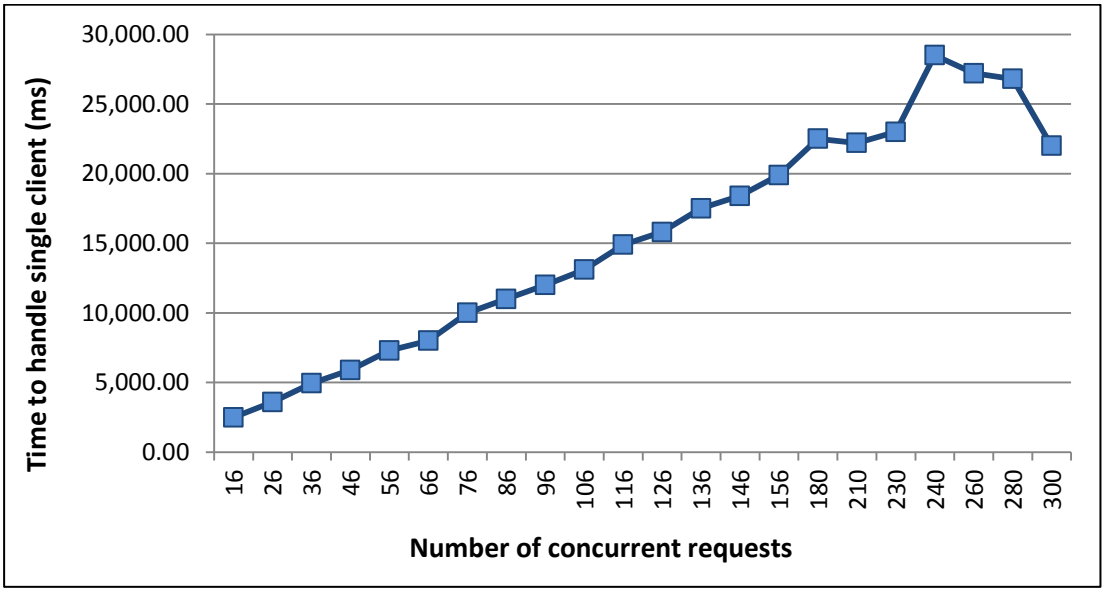

Figure 4.17: Average times taken to handle clients at the MSG.

The results from this analysis show that the MSG has reasonable levels of performance and can be scaled to handle large number of concurrent clients possible in the real-world scenarios. Figure 4.18 further clarifies this by showing that the number of transactions handled by the MSG per second, almost remains steady (in fact growing) even

under heavy load conditions. The MSG successfully handles 6-8 mobile web service invocations per second. The values can significantly grow, when the MSG is established on reasonable servers with high resource and performance capabilities.

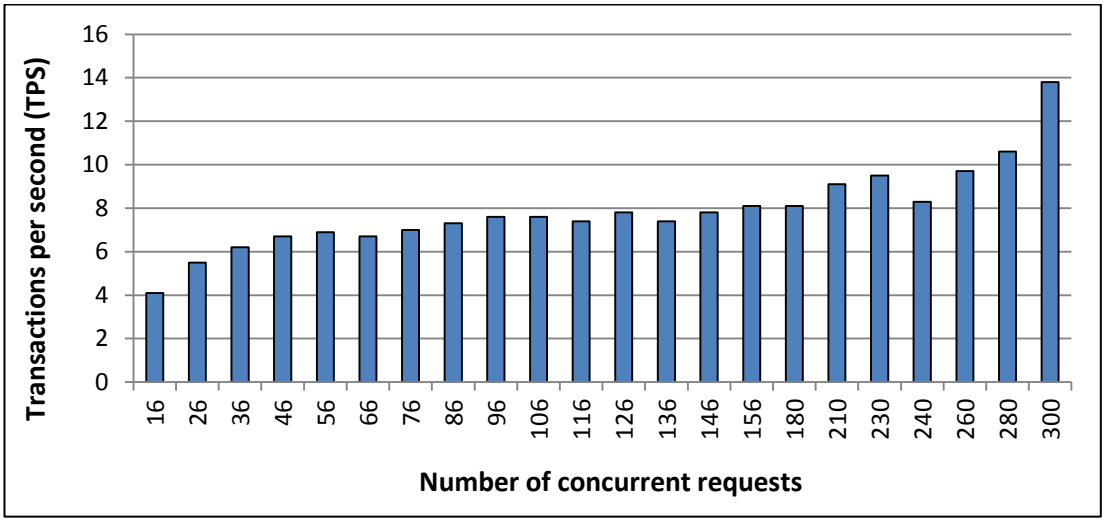

Figure 4.18: Number of transactions handled per second by the MSG.

4.8 Summary

This chapter has introduced the concept of providing mobile services in P2P networks from smartphones. The features, deployment scenario, and realization details of the P2P based web services framework are presented. Enterprise service bus is used to provide SOA features and enterprise level integration of mobile service providers, JXTA virtual P2P network, and existing web services infrastructure. The mobility and discovery issues of mobile web services are discussed; and an alternative for service discovery in P2P networks is suggested using the modules feature of JXTA. The evaluation of the framework clearly showed that MSG has reasonable levels of performance in handling large number of concurrent clients possible in mobile P2P networks. Clearly, this approach opens up numerous scopes for further research in domains like collaborative learning, content sharing, location based services, etc.

Chapter 5

Partitioning Service Execution on Resource Constrained Devices

Applications for mobile devices are getting more and more popular since general consumers have started to utilize the advanced capabilities of their mobile devices like smartphones. However, most of the available mobile devices have low resource capabilities. Compared to stationary nodes, mobile devices have limited capacity, slower processing speed, unreliable communication links and run mostly on battery power. Running complex services on a mobile device will consume most of its resources and may obstruct the device to perform its core functions (e.g. voice calls). It is not feasible and necessary for a single mobile device to execute complex processes independently. Services should be designed in a way to put less workload on the mobile device and delegate heavy-duty tasks to backend servers or to the cloud. This kind of delegation of tasks to servers is referred as *offloading*. Hosting services on mobile devices demands a flexible, light-weight and scalable execution environment, which can exploit resources from its surroundings as necessary.

This chapter proposes a framework for hosting services involving complex business processes on mobile devices. The mobile device acts as the integration point with the support of backend servers and remote web services. The framework provides support for executing functions locally which require the resources of the mobile device, and delegating the heavy-duty tasks to backend servers. The partitioning framework provides a high-level design about how to assign different tasks to be executed on mobile devices and backend nodes. Service partitioning

techniques are presented and different partitioning schemes are analyzed based on the order of the execution process. A comparison between the partitioned and un-partitioned execution engine has shown a significant improvement in the performance.

5.1 Mobile Service Provider Architecture

The thesis presents a light-weight framework for hosting services from mobile devices. The proposed service provider architecture is based on three main modules: *transport handler*, *execution engine*, and *deployed services* (Figure 5.1). Accommodating all parts of the modules on a single node is not feasible for resource constrained devices. Hence, the execution engine can be partitioned to execute the resource consuming partitions on a larger backend node. A backend node is a computing node that executes tasks on request and sends back the results to the provider. The functional details of the individual modules are presented below.

Transport Handler

One of the major advantages of SOAP is its independence of the underlying transport mechanism. This allows web service applications to select the appropriate transport mechanism according to its availability and the requirement of the quality of service. HTTP is used as the transport mechanism in our framework, because it is firewall friendly and most commonly used protocol for exchanging SOAP messages. The transport handler processes the HTTP requests and extracts the SOAP messages to forward them to the execution engine.

Execution Engine

This is the core module of the service provider architecture. This engine is responsible for parsing the incoming request message (SOAP message) and invokes a particular method of a web service class based on the

requested service. The execution engine uses Remote Procedure Call (RPC) as a binding technique between the messaging protocol and the web service implementation. A mapping between the URIs of the deployed services and their class instances is also maintained by the execution engine. This mapping is used to invoke the requested operation of a web service.

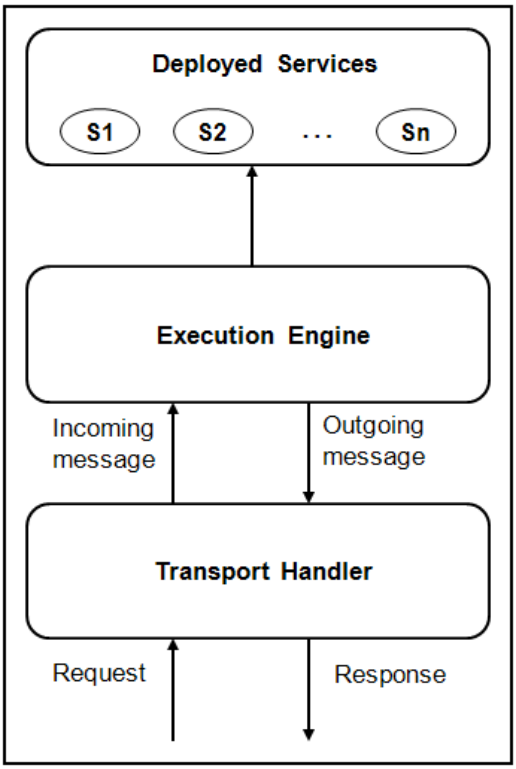

Figure 5.1: Mobile service provider architecture.

Deployed Services

This module contains a stack of the deployed services and the list of the services is maintained in an XML file. A new service can be deployed by simply putting its implementation package on the class path in the XML file. A sample of the deployment configuration of such a web service `searchArticle` is shown below (as presented before in Listing 2.2 in Chapter 2).

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

5.2 Partitioning Service Execution Engine

Partitioning execution engine is conceptually similar to separating an application into different parts that can be executed on different nodes. Static application partitioning is separating application executions at design time. Typical examples of this type of partitioning are client server applications [Tanenbaum and Steen, 2002]. Dynamic application partitioning is offloading some executions to a nearby surrogate node at run time. This type of partitioning is more challenging than static partitioning. More details on this concept can be found in [Hunt and Scott, 1999], [Jamwal and Iyer, 2005] and [Chandra et al., 2002].

The key objective of the partitioning framework is to minimize the execution load on the service provider mobile device. A service application can be executed on the mobile device and a few of its tasks can be offloaded to a powerful backend node. The control of the application remains with the service provider and only the tasks requiring more computing resources are offloaded to the backend node. The service designer can decide which set of tasks to execute on the mobile device and which on the backend node. The tasks that do not depend on resources of the mobile device can be executed on the backend node. Such execution engine partitioning also makes the system scalable in terms of the number of concurrent clients.

The partitioning technique is focused on the processing of SOAP messages in such a way that the execution on the resource constrained mobile device is minimized. At this stage, the target is to split the execution engine into two partitions: one for the mobile device that hosts the mobile execution engine, and the other for the backend node that

hosts the static execution engine. A service execution engine performs a series of tasks while invoking a web service (WS). These tasks may include conforming to WS specifications for address resolution, WS-policy and transaction management, verification of security, and so on. Conforming to each WS specification can be considered as one processing task for the execution engine. For example, verification of XML signature is one task and decrypting a SOAP message can be another task for the execution engine. Likewise, invoking the actual web service application can be considered as another separate task.

To explain the concept of a distributed execution engine, let us consider a WS invocation that requires clients to send encrypted request with identity of the requestor. The tasks that are performed by the WS execution engine are: verifying client identity (T1), decrypting incoming message (T2), verifying message integrity (T3), invoking other web services (T4), incorporating value-added services (T5), and signing response message with service provider's certificate (T6). Figure 5.2(a) shows the sequence of all these tasks performed by a single service execution engine. Figure 5.2(b) shows the execution of these tasks by a distributed service execution engine.

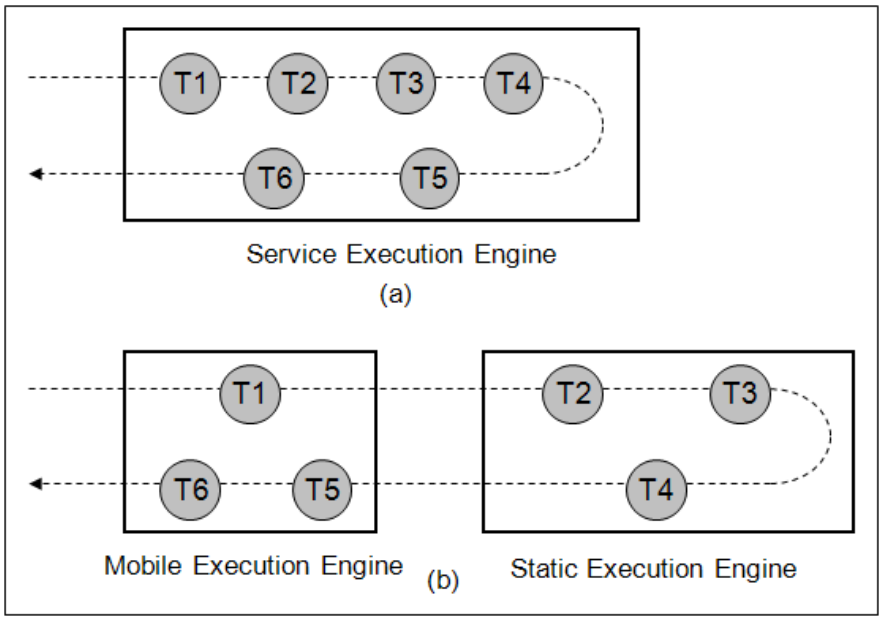

Figure 5.2: Partitioning service execution engine.

As mentioned earlier, the execution engine is divided into two parts: the *mobile execution engine* and the *static execution engine*. The mobile execution engine is designed to be deployed on the mobile device and is responsible to process the tasks that require local resources or require actions of the service provider. The static execution engine is to be deployed on a backend node and is responsible for handling tasks that demand more computing resources.

For example, if signing a response message requires a security certificate of the mobile service owner, then such tasks are better not to be placed on a backend node. Considering these facts, division of tasks between the mobile and the static execution engine is presented in Figure 5.2(b). In this partitioning scheme, the mobile execution engine verifies client identity (T1), incorporates value-added services (T5), and signs the response message (T6). The static execution engine decrypts incoming message (T2), verifies message integrity (T3), and invokes the required Web services (T4).

5.3 Service Partitioning Techniques

Two application partitioning techniques can be used for service partitioning in the proposed framework: *coarse grain partitioning* and *fine grain partitioning*. The first technique suggests partitioning of a service application at the package or class level and wraps these partitions into multiple child services, while the second technique suggests partitioning of a service at the method level.

The objective of partitioning the service application is to offload the complex components of a service from the mobile node with limited resources and execute them on a powerful backend node. The partition of the service that requires to be executed on the mobile device can be put in a separate service or a remotely accessible application (depending on the chosen technique) and deployed on the device. The rest of the components can execute on a powerful backend node that does not suffer from the

resource limitations of a mobile device. The details of these service partitioning techniques are discussed next.

5.3.1 Coarse Grain Partitioning

Coarse grain partitioning is primarily for services that are implemented using packages and classes. With coarse grain partitioning, a service is decomposed into child partitions such that each partition has one or more classes in it and can be executed on a separate node. Although the child partitions can be deployed as independent distributed applications, but to provide interoperability across different partitions the thesis proposes to wrap the child partitions into web services. Wrapping child partitions to web services is better for heterogeneous service execution environments since the web services can be deployed on any node irrespective of the platform it uses.

But processing of multiple (child) web services can add extra overheads due to additional XML/SOAP messaging. For complex systems, the response of multiple child web services can be collected and coordinated by using a workflow engine. The workflow engine is responsible for the arrangement and management of different web services to achieve the desired results. Workflow engines can be built using workflow languages that define the flow of execution of different web services or business processes. The commonly used workflow languages are Web Service Flow Language (WSFL) [Leymann, 2002] and Business Process Execution Language (BPEL) [WS-BPEL, 2002].

5.3.2 Fine Grain Partitioning

Fine grain partitioning is for decomposing a service application at a finer level; method or procedure level for example. This partitioning technique is useful for services that are implemented using native code of the devices. The complexity of performing the partitioning is increased as the granularity level becomes finer and finer.

In fine grain partitioning, it is possible to create partitions as remotely accessible applications by using distributed technologies such as, Remote Method Invocation (RMI) [Tanenbaum and Steen, 2002], Distributed Component Object Model (DCOM) [DCOM, 2007], and Common Object Request Broker Architecture (CORBA) [CORBA, 2007]. Since this technique of partitioning is mainly for implementations that are written in the device's native code, therefore the mobile services in the framework are not partitioned using this technique.

5.4 Service Partitioning Design Guidelines

The criteria for service partitioning include a number of factors; such as, the number of partitions, local resource requirement, frequency of communication between different components of a service, and resource limitations of the mobile device. Thus in the context of mobile services, knowledge on the type of the device (embedded device, handheld device, or smartphone) may also be important for partitioning. Depending on the complexity of the service, there can be multiple partitions of the application to improve performance. Based on these factors, three design guidelines are listed for partitioning services hosted on mobile devices.

- Services hosted on mobile devices will use some local resources of the device and there will be a set of local systems calls in the application. This guideline suggests quarantining part of the application in a separate partition that makes local system calls. The part of the application that does not use local resources can be put in the partition that is to be deployed on a remote node.

- For improving system performance, the communication between different modules of an execution engine should be minimized. The different partitions of a service must be engineered to reduce inter-component communication. This guideline suggests delegating part of the service to a remote node, only if this part does not use local

resources of the mobile device; and moving it to a remote node does not demand a large amount of data transfer between the remote node and the mobile device.

- For optimized performance, this guideline suggests using the *façade design pattern* for multiple services hosted on the same mobile node. The façade design pattern [Gamma et al., 1995] is a structural pattern that provides simpler interface for complex subsystems. One of the most important benefits of the façade pattern is that it reduces network round trips between remote systems. This is especially important for mobile services, since the wireless data transmission is slow. In the context of mobile web services, the façade design pattern can be used to integrate responses of multiple services hosted on the same mobile device. For example, let us assume a service provider offers two services A and B. A workflow engine may require both services A and B to satisfy a service request. The façade design pattern suggests that it would be economical to combine the two services A and B into one service AB, so that the workflow engine can make one call to get the responses of both.

5.5 Mobile Service Partitioning Schemes

Three different partitioning schemes can be devised for mobile web services. All schemes are assumed to have an asynchronous mode of communication between the service providers and clients. In brief these schemes are presented below.

5.5.1 Backend-node Based Scheme

A backend node executes a part of an application of the service provider and sends the results back to it. In this scheme, a client request is directly received by a resource constrained mobile device. The service

application is executed on the mobile device and few of its components are offloaded to a powerful backend node for improving performance. The interactions of different components in this scheme are shown in Figure 5.3(a). On receiving a service request, the mobile device runs some parts of the requested application locally and offloads rest of the execution to a backend node. As shown in Figure 5.3(a), it is the responsibility of the mobile service provider to collect results from the remote partitions, aggregate the results and send the final response back to the client.

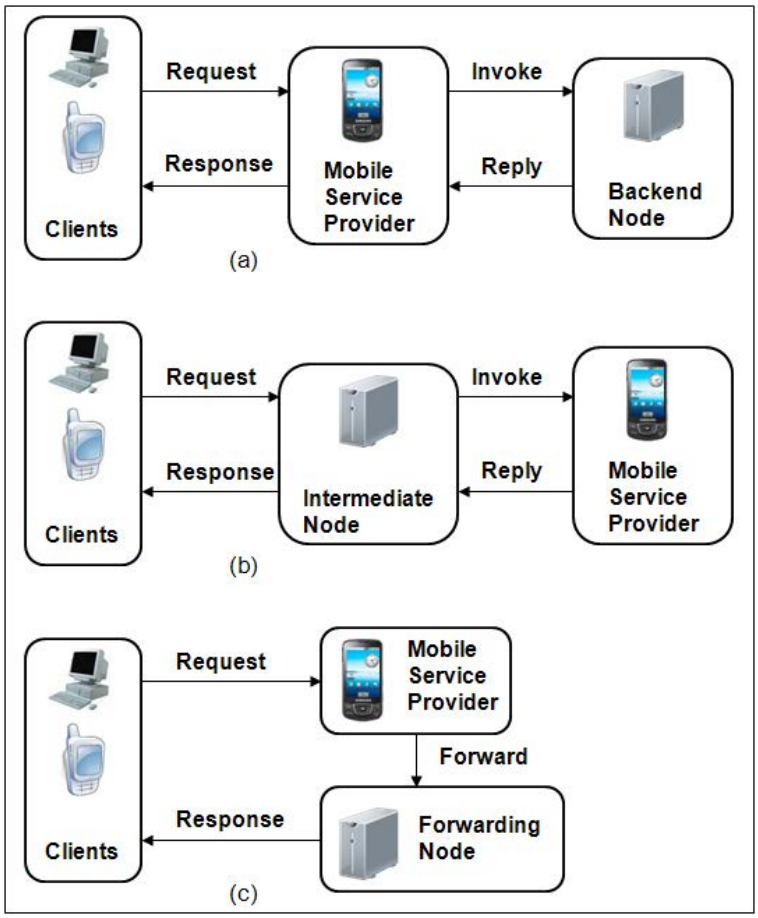

Figure 5.3: (a) Backend-node based scheme, (b) Intermediate-node based scheme, (c) Forwarding-node based scheme.

In this scheme, the mobile service provider itself has the control of coordinating the different partitions. It can either use a design-time or a run-time application partitioning strategy. With the run-time strategy, a number of run-time parameters are used to decide when to offload to a

backend node. An advantage of this scheme is that the mobile device controls the different partitions. This makes the services provisioned from the mobile hosts to the clients. From business point of view, this scheme opens up completely new opportunities for small businesses by being mobile web service providers. The framework proposed in this chapter is based on this scheme.

5.5.2 Intermediate-node Based Scheme

An intermediate node intercepts the client requests and processes partly before forwarding them to the service provider mobile device. In this scheme, an intermediate node works as a *surrogate* node. The surrogate is an interface of the published services to clients. To the external world, a service is assumed to be hosted on the intermediate node. A high level overview of this scheme is shown in Figure 5.3(b). The intermediate node receives the service request, executes some tasks locally and assigns other tasks to the mobile service provider. It is also the responsibility of the intermediate node to create a final response and send it to the client.

There are a number of advantages of using this scheme. The intermediate node can facilitate access to the existing application as a service. The use of an intermediate node allows the service to be implemented using the device's native code while providing an interface to the service consumer. For example, this approach may be useful to collect data from sensor devices and provide the sensor functionality as a web service by using an appropriate wrapper without deploying any service execution environment on the resource constrained sensor device. In the other case, if the service is deployed on the mobile device, the intermediate node can be useful for session management and supporting other extended web service specifications for security and transactions.

In this scheme, the use of an intermediate node as a service proxy ensures availability of the service at all time. The intermediate node is solely responsible for delegating service requests to or collecting results

from the actual service hosted on a mobile device. This scheme is most suitable for design time application partitioning strategies. In this scheme, the mobile service provider has less control on the accessed application. The services that require extensive access to the resources of the mobile device or frequent involvement of device's owner are not suitable for this scheme.

5.5.3 Forwarding-node Based Scheme

The objective of this scheme is to alleviate the drawback of collecting results from backend node and aggregating locally on the mobile service provider. Some tasks can be executed on the mobile device first and then rest of the execution can be moved to a forwarding node. In this scheme, responsibility of sending the final response to the client is delegated to the forwarding node. A high level overview of the interactions of this scheme is presented in Figure 5.3(c).

This scheme is suitable for the applications which require the partition on the mobile device to be executed first. This scheme requires the forwarding node to communicate with clients directly, which may not be feasible in many scenarios (e.g. peer to peer services). Besides, this scheme allows less control to the mobile service provider and introduces additional binding issues between the forwarding node and the client.

5.6 Framework for Partitioned Mobile Services

The partitioning framework makes use of both *design-time* and *run-time* partitioning for the division of tasks between the execution engines (*mobile execution engine* and *static execution engine*). Design-time partitioning is used to separate the tasks which require the resources of the mobile device or mobile owner's input. Service designer can pre-partition the tasks to be executed on the mobile and static execution engines. Run-time partitioning is used to make optimal partitioning

decision for heavy-weight services, where the services may require dynamic allocation of tasks to utilize available resources based on the costs of service components being executed. The design-time partitioning is defined at the time of deploying a service through a predefined partitioning policy. The tasks that can be offloaded to a backend server depending on the availability of resources are identified in the policy statement. The policy uses an XML schema that defines the XML elements for each task execution. A sample partitioning policy is shown in Listing 5.1.

Listing 5.1: Sample partitioning policy.

```
<partition>
    <mobile><IDENTITY required="true"/></mobile>
    <static><WEBSERVICE class-name="location"/></static>
    <mobile><SIGNATURE required="true"/></mobile>
</partition>
```

In this partition, verification of a client identity and signing a response message are assigned to the mobile execution engine. The task of invoking a web service is assigned to the static execution engine. The sequence of tasks performed by the execution engines is the same as specified in the partitioning policy. In this example, there are two `<mobile>...</mobile>` XML blocks. The `<mobile>...</mobile>` block that comes after the `<static>...</static>` block, contains a task to be done on the mobile device after the completion of task on the static execution engine.

The overall framework based on the backend-node based scheme is depicted in Figure 5.4 and the details of the components are discussed in the following subsections. Light-weight and open-source packages kSOAP [kSOAP2, 2007] and kXML [kXML2, 2007] are used for implementing the partitioned execution engines. kXML is an XML parser based on pull parsing and is an implementation of XMLPULL parser API [XMLPULL, 2007]. kSOAP is used for processing SOAP messages.

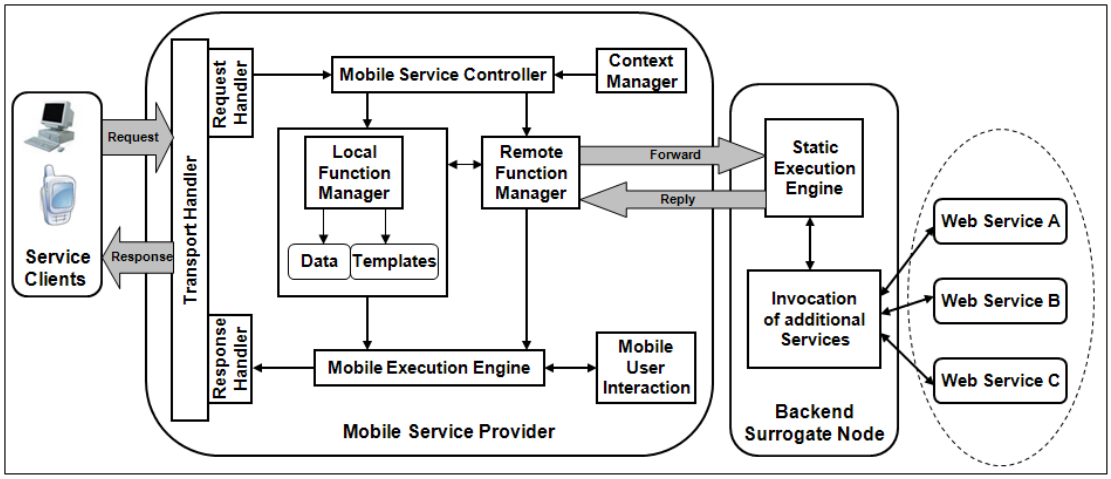

Figure 5.4: A partitioned mobile service provider framework.

5.6.1 Mobile Service Controller

A client request (SOAP message) is first received by the *Transport Handler* in any compatible transport mechanism and the *Request Handler* forwards the message to the *Mobile Service Controller*. The controller decides whether or not to partition the execution, and which tasks are to be offloaded. The partitioning policy is configurable because the partitioning depends both on the nature of a service and the resource capabilities of the mobile device. Depending on the context of available resources, the controller decides whether to allocate more tasks to offload, or to execute all tasks locally.

