UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Carlos Paniagua

# Discovery and Push Notification Mechanisms for Mobile Cloud Services

Master Thesis (30 EAP)

*Supervisor: Satish Narayana Srirama, PhD*

| | | |
|---|---|---|
| Author: | ....................................................... | "...." May 2012 |
| Supervisor: | ....................................................... | "...." May 2012 |
| Professor: | ....................................................... | "...." May 2012 |

TARTU, 2012

# Abstract

In the last lustrum the mobile devices such as laptops, PDAs, smart phones, tablets, etc. have pervaded almost all the environments where people perform their day-to-day activities. Further, the extensive R&D in mobile technologies has led to significant improvements in hardware, software and transmission. Similarly, there are significant developments and standardization efforts in web services domain and basic web services have been widely accessed from smart phones. This has lead to the logical next step of providing web services from the smart phones. The concept of the web service provisioning from smart phones is not new and has been extensively explored by Srirama who proposed the concept of Mobile Host. However, the original implementation considered aged technologies such as JMEE, PersonalJava, SOAP architecture among others. This work updates the Mobile Host to the latest technologies like Android OS and REST architecture and proposes a service engine based on Apache Felix, and OSGI implementation for resource constraint devices.

Moreover, the astonishing speed in developments in mobile computing enable the new generation of applications from domains such as context-awareness, social network, collaborative tools, location based services, etc., which benefit from the Mobile Host service provisioning capabilities. As a result the clients have access to a huge number of services available; therefore, an efficient and effective service discovery mechanism is required. The thesis proposes a directory-based with network overlay support discovery mechanism for large networks with high mobility. The proposed discovery mechanism relies in OWL-S, an ontology for service discovery, invocation, composition, and monitoring of web resources. The work also considers the original service discovery mechanism proposed by Srirama relying in peer-to-peer networks and Apache Lucene, a keyword search engine. The study updates the service search to Apache Solr, the latest development for Apache Lucene. The service discovery was extensively tested and the results are summarized in this work.

Mobile technologies are looking into the clouds for extending their capabilities in storage and processing by offloading data and process intensive tasks. This fosters the development of more complex and rich mobile applications. However, due to the time-consuming nature of the tasks delegated to the clouds, an asynchronous mechanism is necessary for notifying the user when the intensive tasks are completed. Mobile cloud service providers and Middleware solutions might benefit from Mobile Host and its asynchronous notification capabilities. The study presents four push notification mechanisms being AC2DM, APNS, IBM MQTT and Mobile Host based push notification. The work summarizes the results of a quantitative analysis and highlights the strengths and weakness of the four notifications approaches. In addition, it explains CroudSTag realization, a mobile application that aims the social group formation by means of facial recognition that relies in mobile cloud services and Mobile Host to provide its functionality to the user.

# Contents

# List of Figures

# 1

# Introduction

In the last lustrum the mobile devices such as laptops, PDAs, smart phones, tablets, etc. have pervaded almost all the environments where people perform their day-to-day activities. Moreover, nowadays 80% of the world population owns a cell phone and by 2011 there were approximately 5.6 billion wireless subscribers worldwide, almost thrice more than what was expected in 2008. In 2008 the number of subscribers was expected to reach the two billions figure by 2011. Such popularity demands an astonishing speed in the developments for mobile technologies in terms of software and hardware. As a result, extensive R&D in mobile technologies have been carried on by the producers, which have led to significant improvements in hardware, software and transmission. Mobile devices are equipped with embedded sensors, camera, touchscreen, more memory and processing capabilities as well as more efficient power consumption mechanisms. Moreover, due to the release of the iOS and Android OS the mobile applications have increased in number and the complexity of providing more numerous and sophisticated applications also has increased. In addition, transmission rates have increased thanks to 3G and 4G technologies as well as WiFi enabling the mobile devices to access the Internet almost ubiquitously. Altogether those improvements enable the mobile devices to perform tasks that normally run in personal computers. Moreover, such enhancements enabled the next generation of services which can be provided not only from dedicated servers but also from mobile devices paving a broad road of possibilities for mobile web service applications.

In mobile web services domain, the resource constrained smart phones are used as both web service clients and providers (1). Some interesting mobile web service client applications are the provisioning of services like information search, language translation, company news etc. for employees who travel regularly. There are also many public web services like the weather forecast, stock quotes etc. accessible from smart phones. Similarly, with the advent of cloud computing, the mobile applications also started using cloud services, which most often have web service interfaces.

Similarly, the concept of mobile web service provisioning is not new and has been in the ground for some time. Srirama et al. proposed the concept of Mobile Host (1) where the mobile device acts as a service provider. Mobile Host enables seamless integration of user specific services to the enterprise by following web service standards, also on the radio link and via resource constrained smart phones (2). Moreover, Mobile Host fosters the new generation of ubiquitous and context-aware applications enabling the consumption of web services anywhere at any time from the handset. Applications from several domains, such as location based services (3), m-banking, collaboration and content sharing, benefit from such Mobile Host. Furthermore, Mobile Host suits well for the consumption and orchestration of mobile-cloud-services.

Cloud computing (4) is a promising emerging technology in which, typically, the resources scalable on demand are provided "as a service (aaS)" over the Internet to the users who need not have knowledge of, expertise in, or control over the cloud infrastructure that supports them. The provisioning of cloud services occurs at the Infrastructural level (IaaS) or Platform level (PaaS) or at the Software level (SaaS). Mobile technologies are recently drawing their attention to the cloud computing due to the increasing demand of the mobile applications for processing power, storage space and energy.

## 1.1 Motivation

The increasing demand of mobile devices to reap the benefits from the developments in the web services domain required to shift the scope of mobile terminals from acting as simple service consumer to act as service providers. Moreover, the recent improvements and updates in web services domain propose new architectures and protocols for enabling the communication between the clients and providers. For instance, the Representational State Transfer (REST) (5) architecture has emerged as an alternative to the Simple Object Access Protoco (SOAP) enabling the design of web services that focus on system's resources. REST is a lightweight, human readable results and easy to build architecture for the provisioning of web services. Despite both, SOAP and REST architectures are widely used, REST seems to fit better for mobile web service provisioning due the network and resource constrains. Nevertheless mobile web services have manifold opportunities to mobile operators, wireless equipment vendors, third-party application developers, and end users. These services need to be provided according to the standards focusing not only for services but also for resources.

Moreover, the release of Android OS and iOS revolutionized the mobile software domain. It is estimated that by 2011 Android and iOS own the 51% and 24% of the market respectively. As a result of such popularity, mobile software creators and developers have drawn their attention to these platforms to reach a higher number of potential customers and users. Therefore, it is logical to consider those platforms for mobile web service provisioning and client applications.

Meanwhile, due to the popularity of mobile web services and cloud services for mobiles a great amount of applications are available and this number is constantly increasing. The Play Store (Android) and App Store (iOS) provide around 450,000 and 500,000 applications respectively. A large amount of those applications rely their functionality on either mobile web or mobile cloud services. However, these services demand an efficient and effective discovery mechanism to reduce the bottlenecks and to proceed with the mobile web service provisioning and invocation with success. Moreover, the traditional centralized registry mechanisms for web services do not suit for the mobile nature of smart phones. Therefore, the search for an appropriate mechanism for the mobile web service and mobile cloud service discovery is required.

In addition, a key requirement of pervasive mobile applications is that changes occur asynchronously and that it is important that when they occur, mobile users are notified in a timely fashion. Without infrastructure for handling the dispatching of asynchronous events, mobile application developers need to code this as part of their applications. This presents developers with extra complexity above and beyond the core application functionality. Usually implementing such asynchronous behavior in mobile applications is not trivial and it requires developing the same functionality again and again for each application. Furthermore, with the emerging mobile cloud domain, where the services are, typically, very time consuming, an asynchronous behavior for notifications is demanded. As an alternative, Push Notification mechanisms can be used for including asynchronous behavior in the invocation and provisioning of web services and cloud services.

## 1.2 Contributions

Extensive research has been done in the web service and mobile web service domain in terms of service provisioning and discovery. The thesis contributes to this work by migrating the concepts of Mobile Host and mobile web service provisioning to the current generation technologies. Early implementations of Mobile Host were available for Java 2 Platform, Micro Edition (J2ME) and Personal Java and rely on SOAP architecture for the service invocation. The thesis introduces an implementation of Mobile Host for Android and iOS for the provisioning of RESTful web services through Hypertext Transfer Protocol (HTTP).

The recent improvements in mobile technologies foster the next generation of pervasive applications. Application domains such as context-awareness, collaboration, social networks, social media and smart environments among others are being explored by researchers. The thesis contributes to this work by providing a scalable and efficient push notification mechanism for mobile devices. The thesis adapts the concept of Mobile Host for the asynchronous behavior required in pervasive mobile applications and the invocation of cloud-based services. In addition, the thesis presents an extensive analysis of four push notification mechanisms in different platforms and architectures. The thesis contributes by providing a comprehensive overview governing underlying concept for push, key players and their future sets, as well as a quantitative performance evaluation of the mechanism.

The study also presents how the Mobile Host based asynchronous mechanism is used in the Mobile Cloud Middleware (MCM) (6), which eases the invocation of cloud services from multiple cloud providers, in building mobile mashup applications. In this context, the thesis also presents the CroudSTag (7) application, which helps in social group formation from mobiles. CroudSTag recognizes the people who appear in media content such as pictures or videos and joins them together into a social group.

This work also addresses the issues associated with the discovery of mobile web and mobile cloud services. Srirama et al. have studied that a Mobile Enterprise (8) can be established in a cellular network by participating Mobile Hosts and their clients, where the hosts provide user-specific services to the clients as per the web services (WS) standards. However the major problems associated in establishing such a Mobile Enterprise will be the quality of service (QoS) and discovery issues, which were extensively studied. This thesis extends the study and introduces a mobile web and cloud service discovery mechanism. The study proposes a directory-based with overlay support discovery mechanism for large networks with high mobility.

## 1.3 Outline

**Chapter 2**: discusses the state of the art addressed by this thesis. The chapter first introduces the web services and cloud services technologies along with associated standards, protocols and architectural updates. Later, the chapter discusses the nomadic mobile services and mobile web services. Here, the supported device and platforms, standardization efforts and implementations, RESTful transmission mechanisms for mobile web services are discusses in detail.

**Chapter 3**: introduces the mobile service provisioning mechanism and the associated challenges. First, it discusses the mobile service provisioning concept. Later it addresses the service provisioning from resource constraint devices such as smart phones. Here, the developed Mobile Host is explained in detail, describing the architecture and technological choices for iOS and Android OS. Finally, it provides a description of how Mobile Host takes care of the QoS issues associated with the service provisioning.

**Chapter 4**: discusses the presence of Mobile Host in peer-to-peer (P2P) networks. It first describes the concept of mobile services provisioning in P2P networks and the advantages associated in terms of accessibility and discovery. The chapter especially concentrates at the discovery mechanisms and discusses the mobile web services discovery mechanism, proposed by the thesis. Here the discovery approach, its context awareness issues and evaluation results are addressed in detail. Further, the chapter explains how the discovery mechanism can be easily extended not only for mobile service discovery but also for discovery of cloud services in P2P networks.

**Chapter 5**: describes how cloud service provisioning benefits from Mobile Host for delivering asynchronous notifications to mobile devices. It explains the concept of push notification in pervasive mobile applications. Later it describes the most popular push notification providers from the industry such as Google and Apple. Further, it summarizes the results of an extensive analysis of four push notification mechanisms in terms of resource consumptions and QoS, over 3G and Wi-Fi networks.

**Chapter 6**: provides the conclusions about the findings of the thesis.

**Chapter 7**: describes the future research directions such as the integration of Mobile Host in the Mobile Web Service Mediation Framework (MWSMF) (9) and other middleware solutions for the web and cloud service provisioning and push notification services. The chapter also proposes extensions for the service discovery mechanism proposed in the thesis to support discovery for cloud-based and dynamic services.

# 2

# State of the Art

In the last decade extensive research has been conducted in enterprise service integration. Similarly, the developments in mobile devices enable the provisioning of mobile web services. Furthermore, the enterprise service integration benefits from such mobile web service provisioning giving place to the Mobile Enterprise. A Mobile Enterprise can be established in a cellular network by participating Mobile Hosts, acting as web service providers, and their clients. In addition, Mobile Host enables seamless integration of specific services to the enterprise, by following web service standards, on the radio link and provided by resource constrains smart phones. However, establishing such a Mobile Enterprise arises several challenges, like the quality of service (QoS) and discovery aspects, not only from the network perspective but also from the mobile devices users.

In mobile web services domain, the resource constrained smart devices are used as both web service clients and providers (Mobile Host). Mobile terminals accessing the web services cater for anytime and anywhere access to services. Some interesting mobile web service applications are the provisioning of services like information search, language translation, company news, weather forecast, stock quotes etc., accessible from smart phones. Mobile web service clients are also significant in the geospatial and location based services. While mobile web service clients are common, the scope of mobile web service provisioning (MWSP) was studied at RWTH Aachen University since 2003, where Mobile Hosts were developed, capable of providing basic web services from smart phones.

Services provided by the Mobile Host can be integrated with larger enterprise services bringing added value to these services. For example, services can be provided to the mobile user based on his current context. Context details like device and network capabilities, location details etc. can be obtained from the mobile at runtime and can be used in providing most relevant services like maps specific to devices and location information.

While service delivery and management from Mobile Host were thus shown technically feasible, the ability to provide proper quality of service (QoS), especially in terms of security and reasonable scalability, for the Mobile Host is observed to be very critical (10). Similarly, huge number of web services possible, with each Mobile Host providing some services in the wireless network, makes the discovery of these services quite complex. Proper QoS and discovery mechanisms are required for successful adoption of mobile web services into commercial environments. Moreover, the QoS and discovery analysis of mobile web services has raised the necessity for intermediary nodes helping in the integration of Mobile Hosts with the enterprise. Srirama et al. introduced, based on these requirements, a Mobile Web Services Mediation Framework (MWSMF) designed as an intermediary between the web service clients and the Mobile Hosts within the Mobile Enterprise, using the Enterprise Service Bus (ESB) technology.

The scale of the Mobile Enterprise has led the research to the new utility computing paradigm, cloud computing. It has been also observed that load balancing is the key in successful deployment of Mobile Enterprise in commercial environments. A MWSMF was established on a public cloud infrastructure so that the framework can adapt itself to the loads of the mobile operator proprietary networks, thus mainly helping in horizontal scaling and load balancing the MWSMF and its components and consequently the Mobile Enterprise (11).

## 2.1 Mobile Web Service Provisioning

The quest for enabling open Extensible Markup Language (XML) web service interfaces and standardized protocols also on the radio link, with the latest developments in cellular domain, lead to a new domain of applications, mobile web services. The developments in cellular world are two folded; firstly there is a significant improvement in device capabilities like better memory and processing power and secondly with the latest developments in mobile communication technologies with 3G and 4G technologies, higher data transmission rates were achieved. In the mobile web services domain, the resource constrained mobile devices are used as both web service clients and providers, still preserving the basic web services architecture in the wireless environments. While mobile web service clients are quite common these days, the research with providing web services from smart phones is still sparse.

The main benefit with Mobile Host is the achieved integration and interoperability for the mobile devices. It allows applications written in different languages and deployed on different platforms to communicate with Mobile Hosts over the cellular network. Moreover, the paradigm shift of smart phones from the role of service consumer to the service provider is a step toward practical realization of various computing paradigms such as pervasive computing, ubiquitous computing, ambient computing and context-aware computing. For example, the applications hosted on a mobile device provide information about the user as well as his/her context. Moreover, mobile devices also support multiple embedded devices such as dual-camera and Global Position System (GPS). For the hosted services, they provide a gateway to make available their functionality to the outside world. In the absence of such provisioning functionality the mobile user has to regularly update the contents to a standard server, with each update of the device's state and adds extra loads in the networks.