5.6.2 Function Managers

After splitting the tasks, the controller passes the message blocks to *Local Function Manager* and *Remote Function Manager* accordingly. The local function manager further separates the *data* and *templates* from the message blocks to reduce the processing load on the mobile device. The data and templates are passed to the *Mobile Execution Engine* for local execution. The remote function manager forwards message blocks to the *Static Execution Engine* on a backend node. The static engine invokes

additional web services as necessary to process the forwarded messages and replies back the results. These results are then passed to the mobile execution engine to be aggregated with the local execution results. The both execution engines use similar implementation, but they differ in terms of tasks they perform at run-time.

5.6.3 Mobile Execution Engine

This component is responsible for executing the local execution tasks within the mobile device. The user or administrator of the mobile device can administrate the execution engine if necessary. For some request processing, it might be necessary to get the administrator input to proceed or to deliver responses. After getting the data and templates separated from the local function manager, the mobile execution engine performs the tasks costing less computing power. Then it aggregates its results with the results obtained from the backend node. At the last step, the transport handler sends the response message (SOAP message) to the original client from the mobile host.

5.6.4 Context Manager

The *Context Manager* monitors the available resources on the mobile host and helps the controller to allocate tasks accordingly to the execution engines. For example, when the available memory or processing power is low, the controller can allocate more tasks to the backend node if the available bandwidth is sufficient. At the run-time, if available resources become low due to any principal operations (e.g. voice calls) of the mobile device, the controller can offload more tasks. The context manager proposes to partition the tasks by calculating a *context suitability* using the following equation,

$$Context\ suitability = \sum \frac{AR_i - RR_i}{AR_i} \times \omega_i \qquad (5.1)$$

where, AR_i is the available context value of the ith resource, RR_i is the required context value of the ith resource, and ω_i is the weight of the ith resource. If the calculated context suitability value becomes negative, the mobile service controller partitions the tasks between the local function manager and the remote function manager to offload. As shown in Equation (5.1), the context manager has the least context information required to execute a service and proposes to partition based on it.

For instance, let us consider the case of a mobile host with available 500MB memory and 400MHz CPU. For simplicity, we suppose that the weight values ω_1 and ω_2 are 0.5 and 0.5 respectively, and the available bandwidth is sufficient to offload. When a service executes locally and the required resources are 100MB memory and 300MHz CPU, the context suitability will be calculated as,

$$Context\ suitability = \frac{500-100}{500} \times 0.5 + \frac{400-300}{400} \times 0.5$$
$$= 0.4 + 0.125 = 0.525$$

In this case, the context manager will propose to execute the service locally within the mobile host. Now, let us suppose that the mobile host has 300MB memory and 100MHz CPU available. With the same required resources for executing the service, the context suitability in this case is calculated as,

$$Context\ suitability = \frac{300-100}{300} \times 0.5 + \frac{100-300}{100} \times 0.5$$
$$= 0.333 + (-1) = -0.667$$

In this case, as the context suitability is negative, the context manager will propose the mobile service controller to partition the execution and offload the resource intensive tasks to a remote backend node.

5.7 Performance Model

The performance of the system is analyzed by measuring the end to end response time. *Response Time* is the total time taken to invoke a service. It is measured by taking the difference between the times when the service request was sent to the server and when the service response is received by the client.

Partitioning mobile service execution across multiple nodes adds overheads of coordination and communication among the different partitions. Therefore, the performance benefits that are expected from partitioning a mobile service should be enough to outstrip these overheads. Web service hosting requires a service execution environment on the mobile device. This execution environment handles receiving a service request, de-serializing the request message, invoking the requested service, serializing the results into a response message, and sending the response to the client. If we denote the CPU time required by the execution environment by T_{en} , and the CPU time required by the service application itself by T_{app} , then the overall response time T_{mws} is the sum of T_{en} and T_{app} (network delays are not considered).

$$T_{mws} \cong T_{en} + T_{app} \tag{5.2}$$

T_{en} includes the CPU time spent on sending and receiving SOAP messages and executing WS protocols. If the execution engine performs total n number of tasks, then total CPU time required by the service application is:

$$T_{app} = \sum_{i=1}^{n} T_{ti} \tag{5.3}$$

Equation (5.3) only considers CPU time while running all the tasks sequentially on the mobile device. Let us assume that we divide the tasks into two sets to be executed in two partitions. One set of tasks is executed

on the mobile device and the other set of tasks is offloaded to a static remote node. Therefore, equation (5.3) now can be written as,

$$T_{app} = \sum_{i=1}^{m} T_{ti} + \sum_{j=m+1}^{n} T'_{tj} \quad (5.4)$$

where, $i = 1, 2, \ldots, m$ are the tasks executed locally on the mobile device and $j = m+1,\ m+2, \ldots n$ are the tasks executed on a remote node.

There are two additional overheads that will occur if the service application is partitioned across multiple nodes. One overhead arises from the coordination of different partitions, and the other overhead arises from the transfer of data between the partitions. If we denote these two overheads by Δ_{cd} and Δ_{tn} respectively, then Equation (5.4) becomes,

$$T_{app} = \sum_{i=1}^{m} T_{ti} + \sum_{j=m+1}^{n} T'_{tj} + \Delta_{cd} + \Delta_{tn} \quad (5.5)$$

Therefore, the overall response time of a mobile service invocation can be obtained by combining Equations (5.2) and (5.5),

$$T_{mws} \cong T_{en} + \sum_{i=1}^{m} T_{ti} + \sum_{j=m+1}^{n} T'_{tj} + \Delta_{cd} + \Delta_{tn} \quad (5.6)$$

Equation (5.6) gives an estimate of the time required to invoke a mobile hosted service that is partitioned across multiple nodes. In this estimate, the executions on the mobile device and on the remote node are assumed to be not concurrent.

5.8 Performance Evaluation

In this section, the performance of the partitioned execution engine is evaluated by measuring the end to end response time to invoke a service. The prototype implementations are tested and the impact of service partitioning techniques on corresponding response times are presented.

5.8.1 Effect of Partitioning on Performance

To investigate performance of the partitioned execution engine for mobile hosts, a test prototype is developed to compare the end to end response time for different number of concurrent clients. For this investigation, the backend-node based scheme is chosen and a sample service `searchArticle` is deployed on a smartphone. The mobile hosted service `searchArticle` performs the operation `searchArticleByDate` and provides a list of articles written on a particular date by a journalist. The service extracts the input parameter `'date'` from the incoming message and returns the list of articles as a response. The average response time for each input value is measured by sending 10 requests to the service provider and then taking the average.

Test Setup

The service clients are run on a Toshiba Satellite A100 laptop equipped with a RAM of 1GB and 1.83GHz Intel processor. The backend node is a desktop computer equipped with an Intel single core processor of 3.0GHz CPU speed and a RAM of 1GB. Both machines operate with Windows XP professional operating system. The service provider is deployed on a Nokia E63 smartphone equipped with 369MHz ARM11 processor and a RAM memory of 128MB, running on Symbian OS v9.2 operating system. The Java ME technology available on the device is, JSR 172 J2ME Web Services Specification. The light weight execution engine is based on J2ME and deployed on the smartphone. The client machine, the smartphone and the backend node are equipped with wireless interfaces using IEEE 802.11 b/g standard, and they communicate using a wireless local area network.

Test Results

In this test, the execution engine is partitioned at design time. The task of client's authentication checking is partitioned to be offloaded at the backend node. After verifying the signature, the backend node sends the message to the mobile service provider. The service provider does all the tasks of service invocation for the authorized clients and sends back the results to them. In the second test, the same client requests were executed without partitioning the execution. The client's authentication checking is done within the mobile device. Each client operates cyclically and sends one request at a time. The system is stressed by increasing the number of concurrent clients. Table 5.1 shows the average response times obtained from these observations. Figure 5.5 is obtained by plotting the values from Table 5.1.

Table 5.1: Comparison of response times with and without partitioning.

Concurrent clients	Average response time when partitioned (ms)	Average response time when not partitioned (ms)
1	1186.82	1766.53
3	2466.35	4232.21
6	3742.41	7542.37
9	5013.26	13708.82
12	8651.44	18857.34

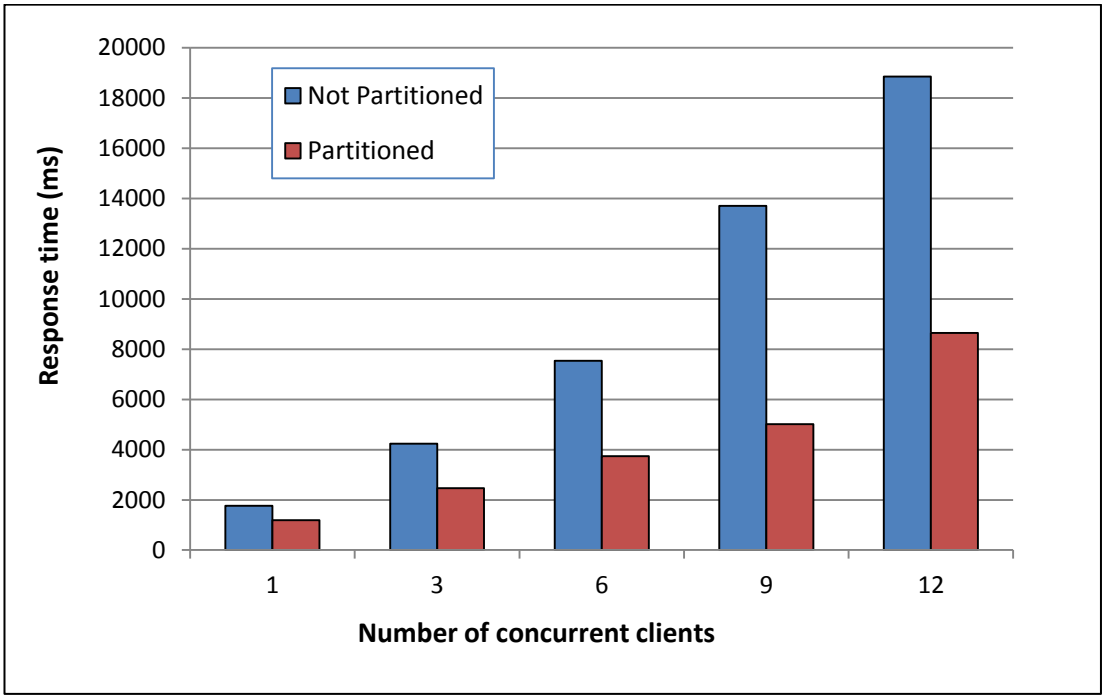

Figure 5.5: Effect of execution engine partitioning on performance.

As we can see for both of the cases, demand for resources increases with an increase of concurrent clients and the mean response time increases. Figure 5 shows that the mean response time for the partitioned execution engine is significantly lower than the mean response time with no partitioning. The result of distributing the execution engine from the mobile host is promising. For a single client the improvement in response time is 32% and for 12 concurrent clients the improvement is 54%.

5.8.2 Effect of Partitioning as Child Services

For this experimental investigation a sample *Location Service* is deployed on a smartphone and the intermediate-node based scheme is chosen to evaluate the effect of partitioning as child services. The functionality of the Location service is to provide the exact location of the device on which the service is deployed. The service provides the address of the location, its elevation, time zone and the population of the area. As the first step, the service fetches the actual GPS coordinates from a GPS receiver. For

this experiment, we emulate the step of getting GPS coordinates by fetching a random set of coordinates from a local file containing more than a thousand locations. In the next step, the service queries a database of locations to find the details of a location closest to the coordinates. The location database is downloaded from a well-known geographical database *GeoNames* [GeoNames, 2009]. In the last step, the location information is serialized as a response message and sent back to the requester.

Test Setup

In the experiments, a Nokia E71 smartphone was used, equipped with 369MHz ARM11 processor and a RAM memory of 128MB, running on Symbian OS v9.2 operating system. It has a built-in GPS receiver with preloaded maps (i.e. Nokia Maps) for location identification. The Java ME technology available on the device is, JSR 172 J2ME Web Services Specification. Services are deployed on the device using a J2ME (J9) runtime environment. The service surrogate (i.e. intermediate node) is deployed on a desktop node that is equipped with a 3.0GHz Intel single core processor and a memory of 1GB. The service clients are running on a Toshiba Satellite A100 laptop equipped with a RAM memory of 1GB and an Intel single core processor of 1.83GHz speed. The inter-communication between the intermediate node (the service surrogate) and the smartphone is based on a wireless local area network. The intermediate node and the smartphone are equipped with wireless adaptors compatible with IEEE 802.11 b/g standard.

Test Results

Location service involves system calls to local resources and has a component for querying a large database that requires a large amount of processing capacity and physical space for storage. For simplicity, the Location service was partitioned into two child partitions. These child

partitions are deployed both as child services (coarse grain partitioning) and as remote distributed objects (fine grain partitioning). SOAP is used for intercommunication in coarse grain partitioning, whereas Remote Method Invocation (RMI) is used for fine grain partitioning.

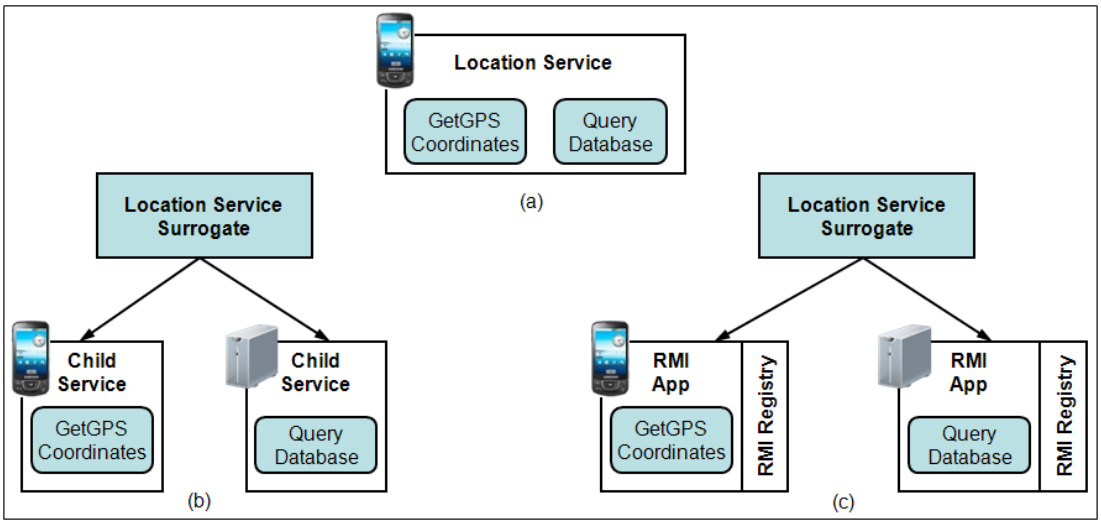

Figure 5.6: Location service (a) un-partitioned, (b) partitioned into child services, (c) partitioned into remotely accessible objects.

Figure 5.6(a) shows the Location service as a whole (un-partitioned), deployed on the smartphone in one of the experiments. Later, the Location service is partitioned into two child services. The `getGPSCoordinates` uses local resources, so this task is kept in one partition and installed on the mobile device. The `queryDatabase` requires accessing a large database with thousands of records containing locations, and does not use any local resources of the mobile device. Therefore, the `queryDatabase` partition along with the database was moved to the backend node. Figure 5.6(b) shows the result of a coarse grain partitioning of the Location service, in which the two partitions are deployed as two child services. Figure 5.6(c) shows the result of a fine grain partitioning of the Location service, in which the two partitions are deployed as two remotely accessible distributed objects. The advantage of using distributed objects is that they can use an optimized transport mechanism for exchanging data.

The performance of using these partitioning techniques on the Location service is observed by sending 100 requests (one after the other) from multiple concurrent clients. The system is tested using one, five, eight, and ten concurrent clients. The results are summarized in Table 5.2 and Figure 5.7.

Table 5.2: Response times with different partitioning techniques.

Concurrent clients	Not partitioned (ms)	Partitioned as child services (ms)	Partitioned as distributed objects (ms)
1	43580.12	3623.15	4675.17
5	208470.55	11660.62	17953.10
8	336051.34	18331.90	28379.47
10	427626.85	23851.35	36110.25

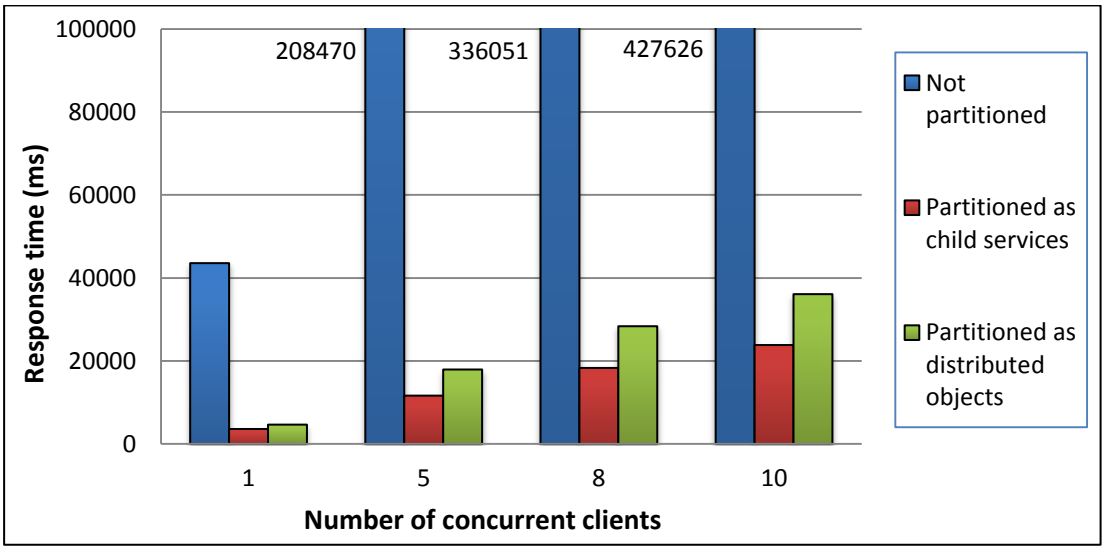

Figure 5.7: Effect of execution engine partitioning as child services.

The results show that the Location service partitioned into two partitions performs much better in comparison to the un-partitioned Location service which is deployed as one entity on the mobile device. The response time achieved with the un-partitioned Location service is

inordinately large and is clearly unacceptable. Among the rest, Location service deployed as two child services (coarse grain partitioning) performs better than the service that is deployed as two distributed objects (fine grain partitioning).

Though the system with distributed objects does not have additional overhead of SOAP/XML processing, but still it performs inferior due to the RMI framework and its implementation available for J2ME. At first, the RMI client uses a registry to get a reference of the remote object and then uses the reference to invoke a particular method on the remote object. Therefore, partitioning a service into multiple RMI based remote objects results in more overhead due to the increased number of messages between the service surrogate and the nodes hosting remote objects. Another reason of the inferior performance of RMI based partitioning is due to its thread creating strategy. A new thread is created for each new request. The RMI runtime keeps the newly created thread for a certain amount of time period to serve another request. If no new request arrives during this period, then the thread is destroyed. This thread creating strategy adds further overheads to service delivery.

Hence, partitioned (coarse grained) service deployed as child services performs better than the partitioned (fine grained) service that is deployed as distributed objects.

5.8.3 Effect of Using Façade Design Pattern on Service Partitioning

In this investigation, the same *Location Service* is deployed along with a *Schedule Service* on a smartphone to evaluate the effect of using the *Façade Design Pattern* on service partitioning. The Schedule service is a simple service that can be used to fetch the schedule detail of an especially skilled person (e.g. doctor, nurse) from his/her device. It is assumed that the person is using a scheduling application for managing his/her appointments and is exposing the schedule as a service. The data

of all private meetings and appointments are assumed to be private and not available to the service to expose.

Test Setup

In this experiment, the same setup is used as the previous experiment in subsection 5.8.2. The Schedule service is deployed along with the Location service on the same mobile device (i.e. Nokia E71 smartphone). The significance of the façade design pattern is investigated for aggregating light weight services if they are deployed on the same mobile node and are required for *service composition* [Zeng et al., 2003]. In this experiment, it is assumed that a service composition entity requires the Location service and the Schedule service to provide the location of a person and his/her current schedule.

Test Results

The steps of partitioning the services and applying the façade design strategy on the partitions are depicted in Figure 5.8. Figure 5.8(a) shows the invocation of the Location service and the Schedule service independently to accomplish a service request for the location of a person and his/her current schedule. Figure 5.8(b) shows that the Location service is partitioned into two child services and the `queryDatabase` child service is deployed on a stationary node. In this case, the composite service surrogate invokes three services to accomplish the goal, i.e. `getGPSCoordinates`, `queryDatabase`, and Schedule service. In Figure 5.8(c), the two services hosted on the same mobile device are combined using the façade pattern and now the composite service surrogate invokes a single service to get the GPS coordinates and the schedule.

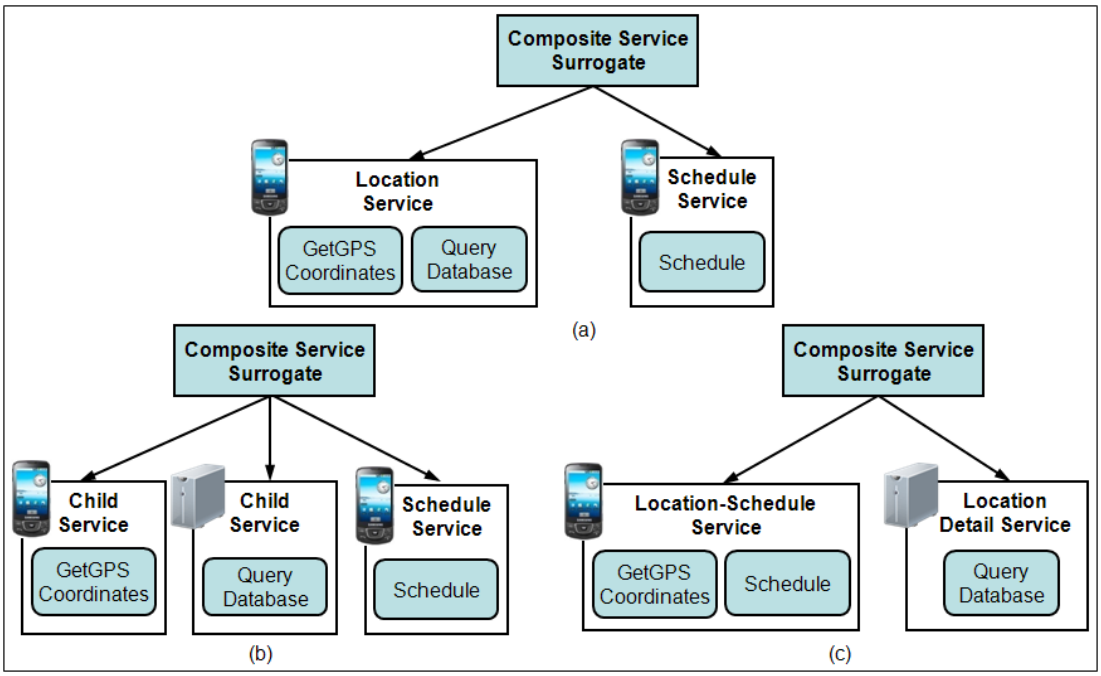

Figure 5.8: Partitioning approach with aggregating two services using façade design pattern.

The invoked composite service is based on the Location service and the Schedule service together. The partitioned composite services with and without using the façade pattern, are then compared by observing the response times of the corresponding composite services. The observations are taken with one, five, and ten concurrent clients and the results are summarized in Table 5.3 and Figure 5.9.

Table 5.3: Response times with and without façade design strategy.

Concurrent clients	Partitioned without façade design strategy (ms)	Partitioned with façade design strategy (ms)
1	5015.32	4660.12
5	17080.15	12290.07
10	32920.74	23695.22

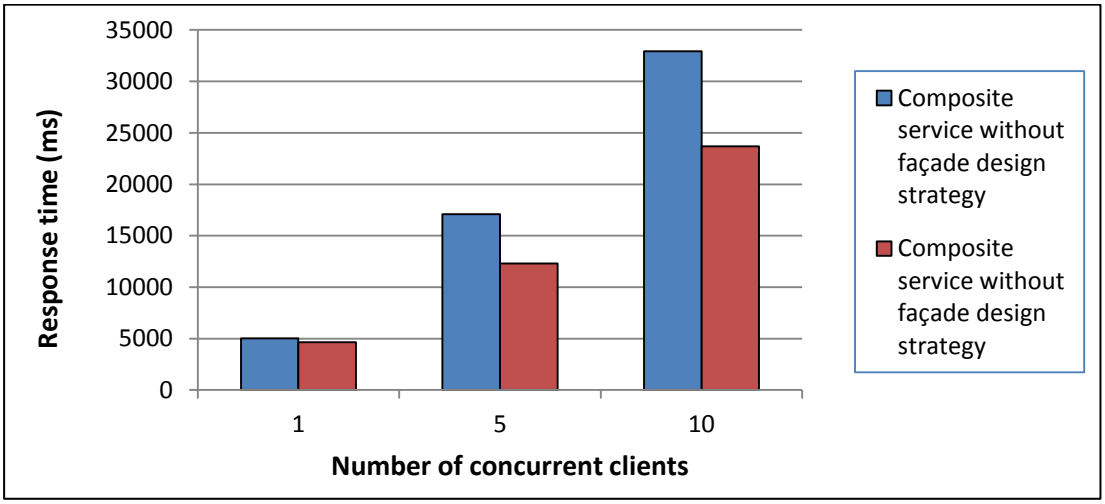

Figure 5.9: Effect of using façade design pattern on partitioning.

As the number of concurrent clients increases, the performance of the system using façade design strategy improves. The response time of invoking the composite service using the façade pattern is observed to be reduced by 7% for one client and by 28% for 5 and 10 concurrent clients, compared to the response time of invoking the composite service without using the façade pattern.

5.9 Summary

This chapter introduced a framework for provisioning web services from resource constrained mobile devices, based on partitioning the application execution of the services. The architecture of the service execution engine has been provided with the details of each component. The proposed architecture facilitates the use of application partitioning techniques to offload parts of the service application to a backend surrogate node. The framework is tested using sample prototypes and some sample web services to observe the effect of applying different partitioning techniques. As we can perceive from the observations, partitioning the computation intensive tasks has led to a significant improvement in response time of the services hosted on mobile devices.

Chapter 6

Approach for Efficient Service Partitioning

Recent years have seen a significant increase in research and development of mobile and wireless networks, including UMTS (Universal Mobile Telecommunications System), mobile ad hoc networks, and sensor networks. The potentials of these network systems lie in their ability to provide users with cost-effective services that have the potential to run anywhere, anytime and on any device without (or with little) user attention. Services with these features are termed as *pervasive services* [Satyanarayanan, 2001]. Hosting pervasive services can bring great convenience for the owners of mobile devices to administrate and manage their business services anytime and anywhere. By hosting services on mobile devices, business users will benefit from collaboration and data sharing, and personal users will benefit from social networking, sharing multimedia contents, incorporating address books, or exchanging calendar schedules etc.

In comparison to desktop nodes, mobile devices usually come with limited memory capacity, slower processing speed, and limited communication bandwidth. In this context, a mobile device can be a smartphone, a PDA, or any other portable computing device. Hosting complex and heavy-weight services on mobile devices demand mechanisms for employing more computing resources to support the mobile devices. Mobile devices can host such type of services by partitioning the tasks and delegating the resource intensive tasks to their backend servers. Offloading is the technique to achieve task delegation to remote servers for employing additional resources on demand for mobile

devices. This chapter presents an efficient approach for service partitioning by considering a combination of the costs of memory, CPU, and bandwidth resources on mobile devices.