Mobile Host is a light weight web service provider built for resource constrained devices such as mobile devices. It has been developed as a web service handler built on top of a normal Web server. The SOAP based web service requests sent by HTTP tunneling are diverted and handled by the web service handler component. The Mobile Host was preliminarily developed in Personal Java on a Sony Ericsson P800 smart phone with a small footprint close to the 100KB. The Mobile Host relies on Open source kSOAP2 for creating and handling the SOAP messages. The key challenges addressed in Mobile Host's development are threefold: to keep the Mobile Host fully compatible with the usual web service interfaces such that clients will not notice the difference; to design the Mobile Host with a very small footprint that is acceptable in the smart phone world; and to limit the performance overhead of the web service functionality such that neither the services themselves nor the normal functioning of the smart phone for the user is seriously impeded.

In (1), it is shown that the Mobile Host can handle the service delivery as well as service administration can be done with reasonable ergonomic quality by normal mobile phone users. It was observed that the processing time is considerably small in comparison with the total response time and rest all being transmission delays. Consequently, the performance of the Mobile Host seems to be directly proportional to data transmission rates. However, nowadays the data transmission rates in mobile devices have achieved considerable improvements with similar or even better transmission rates than domestic networks between 20MB and 100MB. In the same study it is showed that the Mobile Host is capable to handling up to eight concurrent requests for reasonable services of messages sizes (≈2Kb) (2). The implementation also had an extension in J2ME.

Besides to Mobile Host, several other works tried to tackle the challenges involved in the mobile web service provisioning. Farley et al. (12) proposes an architecture for the deployment of mobile web services, highlighting the components for web services to provide a personalized mobile service. However, it does not provide any implementation or evaluation of mobile web services provided from smart phones. Similarly, Ravi, N. et al. (13) explores the integration of Bluetooth service discovery protocol and General Packet Radio Service (GPRS) Internet connectivity into phones, proposing a protocol for provisioning services on smart phones. The SDIPP protocol proposed in this study augments Bluetooth SDP with Web access and personalization. Nevertheless, they claim to extend the Bluetooth capabilities with Web access. The issues inherited by Bluetooth communication persist such as the classes at the discovery time and the short coverage-range. Moreover, with the nowadays developments Wi-Fi communication is most often the technology chosen for short and long coverage-ranges. Similarly, Schall, D. et al. (14) explores the feasibility of web service provisioning from resource constraint devices. This work is more a study of the available toolkits for embedded devices available at that time. Even though, they propose to use Web services from embedded devices to solve interoperability issues in distributed mobile systems they concentrate more in providing performance estimations and design guidelines more than a concrete implementation or architecture.

Asif, M. et al. (15) proposed a light weight Web service provider toolkit. This toolkit supports SOAP messages received through HTTP protocol. Moreover, provides security capabilities for web service provisioning in mobile devices. Finally, this work claims to support up to 12 users. Likewise, Yeon-Seok Kim (16) proposed a light weight framework for mobile web services. The proposed framework supports processing of SOAP messages, execution and migration of services among devices, the management of context and service directory, and the publishing and discovery of services. Nevertheless, the study provides a proper usability and feasibility test, however, it lacks the scalability testing thus scalability is uncertain. Most of the approaches provide lightweight mechanism for mobile web service provisioning, besides security capabilities. Moreover, they support SOAP messages through HTTP protocol. In addition, the communication is established using wireless communication, Wi-Fi or Bluetooth. However, these works only support the SOAP architecture, losing the focus on resources. Moreover, SOAP is a heavy architecture which does not suit the requirements of resource constraints devices. On the other hand, REST, considered in Mobile Host, is a light weight architecture focused on resource description which fits better the mobile web services requirements. Finally, most of the solutions work in LANs or PANs lacking the capabilities for providing web services in the Wide Area Network.

## 2.2 Mobile Enterprise

As mentioned before, a Mobile Enterprise is established in the cellular network when Mobile Hosts and their clients interact among each other, the host providing user-specific services to the clients using WS standards. However, such Mobile Enterprise poses many technical challenges, to the service providers as well as to the mobile operators.

### 2.2.1 Challenges for establishing Mobile Enterprise

As the Mobile Host provides services to the Internet, devices should be safe from malicious attacks. For this, the Mobile Host has to provide only secure and reliable communication in the vulnerable and volatile mobile ad-hoc topologies. In terms of scalability, the Mobile Host has to process reasonable number of clients, over long durations, without failure and without seriously impeding normal functioning of the smart phone for the user.

Similarly, huge number of available web services, with each Mobile Host providing some service in the Internet, makes the discovery of the most relevant service quite complex. Proper discovery mechanisms are required for successfully adoption of such Mobile Enterprise. However, the creation of such discovery mechanism poses critical questions such as where and how the services are going to be published. Whether to use or not to use a service directory support such as the centralized Universal Description, Discovery, and Integration (UDDI). This also raises the question whether centralized nodes can withstand such high loads or new alternatives need to be explored. Moreover, from the mobile operator's perspective a Mobile Enterprise questions what services are expected by the mobile users to be delivered by the operator, if the operator is capable to monitor the communication and have a complete view of the network in such a way that business scenarios can be recognized out of such view, if the operator's infrastructure is capable to scale and adapt itself to such huge oscillating requirements?

### 2.2.2 QoS aspects of the Mobile Host

Providing proper QoS, especially, appropriate security and reasonable scalability, for mobile web service provisioning domain was observed to be very critical. The security analysis of the Mobile Host studied the adaptability of WS Security specification to the MWSP domain and concludes that not all of the specification can be applied to the Mobile Host, mainly because of resource limitations (2). The results of the analysis suggest that the mobile web service messages of reasonable size, approximately 2-5kb, can be secured with web service security standard specifications. The security delays caused are approximately 3-5 seconds. We could also conclude from the analysis that the best way of securing messages in a Mobile Enterprise is to use Advanced Encryption Standard (AES) symmetric encryption with 256 bit key, and to exchange the keys with RSA 1024 bit asymmetric key exchange mechanism and signing the messages with RSA with SHA1. But there are still high performance penalties when messages are both encrypted and signed. Therefore, it is suggested to encrypt only the parts of the message, which are critical in terms of security and signing the message. The signing on top of the encryption can completely be avoided in specific applications with lower security requirements. In terms of scalability, the layered model of web service communication, introduces a lot of message overhead to the exchanged verbose XML based SOAP messages. This consumes a lot of resources, since all this additional information has to be exchanged over the radio link. Thus for improving scalability the messages are to be compressed without effecting the interoperability of the mobile web services. Message compression also improves the energy efficiency of the devices as there will be less data to transmit.

### 2.2.3 Discovery aspects of the Mobile Enterprise

In a commercial Mobile Enterprise with Mobile Hosts, and with each Mobile Host providing some services for the Internet, expected number of services to be published could be quite high. Generally web services are published by advertising WSDL (Web Services Description Language) descriptions in a UDDI registry. But with huge number of services possible with Mobile Hosts, a centralized solution is not the best idea, as they can have bottle necks and can introduce single points of failure. Besides, mobile networks are quite dynamic due to the node movement. Devices can join or leave network at any time and can switch from one operator to another operator. This makes the binding information in the WSDL documents, inappropriate. Hence the services are to be republished every time the Mobile Host changes the network.

Dynamic service discovery is one of the most extensively explored research topics in the recent times. Most of these service discovery protocols are based on the announce-listen model like in Jini. In this model periodic multicast mechanism is used for service announcement and discovery. But these mechanisms assume a service proxy object that acts as the registry and it is always available. For instance, Berger, S. et al. (17) claims to expand web services to mobile devices while exploring the issues that arise due the mobility of devices hosting web services, such as service discovery, device disambiguation and software footprint. This work claims to provide mobile web services from smart phones relying in UUID for publishing the services and its discovery mechanism is rather simple without any consideration of the device context or preferences. Similarly, Steele, R. et al. (18) presents an architecture for discovery and invocation of mobile web services through automatically generated abstract multimodal user interface. This work presents a prototype indented to auto-generate user interfaces based on XForms and VoiceXml from WSDL files. However, this work focuses on the discovery and relies in the UUID mechanism for discovering the mobile web services but lacks the provisioning of services from the devices. Capra, L. et al. (19) presents Q-CALI, a resource discovery framework that claims to enable pervasive application to discover and select the resource that best satisfy the users' needs, taking into account the current execution context and QoS requirements. It introduces its own resource description protocol. However, it lacks of technical details about the discovery protocol and does not addresses the mobility and scalability issues associated to mobile devices. Moreover, since they propose their own resource description protocol integration issues arise when trying to connect with other services such as web services and cloud services. Furthermore, for dynamic ad hoc networks, assuming the existence of devices that are stable and powerful enough to play the role of the central service registries is inappropriate. Hence services distributed in the ad-hoc networks must be discovered without a centralized registry and should be able to support spontaneous peer to peer connectivity.

Considering these developments and the need for distributed registry and dynamic discovery, Srirama et al. (20) studied alternative means of mobile web service discovery and realized a discovery mechanism in the P2P network. In this solution, the virtual P2P network also called the mobile P2P network is established in the mobile operator network with one of the nodes in operator proprietary network, acting as a JXTA super peer. JXTA (Juxtapose) is an open source P2P protocol specification. Once the virtual P2P network is established, the services deployed on Mobile Host in the JXME virtual P2P network are to be published as JXTA advertisements, so that they can be sensed as JXTA services among other peers. JXTA specifies Modules as a generic abstraction that allows peers to describe and instantiate any type of implementation of behavior representing any piece of čodeïn the JXTA world. So the mobile web services are published as JXTA modules in the virtual P2P network. Once published to the mobile P2P network, the services can later be discovered by using the keyword based search provided by JXTA. This approach also considered categorizing the services and the advanced features like context aware service discovery. We address the discovery solution as mobile P2P discovery mechanism. The evaluation of the discovery approach suggested that the smart phones are successful in identifying the services in the P2P network, with reasonable performance penalties for the Mobile Host.

### 2.2.4 Mobile Web Services Mediation Framework

Mobile Hosts with proper QoS and discovery mechanisms, enable seamless integration of user-specific services to the Mobile Enterprise. Moreover services provided by the Mobile Host can be integrated with larger enterprise services bringing added value to these services. However, enterprise networks deploy disparate applications, platforms, and business processes that need to communicate or exchange data with each other or with the Mobile Hosts. The applications, platforms and processes of enterprise networks generally have non-compatible data formats and non-compatible communications protocols. Besides, the QoS and discovery study of the Mobile Host offered solutions in disparate technologies like JXTA. This leads to serious integration problems within the networks. The integration problem extends further if two or more of such enterprise networks have to communicate among themselves. We generally address this research scope and domain, as the Enterprise Service Integration.

The mobile web services mediation framework (MWSMF), introduced by Srirama et al. (9) is established as an intermediary between the web service clients and the Mobile Hosts in mobile enterprise. Enterprise Service Bus (EBS) is used as the background technology in realizing the mediation framework. MWSMF relies on EBS for provisioning services from resource constrained devices.

The mediation framework proposed by Srirama, relied on ServiceMix (21), an open source implementation of ESB, based on the Java Business Integration specification (22). JBI architecture supports two types of components Service Engines and Binding Components. Service engines are components responsible for implementing business logic and they can be service providers/consumers. Service engine components support content-based routing, orchestration, rules, data transformations etc. Service engines communicate with the system by exchanging normalized messages across the normalized message router (NMR). The normalized messaging model is based on WSDL specification. Binding components are used to send and receive messages across specific protocols and transports. The binding components marshal and un-marshal messages to and from protocol-specific data formats to normalized messages.

The HttpReceiver component of the MWSMF receives the web service requests (SOAP over HTTP) over a specific port and forwards them to the Broker component via NMR. The main integration logic of the mediation framework is maintained at the Broker component. For example, in case of the scalability maintenance, the messages received by Broker are verified for mobile web service messages. If the messages are normal Http requests, they are handled by the HttpInvoker binding component. If they comprise mobile web service messages, the Broker component further ensures the QoS of the mobile web service messages and transforms them as and when necessary, using the QoS Verifier service engine component, and routes the messages, based on their content, to the respective Mobile Hosts. The framework also ensures that once the mobile P2P network is established, the web service clients can discover the services using mobile P2P discovery mechanism and can access deployed services across MWSMF and JXTA network.

Apart from security and improvements to the scalability, QoS provisioning features of the MWSMF also include message persistence, guaranteed delivery, failure handling and transaction support. External web service clients, that do not participate in the mobile P2P network, can also directly access the services deployed on the Mobile Hosts via MWSMF, as long as the web services are published with any public UDDI registry or the registry deployed at the mediation framework and the Mobile Hosts are provided with public IPs. This approach evades the JXME network completely. Thus the mediation framework acts as an external gateway from Internet to the Mobile Hosts and mobile P2P network. The framework also provides a bird view of the mobile enterprise to the cellular operator, so that business scenarios can be drawn out of it.

Other work which considers the provisioning of web and cloud services is VOLARE (23), proposed by Papakos, P. VOLARE is a middleware approach that enables dynamic adaptation of cloud service discovery and binding according to the context of a mobile device. The work claims to guarantee QoS level supported by the context, efficient resource utilization and savings in monetary provision cost. It enables the service discovery by intercepting the application requests, monitoring of the client's context, dynamic adaptation to the current QoS levels achievable, binding among the context of the device and the QoS levels required as described by declarative adaption policies. This work focuses in the discovery of cloud services from mobile devices and tries to guarantee QoS levels taking into account the user's context, which is likely volatile in mobile devices. Notwithstanding, it does not consider the provisioning of services from mobile devices neither its interaction with cloud services which might lead to interesting application scenarios.

## 2.3 Cloud Computing

While the MWSMF was successful in achieving the integrational requirements of the Mobile Host and the Mobile Enterprise, a standalone framework still faces the troubles with heavy loads. The problems with scalability are quite relevant in such scenarios and the system should scale on demand. For example number of Mobile Hosts providing the services and the number of services provided by the Mobile Hosts can explode while some events are underway such as the football Euro cup, or the Japanese Earthquakes in 2011. This increases the number of MWS clients the framework has to support. Elasticity of the framework can be defined as its ability to adjust according to the varying number of requests it has to support. The MWSMF considers public clouds to address issues in scalability of mobile operator proprietary networks, to achieve elasticity, horizontal scaling (scaling by adding more nodes to the cluster, rather than increasing performance of a single node) and load balancing (11).

Cloud computing is a style of computing in which, typically, resources scalable on demand are provided as a service (aaS) over the Internet to users who need not have knowledge of, expertise in, or control over the cloud infrastructure that supports them. Cloud computing mainly forwards the idea of utility computing along with virtualization. In the utility computing model, consumers pay based on their usage of computing resources, just like the traditional utility services e.g. water, electricity, gas etc. Just like any utility services model cloud computing benefits from economies of scale. On the other hand, virtualization technologies partition hardware and thus provide flexible and scalable computing platforms. Servers in the cloud can be physical machines or virtual machines. A cloud computing platform dynamically provisions, configures, reconfigures, and de-provisions servers as requested (24). Cloud services are provided on demand and at different levels.

The provisioning of services can be at the Infrastructural level (IaaS) or Platform level (PaaS) or at the Software level (SaaS). In the IaaS, commodity computers, distributed across Internet, are used to perform parallel processing, distributed storage, indexing and mining of data. IaaS provides complete control over the operating system and the clients benefit from the computing resources like processing power and storage, e.g. Amazon Elastic Cloud Computing (EC2) (25). Virtualization is the key technology behind realization of these services. PaaS mainly provides hosting environments for other applications. Clients can deploy the domain specific applications on these platforms, e.g. Google App Engine (26). These applications are in turn provided to the users as SaaS. SaaS are generally accessible from web browsers, e.g. facebook. Web 2.0 is the main technology behind the realization of SaaS. However, the abstraction between the layers is not concrete and several of the examples can be argued for other layers.

There are several public clouds on the market such as Google Apps, Google App Engine and Amazon EC2. Elastic Java Virtual Machine on Google App Engine allows developers to concentrate on creating functionality rather than bother about maintenance and system setup. However, such sandboxing places some restrictions on the allowed functionality. Amazon EC2 on the other hand allows full control over virtual machine, starting from the operating system. It is possible to select a suitable operating system, and platform (32 and 64 bit) from many available Amazon Machine Images (AMI) and several possible virtual machines, which differ in CPU power, memory and disk space. This functionality allows to freely selecting suitable technologies for any particular task. In case of EC2, price for the service depends on machine size, its time up, and used bandwidth in and out of the cloud. Moreover, there are free implementations of EC2 compatible cloud infrastructure e.g. Eucalyptus (27), that help in creating private clouds. Thus the cloud computing application scan initially be developed at the private clouds and later can be scaled to the public clouds. The setup is of great help for the research and academic communities, as the initial expenses of experiments can be reduced by great extent.

To achieve the scalability for the mediation framework, the MWSMF was installed on the Amazon EC2 cloud. Once the Amazon Machine Images (AMI) are configured, stateless nature of the MWSMF allows, fairly easy horizontal scaling by adding more MWSMF nodes and distributing the load among them with the load balancer.

## 2.4 Semantics in Web Service Provisioning

Web Services have been in the ground for the last decade and have been widely embraced by the industry. Moreover, as a result of the extensive research conducted in the field, extensions to composition of web services and semantic support have been included. In the matter of semantics support, McIlraith S.A. et al (28) proposed the markup of Web services in the DARPA Agent Markup Language (DAML) family of Semantic Web markup languages. This markup enables a wide variety of agent technologies for automated Web service discovery, execution, composition and interoperation. The authors present one such technology for automated Web service composition.

Further, Ankolekar A. et al. (29) introduced DAML-S, a DAML+OIL ontology for describing the properties and capabilities of Web Services. DAML-S provides Web Service descriptions at the application layer, describing what a service can do, and not just how it does it. Ankolekar describes three aspects of the ontology: the service profile, the process model, and the service grounding. The work focuses on the grounding, which connects the ontology with low-level XML-based descriptions of Web Services.

Further, Sivashanmugam, K. et al. (30) proposed to develop semantic Web services where by the Web Services are annotated based on shared ontologies, and use these annotations for semantic-based discovery of relevant Web services. However, this work just describes how the services are annotated but does not propose any architecture for the service provisioning. Further, Verma, K. (31) introduced METEOR-S Web Service Discovery Infrastructure. In this work, they present a scalable, high performance environment for Web service publication and discovery among multiple registries. It organizes registries into domains, enabling domain based classification of all Web services by means of a ontology-based approach. METEOR-S enables semantic discovery by adding semantic annotations to Web service specifications either in registries or service descriptions. Moreover, it relies on SAWS algorithm (32) to automate the mapping between WSDL concepts and the domain specific ontologies. The search of services is carried on using templates constructed using ontological concepts. Finally, they provide peer-to-peer support by means of JXTA protocols. Nevertheless this work provides semantic discovery and publications in peer-to-peer networks they do not provide any study or extension to resource constraint devices such as smart phones.

## 2.5 Summary

The developments in the web services domain, the improved device capabilities of the smart phones and the improved transmission capabilities of the cellular networks have lead to the mobile web services provisioning domain. This chapter summarized the challenges and research associated in this domain and establishing the Mobile Enterprise. The QoS aspects of the developed Mobile Host, like providing proper security and scalability, and the discovery of the provided services are described briefly. Further, the QoS and discovery demands of the Mobile Host have raised the necessity for a middleware framework and the features and realization details of the MWSMF were discussed. However to scale of Mobile Enterprise to the loads possible in mobile networks, MWSMF was shifted to the cloud and the overall description of such implementation is addressed. It also described how MWSMF is horizontally scalable, thus allowing to utilize cloud's elasticity to meet load requirements in an easy and quick manner. The chapter also discussed the study and research around semantic web services.

# 3

# Mobile Host in Android

Smart phones are generally used for accessing different types of services like the location based services, mobile web services, mobile cloud services etc. from different providers. However, one can envision providing services from the smart phone with the latest developments in the mobile devices in hardware (embedded sensors, memory, power consumption, touch screen, better ergonomic design, etc.), in software (more numerous and more sophisticated applications due to the release of iOS and Android platforms), in transmission (higher data transmission rates achieved with 3G and 4G technologies) and in Wi-Fi networks ubiquity. Mobile web service provisioning was studied extensively and Mobile Host's QoS issues in terms of security and scalability, discovery issues and integration aspects are addressed thoroughly at (8), as already mentioned in the previous section.

As the popularity of Android rose and with the upcoming of standards like Open Services Gateway initiative (OSGi) framework, we have upgraded the research to the current generation mobile devices and technologies. The OSGi framework is a module system and service platform for the Java programming language. With OSGi, applications or components can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. Mobile Host for Android is realized using Apache Felix, an OSGI implementation for Android. The services run as bundles within Felix and the invocation of the services is through REST protocol. So the services are considered as resources that can be accessed via HTTP requests. Android Software Development Kit (SDK) provides

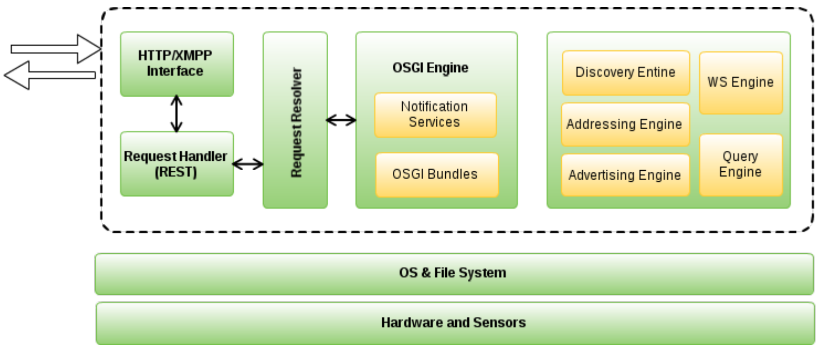


**Figure 3.1:** Architecture of Mobile Host for Android

a mechanism to establish Server Sockets communication between the device and the clients; consequently, the HTTP request can be handled from the device.

## 3.1 Mobile Host Architecture and Realization

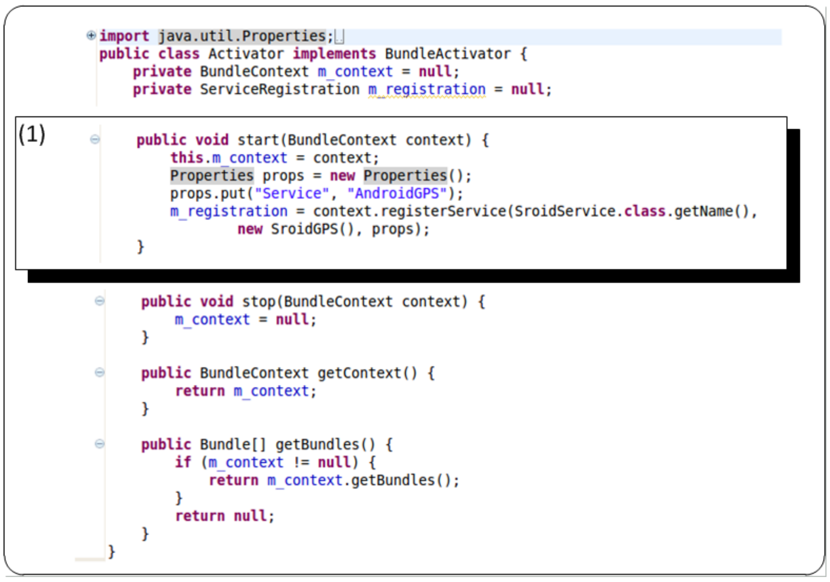
Nevertheless, smart phones are equipped with the hardware for the provisioning of services; several issues arise when providing web services. For instance, from the development perspective the services should be maintainable, easy to install, and most often focus on resources more than in services. In addition, from the provisioning perspective, the mobile nature of smart phones brings challenges such as the addressability, reachability and reliability. The upgrades in Mobile Host address these issues relying in several technologies such as OSGI and ZeroConf. The architecture of Mobile Host for Android is shown in the figure 3.1.

Mobile Host accepts connections through HTTP or Extensible Messaging and Presence Protocol (XMPP). Mobile Host for Android is designed to follow the RESTful architecture. REST is an architecture in which the key resources (entities, collections, services, etc.) are identified by its own URI. The standard methods are mapped to resource-specific semantics. All resources implement the same uniform interface. In this way, for example: the picture "logo.png" requested by the client can be accessed through a GET HTTP request to the URI "/logos/logo.png". Once a request is received, it is passed to the REST handler which parses the request to know the resource requested by the client. After, the REST handler passes the request to the Request Resolver which accesses the OSGI engine for resolving the request. The Request Resolver gets the instance of the service which is running in the OSGi Engine. As mentioned before, the OSGI Framework runs a pool of services, named OSGI Services, which can be managed remotely and easily deployed. Each OSGI Service deployed in the OSGI Engine implements a Java Interface enforcing the service to handle the HTTP method such as GET, POST, DELETE, etc. Once the Request Resolver acquired the service instance it invokes the method corresponding to the HTTP requests. For example, if the HTTP Request was sent as a GET then the Request Resolver will invoke the doGet method of the service. The control is passed to the OSGI Service. The OSGI Service implements the service logic like retrieving pictures, contacts, accessing the GPS location, etc. The OSGI Service also prepares the response which is later delivered to the client. The response can be in any format (XML, JSON, plain text, etc.) or a mime type based on the logic of the service. Finally the OSGI Service writes the response to the socket according to HTTP or XMPP protocol.

## 3.2 Mobile Host Services

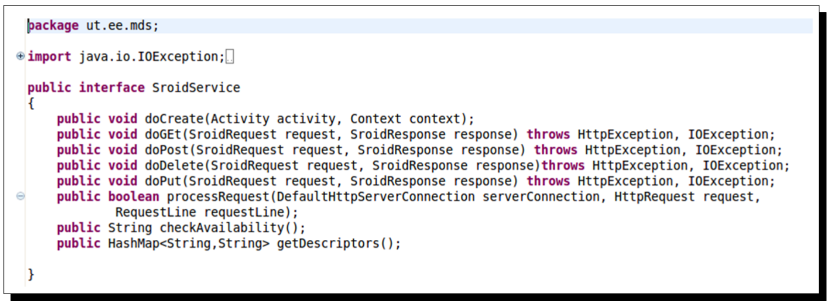
The OSGI Services are deployed as bundles and each bundle needs to be registered in the OSGI framework to make the service available for invocation. During the registration process the OSGI Service provides to the OSGI Engine information about itself such as the name of service. This name is later used by the Request Resolver for invocation purposes. From a developer perspective an OSGI Service needs to implement two Java Interfaces named BundleActivator and SroidService. The BundleActivator contains the methods required for the registration of the service in the OSGI Engine. The figure 3.2 illustrates an Activator for a GPS Service. In this activator the method Start (1) registers the service with the name of "AndroidGPS" to be searchable in the engine. Similarly, the SroidService contains the methods required for the service provisioning. The SroidService interface, shown in the figure 3.3, guarantees that the OSGI Services handle the HTTP methods GET, PUT, DELETE and POST . In this interface the method doCreate has the function of the constructor in a normal class in object oriented programming, and can be used for any variable or process initialization.

Nevertheless Mobile Host is based on REST architecture, the services might need to receive GET or POST parameters sent along with the HTTP request or write information to the socket established between the clients and Mobile Host. For addressing these needs Mobile Host provides of two Java Classes, SroidRequest and SroidResponse



```
import java.util.Properties;
public class Activator implements BundleActivator {
    private BundleContext m_context = null;
    private ServiceRegistration m_registration = null;

(1)
    public void start(BundleContext context) {
        this.m_context = context;
        Properties props = new Properties();
        props.put("Service", "AndroidGPS");
        m_registration = context.registerService(SroidService.class.getName(),
                new SroidGPS(), props);
    }

    public void stop(BundleContext context) {
        m_context = null;
    }

    public BundleContext getContext() {
        return m_context;
    }

    public Bundle[] getBundles() {
        if (m_context != null) {
            return m_context.getBundles();
        }
        return null;
    }
}
```

**Figure 3.2:** Activator Class - GPS Service



```
package ut.ee.mds;

import java.io.IOException;

public interface SroidService
{
    public void doCreate(Activity activity, Context context);
    public void doGEt(SroidRequest request, SroidResponse response) throws HttpException, IOException;
    public void doPost(SroidRequest request, SroidResponse response) throws HttpException, IOException;
    public void doDelete(SroidRequest request, SroidResponse response)throws HttpException, IOException;
    public void doPut(SroidRequest request, SroidResponse response) throws HttpException, IOException;
    public boolean processRequest(DefaultHttpServerConnection serverConnection, HttpRequest request,
            RequestLine requestLine);
    public String checkAvailability();
    public HashMap<String,String> getDescriptors();

}
```

**Figure 3.3:** SroidService Interface

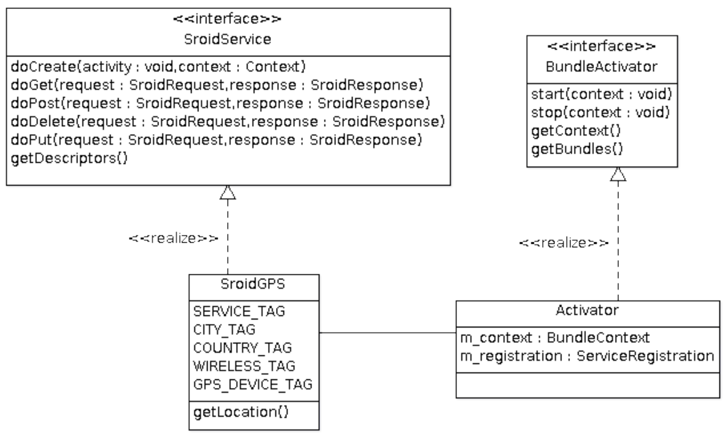

<<interface>>
SroidService
doCreate(activity : void,context : Context)
doGet(request : SroidRequest,response : SroidResponse)
doPost(request : SroidRequest,response : SroidResponse)
doDelete(request : SroidRequest,response : SroidResponse)
doPut(request : SroidRequest,response : SroidResponse)
getDescriptors()
<<interface>>
BundleActivator
start(context : void)
stop(context : void)
getContext()
getBundles()
<<realize>>
<<realize>>
SroidGPS
SERVICE_TAG
CITY_TAG
COUNTRY_TAG
WIRELESS_TAG
GPS_DEVICE_TAG
getLocation()
Activator
m_context : BundleContext
m_registration : ServiceRegistration

**Figure 3.4:** Class Diagram for a GPS Mobile Host Service

respectively. The SroidRequest Class encapsulates the logic for parsing the HTTP request and maps the parameters, if any, to a HashMap accessible from the OSGI Services. Similarly, the SroidResponse Class encapsulates the logic for writing information to the socket previously established by the client with Mobile Host. The Class diagram shown in the figure 3.4 describes the classes and relationships needed to create a Mobile Web Service for Mobile Host in Android.