6.1 Offloading Pervasive Services

Offloading is not a completely new concept. It has been utilized in the form of proxies and surrogates for many years. Offloading has been used for load balancing in distributed systems [Satyanarayanan, 2001]. However, exploiting offloading mechanisms in the domain of service provisioning from resource constrained mobile devices has not yet explored enough. Some research work earlier proposed different offloading mechanisms for resource constrained devices [Chen et al., 2004; Flinn et al., 2002; Li et al., 2001; Balan et al., 2003]. However, they are limited in one way or another, for not considering the dynamic condition of available resources on the devices. Since pervasive services tend to be quite complex and dynamic [Satyanarayanan, 2001], the offloading system should consider multiple types of resource constraints at the same time. It is still lacking of a partitioning solution that considers memory, CPU power, and bandwidth simultaneously. Focusing on only one of them renders the solution to be unfeasible in practice.

The proposed partitioning framework uses an approach which considers the dynamic conditions of the resources, to partition the tasks between a mobile device and its backend nodes. The offloading approach significantly increases the efficiency of services hosted on mobile devices, by considering a combination of the cost values of memory, CPU, and bandwidth resources. The offloading approach considers both the interaction properties and the resource consumption while performing partitioning and offloading. The adaptability and efficiency of the approach lies in its efficient service partitioning algorithm and feasible offloading mechanisms.

6.2 Cost-based Dynamic Offloading

The costs of service components change during service execution on the run-time in mobile environments. Therefore, for better performance and optimization, cost-based dynamic partitioning can guarantee a reliable and robust service provisioning. The offloading approach considers the costs of resource consumption depending on the nature of the provided services and clients requests.

6.2.1 Weighting the Costs of Service Components

In general, there are two approaches to weight the costs of service components: *online-profiling* and *offline-profiling*. The latter uses a pre-defined profile to describe the resource consumption of each service component. An offline-profile needs to be defined during the design period with the knowledge of the execution code. The practical advantage of offline-profiling is the simplicity in its implementation. If an offloading is needed before the start of service application execution, the offline-profiling has to be used. The drawback of offline-profiling is that it cannot reflect dynamic resource changes during application execution. On the other hand, the online-profiling approach generates a multi-cost graph dynamically by monitoring the runtime execution environment. It does not need knowledge of the execution code. The demerit of online-profiling is that it introduces extra overhead due to its real-time and dynamic nature. The proposed offloading approach supports both online-profiling and offline-profiling, depending on the services provided and the runtime execution environment.

6.2.2 Monitoring Dynamic Resource Utilization

To make dynamic offloading decisions, there are two viable approaches for monitoring the runtime resource utilization of the environment: *periodical* and *event-driven* resource snapshots. The former takes

resource utilization snapshots periodically with a time interval. The interval can be smaller for more precise evaluation (e.g. a few milliseconds). However, frequent snapshots increases overhead. Nevertheless, the event-driven approach takes snapshots only when resource utilization changes (e.g. the events of JVM's memory allocation and release). The offloading framework takes the event-driven snapshots and complements them with a periodical triggering of longer interval snapshots (e.g. a few seconds). The following approaches are used to obtain the cost values of service components.

- *Memory usage:* The memory usage of a class changes during its execution. The context manager obtains the memory usage of each class by monitoring the JVM heap.

- *CPU utilization:* It is difficult to measure CPU utilization. For a given time slot if only one class is running, the CPU utilization can go to 100%. Whereas, if two classes are running and the CPU utilization is 100%, it does not imply that the classes' CPU utilizations are 50% each. Therefore, the percentage of occupied CPU is not a feasible way to represent the CPU utilization. A simple approach can be used by calculating the cumulated CPU occupying time of a class during the snapshot intervals, as the CPU processing cost of the class.

- *Bandwidth usage:* Network bandwidth usage comes from the exchanged data between the mobile device and the remote surrogate hosts (e.g. downloading or uploading). The context manager monitors remote data accesses to obtain the bandwidth usage of each class.

6.2.3 Issues for Dynamic Offloading Decision

As mentioned earlier in section 5.6.4, the context manager calculates a context suitability value and helps the mobile service controller to

allocate tasks accordingly. If over run time, available resources become low of the mobile device, the controller can offload more tasks to the backend nodes. Let, m, c, and b represent the values of context suitability and ω_1, ω_2, and ω_3 represent the importance factors for memory, CPU, and bandwidth respectively. Therefore, according to Equation (5.1) the value of context suitability for memory is,

$$m = \frac{M_{avl} - M_{req}}{M_{avl}} \times \omega_1 \tag{6.1}$$

which implies,

if $m > 0$ the tasks should be executed locally; and

if $m < 0$ the heavy-weight tasks should be offloaded.

Likewise, the value of context suitability for CPU is,

$$c = \frac{C_{avl} - C_{req}}{C_{avl}} \times \omega_2 \tag{6.2}$$

which implies,

if $c > 0$ the tasks should be executed locally; and

if $c < 0$ the heavy-weight tasks should be offloaded.

In the case of bandwidth, the effect is different on offloading decisions. More available bandwidth implies that there is more opportunity to optimize the performance by offloading more tasks. Thus, the value of context suitability for bandwidth is,

$$b = \frac{B_{avl} - B_{req}}{B_{req}} \times \omega_3 \tag{6.3}$$

which implies,

if $b > 0$ the heavy-weight tasks should be offloaded; and

if $b < 0$ the tasks should be executed locally.

Considering these issues about memory, CPU, and bandwidth, we can make the following implications for making offloading decisions:

(i). If $\left.\begin{matrix} m > 0 \\ c > 0 \\ b < 0 \end{matrix}\right\}$, then all tasks will be executed locally.

When the available memory and CPU on the mobile host are more than required to execute the tasks locally and the available bandwidth is insufficient to offload them, in such case, all the tasks will be executed locally on the mobile host.

(ii). If $\left.\begin{matrix} m < 0 \\ c < 0 \\ b > 0 \end{matrix}\right\}$, then tasks will be offloaded as many possible.

When the available memory and CPU on the mobile host are less than required to execute the tasks locally and the available bandwidth is sufficient to offload them, in such case, maximum number of tasks that do not require the resources of the mobile host, will be offloaded. The tasks that are not possible to be offloaded will be executed locally on the mobile host.

(iii). If $\left.\begin{matrix} m < 0 \\ c > 0 \\ b > 0 \end{matrix}\right\}$, then memory-intensive tasks will be offloaded.

When the available memory on the mobile host is insufficient to execute the tasks locally and the available CPU is more than required and available bandwidth is sufficient to offload them, in such case, the memory intensive heavy-weight tasks that do not require the resources of the mobile host, will be offloaded. The light-weight tasks that do not require a large portion of memory will be executed locally on the mobile host.

(iv). If $\left.\begin{matrix} m > 0 \\ c < 0 \\ b > 0 \end{matrix}\right\}$, then CPU-intensive tasks will be offloaded.

When the available CPU on the mobile host is insufficient to execute the tasks locally and the available memory is more than required and available bandwidth is sufficient to offload them, in such case, the CPU intensive heavy-weight tasks that do not require the resources of the mobile host, will be offloaded. The light-weight tasks that do not require a large portion of CPU will be executed locally on the mobile host.

(v). If $\left.\begin{matrix} m < 0 \\ c < 0 \\ b < 0 \end{matrix}\right\}$, then tasks execution is delayed.

When the available memory and CPU on the mobile host are insufficient to execute the tasks locally and the available bandwidth is insufficient to offload them, in such case, task execution is delayed due to the lack of available resources. The delayed tasks will be executed locally when sufficient memory and CPU become available, or will be offloaded for execution when sufficient bandwidth becomes available to offload.

(vi). If $\left.\begin{matrix} m < 0 \\ c > 0 \\ b < 0 \end{matrix}\right\}$, then tasks execution is delayed.

When the available memory on the mobile host is insufficient to execute the tasks locally and the available CPU is more than required and available bandwidth is insufficient to offload them, in such case, task execution is delayed as it is neither possible to execute locally nor possible to offload for the lack of available memory and bandwidth. The delayed tasks will be executed locally when sufficient memory becomes available, or will be offloaded for execution when sufficient bandwidth becomes available to offload.

(vii). If $\left.\begin{matrix} m > 0 \\ c < 0 \\ b < 0 \end{matrix}\right\}$, then tasks execution is delayed.

When the available CPU on the mobile host is insufficient to execute the tasks locally and the available memory is more than required and available bandwidth is insufficient to offload them, in such case, task execution is delayed as it is neither possible to execute locally nor possible to offload for the lack of available CPU and bandwidth. The delayed tasks will be executed locally when sufficient CPU becomes available, or will be offloaded for execution when sufficient bandwidth becomes available to offload.

(viii). If $\left.\begin{matrix} m > 0 \\ c > 0 \\ b > 0 \end{matrix}\right\}$, then heavy-weight tasks can be offloaded.

When the memory, CPU, and bandwidth resources on the mobile host are all available more than required, tasks can either be executed locally or can be offloaded for execution. In this case, heavy-weight tasks can still be offloaded for optimizing the performance of mobile services if the cost of offloading is less compared to the cost of executing tasks locally on the mobile host. The cost of executing heavy-weight tasks locally will generally be higher than the cost of offloading them if the available bandwidth is high to interact with the remote server.

6.3 Partitioning Algorithm

The main goal of the partitioning algorithm is to keep the component interaction between the partitions (i.e. the communication cost between the mobile device and surrogates) as less as possible. This section presents the cost modeling of such a pervasive service. Then the partitioning problem is formulated and a $(k+1)$ partitioning algorithm is proposed. Furthermore, this section discusses the *Edge and Vertex*

Matching (EVM) algorithm in detail, which is the core part of the $(k+1)$ partitioning algorithm.

6.3.1 Service Component Weighting Using Multi-cost Graph

An undirected multi-cost graph $G=(V,E)$, is used to represent a pervasive service. The vertex set V represents service components. As Java applications are selected as the partitioning target, each vertex in the multi-cost graph represents a Java class. Each vertex is annotated with 3 cost weights via a 3-tuple $\langle W_{MEM}, W_{CPU}, W_{BW} \rangle$, which represents the normalized memory, CPU time, and bandwidth requirements of the vertex. To reduce computational complexity, a composite vertex weight is used to represent the three weights,

$$w(v) = \varepsilon_1 W_{MEM} + \varepsilon_2 W_{CPU} + \varepsilon_3 W_{BW} \tag{6.4}$$

where ε_1, ε_2, and ε_3 are the importance factors of each weight. The edge set E of the multi-cost graph represents the interactions (including method invocation and data access) amongst classes. For example, an edge $e(v_i, v_j) \in E$ represents the interactions between vertices v_i and v_j. The weight of an edge, $w(e(v_i, v_j))$ is the total number of interactions amongst v_i and v_j. The edge-weights represent only the numbers of interactions. The bandwidth required for interactions is considered in the vertex-weights. Figure 6.1 shows the partitioning concept of the undirected graph, where each vertex represents a class (i.e. task) and each edge represents the interaction between two classes.

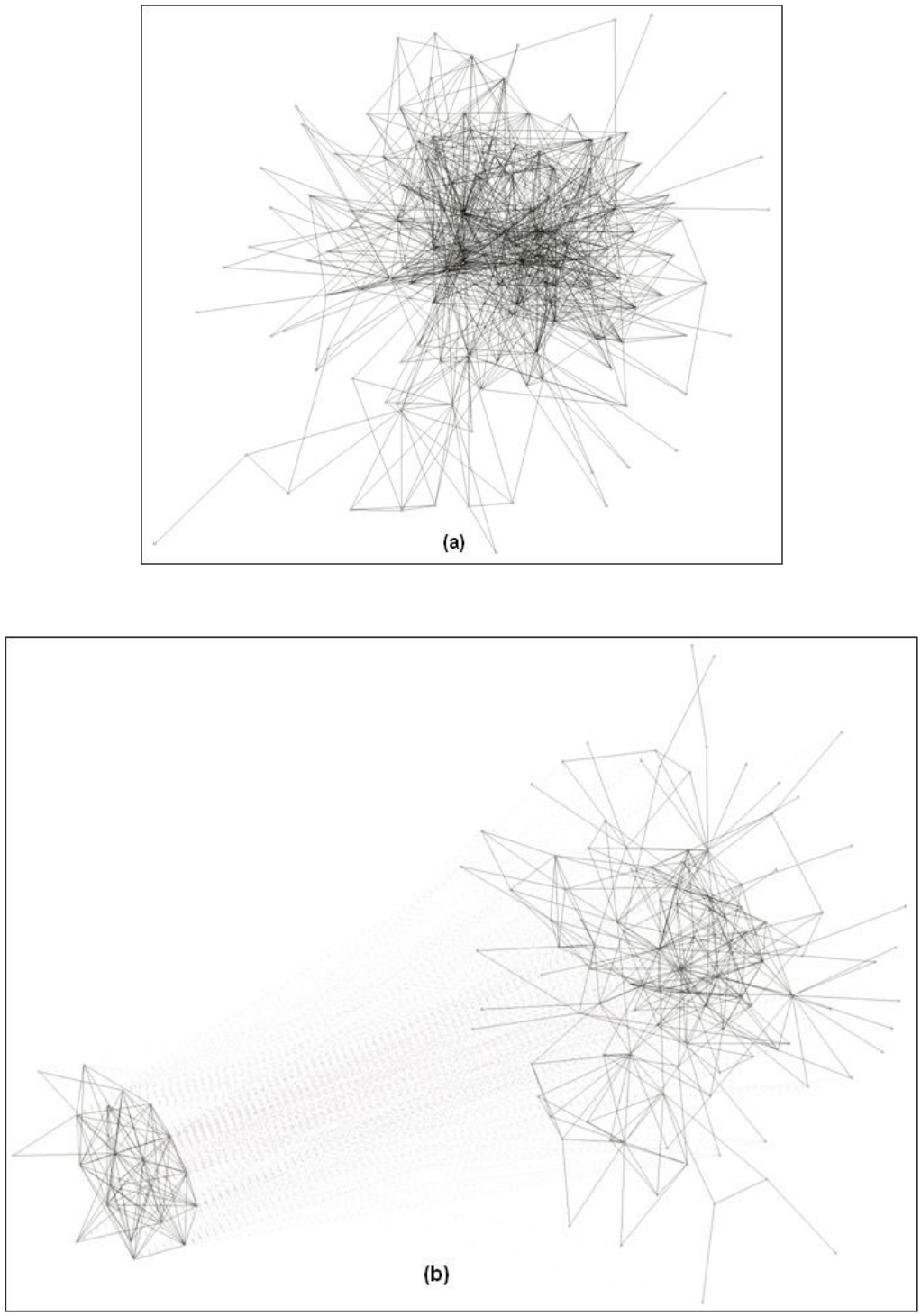

Figure 6.1: Partitioning undirected graph.

The importance factors ε_1, ε_2, and ε_3, defined in Equation (6.4) are utilized to weigh the resources memory, CPU, and bandwidth respectively. The values of these factors can be determined using one of the following means:

- Their values can be set by the service designers based on the knowledge of the source code.
- Alternatively, they can be directly set by the end users according to their real-time scenarios. For instance, if the memory utilization is low on the mobile device, the user can lower the memory factor and increase the CPU and bandwidth factors in order to save processing time.
- These values can be dynamically decided by the offloading systems according to resource availabilities in the mobile device.

6.3.2 $(k+1)$ Partitioning Algorithm

The problem of pervasive service partitioning is similar to that of partitioning a finite element graph into a certain number of disjoint subsets of vertices while fulfilling some given objectives (e.g. minimizing the amount of connection between the subsets). This algorithm is an attempt to find an optimal solution. Given a service's multi-cost graph $G = (V, E)$ and a non-negative integer $k (k \leq |V|)$, the $(K+1)$ partitioning algorithm is intended to find one un-offloadable partition V^U and k disjoint offloadable partitions $V_1^O, V_2^O, \ldots, V_k^O$ satisfying:

(i) $\bigcup_{m=1}^{k} V_m^O = V \setminus V^U$ and $V_m^O \cap V_n^O = \emptyset$ for $1 \leq m, n \leq k$ and $m \neq n$;

(ii) the *edge-cut* of $\forall V_m, V_n$ for $V_m, V_n \in \{V^U, V_1^O, V_2^O, \ldots, V_k^O\}$

$$C^{(m,n)} = \sum_{e(u,v) \in E \wedge u \in V_m \wedge v \in V_n} w(e(u,v)) \quad (6.5)$$

i.e. the sum of the edge-weights whose incident vertices belong to different partitions is minimized subject to the constraints defined by (iii);

(iii) $\forall m : \Psi^m \leq (T^m \pm \delta^m)$, where Ψ^m is the sum of vertex-weights in partition V_m, i.e.,

$$\Psi^m = \sum_{v \in V_m} w(v) \tag{6.6}$$

T^m and δ^m are the constraints that are predefined to represent the *threshold* and the *fluctuation* in partition V_m. $(T^m \pm \delta^m)$ define the lower and upper bounds of the constraints.

The algorithm involves two main steps: *un-offloadable vertex merging* and *coarse partitioning*.

(1) *Un-offloadable vertex merging:* All un-offloadable vertices need be merged into a multinode U. (A multinode is composed of two or more vertices. The multinode will be treated as a normal vertex afterwards). Let Ω represent the set of all unoffloadable vertices. The weight of vertex U is the sum of all the un-offloadable vertices' weight, i.e. $w(U) = \sum_{v \in \Omega} w(v)$. The edges connecting to U are the unions of the edges which connect the un-offloadable vertices being merged, i.e. $e(U, v) = \bigcup_{u \in \Omega} e(u, v)$. The weight of a united edge is the sum of the weights of those edges being merged, i.e. $w\big(e(U, v)\big) = \bigcup_{u \in \Omega} w(e(u, v))$.

(2) $(k + 1)$ *Coarse partitioning:* Let the multi-cost graph with all the un-offloadable vertices merged be the graph G_0. The aim of this step is to coarsen G_0 to the coarsest graph G_i, such that $|G_i| = k + 1$ and all the vertex-weights fulfil the multiple constraints defined. Note that the multinode U from the previous step is treated as a normal vertex during coarsening. The final coarsest graph G_i consists of $k + 1$ multinodes. The multimode including U becomes the un-offloadable partition V^U and the other k multinodes are the k offloadable partitions, i.e. $V_1^O, V_2^O, \dots, V_k^O$. This step is described by the following pseudo-code.

Listing 6.1: The $(k+1)$ coarse partitioning algorithm.

```
1. begin
2.   Define multi-constraints of partitions:
           T^U, δ^U, T_1^O ... T_k^O and δ_1^O ... δ_k^O;
3.   Ψ^U = 0; Ψ_1^O = ··· = Ψ_k^O = 0; // sum of vertex-weights
4.   Appoint the multinode including U as V^U;
5.   while |V| > k + 1 do { // if the graph is not coarse
                                                    enough
6.       Appoint k weightiest multinode as V_1^O ... V_k^O;
7.       if Ψ^U ≥ (T^U − δ^U) then Mark multinode V^U as
                                                      matched;
8.       for m = 1 to k { // check the k partitions
9.          if Ψ_m^O ≥ (T_m^O − δ_m^O) then Mark V_m^O as matched; }
10.      if V^U, V_1^O, ... ,V_k^O are matched and
             hasUnmatchedVertex() then
11.             Partition Failure;
12.      Invoke EVM algorithm to coarsen the graph;
13.      Update V^U, V_1^O, ... ,V_k^O to add new merged vertices;
14.      Update Ψ^U, Ψ_1^O, ..., Ψ_k^O to add new merged
                                        vertex-weights;
15.    }
16. end
```

In lines 7 and 9, the total weight of each multinode (i.e. partition) is checked to examine whether it has already reached the lower bound of the predefined cost constraint. If yes, it is marked as matched and no more vertices will be added in. If all partitions satisfy the cost constraints and there is still an unmatched vertex left, the partition is a failure (lines 10 and 11), which means the partitions cannot be found under predefined constraints. In this case, either the constraints need to be lightened or the service executes without being offloaded. In line 12, the EVM algorithm is invoked for graph coarsening.

6.3.3 The EVM Algorithm

The Edge and Vertex Matching (EVM) algorithm is the core of the $(k+1)$ coarse partitioning. At the graph coarsening phase, a sequence of successively coarser graphs $G_1, G_2, \ldots, G_n$ is constructed from the graph G_0

such that $|V_{i+1}| < |V_i|$ (i.e. the number of vertices in the successively coarser graph is smaller). Two main approaches are proposed in [Karypis and Kumar, 1998a] for coarsening a graph. The first is to merge the highly connected vertices into a multinode, while the second is to find a matching and then to collapse the matched vertices into a multinode. We adopt these two approaches in the algorithm. EVM coarsens the graph by collapsing the heavy edges.

A matching of a graph is a subset of edges with no two edges incident upon the same vertex. The task of finding a maximum matching is to select a maximum subset of such edges. The coarser graph G_{i+1} is constructed from G_i by finding a matching of G_i and collapsing the matched vertices into multinodes. The unmatched vertices are simply copied over to G_{i+1}. Since the goal of collapsing vertices using matching is to decrease the size of the graph G_i, we are trying to find the maximum matching of the graph G_i.

For finding maximum matching, Karypis and Kumar proposed *Random Matching* (RM), *Heavy Edge Matching* (HEM), and *Light Edge Matching* (LEM) in [Karypis and Kumar, 1998a; Karypis and Kumar, 1998b]. All these heuristics only consider the edge-weights. However, in the context of service partitioning for offloading systems, they are not sufficient as most of the resource constraints are related to the vertex-weights. The work in [Karypis and Kumar, 1998c] considered the vertex-weights. However, their focus is to balance constraints in partitions by selecting vertex-pairs with minimized difference for matching.

In the offloading approach, the aim is to keep more highly connected vertices in one partition, satisfying the multiple constraints. In the EVM algorithm, the heavy-edge means the incident vertices are tightly connected (i.e. with heavy edge-weight); whereas, the light-vertex means that more vertices will be merged under the predefined constraints. The basic idea of the algorithm is that when selecting an edge for matching, instead of only comparing the edge-weight, the vertex-weights of the incident vertices are also compared. A *vertex-and-edge-*

composite-weight is used to scale the weight of an edge and its incident vertices. The vertex-and-edge-composite-weight of vertex v in relation to vertex u is,

$$CW(u,v) = \lambda_1 w\big(e(u,v)\big) + \lambda_2 / w(v) \tag{6.7}$$

where, $w\big(e(u,v)\big)$ is the edge-weight, and $w(v)$ is the composite-vertex-weight of v calculated by Equation (6.7). λ_i $(i = 1, 2)$ $(0 \leq \lambda_1, \lambda_2 \leq 1$ and $\lambda_1 + \lambda_2 = 1)$ are the importance factors of edge-weight and vertex-weight respectively. If $\lambda_2 = 0$, then the EVM algorithm becomes heavy-edge matching.

If vertex v is selected to match with u due to the vertex-and-edge-composite-weight $CW(u,v)$ being maximum, then v is called the *tightest-and-lightest vertex* in relation to u. If there is more than one vertex in relation to vertex u that has the same maximum vertex-and-edge-composite-weight, then one of them is selected by the following approach. Let H be the set of such tightest-and-lightest vertices. The vertex is selected if,

$$Adj\ W(u,v) = \sum_{e(v,y)\in E \wedge (e(u,y))\in E \wedge y \neq u} w(e(v,y)) \tag{6.8}$$

is maximized; i.e., the weight sum of the edges that connect v to the vertices which are also adjacent to u is maximized. That means, to choose the one not only tightly linked with vertex u but also tightly linked with the vertices adjacent to u. If still more than one vertices are found, select the first one or a random one in H. The EVM algorithm is as follows.

Listing 6.2: The EVM algorithm.

```
1. begin
2.    Mark all vertices of vertex set V as unmatched;
3.    while hasUnmatchedVertex() do {
4.       u = RandomSelectUnmatchedVertex();
5.       v = GetTheTightestLightestVertex(u,V);
6.       if (v ≠ null) then {
7.          Put edge e(u,v) into the matching;
8.          Mark v as matched vertex; }
9.       Mark u as matched vertex;
10.     }
11. end
```

The *GetTheTightestLightestVertex* function is used to select the tightest-and-lightest vertex, which is implemented as follows.

Listing 6.3: The tightest-and-lightest vertex selection function.

```
1. Function GetTheTightestLightestVertex(u,V)
2. Input: u - the given vertex; V - the vertex set
3. Output: the tightest-and-lightest vertex to u
4. begin
5.    Adj[n] = GetUnmatchedVerticesAdjacentTo(u);
6.    CurrentTightestLightestVertex = null;
7.    CurrentMaxCW = 0; // the maximum composite-weight
8.    NumberOfTightestLightestVertex = 0;
9.    H = null; // The TightestLightestVertex set H
10.   for i = 1 to n {
11.       CW(u,Adj[i]) = GetCompositeWeightOf(u,Adj[i]);
12.       if CW(u,Adj[i]) ≥ CurrentMaxCW then {
13.          if CW(u,Adj[i]) > CurrentMaxCW then {
14.             CurrentMaxCW = CW(u,Adj[i]);
15.           H = null; // Clear TightestLightestVertex set
16.             NumberOfTightestLightestVertex = 1;
17.          } else {
18.             NumberOfTightestLightestVertex++; }
19.       H.add(Adj[i]); // Put current vertex into the set H
20.      }
21.   }
22.   if NumberOfTightestLightestVertex > 1 then
23.      CurrentTightestLightestVertex =
                    GetMostTightestLightestVertex(H);
24. return CurrentTightestLightestVertex;
25. end
```

By using the EVM algorithm, a maximum matching can be found. The vertices being matched will be collapsed into multinodes. The weight vectors of the multinodes are set equal to the sum of the weight vectors of the vertices being merged. Meanwhile, to keep the connectivity information in the coarser graph, the edges of a multinode are the union of the edges of vertices being merged.

Figure 6.2 is an example showing how to select the tightest-and-lightest vertex for matching. The vertices within the areas shaded by the dotted lines are already matched. There are still five unmatched vertices c, d, e, f and g. Suppose, vertex d is now selected randomly; one of its unmatched adjacent vertices c, e, f and g, will be selected to match with vertex d.

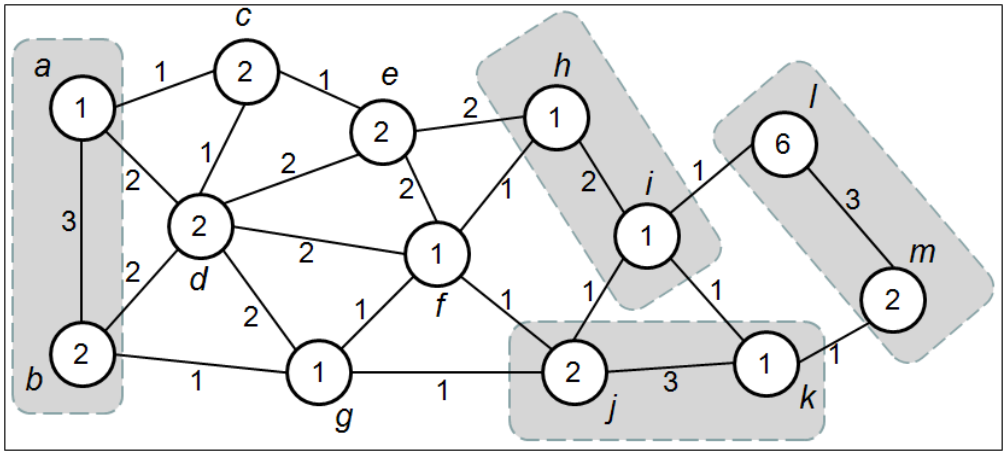

Figure 6.2: An example of EVM algorithm.