The OSGi services can be deployed as Java Archive (JAR) files containing the implementation of both interfaces, Activator and SroidService. When Mobile Host starts, it takes each JAR file from the folder SroidServices, in the Android File System, and deploys the services in the OSGI Engine. Once the services are registered in the OSGI Engine they are ready to be invoking from the Request Resolver.

## 3.3 Sample Web Services Provided by Mobile Host

Mobile Host fosters the next generation of applications that rely in context-awareness and ubiquitous information. The implementation of Mobile Host for Android introduces, as samples, a Location GPS Data Provisioning Service and the Push Notification Messages. Both services can be widely used for ubiquitous and context-awareness scenarios.

### 3.3.1 Location (GPS) Data Provisioning Service (Location Information Service)

This web service provides the location information of the mobile device by using the GPS embedded in the mobile device. As mentioned before, nowadays smart phones are equipped with sensors such as GPS. The GPS is a Department of Defense (DoD) developed, worldwide, satellite-based radio navigation system. The constellation consists of 31 operational satellites and is fully operational since 1995. GPS provides two levels of service, Standard Positioning Service and the Precise Positioning Service. The Standard Positioning Service is a positioning and timing service which will be available to all GPS users on a continuous, worldwide basis with no direct charge. The Precise Positioning Service is a highly accurate military positioning, velocity and timing service which will be available on a continuous, worldwide basis to users authorized by the U.S. However, the standard positioning service, available for common purposes, lacks of accuracy when the receiver device is indoors. As an alternative smart phones can also determine the device location by using Wi-Fi network.

The service developed for Mobile Host supports both location mechanisms and returns the latitude and longitude of the current position of the device in the globe in JavaScript Object Notation (JSON) format. There are several scenarios that can benefit from such service. For instance, we can envision a pro-active service for recommendations. Typically, recommendation services such as foursquare (33) are reactive, it means the mobile user should ask for recommendations to the provider and send its location along with the recommendation request. With the GPS service in Mobile Host the provider can shift to a proactive approach, asking to the device for its current location and then push the recommendations to the device.

### 3.3.2 Push Notification Service

Mobile applications have improved significantly in the last lustrum. They provide more sophisticated functionality to the user but also they demand more resources in hardware and software to support such functionality. Therefore, mobile technologies are looking into the emerging cloud computing domain to satisfy the increasing demand from mobile applications for processing power, storage space and energy. Nevertheless, mobile applications that rely in the clouds provide more sophisticated and rich functionality to the users, they happen to take a long time to provide those functionalities. Extensive research is being carried on in the cloud-based service provisioning. Mobile Cloud Middleware (MCM) and MWSMF are two middleware solutions that enable mobile devices access web services and cloud services from smart phones. However, MCM and MWSMF required an asynchronous mechanism for the communication between the middleware and the mobile devices, mainly but not only for notifications. We have developed a Push Notification Service mechanism that runs on Mobile Host enabling the asynchronous communication between the middleware solutions and the device.

One application developed by means of MCM and Mobile Host is CroudSTag (34). It takes a set of pictures and videos from the cloud and processes them with the aim of social group formation by means of facial recognition technologies. CroudSTag takes approximately 35 minutes to process a three-minute video recorded with a mobile phone in high definition. Such waiting time is not tolerable for the user and mobile application usability perspective. Moreover, the operating system kills a process when it is running out of memory (Android case). CroudSTag relies in the MCM for the mobile cloud services invocation and MCM relies in Mobile Host for notify asynchronously to the device about the CroudSTag's results. The Push Notification Mechanism is discussed in detail in Chapter 5 along with a detailed analysis of other Push Notification approaches.

### 3.3.3 File Browsing Service

This web service enables to access the files such as pictures and documents in the public folder in the mobile device. Following the REST philosophy each files is a resource and can be accessed through its corresponding URL. This service was used for testing the performance of the Mobile Host implementation. The experiments consider a GET request for downloading a picture of 1.5MB size. Tsung, a load testing tool, was used for conducting the experiments. In addition, a Samsung Galaxy SII phone (35), with 8 megapixel color camera with auto focus and flash was considered for the analysis. The phone has a Dual-Core 1.2GHz Cortes-A9 CPU, 1GB of RAM, 16GB for storage. The services and applications were developed based on Android platform, compatible with Android 2.2 API or higher. The experiments were conducted under Wi-Fi and 3G networks. So the tests were taken in a network with upload rate of $\approx$ 4393 kbps

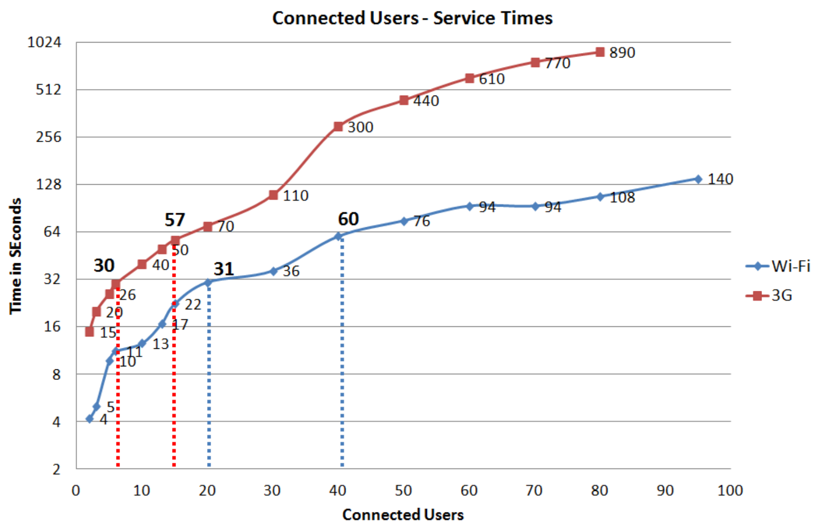


**Figure 3.5:** File Browsing Service - Simultaneous connections serve by Mobile Host

and download rate of ≈ 5648 kbps, respectively and a 3G network with upload rate of ≈ 1487 kbps and download rate of ≈ 4597kbps.

From the graphic in the figure 3.5 we can observe that Mobile Host for Android can handle ≈ 20 concurrent users with a response time of ≈ 31 seconds (in the worst case) in Wi-Fi networks, and ≈ 6 concurrent connections with a response time of ≈ 57 seconds in 3G networks. In this case we are assuming that a waiting time of 30 seconds is reasonable from the user perspective. One should observe that the timestamps are for the slowest invocation in a set of $n$ concurrent requests. Similarly, assuming that the user is whiling to wait 60 seconds for the results, Mobile Host is capable of handling ≈ 40 concurrent connections in Wi-Fi and ≈ 15 connections in 3G.

## 3.4 Summary

The chapter describes the new developments in the realization of Mobile Host. Mobile Host has been updated to the latest technologies such as Android and iOS. Moreover, the Mobile Host architecture for Android considers Open Service Gateway initiative (OSGi) for the service management, and ZeroConf for publishing and advertising the services. The clients can access the Mobile Host services via HTTP or XMPP protocol using REST philosophy in which services are abstracted as resources. Furthermore, the chapter addresses the technical details for developing services for Mobile Host. It explains in detail the technical aspects for developing mobile web services capable to access the device resources. As examples, the chapter presents three services developed using such approach. Finally, the chapter summarizes the performance testing results showing that the Mobile Host for Android presents reasonable performance when providing mobile web services.

4

# Mobile Web and Mobile Cloud Service Discovery

Mobile Host can also provide its services through P2P networks. P2P is a set of distributed computing model systems and applications used to perform a critical function in a decentralized manner. P2P takes advantage of resources of individual peers like storage space, processing power, content, which are all critical for smart phones, and achieves scalability, cost sharing and anonymity, thereby enabling ad-hoc communication and collaboration. In order to reap the benefits of P2P, by achieving increased application scope, and targeting efficient utilization of resources of individual mobile peers, we tried to adapt Mobile Host into P2P networks.

Moreover, mobile applications are increasing in number and complexity and a significant number of those applications rely on mobile web services. This popularity of the mobile web services demands an efficient and effective discovery mechanism to reduce the bottlenecks and to proceed with the mobile web service provisioning and invocation with success. Furthermore, the traditional centralized registry mechanism for web services does not suit for the mobile nature of smart phones. Therefore, the search for an appropriate mechanism for the mobile web service and mobile cloud service discovery is required.

The service discovery requirements associated to the mobile web service provisioning are highly dependent of the size of the network and the mobility of the nodes. Mian et al. (36) characterized the service discovery protocols in the basis of the size of the network and mobility of the nodes. Moreover, for the analysis Mian takes into consideration if the discovery mechanism keeps a directory of the services and if the discovery mechanisms build overlay-networks for routing purposes. Furthermore, Mian defines three levels of mobility (low, medium and high) and three sizes of networks (small, medium and large). Small networks (up to 10 nodes) with low mobility (up to five km per hour, such as in a building) bring different requirements than a large network (greater than 100 nodes) with high mobility (greater than 50 km per hour, such as highway traffic speeds). For example, in large networks and low mobility, overlay networks provide an efficient routing mechanism for searching. Moreover, the low mobility does not bring any issues in terms of maintenance. However, if the mobility turns to be high, the maintenance of the overlay networks becomes an issue. One more example is the small networks with high mobility where the maintenance of directory service is unfeasible and the overlay networks are not necessary. Consequently, the decision to whether include a service directory or whether to build overlay networks is driven by the mobility and size of the mobile networks. Mobile Host supports two discovery mechanisms: a directory-based with overlay support discovery mechanism for large networks with high mobility; and a directory-less with overlay support discovery mechanism for small networks with low mobility.

## 4.1 Mobile Host Service Discovery Mechanism

Considering the need for distributed registry and dynamic discovery in mobile web service provisioning domain, Srirama et al. (37) studied alternative means of mobile web service discovery and realized a discovery mechanism in the P2P network. In this solution, the virtual P2P network also called the mobile P2P network is established in the mobile operator network with one of the nodes in operator proprietary network, acting as a JXTA super peer. JXTA (Juxtapose) is an open source P2P protocol specification. Once the virtual P2P network is established, the services deployed on Mobile Host in the JXME virtual P2P network are to be published as JXTA advertisements, so that they can be sensed as JXTA services among other peers. JXTA specifies Modules as a generic abstraction that allows peers to describe and instantiate any type of implementation of behavior representing any piece of c̈odeïn the JXTA world. So the mobile web services are published as JXTA modules in the virtual P2P network. Once published to the mobile P2P network, the services can later be discovered by using the keyword based search provided by JXTA. This approach also considered categorizing the services and the advanced features like context aware service discovery. This thesis extended the idea by incorporating the latest technologies, updates in mobile invocation, better semantic discovery mechanism etc.

### 4.1.1 Mobile Host Directory-less with Overlay Support Discovery Mechanism for Mobile Ad Hoc Networks

Mobile Ad Hoc Networks, MANET, are self-configuring infrastructure less networks of mobile devices which are connected by wireless links. Each device in a MANET is able to move independently in any direction, and therefore changes its links to other devices periodically. Moreover, each device forwards the traffic unrelated to its own use acting as a router in the network. These networks can be connected to a larger link such as the Internet. However, several challenges arise in this type of networks such as addressing, naming, and service discovery. Most often, MANETs are networks of small size with medium or low mobility. Mobile Host considers ZeroConf for exposing itself and its services to external devices in MANETs. It consists of a set of techniques for automatic configuration and creation of a usable local Internet protocol network. ZeroConf dynamically configures the host in the network assigning them an IP address and also a domain name. Furthermore, ZeroConf provides a mechanism for service discovery and domain resolution. Network users no longer have to assign IP addresses, assign host names, or even type in names to access services on the network. Users simply query what network services are available, and decide from the list. Applications can automatically detect services they need or other applications they can interact with, allowing automatic connection, communication, and data exchange, without requiring user intervention. Mobile Host uses JmDNS, a service discovery protocol which is an implementation of ZeroConf. JmDNS assigns a local domain name to Mobile Host which can be used by other devices to access the services exposed by the host. JmDNS is also totally compatible with other implementations of ZeroConf for other platforms such as Bonjour for Apple.

ZeroConf tackles the Addressing issue by self-assigned link-local addressing. This addressing approach uses a range of addresses reserved for the local network, typically a small LAN or a single LAN segment. The self-assigned addressing simply picks a random IP address in the link-local range and tests it. If the address is not already used, the device is assigned with that name, otherwise it picks another address and checks again whether the name is in used or not. Furthermore, the name-to-address translation capabilities of ZeroConf rely on Multicast DNS (mDNS). In Multicast DNS the DNS-format queries are sent over the local network using IP multicast, therefore no single DNS server with global knowledge is required to answer the queries. Each service or device can provide its own DNS capabilities in such a way that if a query is received asking for its own name the device provides a DNS response with its own address. This name-to-address translation mechanism requires the service names to be unique in the network. Implementations of ZeroConf such as JmDNS (Android) and Bonjour (Apple) automatically rename the service in case of collisions. The service name convention follows a DNS-format which includes the service type, transport protocol, and the domain where the service is exposed. In case of MANETs the domain happens to be "local." and the type should follows the DNS SRV (RFC 2782) Service Types standards (38). For example, the service name *_http._tcp.local.* corresponds to a Mobile Host service of type *http*, for web services, provided through *tcp* protocol in the *local* network.

The final element of ZeroConf is the service discovery. This service discovery enables to other devices to find all the available instances of a particular type of service and to maintain the service directory. Once a device has discovered a service then it resolves the service name to an IP address and port number which can be used later on for the service invocation. The service directory provides a layer for indirection between a service and its current IP address and port. Furthermore, this indirection enables the application to keep a persistent list of available services and resolve an actual network addresses just prior to using a service. The service directory can be relocated dynamically with low network traffic penalties. This dynamical updating capability of the service directory addresses the mobility of the devices which can enter and leave the MANETs at any time. The service discovery is accomplished by sending an mDNS query for a given type a domain. Later, all the matching services reply with their names. The service names received are listed in the local service directory.

ZeroConf is a service-oriented discovery mechanism. The devices query for service not for host providing the services. The service directory stores service names instead of addresses. If, due to the mobility of smart phones, the IP address, port number, or

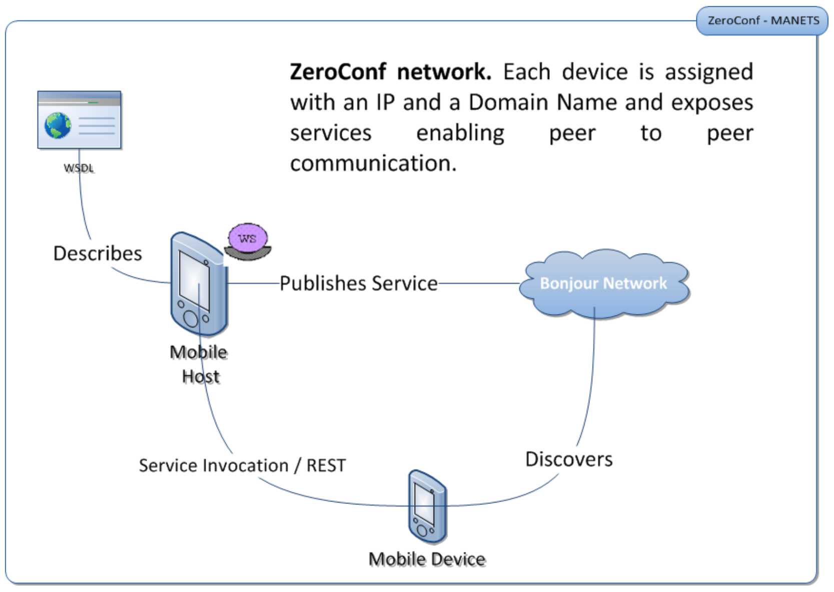

ZeroConf - MANETS
**ZeroConf network.** Each device is assigned with an IP and a Domain Name and exposes services enabling peer to peer communication.
WSDL
Describes
WS
Mobile Host
Publishes Service
Bonjour Network
Discovers
Service Invocation / REST
Mobile Device

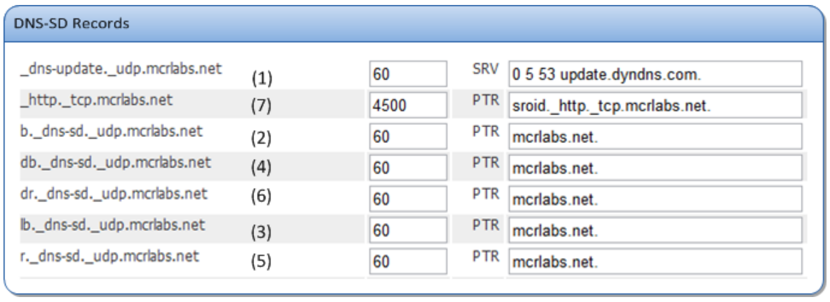
**Figure 4.1:** Discovery Mechanism in Mobile Ad-hoc Networks

host name changes, the devices can still invoke the mobile web services. The figure 4.1 illustrates the discovery mechanism in MANETs.

### 4.1.2 Mobile Host in Wide Area Networks

Mobile Host can easily scale in large networks with high mobility nodes by including Wide Area Bonjour support based on DNS Service Discovery. The DNS Service Discovery enables the service discovery via DNS records in the wide area network and also the self-configuration of devices in order to be accessible from other devices.

The Wide-Area DNS Service Discovery automatically advertises a set of services by simply adding a few records to a DNS server. The first set of records aims the domain enumeration records, inviting the clients to browse the domain. The second set of records is meant to list the services entities. Finally, the third set of records describe each named service, previously defined in the second set of records, with SRV and TXT records. The TXT and SRV records describe the service end point and port. The figure 4.2 illustrates the DNS records necessary for advertising a web service in DNS-SD. From the figure, the record (1) enables the DNS dynamic updates, in this case the updates come from Mobile Host when it updates its information such as services or current address. (2) and (3) enables the domain to be discovered. (4) Set the domain



DNS-SD Records

| | | | | |
|---|---|---|---|---|
| _dns-update._udp.mcrlabs.net | (1) | 60 | SRV | 0 5 53 update.dyndns.com. |
| _http._tcp.mcrlabs.net | (7) | 4500 | PTR | sroid._http._tcp.mcrlabs.net. |
| b._dns-sd._udp.mcrlabs.net | (2) | 60 | PTR | mcrlabs.net. |
| db._dns-sd._udp.mcrlabs.net | (4) | 60 | PTR | mcrlabs.net. |
| dr._dns-sd._udp.mcrlabs.net | (6) | 60 | PTR | mcrlabs.net. |
| lb._dns-sd._udp.mcrlabs.net | (3) | 60 | PTR | mcrlabs.net. |
| r._dns-sd._udp.mcrlabs.net | (5) | 60 | PTR | mcrlabs.net. |

**Figure 4.2:** DNS-SD Records

to be chosen as the default. (5) For this domain to show up in the list of potential registration domains and (6) to be chosen as the default registration domain. Finally, (7) is used to enable clients, that do empty-string domain requests, to browse not only the ”local.” zone.

When a client tries to discover a service, it needs to determine whether to use Wide-Area DNS-SD to browse for services. To do so, it first pre-appends the text $lb.\_dns - sd.\_udp$ to the default DNS domain and then does a query for PTR records. The client should receive as response the domain it is trying to browse for services. For example, if the client tries to discover mobile web services in the domain *mcrlabs.net* the query for PTR records ($lb.\_dns - sd.\_udp.mcrlabs.net$) should return the same *mcrlabs.net* domain. After confirming that the domain is browsable for services the client can discover all the services registered under that domain. The client receives updates about the services via multicast packages sent through the network. Moreover, a client can browse several domains at a time, discovering a wide spectrum of services in the Internet.

Mobile Host discovery mechanism relies in DNS-SD for maintaining a directory of the mobile web services available in the Internet. Consequently, the clients just need to browse service domains and discover the registered services. Nevertheless Mobile Host registers its services in the DNS-SD once when it starts, the mobility nature of the devices require that Mobile Host updates its IP in the DNS Service Discovery records every time it moves from one wireless network to another. When a request comes, the DNS routes it to the most recent address updated by the mobile device. This way the dynamic nature of the devices is addressed and the services can be invoked from other devices in the network. The figure 4.3 illustrates such a scenario.

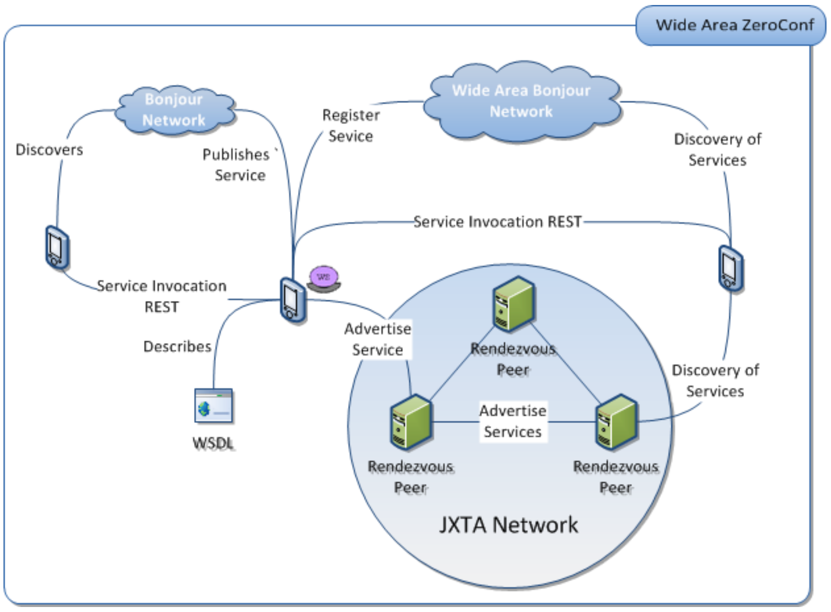


**Figure 4.3:** Discovery Mechanism in Wide Area Networks

## 4.2 Peer-to-Peer Service Discovery in Android

Previous implementations of Mobile Host rely in JXTA networks for advertising, indexing and addressing the services provided by the device. JXTA (Juxtapose) is an open source P2P protocol specification. JXTA defines two main categories of peers named edge peers and super-peers. Further the super-peers can be divided into rendezvous and relay peers. JXTA (Juxtapose) is an open source P2P protocol specification which supports different types of peers to be connected to the network. The general peers are called edge 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.

In the mobile P2P network, the super peer can exist at Base Transceiver Station (BTS) and can be connected to other base stations, thus extending the JXTA network into the mobile operator network. Alternatively, super peer can exist in nodes with capabilities to act as relay and rendezvoud peers. Any Mobile Host or mobile web service client in the wireless network can connect to the P2P network using the node at base station as the rendezvous peer. The super peer can also relay requests to and from JXTA network, to the smart phones. Standalone systems can also participate in such a network as both rendezvous and relay peers, if the operator network allows such functionality, further extending the mobile P2P network (37). This study does not discard this approach but instead we update it to the latest Android developments and extend it by adding ZeroConf and DNS-SD support. Once the virtual P2P network is established, the services deployed on Mobile Host are to be published as JXTA advertisements, so that they can be sensed as JXTA services among other peers, specifically by rendezvous peers. Each mobile web service, deployed in Mobile Host, shares its JXTA advertisement containing the WSDL file of the service.

Generally web services are published by advertising WSDL descriptions in a UDDI registry. However, due to the huge number of services possible with Mobile Host and the dynamics of the mobile networks, a centralized solution for publishing and indexing of services is not the best approach since such centralized solutions can have bottlenecks and can make single points of failure. 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. Lifetimes give the opportunity to control out of date resources without the need for any centralized control mechanism. To extend the life time of an advertisement, the advertisements are to be republished. Later, the services can be discovered by using the keyword based search mechanism provided by JXTA. However, this search happens to be basic and thus returning a large number of services that match the keyword. Moreover, the discovery client in mobile web services scenario is a smart phone. So the result should be rather small and meaningful so that the user can scroll through a list of services and can select the service that fits the best to her/his needs. To tackle this problem this work considers the original Lucene filter mechanism of Mobile Host and proposes a Semantic Service Search.

## 4.3 Keyword-Based Service Search with Solr

Mobile Host extends the search criteria to WSDL level. This means that search parameters are not restricted to the JXTA advertisement details. The search also extends by looking up the WSDL tags and information. This approach assumes that people usually express their opinion by using frequently used words and the frequency of a keyword in WSDL description is also relevant. Hence the basic JXTA search is improved by means of Solr (39). Solr is an open source search platform based on Apache Lucene Project (40). Solr supports full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich documents handling such as Word and Pdf files, geospatial search among others. Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat. Moreover, Solr provides a REST-like HTTP/XML and JSON APIs to access the index data.

The keyword service search mechanism proposed in this work utilizes WSDL for indexing the services discovered in the peer-to-peer JXTA networks. When a new mobile web service advertisement is discovered by the rendezvous peers, the WSDL description extracted from the advertisement is indexed in Solr. In similar fashion, when an advertisement expires, its corresponding WSDL description is removed from the Solr indexes. When a client tries to discover the mobile services, it sends a query to the rendezvous peers in the network along with its service preferences (e.g. environment and context information). The query is sent via HTTP GET on the select URL with the query string in the $q$ parameter. The query string contains the keywords to filter the services. For example, the URL $http://rendezvous:8983/solr/select/?q = notification + in + android\&rows = 10\&indent = on$ queries for the services which contain the keywords $notification$ and *android*. The list of services that matched the search keywords are sent to the client in XML format. The application parses the XML response and displays the list of services to the user who can select the service which fits the best his/her requirements.

In terms of performance this mechanism shows reasonable time responses. From the table 4.1 we can observe that the highest time response ($\approx 3.01$ seconds) occurs under 3G networks. Moreover, the time responses under Wi-Fi networks are below 0.4 seconds. The experiment considered a mobile client application in Android querying an index of 1000 services in Solr. Nevertheless this mechanism performs well and

**Table 4.1:** Semantic and keyword service search performance in seconds

| Mechanism | Wi-Fi LAN | Wi-Fi WAN | 3G |
|---|---|---|---|
| Solr | 0.1947 | 0.3502 | 3.0132 |
| Semantics, OWL-S | 7.0653 | 6.9986 | 7.4729 |

provides short time responses, it lacks the accuracy in the results. For example, when querying for *notification* services in *android* the results included all the services that contain at least one of the keywords thus returning a large list of services that include services for android, not necessarily notification services, and also notification services for other operating systems. The expected result for such a query is the list of services that contains both keywords in their descriptions. Solr prov ides a high performance keyword-based service search but lacks semantics capabilities to enrich the meaning of the results. By adding semantics to the service search mechanism it is possible to improve the quality of the list of services retrieved during the discovery. For example, Solr does not connect the keywords *notification* and *android* in anyway and its search mechanism is limited to find keywords in the service description. In contrast, with semantics it is possible to create the association *notifications for android* thus the search mechanism is able to find notification services specifically for Android, by means of reasoners and semantic rules.

## 4.4 Semantic Search for Discovery of Mobile Web Services

Mobile Host considers ZeroConf and Wide-Area DNS-SD for tackling the naming, addressing and service discovery challenges in the mobile web service provisioning in local and global networks. However, the number of mobile web services is constantly increasing and demands a mechanism for filtering the services in order to provide the service that suit the best for the user requirements in terms of context, resources and functionality. To address this issue we propose a Semantic Search for Discovery of Mobile Web Services.

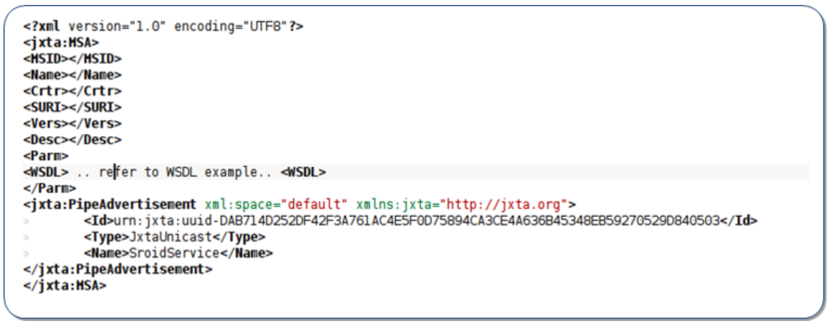
As it was mentioned before, Mobile Host supports JXTA networks for advertising, indexing and addressing the services. Each mobile web service in Mobile Host shares its

JXTA advertisement which includes the WSDL file of the service. WSDL is an XML language to formally describe a Web service. The WSDL description specifies the address, allowable communication mechanisms, interface, and message types of a Web service. In short, a WSDL description provides all the information a client needs to use a Web service. WSDL 2.0 was declared a W3C recommendation in June 2007. This second version of WSDL was created to address issues with WSDL 1.1, many of which had been identified by the Web Services Interoperability (WS-I) organization (41). In addition, WSDL 2.0 has good support for HTTP bindings. Consequently, WSDL is suitable not only for SOAP architecture but also for RESTful architecture. A WSDL description contains all the details of a Web Service such as the service's URL, communication mechanism it understands, what operations can be performed, among other. This information is used by the clients in order to invoke and consume the services.

When Mobile Host enters the JXTA network it advertises its services along with their WSDL specification. The rendezvous peers discover the new edge-peer in the network and stores the advertisements received from the edge-peer. Later, the rendezvous peer parses the JXTA advertisement of the service and extracts the embedded WSDL. Further, the rendezvous peer passes the WSDL to the Semantic Search Engine (SSE). The Semantic Search Engine transforms the service WSDL to its corresponding OWL-S representation. The Semantic Search Engine maintains ontology of mobile and cloud based services for discovery. When the rendezvous peer registers a new service a new instance of the service class is created by the SSE. In similar fashion, when the mobile web service is no longer available, the rendezvous peer senses such event and informs the Semantic Search Engine, which deletes the service instance from the ontology. The details of the Semantic Search mechanism are explained in the next section.