As shown in the figure, the vertex-weight of d is $w(d) = 2$. By using Equation (6.7), we can calculate $CW(d, x)$, for $x \in \{c, e, f, g\}$, to select the tightest-and-lightest vertex. Let $\lambda_1 = \lambda_2 = 0.5$ (i.e. the edge-weight and the vertex weight have the same importance). The vertex-and-edge-composite-weight of the four candidate vertices in relation to d are calculated by,

$$CW(d, c) = 0.5 \times 1 + 0.5/2 = 0.75$$
$$CW(d, e) = 0.5 \times 2 + 0.5/2 = 1.25$$

$$CW(d,f) = 0.5 \times 2 + 0.5/1 = 1.50$$
$$CW(d,g) = 0.5 \times 2 + 0.5/1 = 1.50.$$

We need to select the maximum value as the tightest-and-heaviest vertex in order to put the related edge into the matching. As we can see, there are two tightest-and-heaviest vertices f and g, in relation to d. Equation (6.8) is then used for further selection. In the case of vertex f, it is connected to e and g that are adjacent to d, therefore,

$$Adj\ W(d,f) = w\big(e(f,e)\big) + w\big(e(f,g)\big) = 2 + 1 = 3.$$

In the case of vertex g, it is connected to b and f that are adjacent to d, therefore,

$$Adj\ W(d,g) = w\big(e(g,b)\big) + w\big(e(g,f)\big) = 1 + 1 = 2.$$

As, $Adj\ W(d,f) > Adj\ W(d,g)$, the vertex f is finally selected as the tightest-and-heaviest vertex and the edge $e(d,f)$ is put into the matching.

6.3.4 Complexity of the $(k+1)$ Partitioning Algorithm

The first step of the algorithm merges all un-offloadable vertices. It traverses each vertex in the multi-cost graph to examine if that is un-offloadable. In the worst case, all the vertices may need to be examined. Thus, the worst case complexity of this step is $O(|V|)$.

In the second step, the algorithm coarsens the multi-cost graph by using the EVM algorithm. The worst case is that all the vertices are mesh connected. In this case, the complexity of the $GetTheTightestLightestVertex$ function is $O(|V|)$. Accordingly, the complexity of EVM algorithm is $O(|V|^2)$ and thus, the worst case complexity of the second step is $O(|V|^3)$.

6.4 Experimental Evaluation

In this section, the evaluation of the partitioning approach is carried out by utilizing different applications running on mobile devices. The first experiment is to validate the proposed offloading mechanism. Then the second experiment is to validate the efficiency of the EVM algorithm. These experiments aim to justify the introduction of an offloading mechanism to demonstrate its potential for efficient service provisioning from mobile devices.

6.4.1 Sample Service: π Calculator

In this experiment, a π calculator service is deployed on the mobile device for testing the offloading approach. It consists of two Java classes: `PiCalculator` and `Pi`. The class `Pi` computes the value of π, whereas the class `PiCalculator` handles the graphical user interface (GUI). This interface takes user's input for the required accuracy (i.e. how many decimal places) of the π value; then invokes the π calculation function in class `Pi`; and finally, outputs the result. The π calculator is computation intensive.

Test Setup

The mobile device used in this experiment is a HP iPAQ HX2750 PDA running Microsoft Pocket PC 2003 as the operating system. The service is deployed on the device using a J2ME runtime environment IBM J9. A desktop PC equipped with a 3.0GHz Intel single core processor and 1GB memory, is used to serve as the surrogate. The offloading middleware is based on J2ME and deployed on the PDA. The service host and the surrogate are equipped with IEEE 802.11 b/g wireless interfaces, and they communicate using a wireless local area network.

Test Results

The π calculator runs in three different cases. First, on the mobile device only; then using the offloading middleware involving both mobile device and surrogate; and at last, entirely running on the surrogate machine. In this experiment, the offline-profiling is used and only one surrogate is engaged.

Figure 6.3 shows the time consumption, memory usage and CPU utilization for the calculation, respectively. The Y-axis represents the resource usage and the X-axis represents the accuracy of π (i.e. decimal places). The curves in Figure 6.3(a) show that the response time in the mobile-device-only case is the slowest. It gets faster in the offloading case, because the class `Pi` is offloaded to the surrogate to take advantage of the rich computational resources. The surrogate-only case has the fastest response, since all the classes are running on the surrogate. Note that this specific application does not involve much inter-class interaction. Figure 6.3(b) and Figure 6.3(c) show that the memory usages and CPU utilizations on the mobile device are significantly decreased in the offloading cases. The reason for higher memory usage in the surrogate-only case is the larger JVM heap size setting for providing more memory space on the surrogate.

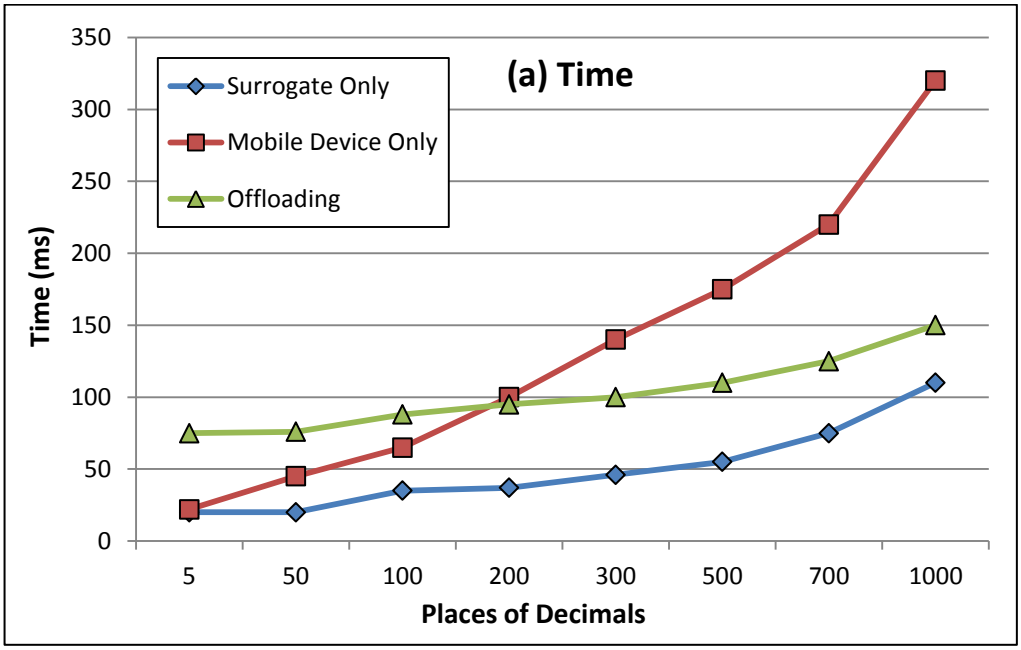

Figure 6.3(a): π calculator response time.

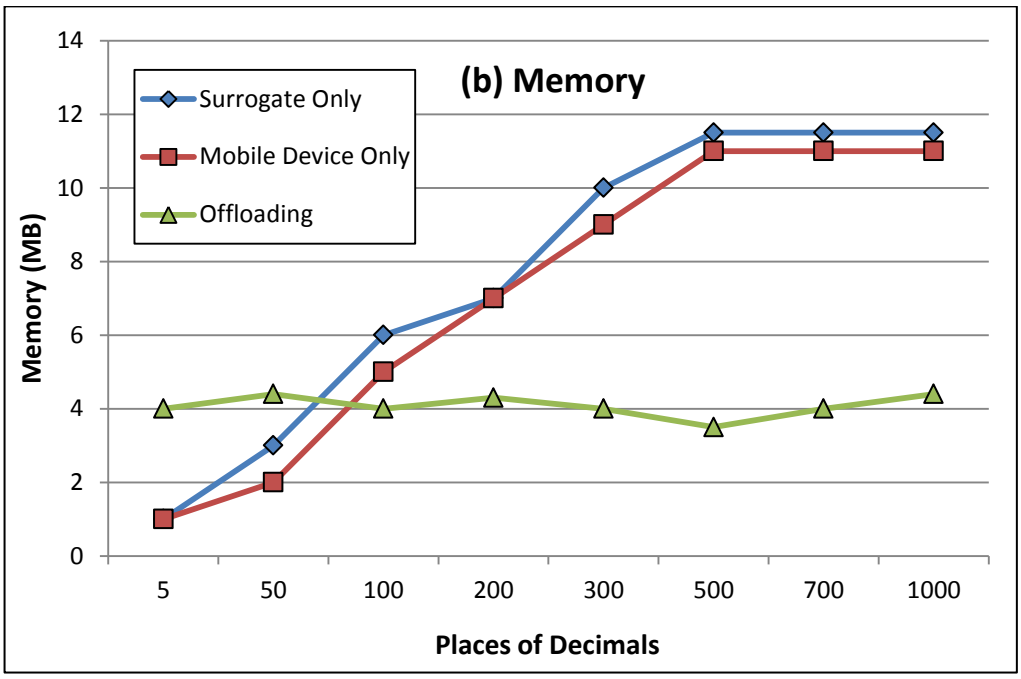

Figure 6.3(b): π calculator memory usage.

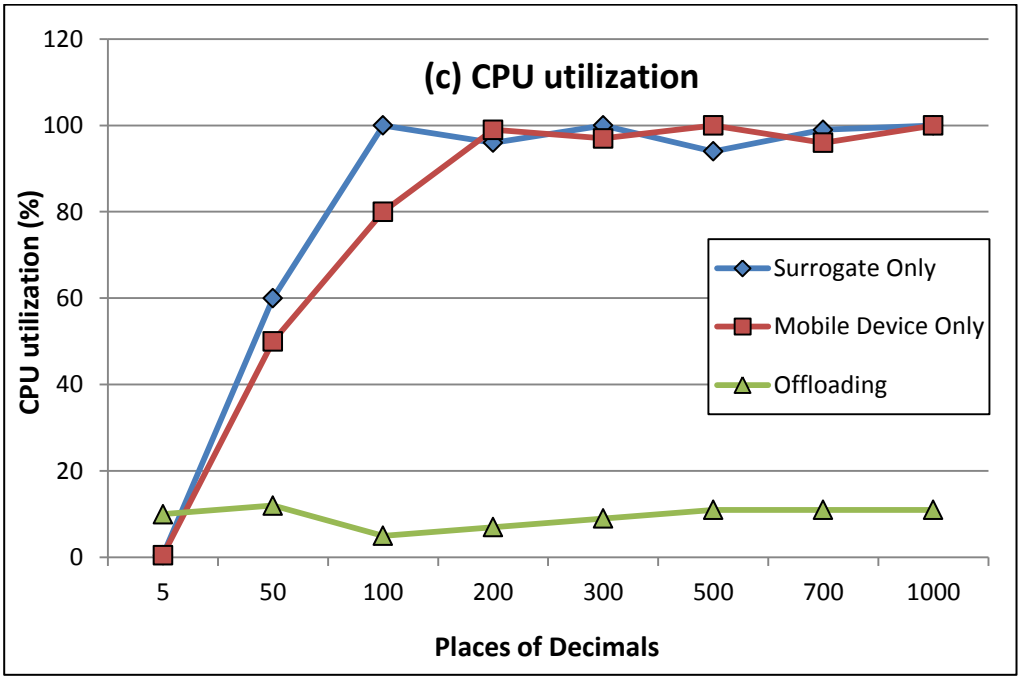

Figure 6.3(c): π calculator CPU utilization.

As can be observed from the figures, the offloading middleware itself caused some overhead. If the calculation is carried out to less than 100 decimal places, the time used for offloading is longer than the time taken without offloading. Memory usage and CPU utilization remain the same if the calculation is performed to less than 80 and 50 decimal places, respectively.

6.4.2 Efficiency of EVM Algorithm

This experiment is performed to validate the efficiency of the EVM algorithm, which is the core of the $(k + 1)$ service partitioning algorithm. This experiment compares the EVM algorithm with Random Matching (RM), Heavy Edge Matching (HEM), and MINCUT [Stoer and Wagner, 1997] algorithms by partitioning services in which the constituent tasks have different resource consumptions. For example, some tasks are computation intensive, whereas the other tasks are memory and bandwidth intensive.

Test Setup

The mobile device used in this experiment is the same as before, a HP iPAQ HX2750 PDA running Microsoft Pocket PC 2003 as the operating system. The services are deployed on the device using a J2ME runtime environment IBM J9. A desktop PC equipped with a 3.0GHz Intel single core processor and 1GB memory, is used to serve as a surrogate. The service `MobileVideo` is designed for the test, which generates and plays MPEG-4 audio/video files and also downloads files from remote hosts. `MobileVideo` is a Java program that integrates the functionalities of the MPEG-4 audio/video generator *AVgen* and the MPEG-4 audio/video player *M4Play*, provided by IBM Toolkit for MPEG-4 [MPEG-4, 2007].

Test Results

The service executes in the following order: (1) downloads video and audio sequences from a remote host; (2) generates an MPEG-4 format video clip from the downloaded sequences; (3) plays the MPEG-4 clip. The execution includes tasks with different levels of resource demands. The tasks involving MPEG-4 generation are computation and memory intensive. The tasks for remote file downloading are bandwidth intensive. The tasks for MPEG-4 playback are computation intensive.

Different matching algorithms are applied on the system to partition `MobileVideo`. The value of k is set to 1 for the $(k+1)$ partitioning algorithm, to partition the application into two partitions (for comparing with MINCUT, as MINCUT only partitions an application into two). The partitioning parameters are set as following: $T^U = 1000 \sim 5000$, $\delta^U = 200$, $T^{O_1} = 25000$, $\delta^{O_1} = 300$, $\lambda_1 = \lambda_2 = 0.5$, $\varepsilon_1 = 0.35$, $\varepsilon_2 = 0.35$, $\varepsilon_3 = 0.30$.

In the evaluation environment, we assume that the bandwidth resource is relatively rich in comparison to the other two resources. Therefore, ε_3 is set to a smaller value. Besides, ε_1 equals to ε_2 due to the

assumption that memory and CPU have equal importance to the `MobileVideo` application.

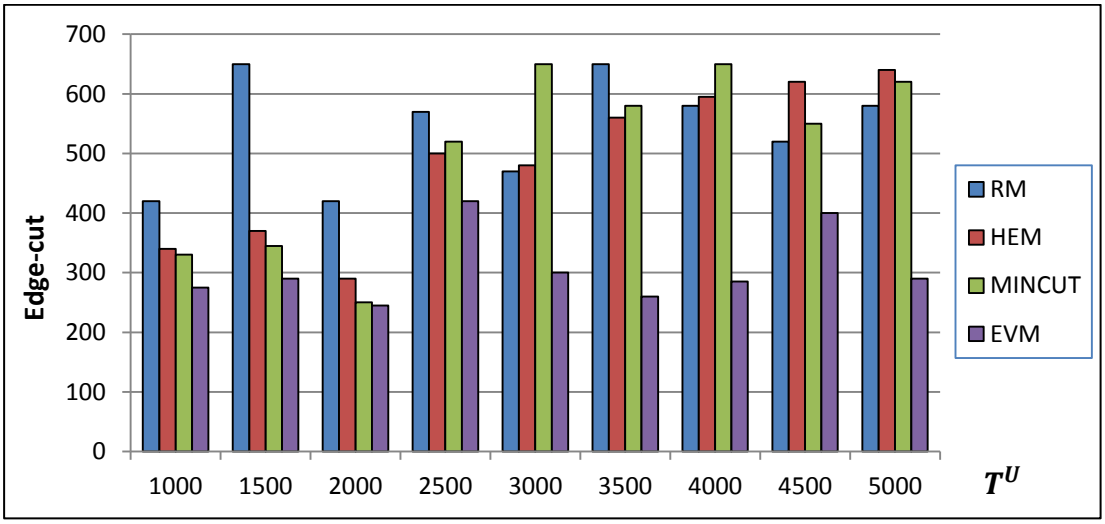

Figure 6.4: Edge-cut comparison for `MobileVideo` application.

Figure 6.4 shows all the edge-cuts with the change of T^U (the threshold of the un-offloadable partition). The values of RM are large and without regularity in variations; this reflects its random-selecting feature. The edge-cuts of HEM gets larger when T^U increases. The MINCUT gives smaller edge-cuts when $T^U \leq 2000$. However, in the rest of the cases it gives larger edge-cuts. Finally, EVM gives significantly smaller edge-cuts and it is threshold insensitive. This is due to selecting the tightest and lightest vertex for matching by the EVM algorithm.

6.5 Summary

This chapter presented an efficient application partitioning approach for pervasive services hosted on resource constrained mobile devices. The partitioning algorithm considers the dynamic condition of the mobile device in terms of available memory, CPU power, and bandwidth in making partitioning decision of the tasks and offloading the computation intensive tasks to backend surrogates. The experimental results show the effectiveness and the feasibility of the proposed approach and algorithms.

Chapter 7

Adaptive Offloading for Pervasive and Cloud Environments

Mobile devices are becoming increasingly pervasive and provide rich computing functionality and strong connectivity with powerful machines ranging from laptops and desktops to commercial clouds. In near future, computing resources in our surroundings will provide plentiful resources to support services in the locality. People will use a multitude of devices to access numerous services from their environment at any time.

In recent years, there has been a growing adoption of mobile devices such as smartphones, PDAs, mobile Internet devices, and netbooks. Examples include the Apple iPhone [iPhone, 2010], Google Android phone [Android, 2010; Nexus, 2010], RIM Blackberry phone [Blackberry, 2010], and netbooks from several vendors. These devices support exciting new features that provide a connected Internet experience to users. Users also have growing demand for running complex resource-demanding applications on mobile devices. Such applications can be the heavy-duty applications using diverse inputs like cameras and sensors. These applications demand increasing amounts of computation, storage, and communications despite the limited capabilities of the mobile devices.

Mobile cloud applications can help to reduce the workload of mobile devices by exploiting remote resources on the cloud and thereby allowing complex services to be hosted on resource-constrained devices. According to ABI Research [ABI, 2010], it has been predicted that by the end of 2014, more than 240 million business customers will be leveraging cloud computing services through mobile devices, which will deliver

annual revenues of 20 billion dollars. This underlines the potential significance of cloud computing for mobile based services.

This chapter investigates about how to achieve adaptive capabilities in making offloading decisions to exploit resources from surroundings, and introduces an adaptive offloading system for pervasive services in cloud environments. The distributed platform monitors the execution needs and resource availability of mobile devices, and dynamically decides the usage of remote resources for offloading applications transparently.

7.1 Boosting Pervasive Applications in Cloud Environments

Resource constraints (limited computation power, memory, and bandwidth) are the major obstacles for mobile devices to host and execute services that require heavy computation. Fortunately, there is increasingly broad availability of tethered computing, storage, and communication resources being spare on commercial clouds, or at wireless hotspots equipped with computational resources (e.g., Cloudlet [Satyanarayanan et al., 2009]), or at users' personal computers. This chapter proposes an approach that allows diverse, resource-constrained devices to host and execute complex services.

In this context, a device is referred to as a *surrogate* that can provide some or all of its resources to other devices to use. A device performing the role of a surrogate for a mobile host, can still work independently for other purposes. Also, surrogates are viewed as having more computing power and memory than the resource constrained mobile devices. The assumption here is that resource constraints can be alleviated by transparently using surrogate resources. A *distributed platform* can be used to achieve the transparent usage of resource from the environment. Such transparent use of surrogate resources is referred to as *offloading*, as the computation is offloaded from the host to the

surrogate. The idea is to enhance a mobile device's run-time capacity so that it can dynamically and transparently establish a distributed platform with other computing resources in its environment. This approach provides the capability for using memory, network and processing resources from surrogate nodes as needed. If the necessary resources are unavailable at the closest surrogate, multiple surrogates could be used by the host, or surrogates could offload to other surrogates to provide access to suitable resources for the mobile host.

Conceptually, the approach automatically transforms a single machine execution (e.g., computation on a smartphone) into a distributed execution that is optimal, given the network connection to remote resources or to the cloud (Figure 7.1). The underlying motivation for such a system is that, as long as execution on the cloud is significantly faster (or more reliable, more secure, etc.) than execution on the mobile device, therefore paying the cost for data transmission between the device and the cloud is worth it [Chun et al., 2011]. It makes sense to partition an application only when the metric (e.g. performance) of the newly partitioned application performs better than that of the existing application.

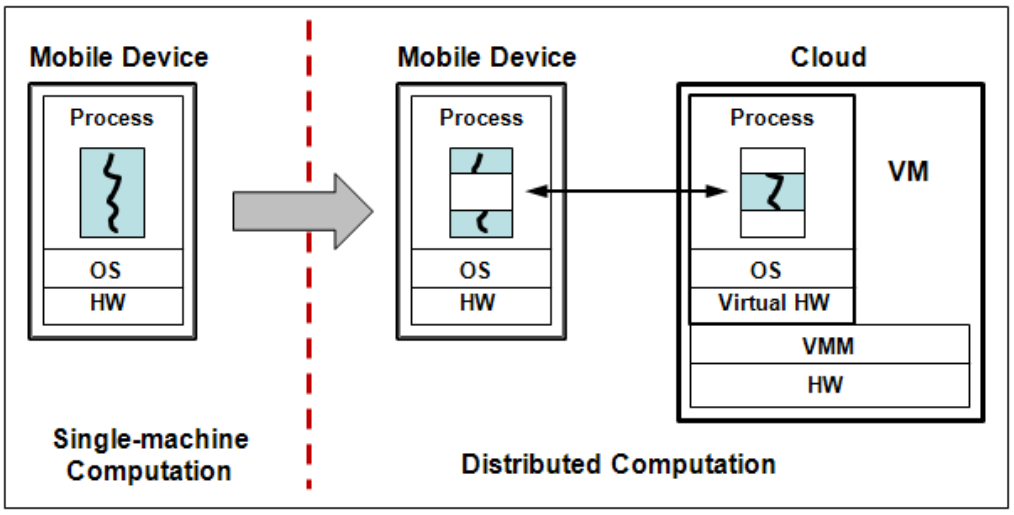

Figure 7.1: Transforming a single-machine execution (mobile device) into a distributed execution (mobile device and cloud) [Chun and Maniatis, 2009].

Furthermore, the decision may be impacted not only by the application itself, but also by the expected workload and the execution conditions, such as network connectivity and CPU speed of both mobile and cloud resources. If a mobile device becomes resource constrained at run-time and believes it can beneficially use remote resources, then the device will transparently offload part of the application execution to them. By monitoring execution needs and resource availability, the platform dynamically decides how much of the remote resources to use. As a result, the distributed platform increases the level of abstraction at which services view the resources of a mobile device.

Benefits of Mobile Cloud Applications

Mobile cloud applications bring the functionalities and services not just to smartphone users but to a much broader range of mobile subscribers, by moving the computing and data storage from mobile devices onto the cloud. The potential benefits of mobile cloud applications are highlighted below.

- Mobile cloud applications can help to overcome limitations of mobile devices, in particular processing power and storage space.
- It also can help to extend the battery life by moving the computation-intensive execution to the cloud.
- Mobile cloud applications can also be a potential solution for the fragmented market of platform dependent mobile applications.
- It can increase security level for mobile devices by facilitating centralized monitoring of services.
- It can also become a one-stop service option for mobile device users, since mobile cloud operators simultaneously can act as virtual network operators, provide e-payment services, and provide software, data storage, etc. as a service.

- A number of new functionalities can be provided by mobile clouds. In particular, provisioning of context-awareness and location-awareness can be enabled for personalization of services.

- Mobile cloud applications can open up cloud based business opportunities for individuals. At present, cloud computing mostly addresses enterprises as the potential consumers for services.

7.2 Decision-Making for Adaptive Offloading

It can be anticipated that many environments in future will contain a multitude of devices with computing capabilities. These devices can be in many forms, including desktops, embedded servers and computers (e.g. meeting room servers), personal computing devices, and so forth. Each device may contain different resources (i.e. processing power or memory), resulting in an environment full of computing resources. Also availability of high Internet bandwidth offers the reliable availability of cloud computing resources.

In pervasive computing environment, resource availability and user mobility are highly dynamic. For example, wireless network bandwidth can fluctuate significantly while a user moves around. To ensure efficient service execution, runtime offloading needs an intelligent decision-making module to adapt to the dynamic changes in a pervasive computing environment. The offloading system should 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 could decide to offload a large amount of application execution to avoid additional offloading in the near future. When the wireless connection is poor, the offloading system could decide to offload only the minimal amount of application execution to overcome resource constraints and avoid high offloading overhead.

Two important decision-making problems are identified for adaptive offloading: (1) adaptive offloading triggering, and (2) efficient application partitioning. For the first problem, the issues to be addressed are: *adaptability* to the pervasive computing environment, *configurability* to different application-specific offloading goals, and *stability* of the offloading system. For the second problem, the issue is efficiently selecting the most effective partitioning from multiple candidate partitions of the application. The most effective application partitioning should be able to meet multiple requirements for offloading simultaneously, such as minimizing wireless bandwidth requirement, minimizing average interaction delay, and minimizing total execution time. To realize the goal of pervasive service delivery without degrading its fidelity, this chapter proposes an adaptive offloading system that can dynamically partition the application and transparently offload the execution to a surrogate. In the following sections, the adaptive offloading system is presented that includes two cooperating parts: (1) *distributed offloading platform*, and (2) *offloading inference engine* (Figure 7.2).

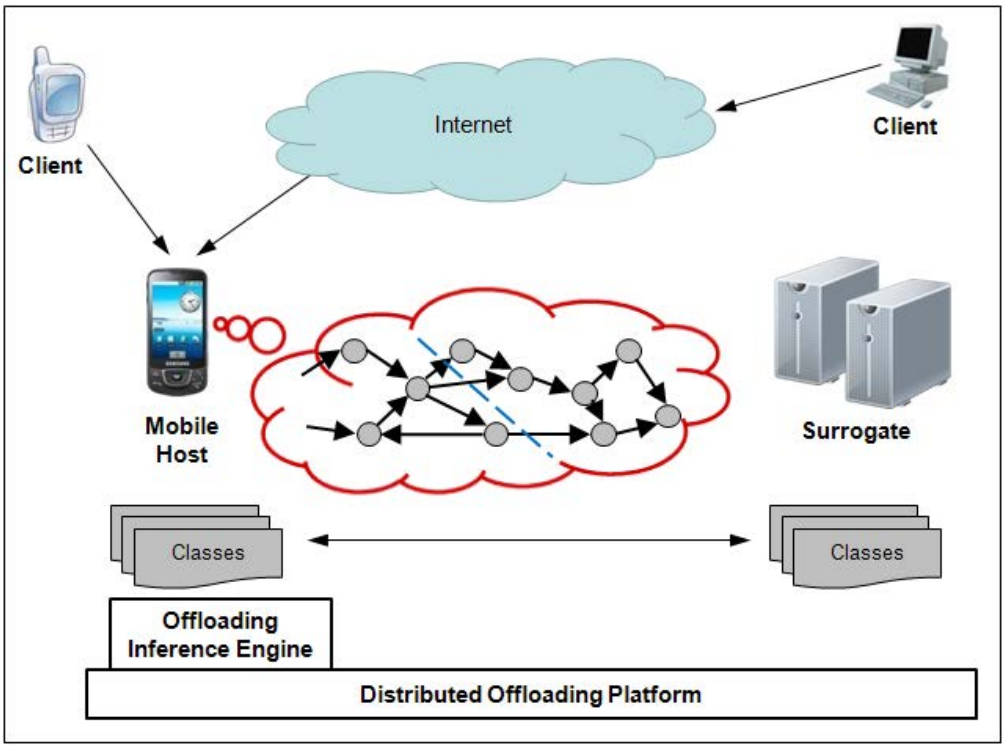

Figure 7.2: Adaptive offloading system architecture.

7.3 Distributed Offloading Platform

The distributed offloading platform refers to a system-level layer that provides a shared execution environment across two or more machines. This enables solving the problems of diversity and resource constraints of mobile service hosts.

7.3.1 Features

A distributed execution platform that supports transparent offloading for resource-constrained mobile devices requires the following features.

- ***Transparent distributed execution:*** Executing a service collaboratively on multiple machines should be possible without the application being aware that multiple machines are being used. In addition, the platform should provide the service an appearance as if the application is executing only on the mobile device. These features allow the platform to hide the complexities of remote execution and allow services to be created more independently of the underlying resources.