### 4.4.1 OWL-S for Mobile Web Service Discovery

Early implementations of Mobile Host rely in text search and contextualization techniques for reducing the service list sent to the user during a service discovery request. Srirama et al. (42) considered JXTA networks for advertising, naming, addressing and discovering the mobile web services. Hence the JXTA search resulted services are ordered according to their relevancy using Apache Lucene tool. Lucene is an open source project that provides a Java based high performance, full-featured text search engine



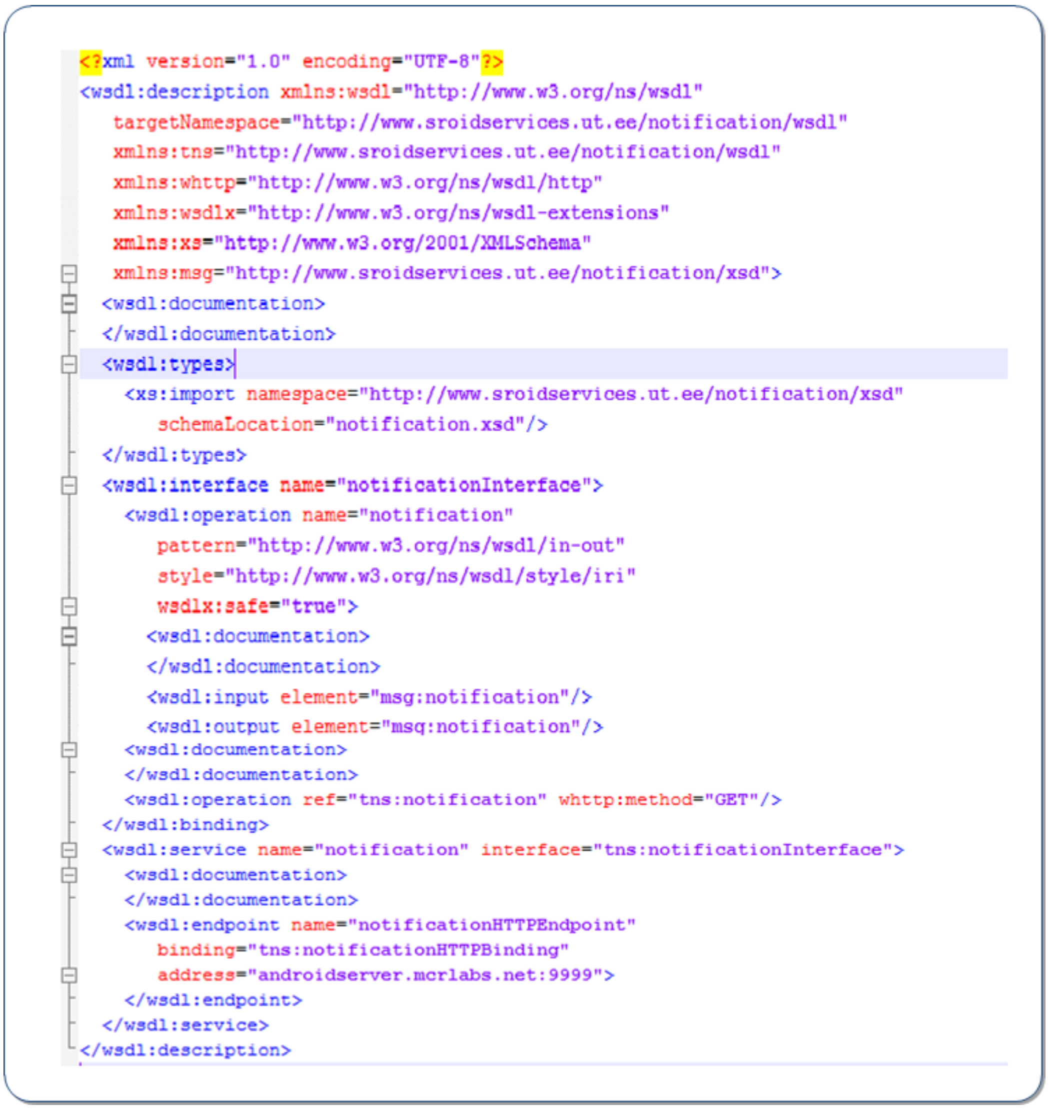
```xml
<?xml version="1.0" encoding="UTF8"?>
<jxta:MSA>
<MSID></MSID>
<Name></Name>
<Crtr></Crtr>
<SURI></SURI>
<Vers></Vers>
<Desc></Desc>
<Parm>
<WSDL> .. refer to WSDL example.. <WSDL>
</Parm>
<jxta:PipeAdvertisement xml:space="default" xmlns:jxta="http://jxta.org">
        <Id>urn:jxta:uuid-DAB714D252DF42F3A761AC4E5F0D75894CA3CE4A636B45348EB59270529D840503</Id>
        <Type>JxtaUnicast</Type>
        <Name>SroidService</Name>
</jxta:PipeAdvertisement>
</jxta:MSA>
```

**Figure 4.4:** JXTA Advertisement - Mobile Host

library. Lucene adds indexing and searching capabilities to user applications. In addition, Lucene can index and search any data that can be mapped to a textual format. Using the tool and its index mechanism the search results are ordered/filtered and the advanced matched services are returned to the discovery client.

In this work we try to shift the discovery mechanism to do not only feature text search but also semantic search. We extend the rendezvous peer functionality by adding semantic support. As mentioned before, when the rendezvous peer discovers a new service provider in the network it stores the corresponding advertisement. The figure 4.4 illustrates the advertisement structure of a sample ”Notification” service spread out through the JXTA network. Further, the rendezvous peer extracts the WSDL from the advertisement and passes it to the Semantic Search Engine. Later, the Semantic Search Engine transforms it into OWL-S representation. The figure 4.6 shows the OWL-S representation corresponding to the WSDL of the Notification Service introduced in the figure 4.5.

OWL-S is an ontology, within the OWL-based framework of the semantic web, for describing web services. It will enable users and software agents to automatically discover, invoke, compose, and monitor web resources offering services, under specified constraints (43). OWL-S aims three tasks Automatic Web service discovery, invocation, and composition and interoperation. The Semantic Search Engine utilizes the OWL-S API to map the WSDL to OWL-S format. OWL-S API provides a Java API for programmatic access to create, read, write, and execute OWL-S described atomic as well as composite services. As a result of this process the service is registered in the JXTA service directory in the form of advertisement but also it is included into the Mobile Web Service Ontology maintained by the Semantic Search Engine. This



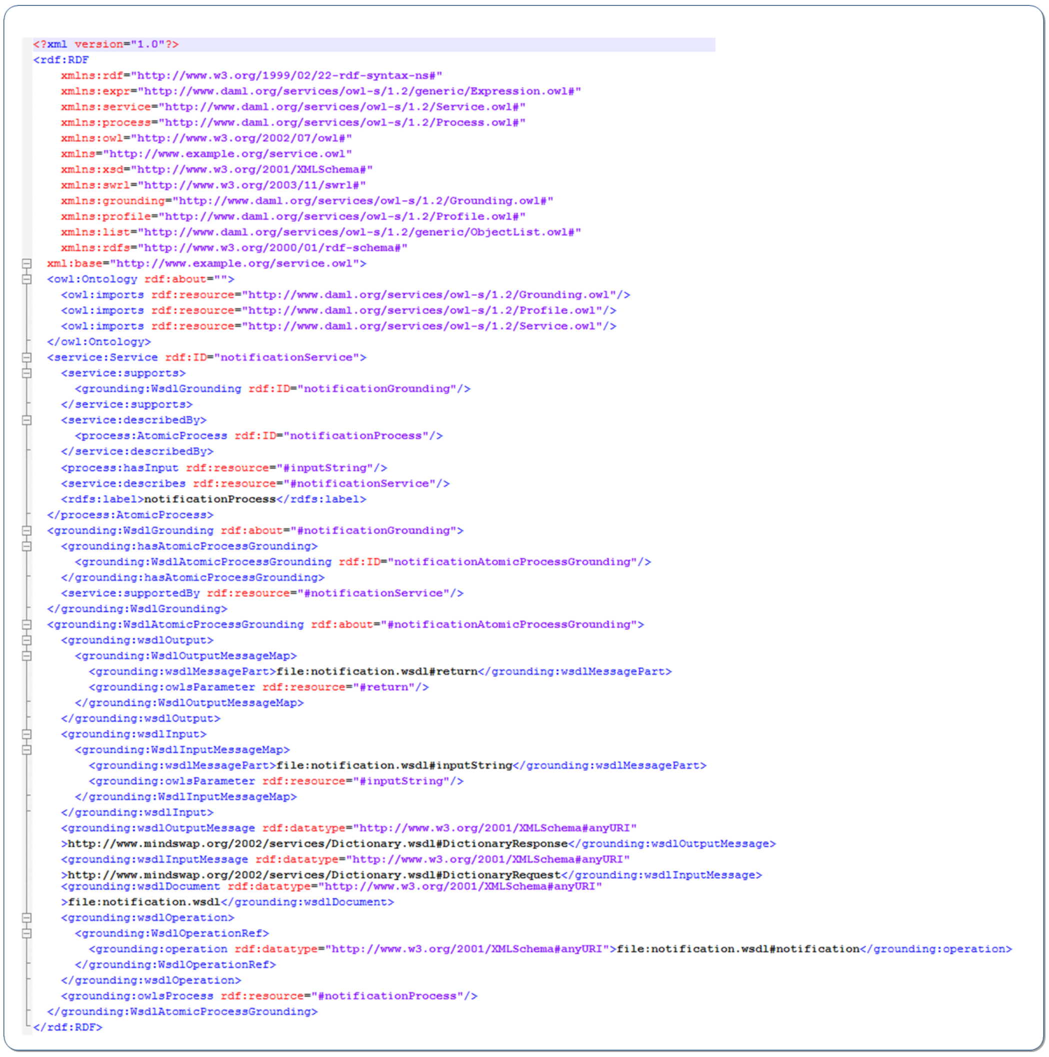
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
   targetNamespace="http://www.sroidservices.ut.ee/notification/wsdl"
   xmlns:tns="http://www.sroidservices.ut.ee/notification/wsdl"
   xmlns:whttp="http://www.w3.org/ns/wsdl/http"
   xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:msg="http://www.sroidservices.ut.ee/notification/xsd">
  <wsdl:documentation>
  </wsdl:documentation>
  <wsdl:types>
    <xs:import namespace="http://www.sroidservices.ut.ee/notification/xsd"
       schemaLocation="notification.xsd"/>
  </wsdl:types>
  <wsdl:interface name="notificationInterface">
    <wsdl:operation name="notification"
       pattern="http://www.w3.org/ns/wsdl/in-out"
       style="http://www.w3.org/ns/wsdl/style/iri"
       wsdlx:safe="true">
      <wsdl:documentation>
      </wsdl:documentation>
      <wsdl:input element="msg:notification"/>
      <wsdl:output element="msg:notification"/>
    <wsdl:documentation>
    </wsdl:documentation>
    <wsdl:operation ref="tns:notification" whttp:method="GET"/>
  </wsdl:binding>
  <wsdl:service name="notification" interface="tns:notificationInterface">
    <wsdl:documentation>
    </wsdl:documentation>
    <wsdl:endpoint name="notificationHTTPEndpoint"
       binding="tns:notificationHTTPBinding"
       address="androidserver.mcrlabs.net:9999">
    </wsdl:endpoint>
  </wsdl:service>
</wsdl:description>
```

**Figure 4.5:** WSDL for the Notification SroidService using REST architecture

Mobile Web Service Ontology is further used when a client queries for a specific type of services to filter the list of services returned in the basis of semantic relevancy and context.

Mobile Host's Semantic Search Engine considers Jena (44) and OWL-S API to support the Service Ontology and to resolve the queries for services from the clients. Jena is a Java framework for building Semantic Web applications. Jena provides a collection of tools and Java libraries for developing semantic web and linked-data apps, tools and servers. When a client needs to discover a service it sends the query through the local network via mDNS to other peers and to the rendezvous peer. The request contains the required service, context information of the client such as geographical location, weather, bandwidth connection, if it is connected through Wi-Fi or 3G, among other, user preferences such as whether to automatically download the content or to be notified about the service availability, the maximum size of downloaded content, etc.

It is logical that the services which are in the same radio-link are discovered faster than those services in the global network. Consequently, the client receives first a list of services published via ZeroConf in the local radio link. Since the number of services in the local networks is most often low these services are filtering using the text-based search provided by framework. Nevertheless, the list of services provided in the local network can be discovered faster, they might not suit the client's requirements and therefore it is necessary to extend the discovery to the services in the wide area network. The wide area network service search is handled by the rendezvous peer. The rendezvous peer receives the service search request and passes it to the Semantic Search Engine (SSE). The SSE keeps an ontology of a large number of services registered in the set of rendezvous peers available in the network. After the query is received, first the services are filtered using the keyword search feature provided by Solr which returns a reduced, but still large, list of services (e.g. 50 out of 1000 services). Later, the SSE performs a SPARQL query over the OWL-S description of the services returned by Solr, retrieving the services that semantically match the client's preferences and needs. SPARQL is a query language initially thought for RDF (Resource Description Framework), a directed, labeled graph data format for representing information in the web. However, Jena and OWL-S API include extensions to support SPARQL queries over ontologies in OWL format. SPARQL can be used to express queries across diverse data sources. SPARQL contains capabilities for querying required and optional



```xml
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:expr="http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#"
    xmlns:service="http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:process="http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.example.org/service.owl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:grounding="http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:profile="http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:list="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.example.org/service.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Service.owl"/>
  </owl:Ontology>
  <service:Service rdf:ID="notificationService">
    <service:supports>
      <grounding:WsdlGrounding rdf:ID="notificationGrounding"/>
    </service:supports>
    <service:describedBy>
      <process:AtomicProcess rdf:ID="notificationProcess"/>
    </service:describedBy>
    <process:hasInput rdf:resource="#inputString"/>
    <service:describes rdf:resource="#notificationService"/>
    <rdfs:label>notificationProcess</rdfs:label>
  </process:AtomicProcess>
  <grounding:WsdlGrounding rdf:about="#notificationGrounding">
    <grounding:hasAtomicProcessGrounding>
      <grounding:WsdlAtomicProcessGrounding rdf:ID="notificationAtomicProcessGrounding"/>
    </grounding:hasAtomicProcessGrounding>
    <service:supportedBy rdf:resource="#notificationService"/>
  </grounding:WsdlGrounding>
  <grounding:WsdlAtomicProcessGrounding rdf:about="#notificationAtomicProcessGrounding">
    <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
        <grounding:wsdlMessagePart>file:notification.wsdl#return</grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource="#return"/>
      </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
        <grounding:wsdlMessagePart>file:notification.wsdl#inputString</grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource="#inputString"/>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlOutputMessage rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.mindswap.org/2002/services/Dictionary.wsdl#DictionaryResponse</grounding:wsdlOutputMessage>
    <grounding:wsdlInputMessage rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.mindswap.org/2002/services/Dictionary.wsdl#DictionaryRequest</grounding:wsdlInputMessage>
    <grounding:wsdlDocument rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >file:notification.wsdl</grounding:wsdlDocument>
    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>
        <grounding:operation rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">file:notification.wsdl#notification</grounding:operation>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>
    <grounding:owlsProcess rdf:resource="#notificationProcess"/>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

**Figure 4.6:** OWL-S representation for the Notification SroidService

graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source graph. In general, the results of SPARQL queries are result sets of RDF graphs.

However, semantic applications are resource demanding and time consuming. Therefore, ontology maintenance and execution of the semantic query is delegated to the rendezvous peers and not to the resource constrained devices. In short, a SPARQL query consist of two parts, the SELECT clause that identifies the variables to appear in the query result set and the WHERE clause that provides the basic graph pattern to match against the data graph. For example, a SPARQL query for service discovery contains the SELECT clause which includes the service and its preferences ($?service$ and $?preference$) and the graph pattern consist of a triple pattern with the association $?hasPreference$ in the object position. The service list, result of the semantic search, is sent asynchronously to the device containing the WSDL of each service. For delivering the results to the device, the Notification Service embedded in Mobile Host is considered. After the results are received by Mobile Host, the service list is shown in the display so the user can decide which service suits better his/her needs. The user can select a service from the service list for invocation. The service WSDL lets the mobile device know about the service invocation details such as the end point, port, and parameters, among others. For example in the WSDL introduced before 4.5 the end point "android.server.mcrlabs.net and port "9999" shall be used for the invoking the service. The invocation process relies in the DNS-SD for addressing the service in such a way that for invoking a service the client application does a simple HTTP request to the corresponding DNS/IP Address and port. The figure 4.7 illustrates the flow of the service publishing process meanwhile the figure 4.8 illustrates how the query and discovery process happens.

The semantic service discovery enriches the service discovery by adding semantics and context-awareness support (service provider and client preferences) to the discovery mechanism, with reasonable performance penalties. The total service query time $Tt$ is:

$$T_t \cong T_{tr} + T_k + \sum_{i=1}^{n}(T_{to_i}) + T_q + T_n \tag{4.1}$$

Where, $T_{tr}$ is the transmission time taken across the radio link for the service query delegation between the mobile phone and the SSE. The value includes the time taken

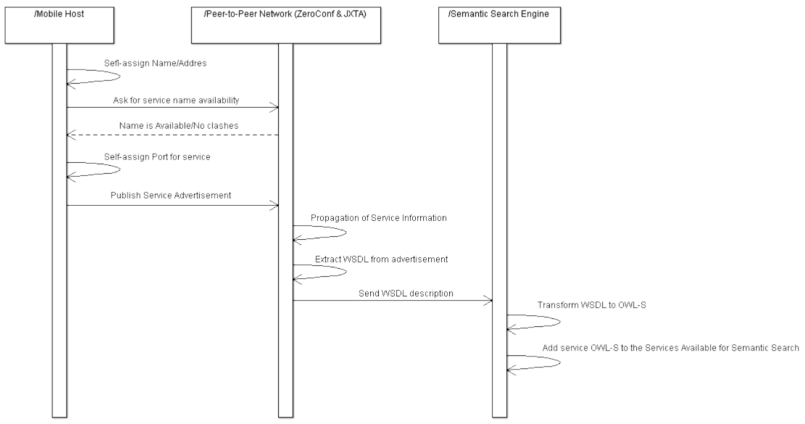

/Mobile Host
/Peer-to-Peer Network (ZeroConf & JXTA)
/Semantic Search Engine
Sefl-assign Name/Addres
Ask for service name availability
Name is Available/No clashes
Self-assign Port for service
Publish Service Advertisement
Propagation of Service Information
Extract WSDL from advertisement
Send WSDL description
Transform WSDL to OWL-S
Add service OWL-S to the Services Available for Semantic Search

**Figure 4.7:** Service Discovery - Publishing Sequence Diagram

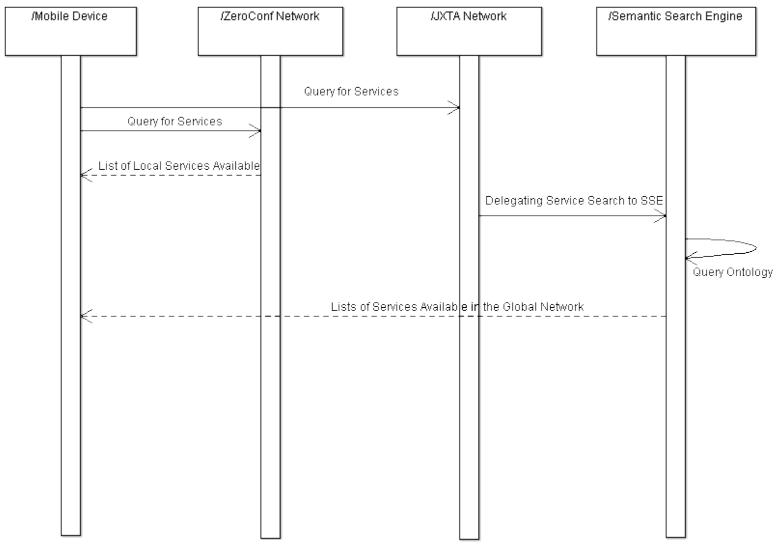


**Figure 4.8:** Service Discovery Sequence Diagram

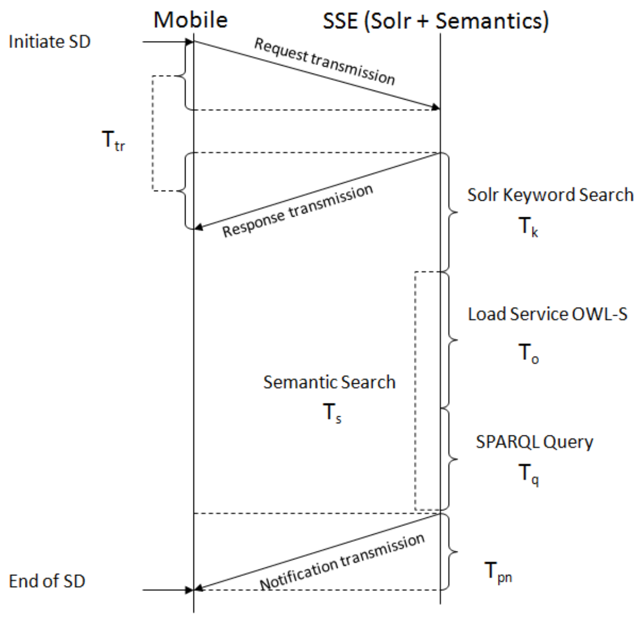


**Figure 4.9:** Service Discovery Cycle - Activities and Timestamps

to transmit the request to the cloud and the time taken to send the response back to the mobile. Apart from these values, several parameters also affect the transmission delays like the TCP packet loss, TCP acknowledgements, TCP congestion control etc. So a true estimate of the transmission delays is not always possible. Alternatively, one can take the values several times and can consider the mean values for the analysis. $T_k$ is the time taken for the keyword text search by Solr. $T_s$ is the sum of the time taken by the SSE to load each OWL-S description in the list of services returned by Solr plus the time taken by the reasoner to query the ontology $Tq$. $\cong$ is considered in the equation as there are also other timestamps involved, like the client processing at the mobile phone, as shown in the figure 4.9. $T_{pn}$, represents the notification time, which is the time taken to send the response of the mobile service discovery to the device.

From the figure 4.10, it can be observed that the mechanism retrieves the list of services in $\approx$ 21 seconds. Most of this time is consumed by the reasoner when querying ontology ($Ts$) after the services are first filtered by Solr ($Tk$), and a little fraction of the total response time ($Tt$) is spent in the network transmission ($Tr$) and the asynchronous notification ($Tn$). On one side Solr presents short time responses but low quality in the list of service retrieved. On the other, the semantic service discovery

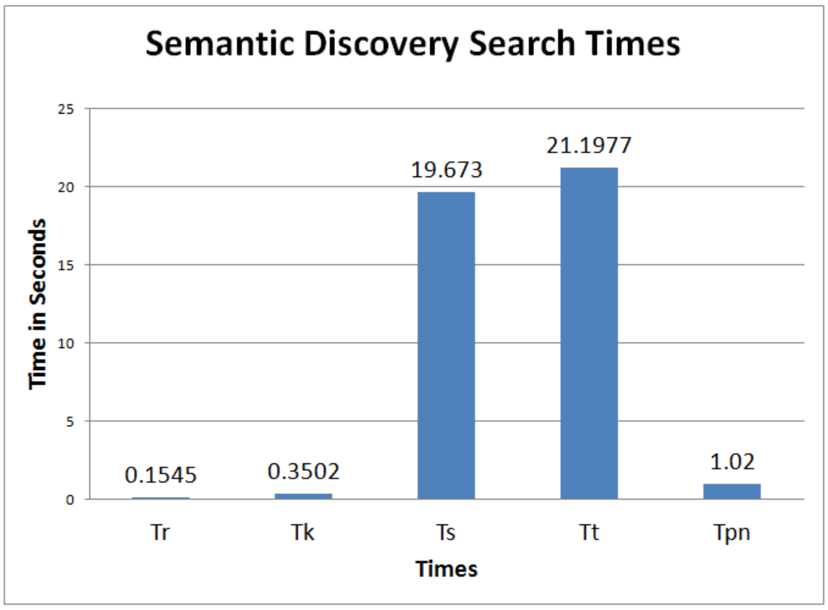


**Figure 4.10:** Service Semantic Search Times

improves the quality of the service search but with some performance penalties. So applying both the mechanisms in tandem would return ideal results. The mechanism was tested using a pool of 1000 service advertisements in the JXTA network. In terms of quality of the results Solr retrieved 50 services out of the 1000 services presented in the index and the semantic search engine returns 5 services out of the 50 returned by Solr, thus improving the quality of the results delivered to the user, fitting better in the mobile devices' screen. In addition, the final list of services is assumed to fulfill the user's requirements better since the search was enriched by means of semantics relationships among the user's context and preferences and the services' description and preferences. However, due to the performance penalties in the semantic service search the asynchronous notification mechanism is required for delivering the query results to the mobile user. The push notification mechanism is discussed in detail in the next chapter.

## 4.5 Summary

This chapter describes the mobile web service discovery mechanism proposed by the thesis. The discovery is a directory-less with overlay support discovery mechanism. It considers peer-to-peer networks for publishing and advertising of services by means of JXTA and ZeroConf protocols and standards. The chapter describes the early peer-to-peer service discovery mechanism based on JXTA peer-to-peer networks and Lucene Framework (as the service search engine) and extends it to a Semantic Mobile Web Services Discovery also over JXTA networks for the discovery but also over ZeroConf networks for the local discovery and global addressing. The approach supports the service discovery in small and medium size networks with low and medium mobility, with the aid of ZeroConf and by supporting the service discovery in large networks such as the wide area network with high mobility of devices, with the aid of JXTA and ZeroConf technologies. The chapter also mentions the realization details of the discovery approach and summarizes the results of the performance analysis, along with the performance model.

# 5

# Push Notification aided by Mobile Host

A key requirement of pervasive mobile applications is that changes occur asynchronously and that it is important that when they occur mobile users are notified in a timely fashion. Without infrastructure for handling the dispatching of asynchronous events, mobile application developers need to code this as part of their applications. This presents developers with extra complexity above and beyond the core application functionality. Most often, the developers need to create again and again the same content delivery mechanism for each application. An alternative is to use a push notification technology that takes care of event notification.

## 5.1 Push Notification Providers

This section describes two commercial push notification offerings: Android Cloud to Device Messaging (AC2DM) and Apple Push Notification Service (APNS), the two widely used push notification mechanisms. In addition, we describe Mobile Host and its role in the push notification mechanism. Moreover, we performed a quantitative performance evaluation of four notification mechanisms (AC2DM, APNS, Mobile Host, and IBM MQTT) and the results are summarized in here.

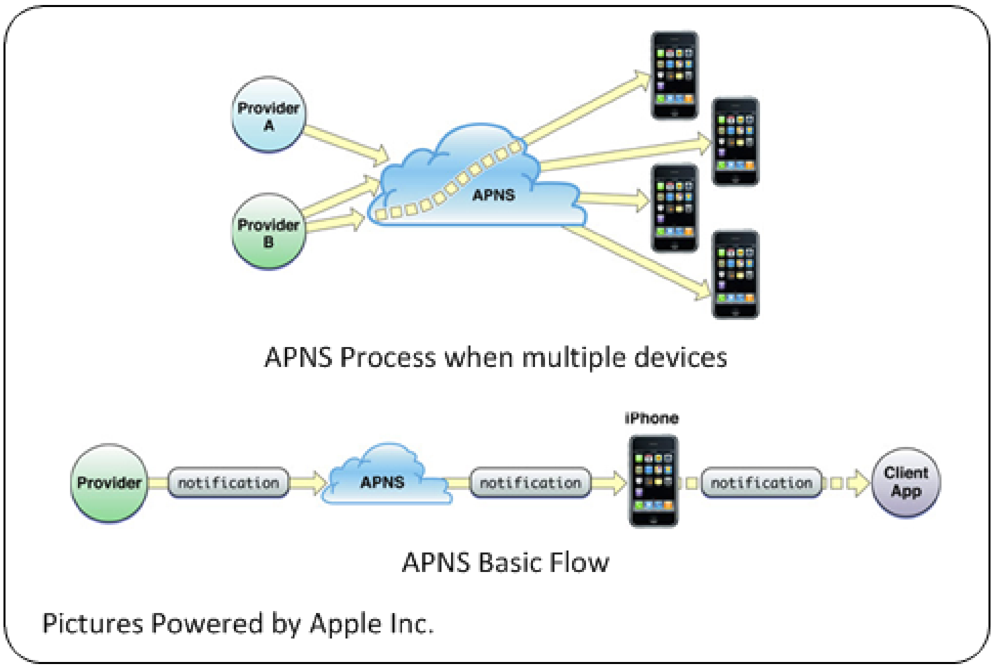


**Figure 5.1:** APNS - General Flow

### 5.1.1 APNS

Apple Push Notification Service is a robust and efficient mechanism which enables the provisioning for remote push notification in devices running iOS. The notifications are delivered by APNS to the devices via a persistent accredited and encrypted IP connection. APNS is a simple and high-capacity transport service including a quality-of-service component providing store-and-forward capabilities. Apple Push Notification service transports notifications from software developers to a given device. The figure 5.1 describes the overall process from the provider until the client application.

During the notification process the service providers, APNS and mobile devices interact with each other through secure communication channels enforced by SSL certificates and keys. When a mobile device connects to APNs it first authenticates and registers itself for the notification service. If the authentication process is successful then APNS sends to the mobile a device token. Each combination of device-application has its own device token that has to be shared with the provider. Further, this device token is used by the providers and APNS for routing the notification to the target device-application. The providers or software developers are the ones who originate the notifications in their server software. When the providers need to notify to the mobile application about the existence of new content or the result of a time-consuming task, they first connect to APNS and send the notification in a JSON format including the device token of the target application-device. Once APNS receives the notification request from the provider it routes the notification to the right device and application based on the device token included in the request, and pass the payload which describes the notification behavior.