- ***Application partitioning:*** Dynamically dividing an application tasks into multiple partitions should be possible at run time that can be placed on different devices. Partitioning may take place at any granularity suitable to the platform or service. The partitioning should create a partition for the mobile device, which is suitable for the device to execute under its constraints.

- ***Adaptive offloading:*** To be effective, it should be possible for the partitioning scheme to consider the available resources and the service's execution patterns. Based on either resource variation triggers or periodic re-evaluation, the platform should be able to adapt to load and execution changes to maintain good partitioning decisions.

- ***Beneficial offloading:*** The platform should only offload a portion of the tasks if doing so benefits the mobile host. Offloading is defined as being beneficial if it improves the performance of the service (e.g. overcomes limitations with available processing speed or memory). It should also be possible for the user to specify what is beneficial. For example, a mobile device owner may choose to keep memory space available for performing other operations on the mobile device and offload more tasks of the service application to allow the device to continue functioning.

- ***Ad-hoc platform creation:*** It should be possible to create or terminate the distributed platform between a mobile host and a surrogate at run-time. The mobile host should be able to determine which surrogate is the most appropriate to be used, based on factors such as latency of access and resource availability.

Several of these features are available in particular platforms or in other research contexts, but no framework is available supporting them in the context of hosting pervasive services on mobile devices.

7.3.2 The Platform

The platform is visualized as shown in Figure 7.3. While the service executes, the platform on the mobile host monitors the application execution and the state of system resources such as memory, processing power, and network bandwidth. When a trigger event occurs, such as resources running low or periodical re-evaluation, the host platform analyzes the context information and decides whether offloading should occur. If it appears to be beneficial to offload, the host platform will select the tasks and will offload them to one or more surrogates running the distributed platform. The application will then continue to execute and monitoring will resume. The mobile host transparently communicates to pass execution or to access the surrogate node, and the execution on the

surrogate node transparently refers back to the mobile host for data accesses and method invocations on the host.

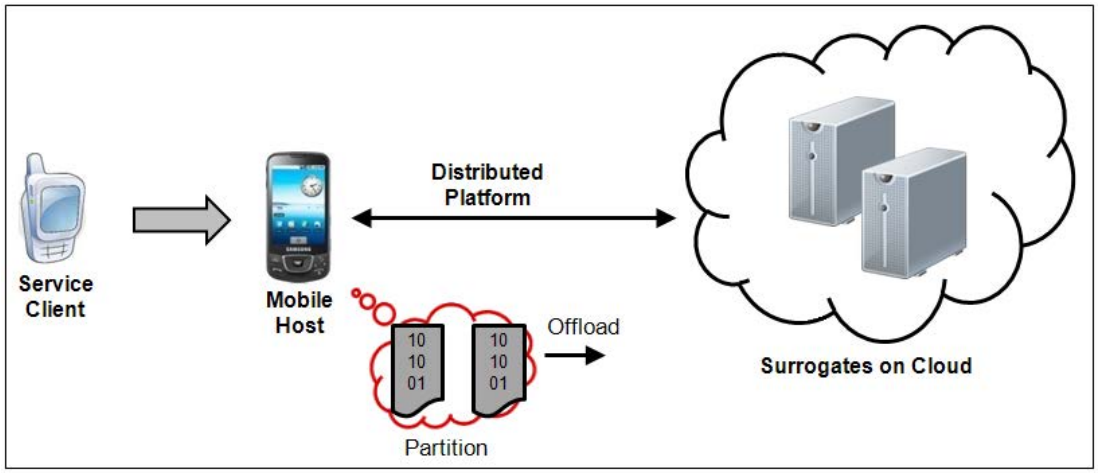

Figure 7.3: Interaction and operation of the distributed platform.

7.3.3 Benefits

A distributed service execution platform with the above mentioned features will provide the following benefits:

- Applications can be created generically for a diverse set of devices, as resource constraints will require less consideration.
- The division of responsibility between a mobile host and a surrogate node can be dynamically altered based on the available resources on the network, or the surrogate, or the mobile host.
- Component-oriented and monolithic applications can all be supported, as long as components can be offloaded independently and information on component interactions can be gathered.
- Resources of surrogate nodes can be used as needed depending on the context, i.e. when doing so is beneficial for the mobile host and service application.

7.3.4 Assumptions

This work focuses on investigating the possibilities and benefits of a distributed platform incorporating all the above mentioned features in the context of resource-constrained mobile devices hosting services. The following assumptions are made as a basis of the investigation and to limit its scope.

- The reliability of communication connectivity between mobile hosts and surrogates will be good enough that errors are not the obstacle for distributed communication. Otherwise, the reliability can be orthogonally added to the platform using existing techniques such as replication/redundancy.
- Computing resources in the environment will be always more than enough compared to how much it is required by the platform at any particular time.
- Physical device limitations such as screen size and user interface constraints will be handled by orthogonal techniques such as transcoding.

7.4 Approach for Distributed Offloading Platform

The intended transparent distributed execution can be implemented based on Java virtual machine (JVM). Using Java resolves many of the standard heterogeneity problems and allows hosting numerous applications that have already been developed in Java. The detail on how to realize the transparent distributed platform is discussed in the following subsections.

7.4.1 Componentization

From a high level view, a Java application is written as a single monolithic unit or as a group of interacting components. Some research has been done on statically partitioning applications that are composed of high-level components [Hunt and Scott, 1999]; whereas, we require dynamic partitioning on run-time. In addition, we need to handle monolithic applications, which account for most of the applications developed in Java already.

Generally, all Java applications can be considered component oriented, as they are composed of *objects* and *classes*. Thus for componentization, objects and classes are the two levels of component granularities. Each level influences the overhead of execution monitoring, the flexibility of offloading, and the type of support required for remote execution. The offloading platform considers classes as the application components; because, (1) classes represent a natural unit for service operations, (2) classes enable more precise offloading decisions than coarser component granules, and (3) classes enable to avoid from manipulating a large execution graph with too many fine-grained objects (e.g. a simple image-editing Java program created 16,994 distinct objects during 174 seconds of execution).

7.4.2 Surrogate Discovery

In the offloading platform, the mobile device runs as the *master* device and each surrogate runs as a *slave* device. The slave device runs a JVM and the monitoring modules. During runtime if the mobile device decides that offloading is needed, it triggers a new offloading action. The mobile device then initiates a discovery protocol to find a nearby surrogate that will accept the executions to be offloaded. A surrogate can be discovered using a wireless broadcast of a '*surrogate discovery*' message or a more complex discovery protocol such as UPnP [UPnP Forum, 2003] or Jini [Sun Microsystems, 2001]. The mobile device then transfers the byte-code

and data it will execute remotely on the surrogate. The surrogate loads the related classes and awaits RPC requests from the mobile device to continue the task execution.

However, in certain cases surrogate discovery could be a lengthy process, though offloading would be triggered when urgent action is needed. Therefore, another option could be to perform surrogate discovery at an earlier time. For instance, surrogate discovery can be done at the configuration phases or at the time when the mobile device enters a new environment.

7.4.3 Transparent Distributed Execution

To support transparent offloading, a mechanism is required for transparent RPCs between virtual machines. Java's existing support for remote execution (RMI) does not provide transparent mapping of calls into RPCs between machines. For diverse clients, RMI requires new client/server applications to be written for each combination. Such limitation prevents from using RMI to support a fully transparent execution of applications across multiple machines.

This limitation can be overcome by modifying the JVMs and allowing objects to be migrated transparently between the mobile device and surrogate node. In a JVM, an object is uniquely identified by an object reference. To support remote execution, the JVMs can be modified to flag object references to remote objects. Then by using these hooks the offloading platform can convert remote accesses into transparent RPCs between the JVMs. A JVM that receives an RPC request uses a pool of threads to perform execution on behalf of the other JVM. Using this approach, threads are not migrated. Rather, invocations and data accesses follow the placement of objects. However, this approach brings up several issues that must be addressed to support transparent distributed execution between the JVMs.

Native methods: Native methods cannot be offloaded because they are implemented using native code and may have different effects on different platforms. To solve this issue, native invocations are directed back to the master JVM on the mobile device. This gives an application the appearance of executing on the mobile device, even though part of their execution is on a surrogate node.

Static functions and data: Static functions and data are statically shared between application components. Like native methods, the JVM may also contain some static data that is specific to the host where it is located. Therefore, to ensure consistency all accesses to static data are directed back to the master JVM on the mobile device.

Object references: Each JVM has a private object reference name-space and does not understand an object reference from another JVM. To overcome name-space limitations, the JVMs can be modified to map each other's references into their own name-space. Each JVM keeps local references for remote objects as placeholders. When a JVM invokes a method on the other JVM, it sends an operation referring to the object using its local object reference. The receiving JVM then maps the first JVM's local reference to its own local reference for the object. As a result, each JVM maintains its object reference mappings when objects and object references move between the two JVMs.

7.4.4 Partitioning Execution

The rationale behind partitioning is that if two components interact frequently (e.g. because of invocations), the graph will reflect this with a high-weighted edge. A partitioning policy should have a high probability of placing frequently interacting components on one machine, as because splitting them across the network could severely affect performance. Execution history can reflect future execution and can be used to predict the future behavior. Combining execution history with resource

availability will allow a partitioning policy to effectively create partitions that balances the service's resource requirements, the device's resource availability, and the user's preferences.

The offloading platform attempts to create partitions suitable for resource-constrained devices to satisfy the partitioning policy. To achieve this, the offloading platform generates a set of possible candidate partitions and selects the partitioning that best satisfies the partitioning policy. The approach begins by placing all of the classes that cannot be offloaded, into one partition. That means, placing the classes that contain native methods, into the first partition. This is the partition that will remain on the mobile device and will not be offloaded. Then the approach continues by moving one node at a time. It bisects the graph along the cut with the maximum interactions or the highest interaction weight (Figure 7.4). Therefore, the next node to move to the first partition will be the node having highest connectivity with the first partition. This process is repeated until the first partition contains all but one of the nodes.

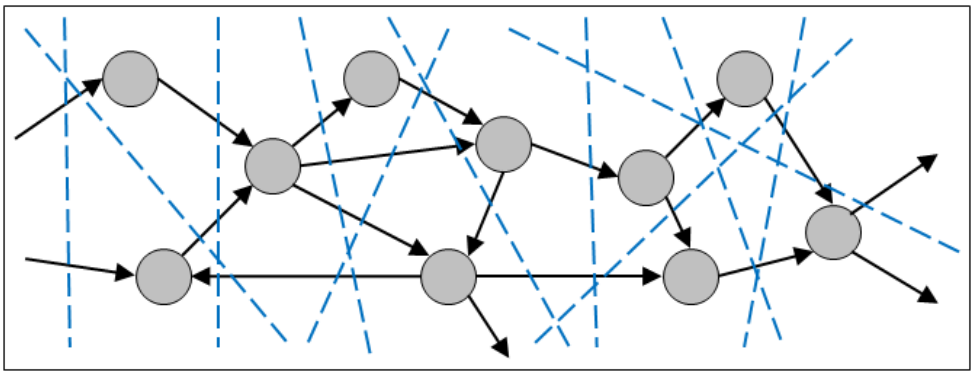

Figure 7.4: Multiple partitioning of execution graph.

All the intermediate partitions are evaluated according to the partitioning policy. The selected partitioning may not have the minimum interaction cost, but will satisfy the overall policy best. A particular intermediate partitioning is evaluated by applying a cost function that represents part of the partitioning policy. The cost function returns the historical information of the interactions between the two partitions. The

partitioning policy then determines whether any part of the application can be beneficially offloaded. If so, it determines whether components can be offloaded without affecting other constraints severely. For example, a constraint can be that at least a certain amount of memory must be freed by the partitioning. If no such partitioning exists, offloading will not occur. Conceptually, this policy offloads a significant amount of processing while placing minimal load on the network bandwidth.

Algorithm to generate possible candidate partitions

Let $G = (V, E)$ represent the current application execution graph, where V is the set of nodes and E is the set of edges. Each edge is associated with a cost value reflecting the inter-class interactions. Let P_M represent the partition on the mobile device, and P_S represent the partition on the surrogate. At the beginning both P_M and P_S are initialized as empty. The candidate partition generating algorithm is as follows.

- ***Step 1.*** Merge all the nodes that cannot be offloaded to the surrogate (i.e. nodes containing native methods), into one node v_1. If there are k edges from a node to the merged nodes, then substitute the k edges with one edge. The new edge has an edge cost equal to the sum of the costs of k old edges. Suppose there are n nodes after merging. Let $P_M = \{v_1\}$ and $P_S = \{v_2, \dots, v_n\}$.

- ***Step 2.*** Among the neighbors of v_1 representing the partition on the mobile device, select the one that has the largest edge cost to v_1. Suppose the selected node is v_i. Merge v_i with v_1 and move v_i from P_S to P_M. Consider the cut $\Gamma \triangleq < P_M, P_S >$ as one of the candidate partitions. This candidate partition is recorded with its information about the partitioning cost and the two partitions.

- ***Step 3.*** Repeat Step 2 until all nodes have been merged with v_1.

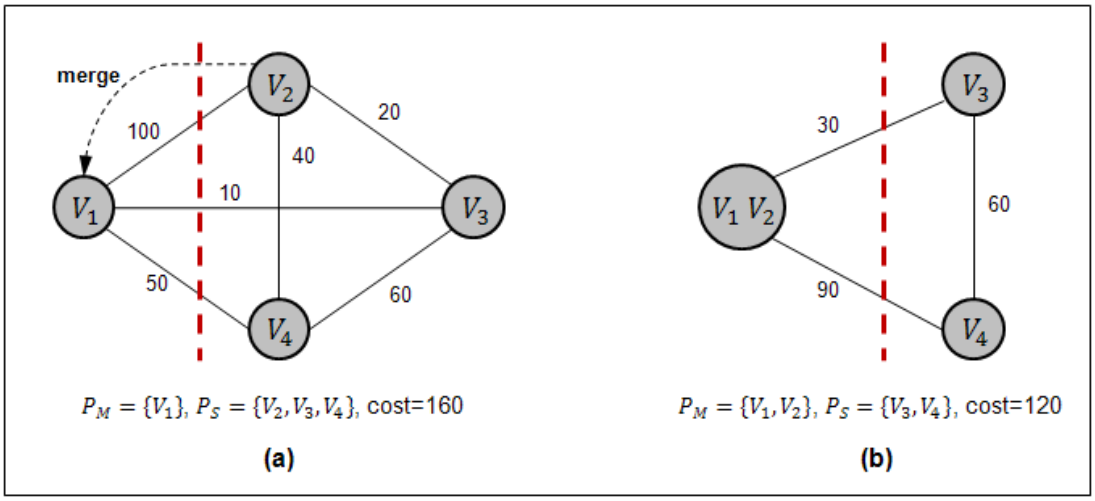

Figure 7.5: Illustration of candidate partition generation.

Figure 7.5 illustrates the process of the candidate partition generation. The original execution graph with cost annotation is shown by Figure 7.5(a). First, v_2 is selected among the neighbors of v_1, which has the largest edge cost. Then v_2 is merged with v_1, generating a new graph illustrated by Figure 7.5(b). From this new graph, another candidate partition can be derived. The merging process will continue until all four nodes are merged into one. The offloading platform can then select the best partitioning from the above candidate partitions, based on their partitioning cost metric.

7.4.5 Application Execution Monitoring

To be able to partition an application efficiently, its execution information needs to be collected on the runtime. This is accomplished by augmenting the JVM's code for method invocations, data accesses, object creation, and object deletion. The information is obtained at the object level and aggregated to the class level. The system monitors the amount of memory occupied by a class, the number of interactions between two classes, and the amount of information exchanged between two classes.

The execution information is characterized by a fully connected weighted graph to reflect the application's execution history. Each node

represents a class and is annotated with the amount of memory occupied and CPU processing cost by the class. Each edge represents the interaction between two classes and is annotated with the number of interactions and the total amount of information transferred between the classes.

7.4.6 Resource Monitoring

For a partitioning to remain effective, the partitioning process needs to adapt to the resource changes in the environment. The offloading platform needs to monitor the resources of the mobile device, the surrogate, and the wireless network. The viable resource monitoring techniques are briefly discussed in the previous chapter in subsection 6.2.2.

The available memory on the mobile device and surrogate are monitored by tracking the amount of free space in the Java heap, which is obtained from the garbage collector of the JVM. However, measuring CPU utilization can be difficult. The percentage of CPU occupied is not a feasible way to represent a class's CPU utilization. A viable approach is to calculate the cumulated CPU occupying time of a class and normalizing it as the CPU processing cost of the class. The wireless bandwidth can be estimated by passively observing ongoing traffic through the offloading platform, or by actively measuring the remote data accesses to obtain bandwidth usage of each class.

Whenever any significant change happens (e.g., a certain amount of memory or CPU is consumed, or a sufficiently large wireless bandwidth fluctuates), the offloading inference engine decides whether offloading should be triggered.

7.5 Adaptive Offloading Inference Engine

The design detail of the offloading inference engine is presented in this section. The offloading inference engine performs the decision making tasks for triggering offloading and selecting partitioning. Offloading can add overhead to the service application's execution. This overhead includes the cost of transferring objects between the mobile device and the surrogate, and performing remote data accesses and function invocations over a wireless network. One of the goals of the inference engine is to minimize the offloading overhead while relieving the memory or CPU constraints on the mobile device. The offloading inference engine is illustrated in Figure 7.6.

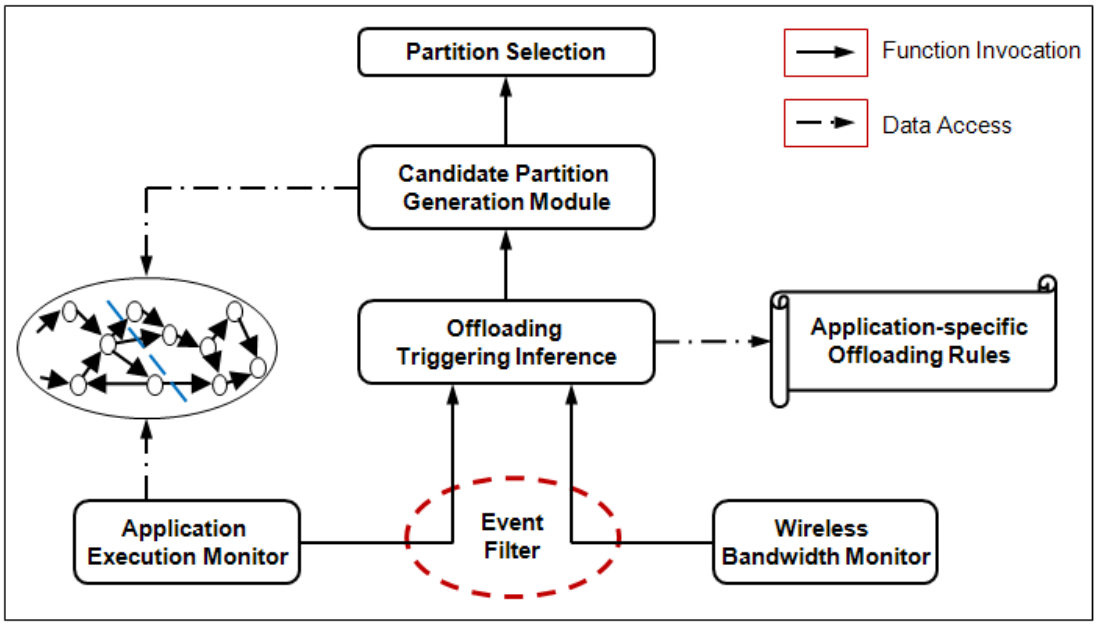

Figure 7.6: Offloading inference engine architecture.

7.5.1 Trigger Offloading based on a Fuzzy Control Model

To trigger offloading, the offloading inference engine examines the current resource consumption of the application and the available resources in the pervasive computing environment. It then decides whether offloading should be triggered given the user's offloading goals. If

offloading is to be triggered, it decides what level of resource utilization should be done on the mobile device. That means, how much memory or CPU should be freed up by offloading task objects to the surrogate. At a first glance, the problem can be solved using a simple threshold-based approach. For example, threshold-based rules can be predefined in the offloading inference engine; such as, *"if the current amount of free memory on the mobile device is less than 20% of its total memory, then trigger offloading and offload enough task objects to free up at least 40% of the mobile device's memory"*. However, such a simple approach cannot meet the challenges of adaptability, configurability, and stability requirements (discussed in section 7.2) in dynamically changing environments.

The offloading inference engine addresses this problem with a *Fuzzy Control model* [Li and Nahrstedt, 1999; Passino and Yurkovich, 1998], which has been shown to be effective for flexible, expressive, and stable coarse-grained application adaptations. The use of this approach in the offloading inference engine is novel because it applies the model to coarse-grained application adaptation via runtime offloading. The Fuzzy Control model includes: (1) linguistic decision-making rules provided by application developers, (2) membership functions, and (3) a generic fuzzy inference engine based on fuzzy logic theory.

The term '*fuzzy*' refers to the ability of dealing with imprecise or vague inputs. Instead of using complex mathematical equations, fuzzy logic uses linguistic descriptions to define the relationship between the input information and the output action. Based on the Fuzzy Control model, the *offloading rules* can be specified for the offloading inference engine as shown in Listing 7.1.

Listing 7.1: Offloading rules for the offloading inference engine.

```
if (AvailMem is low) and (AvailBW is high)
             then NewMemSize:=low;
if (AvailMem is low) and (AvailBW is moderate)
             then NewMemSize:=average;
if (AvailMem is high) and (AvailBW is low)
             then NewMemSize:=high;

if (AvailCPU is low) and (AvailBW is high)
             then NewCPUalloc:=low;
if (AvailCPU is low) and (AvailBW is moderate)
             then NewCPUalloc:=average;
if (AvailCPU is high) and (AvailBW is low)
             then NewCPUalloc:=high;
```

The `AvailMem`, `AvailCPU`, and `AvailBW` are *input* linguistic variables that represent the current available memory, available CPU, and available wireless bandwidth respectively. The `NewMemSize` and `NewCPUalloc` are the *output* linguistic variables representing the new memory utilization and the new CPU utilization respectively, on the mobile device. If any of these rules is matched by the current system conditions, the offloading inference engine triggers offloading and derives the offloading memory size or CPU allocation (i.e., current consumption – new utilization). If the difference is negative, it means that some task objects should be pulled back from the surrogate to adapt to the low wireless bandwidth. The application developer or the user can easily configure the offloading inference engine using the linguistic offloading rules.

To interpret the linguistic offloading rules, the offloading inference engine needs to establish mappings between numerical and linguistic values for each linguistic variable. *Low*, *moderate*, and *high* are the linguistic values. In fuzzy logic, the mapping between the numerical value of a linguistic variable and its linguistic values are defined by a *membership function*. A membership function represents a fuzzy set on the *universe of discourse* [Passino and Yurkovich, 1998]. The membership

function gives the grade or degree of membership within the set of any element of the universe of discourse.

The membership function maps the elements of the universe onto numerical values in the interval [0, 1]. A membership function value of *zero* implies that the corresponding element is definitely not an element of the fuzzy set, while a value of *unity* means that the element fully belongs to the set. A grade of membership in between corresponds to the fuzzy membership to the set. Each fuzzy set spans a region of input (or output) value graphed with the membership. Any particular input is interpreted from this fuzzy set and a degree of membership is interpreted. The membership functions normally overlap to allow smooth mapping of the system.

For example, Listing 7.2 shows the membership function definition for the linguistic variable `AvailMem`. Figure 7.7 gives the graphical representation of the corresponding membership function. In this example, if the numerical value of `AvailMem` is within the range [0, 800], the offloading inference engine's stochastic confidence that `AvailMem` belongs to the set of linguistic value '*low*' is 100%. Whereas, if the numerical value of `AvailMem` is within the range [800, 900], the confidence that `AvailMem` belongs to '*low*' is the linear decreasing function from 100% to 0%.

Listing 7.2: Membership function definition for linguistic variable `AvailMem`.

```
lingvar AvailMem on [0, 8000] with
     class low is 0 0 800 900
     class moderate is 850 1500 4000 4200
     class high is 4150 5500 7500 8000
end
```

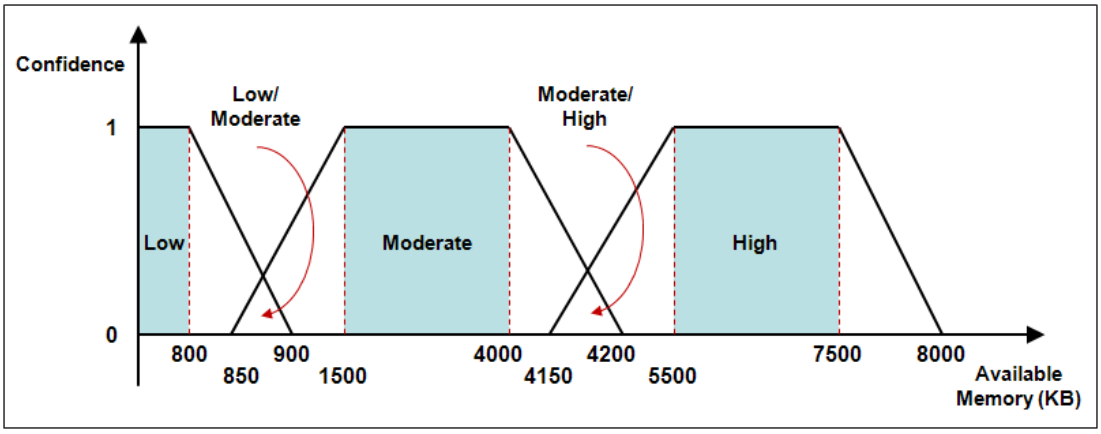

Figure 7.7: Illustration of membership function `AvailMem`.

In another example, Listing 7.3 shows the membership function definition for the linguistic variable `AvailCPU`. Figure 7.8 gives the graphical representation of the corresponding membership function. In this example, if the numerical value of `AvailCPU` is within the range [0, 100], the offloading inference engine's stochastic confidence that `AvailCPU` belongs to the set of linguistic value '*low*' is 100%. Whereas, if the numerical value of `AvailCPU` is within the range [100, 140], the confidence that `AvailCPU` belongs to '*low*' is the linear decreasing function from 100% to 0%.

Listing 7.3: Membership function definition for linguistic variable `AvailCPU`**.**

```
lingvar AvailCPU on [0, 500] with
     class low is 0 0 100 140
     class moderate is 120 170 270 320
     class high is 300 350 450 500
end
```

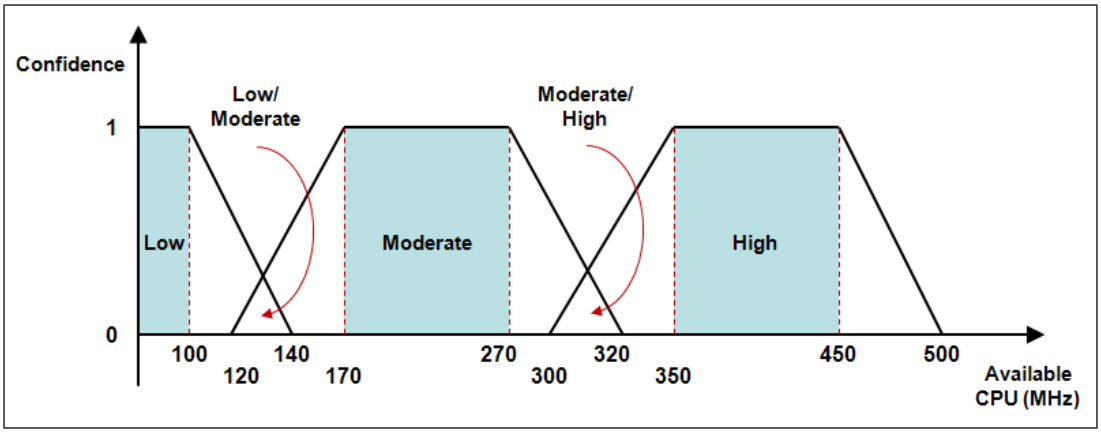

Figure 7.8: Illustration of membership function `AvailCPU`**.**

The intersection between different linguistic values (e.g. the values within [850, 900] in Figure 7.7, or the values within [120, 140] in Figure 7.8) represents uncertainty in stochastic confidence and the result can belong to either linguistic value *'low'* or *'moderate'* but with different confidence probabilities. Membership functions are part of the rule specifications provided by the application developer. If the current system and network conditions match any specified rule, an offloading action is triggered. In comparison to simple threshold-based offloading triggering, the Fuzzy Control model allows us to implement more expressive and configurable triggering conditions.

The generic fuzzy inference engine implements the fuzzy-logic based mapping and non-linear adaptation process. It takes the confidence values of fuzzy sets (e.g. *low*, *average*, and *high*) as inputs and generates outputs in the form of confidence values of fuzzy sets for output variables (e.g. `NewMemSize`, `NewCPUalloc`). Hence, to use the generic fuzzy

inference engine, the offloading inference engine provides two functions. The function *fuzzification* is to prepare input fuzzy sets for the generic fuzzy inference engine. Fuzzification is the process of decomposing the input values into one or more fuzzy sets. The function *defuzzification* is to convert the output fuzzy sets to actual offloading decisions; such as, the new memory or CPU utilization on the mobile device. Defuzzification is the inverse transformation which maps the linguistic output variable into a crisp numeric value that best represents the inferred offloading decisions.

7.5.2 Efficient Partitioning Selection

The offloading inference engine selects the best application partitioning from a group of candidate partitions generated by the offloading platform. First, the offloading inference engine considers the target memory utilization or target CPU utilization on the mobile device to discard partitioning plans that do not meet the minimum requirement. Then the offloading inference engine selects the best partitioning from the remaining candidate partitions by using a composite cost metric. In the case of offloading to overcome the memory or CPU constraints of mobile devices, the user can have multiple offloading requirements. Such as, minimizing wireless bandwidth overhead, minimizing average response time stretch, and minimizing total execution time. The wireless bandwidth cost comes from two factors: (1) migration of task objects during offloading, and (2) remote function calls and remote data accesses. The average response time stretch is decided by the total number of all remote invocations. The total execution time stretch caused by offloading includes all offloading delays and remote interaction delays.

The offloading inference engine addresses the problem by considering different inter-class dependencies and interactions during application execution. For each neighbor node v_k of v_i, $b_{i,k}$ denotes the total amount of data traffic transferred between v_i and v_k. $f_{i,k}$ defines the

total interaction number, and M_k represents the current memory size of v_k. The offloading inference engine uses a composite cost metric for the application execution graph edge between v_i and v_k: $C_k \triangleq < b_{i,k}, f_{i,k}, M_k >$. Such a composite cost metric is most effective for meeting different offloading requirements, as it collectively considers the bandwidth, interaction frequency, and memory size of the candidate neighbor during the coalescing process. The cost of a candidate partition is the aggregated costs of all edges whose end-points belong to different partitions. The offloading inference engine then selects the best candidate partitioning that minimizes the partitioning cost.

Suppose, b^{max}, f^{max}, and M^{max} represent the upper bound of all possible $b_{i,k}$, $f_{i,k}$, and M_k. The comparison of any two composite cost metrics C_k and C_l (for node v_l) is defined as follows: $C_k \geq C_l$ if and only if,

$$w_1 \cdot \frac{b_{i,k}-b_{i,l}}{b^{max}} + w_2 \cdot \frac{f_{i,k}-f_{i,l}}{f^{max}} + w_3 \cdot \frac{M_l-M_k}{M^{max}} \geq 0 \tag{7.1}$$

$$\sum_{i=1}^{3} w_i = 1, \quad 0 \leq w_i \leq 1, \quad 1 \leq i \leq 3. \tag{7.2}$$

The above equations imply that the offloading inference engine always keeps the classes on the mobile device that are more active (i.e., that have larger bandwidth requirements and interaction frequencies) and occupy smaller amount of memory space. Meanwhile, the offloading inference engine intends to offload the classes to the surrogate that are more inactive (i.e., that have smaller bandwidth requirements and interaction frequencies) and occupy larger amount of memory space.

To allow customization, $w_i (1 \leq i \leq 3)$ is used to represent the importance factor of the ith metric (e.g. wireless bandwidth, remote interaction delay, and memory size of the class) in making the offloading decision. These weights can be adaptively configured according to application requirements and user preferences. For example, if the user cares more about the interaction delay while having plentiful wireless

bandwidth, in such case w_1 can be set to a lower value and w_2 to a higher value of importance.

7.5.3 Class Granularity Consideration

As mentioned before, classes were selected as the execution graph nodes. However, in practice it has been observed that the memory sizes of some classes are too large to be treated as single nodes. For example, in an experiment the string class in JavaNote application occupied 5.9MB memory during execution. If such large classes are offloaded, they will cause large migration and remote invocation overhead. On the other hand, if they are not offloaded, the memory constraint of the mobile device cannot be subdued.

Hence, if the memory size of a class exceeds a certain threshold, a new node is created in the execution graph for the class. All the objects belonging to the large class are distributed into two sets, each of which represents a node in the execution graph. Thus, the large class node is split into multiple nodes with smaller memory sizes to enable more precise control over offloading execution. The decision-making algorithm used by the offloading inference engine in splitting large classes is illustrated in Listing 7.4.

Listing 7.4: Decision-making algorithm used by offloading inference engine.

```
M_i: memory size for Java class i;
EG = {N_0, N_1, ..., N_n}: execution graph;
CMT: the maximum memory size for a class node;
NewMemoryUtilization ← −1;

while offloading service is on
     while (no significant changes happen)
          perform executions and update EG
                                        accordingly;
          while (∃M_i > CMT)
               create a new node to represent class i;
     // make the adaptive offloading triggering decision
     // set numerical values for all input linguistic
                                                variables
     SetLingVar();
     // map the numerical values to the linguistic values
     fuzzify();
     FuzzyInferenceEngine();
     // map the linguistic values to the numerical values
     defuzzify();
     if (NewMemoryUtilization == −1)
          then offloading is not triggered;
     else // make the partitioning decision
          merge all un-offloadable classes into a
                                                  node N;
          while (size(EG) > 1)
               merge (N, one of its neighbors NB_j);
               if (current cut is better)
                    bestPos = NB_j;
     Partition_mobiledevice = {N_0, ..., N_bestPos};
     Partition_surrogate = {N_bestPos+1, ..., N_n};
```

7.6 Performance Evaluation

In this section, the performance of the dynamic offloading platform is evaluated using extensive trace-driven simulation experiments. For this study, the prototype is designed based on the distributed offloading platform described in section 7.3. The prototype is developed by modifying ChaiVM version 5.1, which is HP's personal JVM for embedded and real-

time systems. It supports the distributed execution of monolithic Java applications and performs offloading from a mobile host device to a single surrogate device.

In the experiments, it is assumed that graph partitioning is performed solely on the JVM of the mobile device. Distributed monitoring of the application execution and distributed partitioning of the execution graph would be more suitable in a real-world solution. To simplify the platform, it is also assumed that both VMs (on the mobile host and the surrogate) have access to the application's byte-codes. This allows both VMs to have common knowledge about the application. In a real-world solution, the surrogate may need to acquire the necessary byte-codes from the mobile device or have them installed.

Setup and Methodology

The application execution traces are collected on a Linux desktop machine. The execution trace files record method invocations, data field accesses, and object creations or deletions by querying the instrumented JVM. The wireless network traces are collected using the Ping system utility on a Toshiba A300 laptop with an IEEE 802.11 WaveLAN network card and available bandwidth measurement tools [Hu and Steenkiste, 2003]. The surrogate device is a desktop located in an office room and attached to the 10Mbps corporate wireless network using a network switch leading to the access point.

The mobile roaming scenario for the evaluation is conducted in a three storied building equipped with wireless access points at all levels. The wireless network traces are obtained by having a person with the mobile device starting from the office room, entering an elevator and riding up to the basement, exiting the elevator and then walking to a stairway. The measured wireless bandwidth maintains around 4.8Mbps until the person enters the elevator where it drops to about 2.4Mbps. Then it rises to about 3.6Mbps when the person walks through the

ground floor. The size of the parameters used for function interactions and data accesses is quite small (i.e., < 64 bytes in all execution traces). Therefore, only the round-trip time (RTT) is measured for small packets, which is about 2.4ms on average.

The prototype is driven by the execution and network traces described above. The simulation emulates a remote function invocation by stretching the total execution time. The remote data access delay is the time duration between sending a request to the remote site and receiving the requested data, which is approximately equal to the RTT. The remote function call delay is the time duration to redirect a function request to the remote site, which is close to $RTT/2$ each way. The offloading delay is simulated by increasing execution time using the calculation, $\frac{\sum Memory_{classes\ to\ be\ offloaded}}{current\ available\ bandwidth}$.

In all the evaluations, *Adaptive Offloading Engine (AOE)* denotes the proposed approach with the offloading inference engine. For comparison, two other common heuristic approaches are also implemented in making offloading decisions: *random* and *least recently used (LRU)*. Unlike the proposed approach, which adaptively triggers offloading by comprehensively considering both memory and wireless network conditions, *random* and *LRU* adopt a simple fixed rule, which triggers offloading when the available memory is lower than 5% of the total memory and sets the new target memory utilization as 80% of the total memory. In all of the experiments, the adaptive offloading platform uses the offloading triggering rules presented in subsection 7.5.1.

The complete offloading inference engine is run with partitioning selection using the composite metric defined by Equation (7.1). The weights $w_i (1 \leq i \leq 3)$ in Equation (7.2) are all set as 1/3 to specify equal importance. Moreover, the *random* and *LRU* algorithms do not consider the problem with large classes. Whereas, *AOE* splits the large class node into smaller nodes with memory sizes smaller than 500KB. For application partitioning, the *random* algorithm randomly selects some classes to keep on the mobile device and offloads the rest of the classes to

the surrogate. The *LRU* algorithm offloads those classes that are least recently used according to their access frequencies.

Applications Used in the Observations

Table 7.1 lists the descriptions of three applications used in the experiments. *Dia* is a simple Java image editor. For the execution trace, a 180 KB image file was opened with Dia and was dragged around. The application is quite memory intensive as it loads a large image file and allows editing on the mobile device. *Biomer* is a graphical molecular editor that can be used to visualize and edit the chemical structure of various molecules. For the execution trace, three complex molecules were drawn. The application is both highly memory and CPU intensive as performed. *JavaNote* is a Java text editing application. Its execution trace is extremely memory intensive as JavaNote was used to read a large text file of 600 KB. This causes JavaNote to keep creating and deleting objects of the string class. The maximum memory capacity of the mobile device (i.e., the Java heap size) was set to 8 MB for Dia, to 8 MB for Biomer, and to 7 MB for JavaNote, according to their peak memory requirements.

Table 7.1: Descriptions of applications used in the experiments.

Application	Description	Operation	Lifetime	Peak memory requirement	Oversizing	Classes
Dia	Image editor	Open a 180 KB image file and drag it around (memory intensive)	174s	8,949 KB	11% (8MB)	100
Biomer	Graphical molecular editor	Create three complex molecules (both memory and CPU intensive)	261s	10,668 KB	34% (8MB)	105
JavaNote	Text editor	Open a 600 KB text file (extremely memory intensive)	268s	7,972 KB	14% (7MB)	85

7.6.1 Evaluation of Different Performance Metrics

Three different performance metrics are considered in evaluating the *AOE* with *random* and *LRU* algorithms to execute the three applications in the experiments. The performance metrics are: total offloading overhead, average interaction delay, and total bandwidth requirement. The investigations and observations on these metrics are discussed below.

Total Offloading Overhead

The first performance metric considered is the *total offloading overhead*, which consists of migration overhead, remote data access overhead, and remote function call overhead. These overheads extend the total

execution time of the three applications. Figure 7.9 illustrates the total offloading overheads of the three applications by *random*, *LRU*, and *AOE* algorithms respectively.

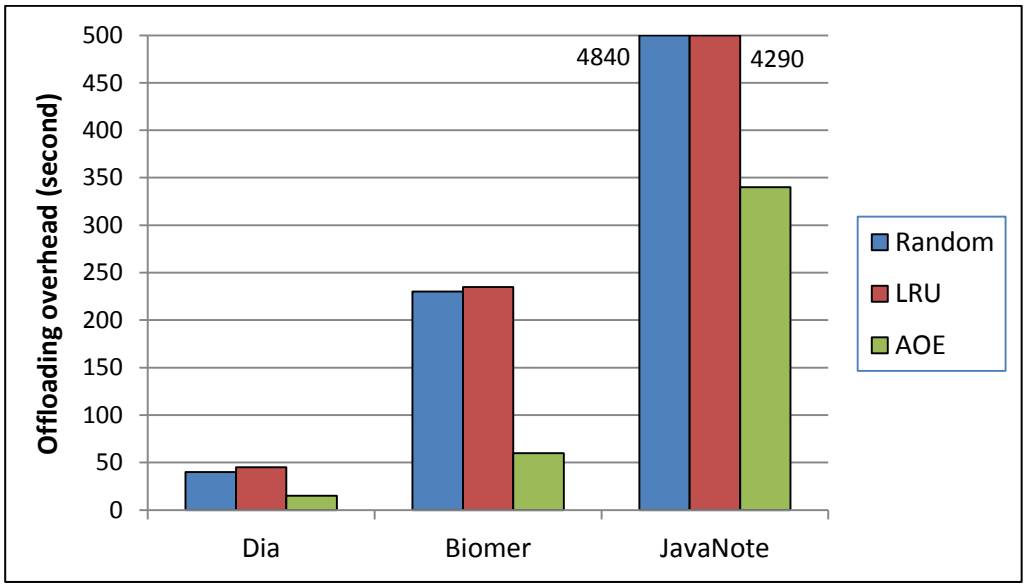

Figure 7.9: Total offloading overhead by different offloading algorithms.

The results show that the *AOE* algorithm consistently selects classes more accurately, thereby achieving much less overhead than the *random* and *LRU* algorithms. The reason behind this is that unlike *AOE*, both *random* and *LRU* do not consider inter-class dependencies and cannot adaptively trigger the offloading action according to the fluctuations in the wireless network. Moreover, *random* and *LRU* do not solve the problem with large classes, which causes large migration overhead during offloading. The proposed approach *AOE* especially reduces the offloading overhead for the highly memory intensive applications Biomer and JavaNote. The performance improvements by *AOE* can be as high as 66% for Dia, 73% for Biomer, and 94% for JavaNote while compared with *random* and *LRU*. This demonstrates that the combined selection metric (Equation 7.1) selects the appropriate application partitioning very effectively. The reason is that the *AOE* algorithm comprehensively considers all inter-class interactions to properly split the application into two least-connected partitions.

Average Interaction Delay

The second performance metric considered is the *average interaction delay*; measured by, $\frac{(\sum RemoteDataAccess*RTT+\sum RemoteFunctionCall*RTT/2)}{\sum(DataAccess+FunctionCall)}$. This metric represents the average interaction time stretch caused by remote data accesses and remote function calls. The average interaction delay metric is very important for interactive applications because they are sensitive to the response time of each interaction. It is of primary importance that the dynamic offloading does not significantly compromise the responsiveness of the application.

For each remote data access, the interaction delay is the time required to send the request to the remote site and to receive the requested data from the remote site. This is close to the *RTT* of the wireless connection. For each remote function call, the interaction delay is the time required to send the function request and its parameters to the remote site. This is close to half of the *RTT* for the wireless connection.

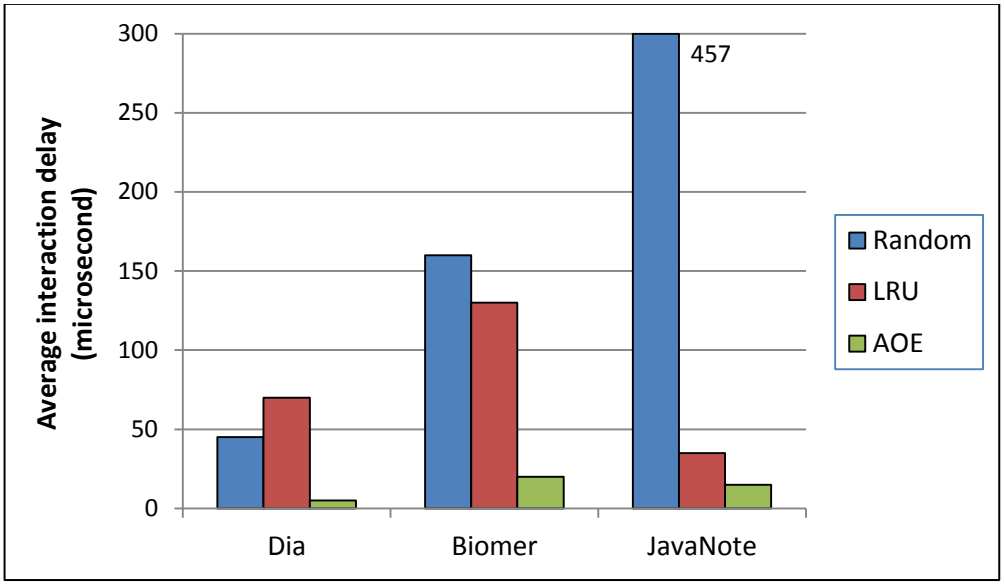

Figure 7.10: Average interaction delay by different offloading algorithms.

Figure 7.10 illustrates the average interaction delays for the *random*, *LRU*, and *AOE* offloading algorithms respectively. The results

again show that *AOE* approach achieves the best performance. The delay reduction can be as high as 87.5% for Biomer, 93% for Dia, and 97% for JavaNote while compared with *random* and *LRU*. The reason for the delay reduction is that *AOE* algorithm reduces the number of remote data accesses and remote function calls to the minimum, by explicitly considering interactions between classes during offloading.

Total Bandwidth Requirement

The third performance metric considered is the *total bandwidth requirement*, which is measured by the sum of the total size of the migrated objects and the total size of the parameters that are passed during remote interactions. Figure 7.11 shows the bandwidth requirements for the *random*, *LRU*, and *AOE* offloading algorithms respectively. Again the observation shows that *AOE* approach requires much less bandwidth to offload than the *random* and *LRU* approaches. For all the three applications, *AOE* approach demands much less bandwidth consumption to offload the executions, by generating less remote function calls and less remote data accesses during offloading.

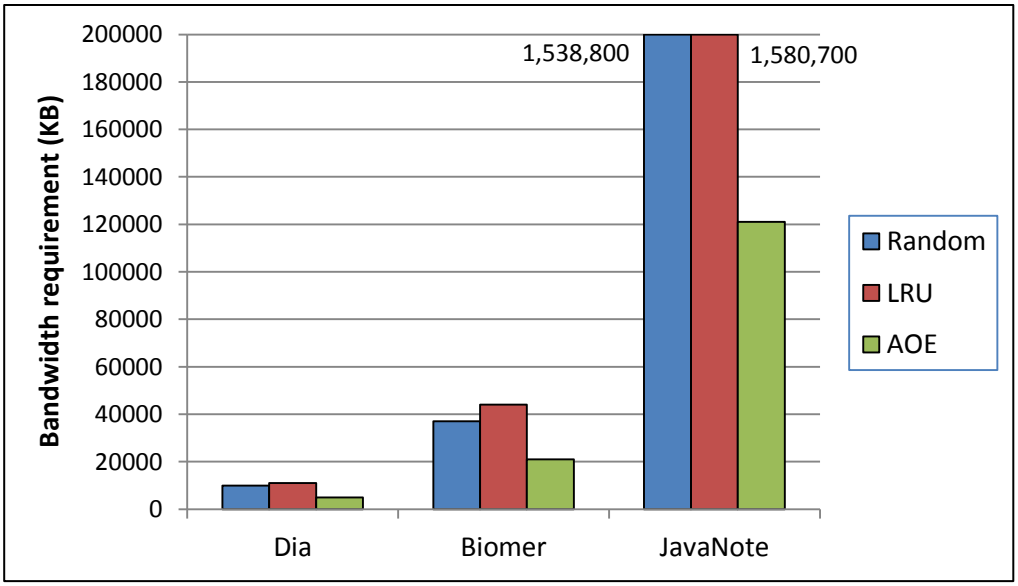

Figure 7.11: Total bandwidth requirements by different approaches.

7.6.2 Evaluation of Other Factors Affecting Performance

In this section, three important factors are examined related to the offloading overhead which affects the performance of offloading. The factors are: dynamic selection of partitioning policies, component granularity, and native method execution. The investigations and observations are discussed below.

Effect of Policy on Performance

To evaluate the effect of triggering and partitioning policies on the remote execution overhead, the same execution traces were repartitioned under multiple policies. The partition triggering threshold was varied from 2% to when 50% of memory remained free, the tolerance to low-memory signals from the garbage collector was varied from one to three events, and the minimum amount of memory to free was varied from 10% to 80%.

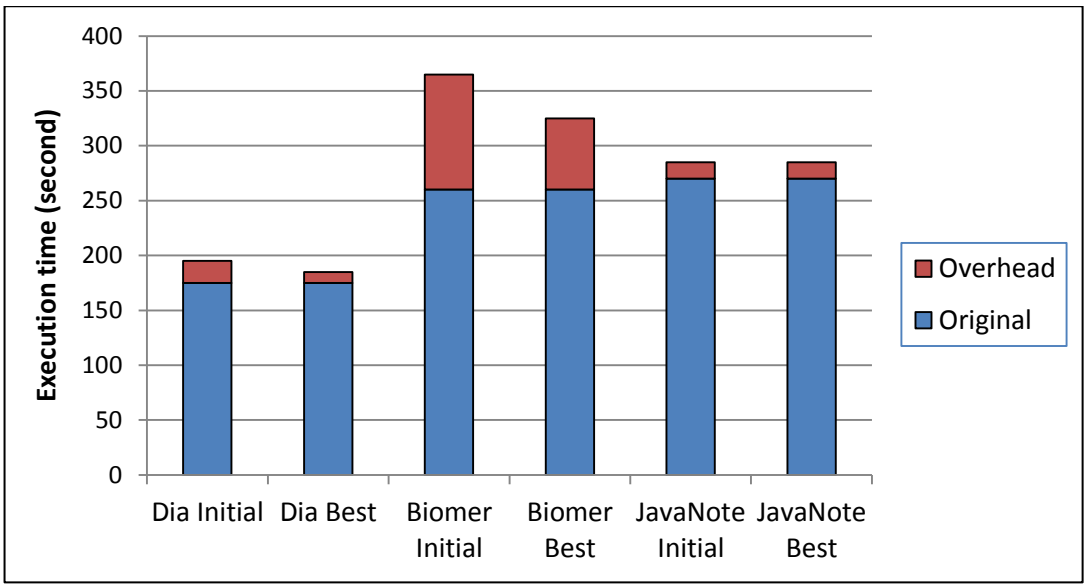

Figure 7.12: Effect of different partitioning policies on execution overhead.

Figure 7.12 illustrates the minimum overhead observed for the best policy for each application. From the experiments, the remote execution overhead was reduced for Biomer and Dia by 30% to 43%, but remained the same for JavaNote. Both Biomer and Dia happened to

perform best with a triggering threshold of 50%, one low-memory signal, and a minimum of 80% memory freed. However, JavaNote performed best with a triggering threshold of 5%, three low-memory signals, and a minimum of 20% memory freed. This indicates that the system needs to be able to dynamically select among partitioning policies and adjust policy parameters to achieve the best performance. This can be achieved based on analysis, knowledge about the type of the application, and so forth.

Effect of Granularity on Partitioning

One consequence of using a class as the unit of placement is that the total amount of memory associated with the objects in a single class can represent a large percentage of the memory available for offloading. As all of the objects of a class must be placed on the same site, therefore fewer options remain available for the partitioning policy to be effective. For example, in JavaNote the primitive character arrays (which contain the document being edited, menu items, etc.) account for a large percentage of the available memory. A class can be composed of groups of unrelated objects that are used by the application in different ways. Using a class as the unit of placement forces all of its objects to reside on one site, even though some of the objects can be highly referenced on the other site. Therefore, in some particular cases forcing all of the objects to reside on one site can cause a higher remote execution overhead. For such particular cases, it might be desirable to be able to offload just a selected group of objects from a class to improve performance.

Effect of Native Methods on Performance

The next observation is to study how much the applications depend on native methods that must execute on the mobile device. The number of references from the execution on the surrogate to native methods is measured, which are required to execute on the mobile device. Figure

7.13 shows that for some applications native methods can account for quite a large percentage of remote accesses (e.g., JavaNote, Dia); while for others it is a relatively small percentage.

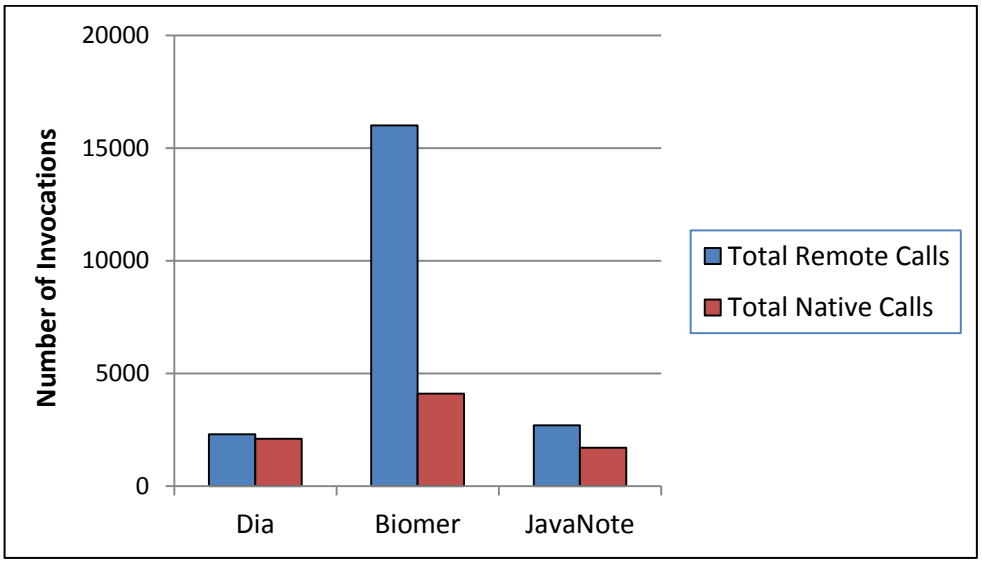

Figure 7.13: Effect of native method invocations on performance.

In fact, many of the native methods do not need to be executed only on the mobile device. The reason is that they are stateless and/or idempotent operations, such as string copy or mathematical functions. However, some native methods which use the resources of the mobile device will always have to be executed locally, such as graphical framebuffer access. Given this, it can be expected that the number of remote calls caused by native methods can be significantly reduced by annotating the methods in the standard Java library according to their operation types. Further, significant gains can be achieved by identifying the native methods that are stateless and executing them on the device where they are invoked. In this approach, the native methods must have the same interface and behavior on both devices.

7.7 Summary

This chapter presented an adaptive offloading system for mobile hosted pervasive services, which includes the offloading platform support and the offloading inference engine. The adaptive offloading system makes offloading decisions without any prior knowledge about the application's execution or network conditions in the pervasive environment. Firstly, decision-making issues in adaptive offloading triggering are addressed. Secondly, the Fuzzy Control model is used to achieve adaptability, configurability and stability when making the triggering decision for offloading. Thirdly, a composite metric for selecting the efficient partitioning is proposed that can simultaneously satisfy multiple user requirements. The extensive evaluations performed show that with the offloading inference engine, runtime offloading can effectively relieve resource constraints for mobile devices, achieving much lower overhead than other common approaches. The prototype experiments show that the execution and memory overheads introduced by the adaptive offloading system are acceptable in terms of achieved performance.

Chapter 8

Conclusions and Future Work

This chapter concludes the thesis with a summary of the findings and discusses the future research directions from this work. The chapter discusses what has been achieved in this research and explains whether the goals set at the beginning of the thesis have been accomplished. Finally, the chapter draws some concluding remarks based on the outcomes and potentials of the thesis.

8.1 Summary of Thesis Contributions

This thesis has made a direct contribution to the field of service provisioning from resource constrained mobile devices, by providing a comprehensive framework that can integrate web services and P2P platforms. The thesis concludes that it is now feasible to deliver business services from mobile devices like smartphones due to the advances in mobile technologies and the use of task offloading mechanisms in pervasive environments. The work incorporates vigorous investigations into the performance characteristics of the offloading system, in considering the dynamic condition of processing power, memory space, and communication bandwidth of the mobile device. The most substantial contributions of the thesis are summarized below.

- ***Scalability and integration of mobile hosted services:*** Communication using verbose XML based SOAP messages introduces message overhead which hinders the scalability of

mobile hosted services. SOAP message based invocation consumes resources of the mobile device extensively. On the other hand, compression of SOAP messages leads to additional execution workload, which causes performance latencies.

Therefore, the thesis thoroughly studied the performance penalties of different compression techniques to identify the best encoding method for mobile hosted service provisioning. The observations identified BinXML as the most suitable compression option available for mobile hosted service invocations. The integration issues for mobile hosted services were also found to be compatible with traditional web services. The study showed the suitability of the ESB technology, the JBI specification, and the ServiceMix tool in realizing the integration framework.

- ***Mobile service provider in P2P environments:*** The thesis has introduced the idea of using smartphones to provide mobile services in P2P networks. The features, deployment scenario, and realization details of the P2P based web services framework have been presented. The thesis realized mobile service provisioning using the JXTA network in P2P environments. 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 of mobile P2P services with traditional web services platforms. An alternative service discovery mechanism is proposed in P2P networks using the modules feature of JXTA. The evaluation of the framework clearly shows that the mobile services gateway has reasonable levels of performance in handling the large number of concurrent clients possible in mobile P2P networks.

- ***Partitioning mobile hosted service execution:*** The thesis introduced a framework for partitioning the application execution of the services provided from resource constrained mobile devices. Hosting complex services on mobile devices demands a flexible and scalable execution environment, which can exploit backend servers to perform heavy-duty executions in a distributed manner. The framework facilitates the use of application partitioning techniques to offload some workload of the services to backend nodes, and keeps the web service interfaces on mobile devices.

  The architecture of the partitioned service execution environment is designed using the light-weight and open-source packages kXML and kSOAP. The components of the service execution engines are provided with their functionalities in the architecture. The framework is tested using sample prototypes and web services to observe the performance of different partitioning techniques. The experiments show that partitioning the computation intensive tasks leads to a significant improvement in response time of the services hosted on mobile devices.

- ***Efficient partitioning approach:*** The thesis formulated an efficient application partitioning approach for pervasive services hosted on resource constrained mobile devices. An ideal partitioning solution must consider the dynamic conditions of the resources in partitioning the tasks between a mobile device and its backend nodes. The proposed partitioning algorithm considers the context of memory space, computation power, and transmission bandwidth to ascertain the dynamic condition of the mobile device in partitioning the computation intensive tasks for offloading.

  The thesis proposed an efficient $(k+1)$ partitioning algorithm that considers a combination of the cost values of resources for the classes that can be offloaded. The partitioning approach also considers the interaction properties and

dependencies between the interacting classes while performing execution partitioning. Experimental results have shown the effectiveness of the proposed approach, by significantly increasing the efficiency of mobile hosted services in dynamic resource conditions.

- ***Adaptive offloading system:*** The thesis devised an adaptive offloading system for mobile hosted pervasive services, which can beneficially offload execution from mobile devices. The adaptive offloading system can trigger offloading at the right time and offload the right tasks to achieve low offloading overheads. The proposed offloading system includes two cooperating parts; namely, the distributed offloading platform and the offloading inference engine. The offloading decisions are made without prior knowledge about the application's execution or network conditions in the pervasive environment.

  The adaptive offloading system uses the Fuzzy Control model to make effective offloading decisions and to achieve adaptability, configurability, and stability in triggering execution offloading. A composite cost metric which represents the interaction properties between executing classes is used by the offloading system for efficiently selecting class partitioning at run time. Experimental evaluation has shown that the offloading inference engine can effectively relieve resource constraints of mobile devices, achieving much lower overhead than commonly used previous approaches. The prototype experiments have demonstrated that the execution and memory overheads introduced by the adaptive offloading system are acceptable in terms of achieved performance.

8.2 Future Research Directions

While the thesis contributes a complete framework for provisioning services from resource constrained mobile devices, it also raises several issues which need to be explored further in the future. In particular, security of mobile hosted services, context-aware service discovery, mobility of the service provider in handling surrogates, and support for engaging multiple surrogates are the most important research issues to be addressed.

- ***Security of mobile hosted services:*** For the traditional stationary web services, many standardized security specifications, protocols, and implementations exist. But not much has been explored and standardized for mobile hosted services in wireless environments. The mobile service provider has to provide secure and reliable communication in volatile mobile networks. Basic service-level authentication and user-intervened authorization are not sufficient for providing proper end-point security for mobile hosted services. Therefore, maintaining the private keys of the users and hardware level support for securing mobile hosted services, are important issues that must be explored further.

- ***Context-aware service discovery:*** Mobile hosted service discovery is another important direction for further research. Commonly used service discovery mechanisms for stationary networks, are obviously not suited for mobile hosted services discovery. Effective discovery mechanisms that support the dynamic and spontaneous nature of mobile environments need to be developed. The thesis has demonstrated an alternative for publishing and discovery of mobile web services with the help of P2P networks. To facilitate effective service discovery in mobile environments, a context-aware service discovery mechanism is

needed, which can provide semantic matching for mobile hosted services. In addition, mechanisms for deploying mobile hosted services in decentralized P2P networks need to be developed, so that services can be discovered in ad-hoc environments.

- ***Mobility support in handling surrogates:*** Mobility of the mobile service providers is another important issue to be examined in terms of supporting execution offloading. New strategies are needed to handle situations where a mobile service host moves from one surrogate's region to that of another surrogate. Cloud based architectures can be a potential solution for the mobility management of mobile hosts and supporting execution offloading to surrogates. Another interesting issue for further research in this direction is supporting the use of multiple surrogates for offloading. That would essentially enable concurrent execution of multiple partitions on multiple surrogates in parallel.

8.3 Concluding Remarks

This thesis has opened up numerous scopes and opportunities for further research in provisioning business services from mobile devices. The emergence of smartphones and the development of cloud based resources clearly indicate the future prospect of mobile hosted services. The thesis mainly focused on designing a light-weight and efficient framework for providing services involving complex business processes on mobile devices. The framework provides a distributed platform for executing functions locally which require the resources of the mobile device and offloading the resource-demanding tasks to surrogates. The offloading system adaptively considers both the interaction properties and the resource consumptions while performing execution partitioning and offloading.

Nevertheless, some factors can affect the performance of the proposed framework. Firstly, error modes (e.g., loss of communication with the surrogate) can affect the distributed execution of service applications. In this regard, the assumption is that wireless networking technologies such as Ultra-Wideband in the future will provide fairly good communication coverage and connectivity. Secondly, not all applications are readily possible to be partitioned, because of being badly written (e.g., use of a single large class rather than an object-oriented approach). However, it can be expected that most service applications do not fall into this category and are good candidates for partitioning to offload their execution.

It is envisaged that the future phases of this research will realize the convergence of various computing domains; such as, pervasive computing, ubiquitous computing, context-aware computing, service-oriented computing, and mobile cloud computing. As a result of this research, the expectation is that mobile hosted services will emerge as a universal technology, capable of supporting all kinds of personalized services.

Bibliography

3GPP (2007), Third Generation Partnership Project, [http://www.3gpp.org/].

4GPress (2005), World's First 2.5Gbps Packet Transmission in 4G Field Experiment, 4G Press, [http://www.4g.co.uk/PR2006/ 2056.htm].

ABI Research (2010), [http://www.abiresearch.com/].

AgileDelta (2007), Efficient XML: Lightning-Fast Delivery of XML to More Devices in More Locations, [http://www.agiledelta.com/product_efx.html].

Allan, R. (2004), Systinet WASP, [http://tyne.dl.ac.uk/ReDRESS/WebServices/].

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004), *Web Services Concepts, Architectures and Applications*, Springer Verlag.

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002), SETI@home: An experiment in public-resource computing, *Communications of the ACM*, 45(11):56–61.

Android (2010), Android dev phone 1, [http://www.code.google.com/android/dev-devices.html].

Apache Geronimo (2007), Welcome to Apache Geronimo - Home Page, [http://geronimo.apache.org/].

Apache Software Foundation (2007a), Apache Axis2/Java - Version 1.2, [http://ws.apache.org/axis2/].

Apache Software Foundation (2007b), Apache Benchmark, [http://svn.apache.org/repos/httpcomponents/httpcore/truk/contrib/benchmark/].

Apache Software Foundation (2007c), Apache ServiceMix, [http://incubator.apache.org/servicemix/home.html].

Apache Software Foundation (2007d), Web Services – Axis, [http://ws.apache.org/axis/].

Apache ServiceMix (2007), Apache ServiceMix 3.x Users' Guide, Apache ServiceMix community, [http://incubator.apache.org/servicemix/users-guide.html].

Apte, N., Deutsch, K., and Jain, R. (2005), Wireless SOAP: Optimizations for Mobile Wireless Web Services, In *the posters of the 14th international conference on World Wide Web*, pages 1178 – 1179, ACM Press.

ARM (2005), ARM Product Backgrounder, Technical report, ARM, [http://www.arm.com/miscPDFs/3823.pdf].

Balan, K., Satyanarayanan, M., Park, S. and Okoshi, T. (2003), Tactics-Based Remote Execution for Mobile Computing, In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, California, USA, pp. 273–286.

Balani, N. (2003a), Deliver Web services to mobile apps, IBM DeveloperWorks, [http://www-128.ibm.com/developerworks/wireless/edu/].

Balani, N. (2003b), Using kXML to access XML files on J2ME devices, IBM DeveloperWorks, [http://www-128.ibm.com/developerworks/edu/].

Ballinger, K., Ehnebuske, D., Ferris, C., Gudgin, M., Karmarkar, A., Liu, C. K., Nottingham, M., and Yendlurif, P. (2007), Basic Profile Version 1.2, Technical report, The Web Services-Interoperability Organization (WS-I), [http://www.ws-i.org/Profiles/BasicProfile-1.2.html].

BEA AquaLogic (2007), BEA AquaLogic Service Bus 2.6: The Enterprise Service Bus for the Agile Business (Product Data Sheet), Technical report, BEA Systems, Inc.

Bellwood, T. (2002), UDDI Version 2.04 API Specification, Technical report, UDDI Committee Specification, [http://uddi.org/pubs/ProgrammersAPI_v2.htm].

Benatallah, B. and Maamar, Z. (2003), Introduction to the special issue on M-services, *IEEE transactions on systems, man, and cybernetics - part a: systems and humans*, 33(6):665–666.

Belov, N., Braude, I., Krandick, W., and Shaffer, J. (2005), Wireless Internet Collaboration System on Smartphones, In *3rd International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS 2005), a CAiSE'05 workshop*.

Bilorusets, R., Box, D., Cabrera, L. F., Davis, D., Ferguson, D., Ferris, C., Freund, T., Hondo, M. A., Ibbotson, J., Jin, L., Kaler, C., Langworthy, D., Lewis, A., Limprecht, R., Lucco, S., Mullen, D., Nadalin, A., Nottingham, M., Orchard, D., Roots, J., Samdarshi, S., Shewchuk, J., and Storey, T. (2005), Web Services Reliable Messaging Protocol (WS-ReliableMessaging), Technical report, BEA Systems, IBM, Microsoft Corporation Inc. and TIBCO Software Inc.

Blackberry (2010), Blackberry smartphones. [http://na.blackberry.com/eng/].

Bolcer, G. A., Gorlick, M., Hitomi, A. S., Kammer, P., Morrow, B., Oreizy, P., and Taylor, R. N. (2000), Peer-to-Peer Architectures and the Magi*TM* Open-Source Infrastructure, Technical report, Endeavors Technology, Inc.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004), Web Services Architecture, Technical report, W3C Working Group Note, [http://www.w3.org/TR/ws-arch/].

Borck, J. R. (2005), Enterprise service buses hit the road, Infoworld, pages 26–40, [http://www.infoworld.com/article/].

Boyer, J. (2001), Canonical XML, Version 1.0, Technical report, W3C Recommendation, [http://www.w3.org/TR/xml-c14n].

Brisaboa, N. R., Luaces, M. R., Parama, J. R., and Viqueira, J. R. (2007), Managing a Geographic Database from Mobile Devices Through OGC Web Services, In *International Workshop on DataBase Management and Application over Networks (DBMAN 2007)*.

Brown, N. and Kindel, C. (1998), Distributed Component Object Model Protocol – DCOM/1.0, Technical report, Microsoft Corporation, [http://samba.osmirror.nl/samba/ftp/specs/draft-brown-dcom-v1-spec-03.txt].

Burbeck, S. (2000), The Tao of e-business services - The evolution of Web applications into service-oriented components with Web services, IBM DeveloperWorks, [http://www-128.ibm.com/developerworks/webservices/library/ws-tao/].

Carlsson, B. and Gustavsson, R. (2001), The Rise and Fall of Napster - An Evolutionary Approach, In *Proceedings of the 6th International Computer Science Conference on Active Media Technology*, pages 347 – 354, Springer-Verlag.

Chandra, D., Fensch, C., Hong, W., Wang, L., Yardımci, E. and Franz, M. (2002), Code Generation At The Proxy: An Infrastructure-Based

Approach To Ubiquitous Mobile Code, In *Proceedings of the 5th ECOOP Workshop on Object-Orientation and Operating Systems (ECOOP-OOOSWS02)*, Malaga, Spain.

Chappell, D. A. (2004), *Enterprise Service Bus*, O'Reilly Media, Inc.

Chatti, M. A., Srirama, S., Kensche, D., and Cao, Y. (2006), Mobile Web Services for Collaborative Learning, In *Fourth IEEE International Workshop on Wireless, Mobile and Ubiquitous Technology in Education - (WMUTE'06)*, pages 129–133.

Chen, G., Kang, B. and Kandemir, M. (2004), Studying energy trade-offs in offloading computation/compilation in java-enabled mobile devices, *IEEE Transactions on Parallel and Distributed Systems 15(9)*.

Cheney, J. (2001), Compressing XML with Multiplexed Hierarchical PPM Models, In *Proceedings of IEEE Data Compression Conference*, pages 163–172.

Chun, B. G., and Maniatis, P. (2009), Augmented smartphone applications through clone cloud execution, In *Proceedings of HotOS 2009*.

Chun, B. G., Ihm, S., Maniatis, P., Naik, M. and Patti, A. (2011), CloneCloud: Elastic Execution between Mobile Device and Cloud, In *Proceedings of EuroSys'11*, Salzburg, Austria.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001), Web Services Description Language (WSDL) 1.1, Technical report, W3C Note, [http://www.w3.org/TR/wsdl].

CIA (2006), Smartphones to Outsell PDAs by 5:1 in 2006, Computer Industry Almanac Inc., [http://www.c-i-a.com/pr0306.htm].

Clarke, I., Miller, S. G., W.Hong, T., Sandberg, O., and Wiley, B. (2002), Protecting Free Expression Online with Freenet, *IEEE Internet Computing*, 6(1):40–49.

Cleary, J. G. and Teahan, W. J. (1997), Unbounded Length Contexts for PPM, *The Computer Journal*, 40(2/3):67–75.

Cunnings, R., Fell, S., and Kulchenko, P. (2001), SMTP Transport Binding for SOAP 1.1, [http://www.pocketsoap.com/specs/smtpbinding/].

Contreras, P., Zervas, D., and Murtagh, F. (2005), Web Services in Natural Language: Towards an Integration of Web Service and Semantic

Web Standards in the JXTA Peer to Peer Environment, [http://thames.cs.rhul.ac.uk/wstalk/papers/].

CORBA (2007), Common Object Request Broker Architecture, [http://www.corba.org/].

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. (2002), Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI, *IEEE Internet Computing*, 6(2):86–93.

Cutting, D. (2007), Apache Lucene, [http://lucene.apache. org/].

Dao, T. (2005), UDDI Explorer: Tool for searching web services, The Code Project.

Davis, D. and Parashar, M. (2002), Latency Performance of SOAP Implementations, In *IEEE Cluster Computing and The Grid 2002.*

DCOM (2007), Distributed Component Object Model (DCOM), Microsoft Corporation, [http://msdn2.microsoft.com/en-us/library/ms809332.aspx].

Dustdar, S. and Treiber, M. (2006), Integration of transient Web services into a virtual peer to peer Web service registry, *Distributed and Parallel Databases*, 20:91–115.

Ellis, J. and Young, M. (2003), J2MEWeb Services 1.0 - Final Draft (JSR 172), Technical Report 11, Sun Microsystems, Inc.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., and Newling, T. (2004), *Patterns: Service-Oriented Architecture and Web Services*, IBM Redbooks, [http://www.redbooks.ibm.com/abstracts/sg246303.html].

Engelen, R. A. and Gallivan, K. A. (2002), The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks, In *The proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002).*

Ericsson (2003a), Enhanced Data Rates for GSM Evolution (EDGE) - Introduction of high-speed data in GSM/GPRS networks, Technical report, Ericsson AB, [http://www.ericsson.com/technology/whitepapers/edge_wp_technical.pdf].

Ericsson, M. (2003b), A study of compression of XML-based messaging, Technical report, Växjö University.

Ericsson, M. and Levenshteyn, R. (2003), On optimization of XML-based messaging, In *Second Nordic Conference on Web Services (NCWS 2003)*, pages 167–179.

ETC (2007), New Media Review, European Travel Commission, [http://www.etcnewmedia.com/review/default.asp?sectionid=10&overviewid=6].
ETSI (1997), GSM Technical Specification, GSM 03.64: General Packet Radio Service (GPRS), Technical report, European Telecommunications Standards Institute (ETSI).

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999), Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, Technical report, Network Working Group, [http://www.w3.org/Protocols/rfc2616/rfc2616.html].

Flinn, J., Park, S. and Satyanarayanan, M. (2002), Balancing performance, energy, and quality in pervasive computing, In *Proceedings of 22nd IEEE International Conference on Distributed Computing Systems*, Austria.

Forum Nokia (2004), Introduction to Web Services in Nokia Devices – Version 1.0, Technical report, Forum Nokia, [http://sw.nokia.com/].

Forum Nokia (2007a), Nokia Mobile Web Services Framework, Architecture, APIs, SDK, Forum Nokia, [http://www.forum.nokia.com/main/resources/technologies/web_services/].

Forum Nokia (2007b), Nokia Series 60 Platform, Forum Nokia, [http://www.forum.nokia.com/main/platforms/s60/].

Forum Nokia (2007c), Nokia Series 80 Platform, Forum Nokia, [http://www.forum.nokia.com/main/platforms/s80/].

Fox, D. and Box, J. (2004), *Building solutions with the Microsoft .NET compact framework architecture and best practices for mobile development*, Addison-Wesley, Boston, ISBN 0321197887 (9780321197887), ID: 52948769.

Frank, S. P. (2004), Personal Java and Inferno for Today's Consumer Devices, *Java Developers Journal*, 3(3).

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, 1st edition.

Gehlen, G., Aijaz, F., and Walke, B. (2006), Mobile Web Service Communication over UDP, In *Proceedings of the 64th IEEE Vehicular Technology Conference*.

GeoNames (2009), GeoNames geographical database, [http://www.geonames.org/].

Girardot, M. and Sundaresan, N. (2000), Millau: an encoding format for efficient representation and exchange of XML over the Web, In *Proceedings of the Ninth International World Wide Web Conference*.

Google (2007), Google Maps, [http://maps.google.com/].

Google Mobile (2007), Search, find and browse with Google on your mobile device, [http://www.google.com/mobile/search/].

Gospodnetic, O. and Hatcher, E. (2007), Meet Lucene, [http://www.developer.com/java/other/article.php/].

Gottschalk, K., Graham, S., Kreger, H., and Snell, J. (2002), Introduction to Web Services architecture, *IBM Systems Journal: New Developments in Web Services and E-commerce*, 41(2):178–198.

Greenberg, I. (2005), Memory breakthroughs will propel smartphone development – Transition from NOR to NAND flash memory a key, *Smartphone & Pocket PC magazine*.

Grimshaw, A. S., Wulf, W. A., and THE LEGION TEAM (1997), The Legion vision of a worldwide virtual computer, *Communications of the ACM*, 40(1):39–45.

GSM1 2009, GSM World: GSM - The Wireless Evolution, viewed 14 May 2009, [http://www.gsmworld.com/technology/].

Gu, X., Nahrstedt, K., Messer, A., Greenberg, I. and Milojicic, D. (2003), Adaptive offloading inference for delivering applications in pervasive computing environments, In *Proceedings of 1st International Conference on Pervasive Computing and Communications*, IEEE Press.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H. F. (2003), SOAP Version 1.2 Part 2: Adjuncts, Technical report, World Wide Web Consortium Recommendation, [http://www.w3.org/TR/REC-soap12-part2-20030624/].

Gudgin, M., Combs, H., Justice, J., Kakivaya, G., Lindsey, D., Orchard, D., Regnier, A., Schlimmer, J., Simpson, S., Tamura, H., Wright, D., and Wolf, K. (2004), SOAP-over-UDP, Technical report, BEA Systems Inc.,

Lexmark, Microsoft Corporation, Inc, and Ricoh, [http://specs.xmlsoap.org/ws/2004/09/soap-over-udp/].

Gudgin, M., Mendelsohn, N., Nottingham, M., and Ruellan, H. (2005), SOAP Message Transmission Optimization Mechanism, Technical report, W3C Recommendation, [http://www.w3.org/TR/soap12-mtom/].

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. J., Nielsen, H. F., Karmarkar, A., and Lafon, Y. (2007), SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), Technical report, World Wide Web Consortium Recommendation, [http://www.w3.org/TR/soap12-part1/].

Gulbrandsen, A., Vixie, P., and Esibov, L. (2000), A DNS RR for specifying the location of services (DNS SRV) (RFC 2782), Technical report, Network Working Group, [http://rfc.sunsite.dk/rfc/rfc2782.html].

Hajamohideen, S. H. (2003), *A Model for Web Service Discovery and Invocation in JXTA*, Master's thesis, Technical University Hamburg-Harburg.

Halteren, A. and Pawar, P. (2006), Mobile Service Platform: A Middleware for Nomadic Mobile Service Provisioning, *2nd IEEE International Conference On Wireless and Mobile Computing, Networking and Communications (WiMob 2006).*

Hanson, J. J. (2005), ServiceMix as an enterprise service bus: Use ServiceMix 2.0 as a service-oriented message routing framework, JavaWorld.com, [http://www.javaworld.com/javaworld/jw-12-2005/jw-1212-esb.html].

Hao, W., Gao, T., Yen, I-L., Chen, Y. and Paul, R. (2006), An Infrastructure for Web Services Migration for Real-Time Applications, In *Proceedings of 2nd IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*, pp. 41-48, Shanghai, China.

Harjula, E., Ylianttila, M., Ala-Kurikka, J., Riekki, J., and Sauvola, J. (2004), Plug-and-play application platform: Towards mobile peer-to-peer, In *Third International Conference on Mobile and Ubiquitous Multimedia (MUM2004)*, pages 63–69.

Harrison, R. (2003), Symbian OS C++ for Mobile Phones, John Wiley and Sons Ltd.

Hassan, M. (2009), Mobile Web Service Provisioning in Peer to Peer Environments, In *Proceedings of IEEE International Conference on*

Service-Oriented Computing and Applications (SOCA 2009), pp. 138-141, Taipei, Taiwan.

Hassan, M., Zhao, W. and Yang, J., (2010), Provisioning Web Services From Resource Constrained Mobile Devices, In *Proceedings of Third IEEE 2010 International Conference on Cloud Computing (CLOUD 2010), pp. 490-497*, Miami, Florida, USA.
Heine, G. (1999), *GSM Networks: Protocols, Terminology and Implementation*, ISBN 0-8900-6471-7.

Helal, A., Haskell, B., Carter, J., Brice, R., Woelk, D., and Rusinkiewicz, M. (1999), *Any Time, Anywhere Computing; Mobile Computing Concepts and Technology*, Kluwer Academic Publishers.

Hemmati, H., Ranjbar, A., Niamanesh, M. and Jalili, R. (2005), A Model to Support Context-Aware Service Migration in Pervasive Computing Environments, In *Proceedings of 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, VTT Elektroniikka, Florida, USA.

Heuer, J., Thienot, C., and Wollborn, M. (2002), *Introduction to MPEG-7: Multimedia Content Description Interface*, Chapter Binary Format, pages 61–80, Jon Wiley and Son.

Hirsch, F., Kemp, J. and Ilkka, J. (2006), *Mobile Web Services: Architecture and Implementation*, John Wiley and Sons, ISBN 0-470-01596-9.

Hodges, J. and Morgan, R. (2002), Lightweight Directory Access Protocol (v3): Technical Specification (RFC 3377), Technical report, Network Working Group, [`http://www.rfc-editor.org/rfc/rfc3377.txt`].

Hu, N. and Steenkiste, P. (2003), Evaluation and Characterization of Available Bandwidth Probing Techniques, *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*.

Hunt, G. and Scott, M. (1999), The Coign Automatic Distributed Partitioning System, In *Proceedings of the 3rd Symposium on Operating System Design and Implementation (OSDI'99)*, New Orleans, LA, U.S.A., pp. 187-200.

IBM Corporation (2007a), IBM WebSphere Studio Device Developer, IBM developer-Works, [`http://www-306.ibm.com/software/wireless/wsdd/`].

IBM Corporation (2007b), WebSphere Enterprise Service Bus, WebSphere Software, [http://www-306.ibm.com/software/integration/wsesb/].

IBM Corporation (2008), IBM WebSphere MQ - Home page, WebSphere Software, [http://www-01.ibm.com/software/websphere/].

Ichikawa, K. (2002), The View of NTT DoCoMo on the Further Development of Wireless Internet, In *Tokyo Mobile Round Table Conference*.

iPhone (2010), Apple iPhone, [http://www.apple.com/iphone].

Imielinski, T., Viswanathan, S., and Badrinath, B. R. (1994), Power efficient filtering of data on air, In *Proceedings of the 4th international conference on extending database technology on Advances in database technology*, pages 245 – 258, Springer-Verlag.

IONA Technologies (2005), ESB: evolving beyond EAI, Technical report, IONA Technologies, [http://www.itarchitects.ca/whitepaper/ESB-EvolvingBeyondEAI.pdf].

ITU (2010), International Telecommunication Union, [http://www.itu.int/ITU-D/ict/statistics/at_glance/].

java.net (2008), Project Open ESB, [https://open-esb.dev.java.net/index.html].

Jamwal, V. and Iyer, S. (2005), Automated Refactoring of Objects for Application Partitioning, In *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)*, Taipei, Taiwan, pp. 671-678.

JBoss (2007), JBoss Application Server, [http://www.jboss.org/products/jbossas/].

Johnson, D., Perkins, C. and Arkko, J. (2004), Mobility Support in IPv6 (RFC 3775), Technical report, Network Working Group, [http://www.ietf.org/rfc/rfc3775.txt].

Johnson, R. (2005), Introduction to the Spring Framework, TheServerSide.com, [http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework].

JXTA (2008), JXTA home page, [http://www.jxta.org/].

JXTA Community (2007a), JXTA-SOAP, java.net, [https://soap.dev.java.net/].

JXTA Community (2007b), Project JXTA Community, java.net, [https://jxta.dev.java.net/].

Kangasharju, J. (2007), Efficient implementation of xml security for mobile devices, *IEEE International Conference on Web Services, ICWS 2007*, 00:134–141.

Kangasharju, J., Lindholm, T. and Tarkoma, S. (2006), On encrypting and signing binary xml messages in the wireless environment, In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 637–646, Washington DC, USA, IEEE Computer Society, ISBN 0-7695-2669-1.

Kangasharju, J., Lindholm, T. and Tarkoma, S. (2007), XML messaging for mobile devices: From requirements to implementation, *Computer Networks*, 51(16):4634–4654.

Karypis, G. and Kumar, V. (1998a), A fast and highly quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Scientific Computing*.

Karypis, G. and Kumar, V. (1998b), Multilevel *k*-way partitioning scheme for irregular graphs, *Journal of Parallel and Distributed Computing 48 (1) 96–129*.

Karypis, G. and Kumar, V. (1998c), Multilevel algorithms for multi-constraint graph partitioning, In *Proceedings of Super Computing'98*, Orlando, USA.

Kazaa (2009), Kazaa Home Page, [http://www.kazaa.com/].

Keen, M., Bishop, S., Hopkins, A., Milinski, S., Nott, C., Robinson, R., Adams, J., Verschueren, P., and Acharya, A. (2004), *Patterns: Implementing an SOA using an Enterprise Service Bus*, IBM RedBooks, [http://www.redbooks.ibm.com/redbooks/pdfs/sg246346.pdf].

Kefali, U. (2004), *Development and Performance Evaluation of a Simple Object Access Protocol (SOAP) Profile for Block Extensible Exchange Protocol (BEEP)*, Master's thesis, RWTH Aachen University.

Kim, Y. K. and Prasad, R. (2006), *4G Roadmap and Emerging Communication Technologies*, Artech House Publishers.

Knudsen, J. (2002), Getting Started with JXTA for J2ME, Sun Developer Network, [http://developers.sun.com/mobility/midp/articles/jxme/].

kSOAP (2007), kSOAP - An Open Source SOAP for the kVM, ObjectWeb, [http://ksoap.objectweb.org/].

kSOAP2 (2007), kSOAP2 - An efficient, lean, Java SOAP library for constrained devices, SourceForge.net, [http://sourceforge.net/projects/ksoap2].

Kurose, J. F. and Ross, K. W. (2001), *Computer Networking - A Top-Down Approach Featuring the Internet*, Pearson Education Addison Wesley.

Kwok, S. H., Lui, S., Cheung, R., Chan, S., and Yang, C. C. (2003), Searching Behavior in Peer-to-Peer Communities, In *International Conference on Information Technology: Computers and Communications (ITCC 03)*, page 130, IEEE Computer Society.

kXML2 (2007), kXML2 project, SourceForge.net, [http://kxml.sourceforge.net/kxml2/].

LA (2007), Liberty Alliance project, [http://www.projectliberty.org/].

Lai, K. Y., Phan, T. K. A., and Tari, Z. (2005), Efficient SOAP Binding for Mobile Web Services, In *Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)*.

Lee, C., Helal, A., Desai, N., Verma, V., and Arslan, B. (2003), Konark: A System and Protocols for Device Independent, Peer-to-Peer Discovery and Delivery of Mobile Services, *IEEE transactions on systems, man, and cybernetics - part a: systems and humans*, 33(6):682–696.

Leymann, F. (2002), Web Services Flow Language (WSFL1.0), [http://www.306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf].

Leymann, F. (2005), The Service Bus: Services Penetrate Everyday Life, In *Proceedings of the Third International Conference on Service-Oriented Computing (ICSOC 2005)*, Netherlands.

Li, B. and Nahrstedt, K. (1999), A Control-based Middleware Framework for Quality of Service Adaptations, *IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms*, 17(9).

Li, Z., Wang, C. and Xu, R. (2001), Computation offloading to save energy on handheld devices: A partition scheme, In *Proceedings of International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ACM Press.

Liefke, H. and Suciu, D. (1999), XMill: an efficient compressor for XML data, Technical report, University of Pennsylvania.

LocatioNet (2007), *amAze* - Free GPS Navigation for your mobile phone, LocatioNet Solutions, [http://www.amazegps.com/].

Loup Gailly, J. (2007), The GZip Home page, [http://www.gzip.org/].

MacVittie, L. (2006), Review: ESB Suites - Make Way for the Enterprise Service Bus, Technical report, Network Computing.

Martin, B. and Jano, B. (1999), WAP Binary XML Content Format, Technical report, W3C NOTE, [http://www.w3.org/TR/wbxml/].

McFaddin, S., Narayanaswami, C. and Raghunath, M. (2003), Web Services on Mobile Devices – Implementation and Experience. In *Proceedings of the Fifth IEEE Workshop on Mobile Computing Systems & Applications (WMCSA'03)*, pp. 100-109, Monterey, California, USA.

Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. (2003), Peer-to-Peer Computing. Technical report, HP Laboratories Palo Alto, [http://www.hpl.hp.com/techreports/].

Mitra, N. and Lafon, Y. (2007), SOAP Version 1.2 Part 0: Primer (Second Edition), Technical report, W3C Recommendation.

Moyo, T., Irwin, B. and Wright, M. (2006), Securing mobile commerce interactions through secure mobile web services, In *8th Annual Conference on WWW Applications: South Africa*.

MPEG-4 (2007), IBM toolkit for MPEG-4, [http://www.alphaworks.ibm.com/tech/tk4mpeg4].

MSDN (2007a), .NET Compact Framework, Microsoft Corporation, [http://msdn2.microsoft.com/en-us/netframework/aa497273.aspx].

MSDN (2007b), .NET Framework, Microsoft Corporation, [http://msdn2.microsoft.com/en-us/netframework/].

MSDN (2007c), Web Services Enhancements (WSE), Microsoft Corporation, [http://msdn2.microsoft.com/en-us/webservices/].

Nadalin, A. (2003), SOAP Message Security: Minimalist Profile (MProf), Draft Technical Specification, OASIS Open, [http://docs.oasis-open.org/wss/].

Nadalin, A., Kaler, C., Monzillo, R., and Hallam-Baker, P. (2006), Web Services Security: SOAP Message Security 1.1(WS-Security 2004), Technical Specification, OASIS Open, [http://www.oasis-open.org/committees/download.php/16790/].

NAVSTAR (1995). Global Positioning System Standard Positioning Service Signal Specification - 2nd Edition, Technical report, U.S. Coast Guard Navigation Center, [http://www.navcen.uscg.gov/pubs/gps/sigspec/gpssps1.pdf].

NAVSTAR (2007), USNO NAVSTAR Global Positioning System, NAVSTAR GPS Operations, [http://tycho.usno.navy.mil/gpsinfo.html].

Nexus (2010), Nexus One, [http://www.google.com/phone].

Novak, L. and Svensson, M. (2001), MMS - building on the success of SMS, Ericsson Review No. 3, pages 102–109, [http://www.ericsson.com/ericsson/corpinfo/publications/review/2001_03/].

NTT DoCoMo (2005), NTT DoCoMo Unveils 505i Series i-mode compatible Mobile Phones Equipped for Macromedia Flash and Enhanced Java-Based Applications, NTT DoCoMo Press Releases, [http://www.nttdocomo.com/pr/2003/000946.html].

NTT DoCoMo (2007a), i-mode Business Model, [http://www.nttdocomo.com/services/imode/business/index.html].

NTT DoCoMo (2007b), i-mode Technology, [http://www.nttdocomo.com/technologies/present/imodetechnology/index.html].

NTT DoCoMo (2007c), NTT DoCoMo: HOME, [http://www.nttdocomo.com/].

OMA (2004), Open mobile alliance overview, Technical report, Open Mobile Alliance Group, [http://www.openmobilealliance.org/docs/].

OMA (2006a), Mobile Web Services Requirements - Version 1.1, Technical report, Open Mobile Alliance Group, [http://www.openmobileallian ce.org/release_program/docs/].

OMA (2006b), OMA Web Services Enabler (OWSER): Core Specifications - Version 1.1, Technical report, Open Mobile Alliance Group, [http://www.openmobilealliance.org/release_program/docs/].

OMG (2004), Common Object Request Broker Architecture (CORBA): Core Specification, Technical report, Object Management Group, [http://www.omg.org/docs/formal/04-03-12.pdf].

Ozzie, J. and Burton, A. (2003), Delivering Collaboration Services to Information Workers: Extending SharePoint with Groove, In *Presentation at Microsoft SharePoint products and technologies Developer's conference*.

Passino, K. M. and Yurkovich, S. (1998), *Fuzzy Control*, Addison Wesley Longman, Menlo Park, CA.

Pawar, P., Srirama, S., van Beijnum, B. J. and van Halteren, A. (2007), A Comparative Study of Nomadic Mobile Service Provisioning Approaches, In *International Conference and Exhibition on Next Generation Mobile Applications, Services and Technologies (NGMAST 2007)*, pages 277–286, IEEE Computer Society.

Perera, A. (2007), WSO2 ESB Performance Testing Round 2, [http://wso2.org/library/2259].

Pham, L. and Gehlen, G. (2005), Realization and Performance Analysis of a SOAP Server for Mobile Devices, In *Proceedings of 11th European Wireless Conference*, vol. 2, pp. 20-27, Nicosia, Cyprus.

Plebani, P., Cappiello, C., Comuzzi, M., Pernici, B. and Yadav, S. (2011), MicroMAIS: executing and orchestrating Web services on constrained mobile devices, *Software: Practice and Experience (2011)*, John Wiley & Sons, Ltd.

PolarLake (2005), Understanding the ESB: What it is, why it matters, and how to choose one (White Paper), Technical report, PolarLake, [http://www.polarlake.com/files/esb.pdf].

Pouwelse, J., Garbacki, P., Epema, D., and Sips, H. (2005), The bittorrent P2P file-sharing system: Measurements and analysis, In *Proceedings of 4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*.

Pratistha, D., Nicoloudis, N. and Cuce, S. (2003), A Micro-Services Framework on Mobile Devices, In *Proceedings of International Conference on Web Services*, Nevada, USA.

Pratistha, D., Zaslavsky, A., Cuce, S. and Dick, M. (2005a), Performance Based Cost Models for Improving Web Service Efficiency Through

Dynamic Relocation, In Proceedings of 6th International Conference on Electronic Commerce and Web Technologies (EC-Web), pp. 248-287, Copenhagen, Denmark.

Pratistha, D., Zaslavsky, A., Cuce, S. and Dick, M. (2005b), A Generic Cost Model and Infrastructure for Improving Web Service Efficiency through Dynamic Relocation, In *Proceedings of International Conference on Web Services (ICWS 2005)*, pp. 381-388, Florida, USA.

Pulkkinen, M., Naumenko, A., and Luostarinen, K. (2007), Managing Information Security in a Business Network of Machinery Maintenance Services Business – Enterprise Architecture as a Coordination Tool, *Special Issue on Methodology of Security Engineering for Industrial Security Management Systems, Journal of Systems and Software*, ELSEVIER.

Qu, C. and Nejdl, W. (2004), Interacting the Edutella/JXTA Peer-to-Peer Network with Web Services, In *Proceedings of the 2004 International Symposium on Applications and the Internet (SAINT'04).*

Ripeanu, M., Foster, I., and Iamnitchi, A. (2002), Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design, *IEEE Internet Computing Journal*, 6(1):51-57.

Riva, O., Nadeem, T., Borcea, C. and Iftode, L. (2010), Mobile Services: Context-Aware Service Migration in Ad Hoc Networks, *IEEE Transaction on Mobile Computing*, IEEE Educational Activities Department.

Rodriguez, S. (2002), XML optimization, [http://www.codeproject.com/soap/betterxml.asp].

Rollman, R. and Schneider, J. (2004), Mobile web services, In XML 2004 Proceedings by SchemaSoft, [http://www.idealliance.org/proceedings/xml04/papers/MobileWebServices.pdf].

Rysavy, P. (1998), General Packet Radio Service (GPRS), GSM Data Today online journal. [http://www.rysavy.com/Articles/GPRS/GPRS.htm].

Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M., and Pelegri-Llopart, E. (2003), Fast Web Services, [http://java.sun.com/developer/technicalArticles/WebServices/fastWS/].

Sandoz, P., Triglia, A., and Pericas-Geertsen, S. (2004), Fast Infoset, [http://java.sun.com/developer/technicalArticles/xml/fastinfoset/].

Satyanarayanan, M. (2001), Pervasive computing: Vision and challenges, *IEEE Personal Communications*.

Satyanarayanan, M., Bahl, V., Caceres, R. and Davies, N. (2009), The Case for VM-based Cloudlets in Mobile Computing, *Pervasive Computing*, 8(4).

Schneider, J. (2001), Convergence of Peer and Web Services, OpenP2P, [http://www.openp2p.com/pub/a/p2p/2001/07/20/convergence.html].

Schroth, C. and Janner, T. (2007), Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services, *IEEE IT Professional Magazine*, vol. 9, issue 3, pp. 36 – 41.

Schulte, R. W. (2007), The Enterprise Service Bus: Communication Backbone for SOA, Gartner Inc.

Seward, J. (2005), bzip2 and libbzip2, version 1.0.3 - A program and library for data compression, [http://www.bzip.org/1.0.3/bzip2-manual-1.0.3.html].

Silva, R. D. (2001), SOAP and Embedded Systems - Draft 0.4, Technical report, ConnectTel, Inc.

Silva, R. D. (2007), eSOAP - Features. [http://esoap.ultimodule.com/bin/esoap/templates/].

Skype (2009), Skype Home page, [http://www.skype.com/].

Srirama, S. (2004), *Concept, implementation and performance testing of a mobile web service provider for smart phones*, Master's thesis, RWTH Aachen, Germany.

Srirama, S., Jarke, M. and Prinz, W. (2006a), Mobile web service provisioning, *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, pp. 120-128, Guadeloupe, French Caribbean.

Srirama, S., Jarke, M., and Prinz, W. (2006b), Mobile Host: A feasibility analysis of mobile Web Service provisioning, In *4th International Workshop on Ubiquitous Mobile Information and Collaboration Systems, UMICS 2006, a CAiSE'06 workshop*, pp. 942–953.

Srirama, S. (2007), Mobile Web Service Provisioning, [http://www-i5.informatik.rwth-aachen.de/lehrstuhl/staff/srirama/MWSP.html].

Srirama, S., Jarke, M., and Prinz, W. (2007a), Security analysis of mobile web service provisioning, *International Journal of Internet Technology and Secured Transactions (IJITST)*, 1(1/2):151–171.

Srirama, S., Jarke, M., and Prinz, W. (2007b), A performance evaluation of mobile web services security, In *Proceedings of 3rd International Conference on Web Information Systems and Technologies (WEBIST 2007)*, Barcelona, Spain.

Srirama, S. and Naumenko, A. (2007), Secure Communication and Access Control for Mobile Web Service Provisioning, In *Proceedings of International Conference on Security of Information and Networks (SIN 2007)*.

Stoer, M. and Wagner, F. (1997), A simple min-cut algorithm, *Journal of the ACM*, 44(4):585–591.

Sun Microsystems (2000), J2ME Building Blocks for Mobile Devices - White Paper on KVM and the Connected, Limited Device Configuration (CLDC), Technical report, Sun Microsystems, Inc. [http://java.sun.com/products/cldc/wp/KVMwp.pdf].

Sun Microsystems (2001), Jini Architecture Specification - Version 1.2, Technical report, Sun Microsystems, Inc., [http://www.sun.com/software/jini/specs/jini1.2html/].

Sun Microsystems (2005), CDC: Java™ platform technology for connected devices, Technical report, Sun Microsystems, Inc. [http://java.sun.com/products/cdc/].

Sun Microsystems (2007a), J2ME Web Services APIs (WSA), JSR 172, Sun Developer Network, [http://java.sun.com/products/wsa/].

Sun Microsystems (2007b), The Java ME Device Table, Sun Developer Network, [http://developers.sun.com/mobility/device/].

Sun Microsystems (2007c), Java™ 2 Platform, Micro Edition (J2ME™) Web Services Specification – Datasheet, Technical report, Sun Microsystems, Inc., [http://java.sun.com/j2me/docs/].

Sun Microsystems (2007d), PersonalJava, Sun Developer Network, [http://java.sun.com/products/personaljava/].

Sun Microsystems (2007e), Sun JavaWireless Toolkit for CLDC, Sun Developer Network, [http://java.sun.com/products/sjwtoolkit/].

Sun Microsystems (2007f), The Java ME Platform - the Most Ubiquitous Application Platform for Mobile Devices, Sun Developer Network, [http://java.sun.com/javame/index.jsp].

Sun Microsystems (2007g), The Jini™ Technology Surrogate Architecture Overview, Technical report, Sun Microsystems, Inc., [https://surrogate.dev.java.net/doc/overview.pdf].

Sun Microsystems (2008), Java Development Kit (JDK), v 1.1 - Archive Home page, Sun Developer Network.

Symbian (2007), Symbian OS - The mobile operating system, [http://www.symbian.com/].

Tanenbaum, A. and Steen, M. (2002), *Distributed Systems: Principles and Paradigms*, 1st edition, Prentice Hall.

Tate, B. (2005), Secrets of lightweight development success, Part 1: Core principles and philosophies, IBM DeveloperWorks.

Teder, A. (2006), *The problem sets of Java Micro Edition technology*, Master's thesis, University of Tartu, 2006.

Ten-Hove, R. and Walker, P. (2005), Java™ Business Integration (JBI) 1.0 - JSR 208 Final Release, Technical report, Sun Microsystems, Inc.

Ten-Hove, R. (2006), JBI Components: Part 1 (Theory), [https://open-esb.dev.java.net/public/pdf/JBI-Components-Theory.pdf].

Thomas, N. and Buckley, W. (2003), The Enterprise Service Bus: A Developer-Friendly Integration Engine, *Web Services Journal*, 3(10):30.

Tian, M., Voigt, T., Naumowicz, T., Ritter, H. and Schiller, J. (2004), Performance considerations for mobile web services, *Computer Communications*, 27(11):1097–1105.

TomTom (2007), TomTom, portable GPS car navigation systems, [www.tomtom.com/].

Tourzan, J. and Koga, Y. (2006), Liberty ID-WSF Web Services Framework Overview - Version: 2.0, Technical report, Liberty Alliance Project, [http://www.projectliberty.org/liberty/].

Traversat, B., Abdelaziz, M., Doolin, D., Duigou, M., Hugly, J.-C., and Pouyoul, E. (2003), Project JXTA-C: Enabling a Web of Things, In *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, page 282, IEEE Computer Society.

UIQ (2007), UIQ Technology Home, [http://www.uiq.com/].

Umtsworld (2002), Overview of the Universal Mobile Telecommunication System, UMTS world, [http://www.umtsworld.com/technology/overview.htm].

UPnP Forum (2003), Universal Plug and Play Device Architecture, Technical report, UPnP Forum, [http://www.upnp.org/resources/documents/].

Vinoski, S. (2005), Java Business Integration, *IEEE Internet Computing*, pages 89–91.

Waterhouse, S., Doolin, D. M., Kan, G., and Faybishenko, Y. (2002), Distributed Search in P2P Networks, *IEEE Internet Computing*, pages 68–73.

Werden, S., Evans, C., and Goodner, M. (2003), WS-I Usage Scenarios, Technical report, Web Services Interoperability Organization, [http://www.ws-i.org/SampleApplications/SupplyChainManagement/UsageScenarios-1.01.pdf].

Whiteside, A. (2007), OGC Web Services Common Specification, Technical report, Open Geospatial Consortium Inc.

Wikman, J. (2006), Providing HTTP Access to Web Servers Running on Mobile Phones, Nokia Research Center, Helsinki, [http://research.nokia.com/tr/NRC-TR-2006-005.pdf].

Wilson, B. J. (2002), *JXTA*, New Riders Publishing.

Wingfoot (2007a), Wingfoot SOAP 1.06 - User Guide, Technical report, Wingfoot Software, [http://www.wingfoot.com/docs/WSOAP_User_Guide1_06.html].

Wingfoot (2007b), Wingfoot SOAP Client, [http://www.wingfoot.com/products.html].

WS-BPEL (2002), Business Process Execution Language for Web Services version 1.1, [http://www.ibm.com/developerworks/library/specification/ws-bpel/].

WS-I (2004), Interoperability: Ensuring the Success of Web Services - An Overview to WS-I, Web Services Interoperability organization (WS-I), [http://www.ws-i.org/docs/].

WSO2 (2007), WSO2 ESB Benchmark, [https://www-lk.wso2.com/benchmark/].
WS-Talk (2007), WS-Talk Project, [http://thames.cs.rhul.ac.uk/wstalk/].

XMLPULL (2007), XMLPULL Parser API, [http://www.xmlpull.org/].

Yang, B. and Garcia, M. H. (2003), Designing a Super-peer Network, In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, IEEE Computer Society, pp. 49-60.

Yang, X., Bouguettaya, A., Medjahed, B., Long, H., and He, W. (2003), Organizing and Accessing Web Services on Air, *IEEE transactions on systems, man, and cybernetics - part a: systems and humans*, 33(6):742–757.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. and Sheng, Q. Z. (2003), Quality Driven Web Services Composition, In *Proceedings of the 12th International World Wide Web (WWW) Conference*.

Zheng, P. and Ni, L. M. (2006), Spotlight: the rise of the smart phone, *Distributed Systems Online, IEEE, 7(3).*

Ziv, J. and Lempel, A. (1977), A Universal Algorithm for Sequential Data Compression, *IEEE Transactions on Information Theory*, 23(3):337–344.

Appendix

Hypertext Transfer Protocol (HTTP)

HTTP is the application layer protocol used to transfer or convey information on intranets and the Web. It is implemented in two communication peers: the client and the server, executed on different end systems, and communicating through HTTP messages. The protocol defines the method and the structure of message for the communication.

The HTTP client first initiates a Transmission Control Protocol (TCP) connection to the server port (default port for HTTP is port 80). Once the TCP connection is established, the client sends the HTTP request message to the server through the socket associated with the TCP connection. The HTTP server receives the request through the socket, extracts the path name of the web page, and retrieves the page from the storage (file system or RAM) of the server, encapsulates the object in an HTTP response message, and sends it to the client using the socket. Once the client receives the message, the TCP connection is closed. The scenario explained here uses the non-persistent TCP connection mode, as the TCP connection closes after processing the client request. A persistent TCP connection is also possible, where the server maintains the connection even after sending the response. Subsequent communication between the same client and server utilizes the already established and still open connection.

The HTTP specification defines two types of message formats, HTTP request and HTTP response messages. The structures of the request and response messages are shown in Figure A.1 and Figure A.2 respectively. The *method* field of the HTTP request indicates the method to be performed on the object identified by the URL. HTTP defines eight

methods indicating the desired action to be performed on the identified resource. Method `GET` means retrieval of the data the URL identifies. Method `HEAD`, which is same as `GET`, returns only the HTTP headers and no document body. Method `POST` submits the data to be processed, to the identified resource. This may result in the creation of a new resource or the updates of existing resources or both. Other methods supported by HTTP include `PUT`, `DELETE`, `TRACE`, `OPTIONS` and `CONNECT`.

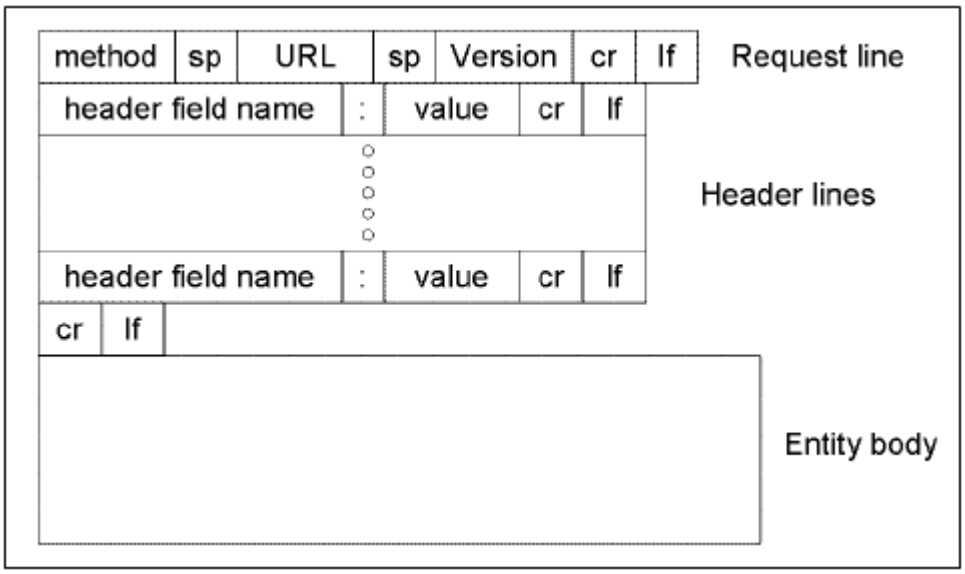

Figure A.1: Structure of HTTP request message [Kurose and Ross, 2001].

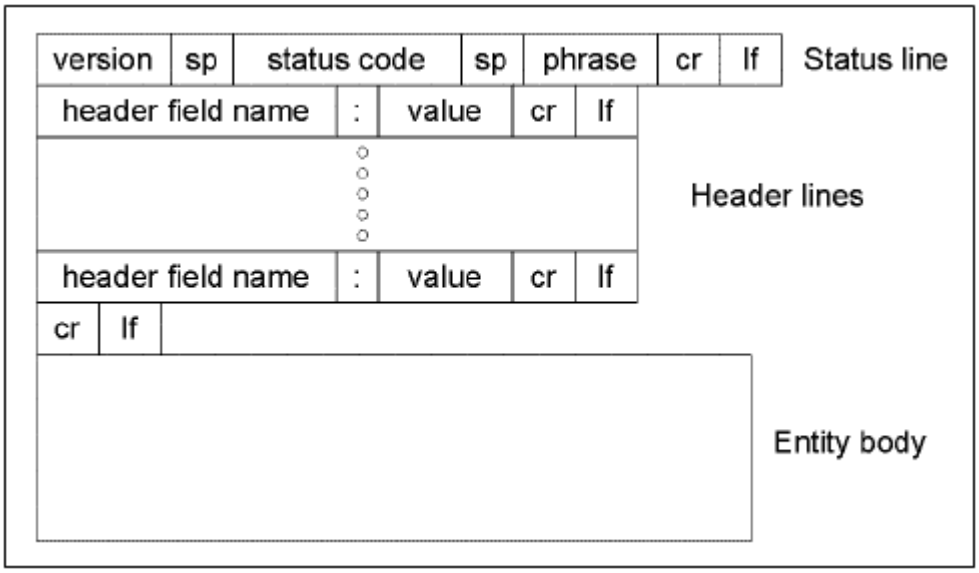

Figure A.2: Structure of HTTP response message [Kurose and Ross, 2001]

HTTP has evolved into multiple, mostly backward-compatible protocol versions. The *version* field of both request and response messages specify the version of HTTP being used by them. In the beginning of the request, the client tells the version it uses, and the server uses the same or earlier version in the response. The supported values are `HTTP/1.0` or `HTTP/1.1` or `HTTP/1.2`.

Similarly, HTTP *header fields* of the HTTP request and HTTP response messages define various characteristics of the data that is requested or the data that has been provided. Some of the prominent header fields are: `Content-Length` that specifies the length of the content in bytes, `Server` that specifies the name of the server that processed the message, etc. The *status code* of the HTTP response message and the phrase associated with it indicate the result of the HTTP request. Some of the prominent codes include `200 OK` which is standard response for successful HTTP requests, `404 Not Found` which indicates that the requested document does not exist on the server, `505 HTTP Version Not Supported`, etc.

Last, but not least critical of the HTTP messages is the *entity body* which contains the requested object itself. The body thus constitutes the main payload of the HTTP messages. The entity body is used with `POST` method and not with `GET` method. The body can be any ASCII (American Standard Code for Information Interchange) message. Thus, in SOAP over HTTP transportation of web service messages, the SOAP request is encapsulated into the HTTP request message body and the SOAP response is encapsulated into the body of the HTTP response.