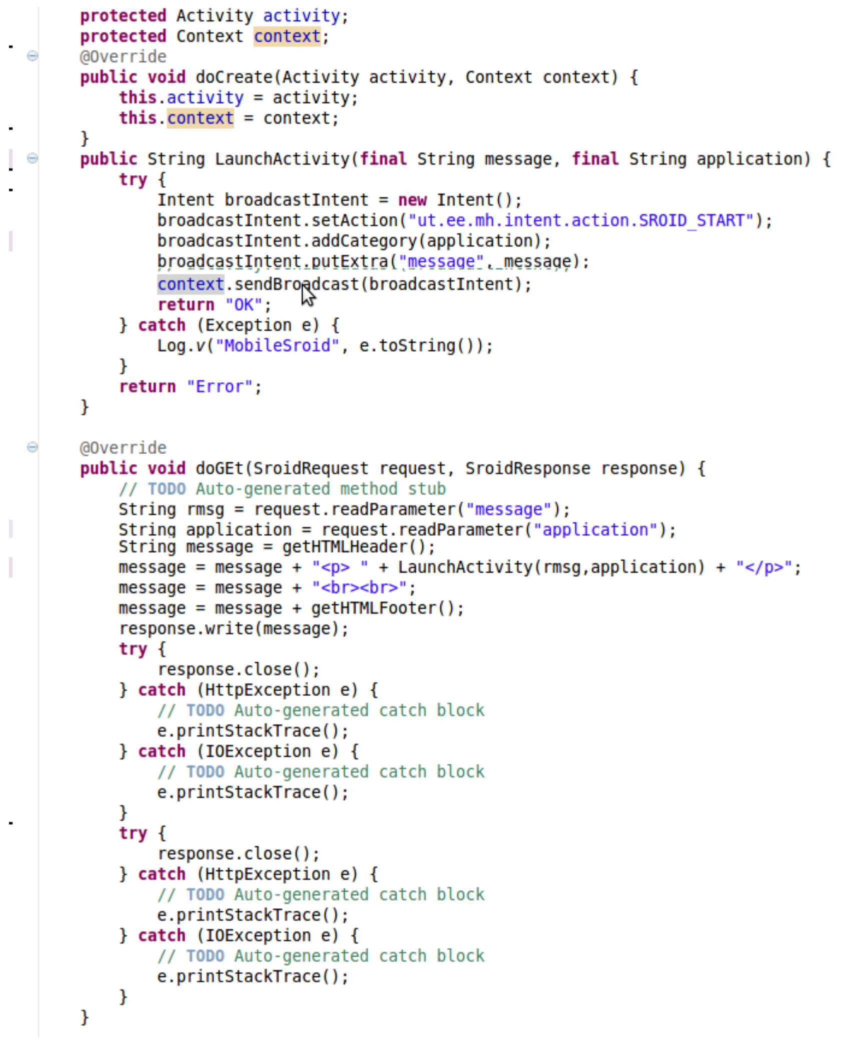
The payload specifies how the users are to be alerted when a notification is received. This payload can be up to 256 bytes. Moreover, the payload consists of a JSON dictionary object which adheres to RFC 4627. The dictionary describes how the user is alerted when the notification arrives. Properties such as the message to be displayed, a number to badge the application icon, a sound to play among others are described in the properties dictionary. The payload is deliberately short since it has been designed to notify about the existence of new content in the server, by delivering a short text message and alerting the user through a dialog and sound. The application can parse the text in the notification following its own rules and semantics and start a new task if it is necessary. For instance, when a new Facebook notification arrives, the notification is displayed and the OS launches the Facebook App but the logic and rules for pulling the new content from Facebook are included inside the Facebook app. Consequently, the mobile application needs to do extra calls to the content provider interface in order to pull the new content available.

### 5.1.2 Mobile Host aided push notification

Smart phones are generally used for accessing different types of services like the location based services, mobile web services, mobile cloud services etc. from different providers. However, one can envision providing services from the smart phone with the latest developments in the mobile devices in hardware (embedded sensors, memory, power consumption, touch screen, better ergonomic design, etc.), in software (more numerous and more sophisticated applications due to the release of iOS and Android platforms), in transmission (higher data transmission rates achieved with 3G and 4G technologies) and in Wi-Fi networks ubiquity. In this thesis we propose updates and extensions to the Mobile Host developed by Srirama et al.

As the popularity of Android rose and with the upcoming of standards like Open Services Gateway initiative (OSGi) framework, we have upgraded the research to the current generation mobile devices and technologies. The OSGi framework is a module system and service platform for the Java programming language. With OSGi, applications or components can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. Mobile Host for Android is realized using Apache Felix, an OSGI implementation for Android. The services run as bundles within Felix and the invocation of the services is through REST protocol. So the services are considered as resources that can be accessed via HTTP requests. Android SDK provides a mechanism to establish Server Sockets communication between the device and the clients; consequently, the HTTP request can be handled from the device. Mobile Host exposes itself and its services to external devices through ZeroConf. It consists of a set of techniques for automatic configuration and creation of a usable local Internet protocol network. ZeroConf dynamically configures the host in the network assigning them an IP address and also a domain name. Furthermore, ZeroConf provides a mechanism for service discovery and domain resolution. Mobile Host uses JmDNS, a service discovery protocol which is an implementation of ZeroConf. JmDNS assigns a local domain name to Mobile Host which can be used by other devices to access the services exposed by the host. JmDNS is also totally compatible with other implementations of ZeroConf for other platforms such as Bonjour for Apple. In addition, Mobile Host also includes Wide Area Bonjour support based on DNS Service Discovery. The DNS Service Discovery enables the service discovery via DNS records in the wide area network and also the self-configuration of devices in order to be accessible from other devices. The mobile device updates its IP in the DNS Service Discovery records every time it moves from one wireless network to another. When a request comes the DNS routes it to the most recent address updated by the mobile device. This way the dynamic nature of the devices is addressed and the services can be invoked from other devices in the network.

Push notification can be offered using the feature of the Mobile Host and as a service from the device. Since the Mobile Host is directly accessible from external devices via JmDNS and offers services, the external applications can directly send notification messages to the device. The applications send the complete message payload as part



```
protected Activity activity;
protected Context context;
@Override
public void doCreate(Activity activity, Context context) {
    this.activity = activity;
    this.context = context;
}
public String LaunchActivity(final String message, final String application) {
    try {
        Intent broadcastIntent = new Intent();
        broadcastIntent.setAction("ut.ee.mh.intent.action.SROID_START");
        broadcastIntent.addCategory(application);
        broadcastIntent.putExtra("message", message);
        context.sendBroadcast(broadcastIntent);
        return "OK";
    } catch (Exception e) {
        Log.v("MobileSroid", e.toString());
    }
    return "Error";
}

@Override
public void doGEt(SroidRequest request, SroidResponse response) {
    // TODO Auto-generated method stub
    String rmsg = request.readParameter("message");
    String application = request.readParameter("application");
    String message = getHTMLHeader();
    message = message + "<p> " + LaunchActivity(rmsg,application) + "</p>";
    message = message + "<br><br>";
    message = message + getHTMLFooter();
    response.write(message);
    try {
        response.close();
    } catch (HttpException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    try {
        response.close();
    } catch (HttpException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

**Figure 5.2:** Notification Service Implementation in Mobile Host

of the request to the notification service on the Mobile Host. With this approach there is no limit to the payload delivered to the device. The figure 5.2 illustrates an implementation of a notification service in Mobile Host.

### 5.1.3 Android Cloud to Device Message Framework

Android Cloud to Device Messaging (AC2DM) is a service that aids developers to send small data from servers to their mobile application on Android devices. AC2DM provides a simple and lightweight mechanism that servers can utilize to tell mobile application to contact the server directly in order to fetch new data available. Moreover, AC2DM handles all the aspects of queuing the messages and delivery to the target application running in the target device.

AC2DM allows third-party application services to send lightweight messages to their Android applications. It is not designed for sending heavy content rather it is intended to notify the Android applications about new data on the server so the application can fetch it following its own logic. The application does not need to be running in order to receive messages. When the notification arrives the OS wakes up the application via Intent Broadcasts. However, the application needs to be configured in development time with the proper broadcast receivers and permissions. Further, when the notification is received AC2DM passes the raw message data straight to the target application. Later the application has full control and can handle it according to its own logic and semantics. For example, the target application can trigger an alert and start synchronization data process. AC2DM is available for devices with the Android 2.2 and the Market Application installed. The service uses an existing connection for Google services; therefore, it also requires the users to set up their Google account on the mobile device.

As mentioned before, the mobile application needs to be set up in development time for receiving notifications. In short, the developer needs to implement a Broadcast Receiver and set up the proper permission in the AndroidManifest of the application. The Broadcast Receiver is meant to handle the Broadcast Intent fired by the OS when the notifications arrive. In addition, Internet, com.google.android.c2dm.intent.RECEIVE, and com.google.android.c2dm.intent.REGISTER permissions are required, when the application starts to register itself to AC2DM. As a result of the registration process the device receives an authorization TOKEN from AC2DM. The device must share its token with the service provider. Later, the service provider includes the token to tell AC2DM to what device the notification is to be sent.

The process to send a notification to mobile devices is as follows. First, the service provider requires a ClientLogin authorization token provided by Google for accessing the notification service. The ClientLogin token authorizes the service provider to send messages to a particular Android application. Consequently, the service provider requires its own ClientLogin token, to access the messaging service, and the registration ID's (Tokens) from its clients, to route the messages to the right target device. Further, the application sends a message to the AC2DM servers which queues are used to store

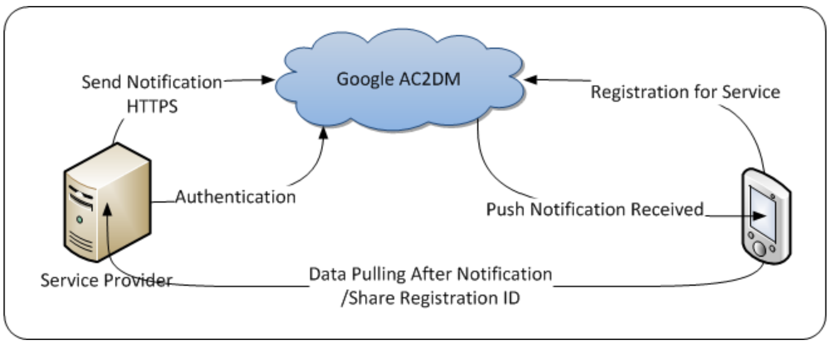


**Figure 5.3:** Google AC2DM - General Flow

the messages in case the device is not accessible. As soon as the device is reachable Google sends the message to the device. When the notification is received in the phone, the OS extracts the raw key/value pairs from the message's payload and passes them to the targeted Android application in a com.google.android.c2dm.intent.RECEIVE Intent as a set of extras. The application receives the notification through the Broadcast Receiver. From this point, the application can handle the notification following its own rules and logic. Finally, AC2DM message size limits itself to 1024 bytes and currently the quota of the service is limited to approximately 200,000 messages per day. The figure 5.3 illustrates the overall process and architecture.

### 5.1.4 IBM MQ Telemetry Transport Protocol

MQTT is a lightweight messaging protocol designed for devices with limited resources and low-bandwidth, high-latency or unreliable networks. The protocol attempts to minimize network bandwidth and device resources. It guarantees different levels of delivery assurance. Some of the main features in this mechanism include the publish/subscribe messaging model for one-to-many message distribution, agnostic of message content when transporting payload, communicate between clients using TCP/IP network protocol, few networks overheads, "Last Will and Testament" publication mechanism for alerting when a client disconnects unexpectedly. Moreover, this mechanism provides three Quality of Service levels. The first level of QoS, "At most once", where messages are delivered with the best effort of the underlying TCP/IP network. In this level messages can be lost or duplicated. Moreover, this level suits well in scenarios where the lost or duplication of a single value is not critical. For example, in a mobile application that gathers accelerometer data every second during the entire day the loss of a single value or some values do not impact the final results. The second level of quality of service, "At least once", guarantees that the messages are delivered at least once. However, duplicate messages can occur. The last level of QoS, "Exactly once", guarantees the messages to be delivered exactly once.

MQTT has an intermediary-based architecture comprising of two main components, a single Broker and the clients that exchange messages. The broker is an intermediary that is aware of all the clients in the network. The clients rely in the Broker for the message delivery. Moreover, MQTT follows a ẗopicsb̈ased publish/subscribe model. A topic is a hierarchical namespace that defines the taxonomy of information sources. Later, the clients can subscribe to the topics of their interest. Further, when a message is published under a topic all the subscribers of that topic receive the message. In addition, MQTT utilizes a keep-alive protocol to monitor the clients' connectivity and accounts for unreliable network connectivity.

The mobile devices register with the Broker before they are able to publish or receive messages. The device sends its clientID, which is a 23 characters long string that is unique across all the clients handled by the Broker. When a client tries to connect with an existing clientID, the current client is disconnected before the new client is reconnected. Once the device is registered with the Broker, the client is able to subscribe to one or more topics. If a mobile device wants to broadcast a message, it sends the message to the broker with its corresponding topic. The Broker then broadcast the message to all the devices subscribed to the message topic. Moreover, the devices need to notify periodically to the Broker about their presence in the network. Therefore, the handset sends PINGREQ (pings) messages to the Broker as part of the keep-alive protocol. If the Broker stops receiving the PINGREQ messages from the client it assumes that the devices is no longer present in the network and disconnects it. It is the client's responsibility to send the messages before the keep-alive period expires. Finally, the messages are not guaranteed to be delivered in order and the delivery order depends on several factors such as the utilization of multi-threads from the client.

## 5.2 Quantitative Analysis

Nevertheless APNS and AC2DM are the most widely used push notification mechanisms. However, they happen to be inappropriate under some scenarios. For example, when real-time notifications are required the AC2DM's reliability does not fulfill the time constraints. On the other hand, APNS' reliability fulfills soft-real time constrains but under high resource consumption penalties; therefore, it is not suitable under high resource constrained scenarios. Consequently, alternative solutions need to be explored such as IBM MQTT and Mobile Host. This section describes and summarizes the quantitative performance evaluation process of four notification mechanisms, AC2DM, APNS, Mobile Host, and IBM MQTT.

### 5.2.1 Description of the experiments

The study is a result of the cooperation between the University of Tartu, Estonia, and University of Auckland, New Zealand. During the last months we have been working together to perform a complete study and compare the performance of several push notification mechanisms in Europe and Australasia. The evaluation considers parameters such as battery consumption, reliability, bandwidth consumption, and latency. The push notification mechanisms evaluated include APNS, AC2DM, IBM MQTT and Mobile Host.

The mobile devices considered in the experiments include an iPhone 4, capable for UMTS/HSDPA/HSUPA networks, with Wi-Fi support (802.11n 2.4GHz only), equipped with assisted GPS and GLONASS, digital compass, 8-megapixel iSight camera, built-in rechargeable lithium-ion battery, three-axis gyro and accelerometer sensor, 16GB storage, 512 MB RAM, 1 GHz Cortex-A8 CPU and PowerVR SGX535 GPU, running iOS 5.1. Also, a Samsung Galaxy SII phone, capable for HSDPA networks, with Wi-Fi support (802.11 bgn), equipped with GPS, digital compass, 8-megapixel camera, standard battery lithium-ion 1650 mAh, three-axis gyro and accelerometer sensor, 16GB storage, 1GB RAM, Dual-core 1.2 GHz Cortex-A9 CPU and PowerVR SGX540 GPU, running Android 4.0 is considered for the experiments.

In case of iOS/APNS the test harness software consists of a set of scripts that run in a server which acts as the service provider. Also, the harness software includes a client iPhone application that receives the notifications and acts as the client. For the server side the gem library jpoz/APNS (45) written in Ruby on Rails (46) was considered. jpoz/APNS is a library that enables to send push notifications from Ruby scripts. The scripts running in the service providing node receives three parameters, the device token, the frequency of messages per minute, and the duration of the experiment in minutes. From the client application perspective, when a notification arrives, it parses the notification's JSON content and extracts the message and the server Timestamp. Right after, the application creates a new entry in a Sqlite database embedded in the application, storing the server Timestamp (when the notification was sent), the client application Timestamp (when the notification was received) and the message received (only for logs).

The AC2DM and Mobile Host harness software follow the same architecture. A server which acts as the service provider runs a set of scripts written in python for sending the notification through the AC2DM service or directly to Mobile Host. In case of AC2DM the scripts receive the registration token assigned by AC2DM to the device, the frequency of messages per minute, and the duration of the experiment. Similarly, the scripts that send notification to Mobile Host, receive the Mobile Host DNS name assigned by the DNS-SD, the frequency of the notifications and the duration of the experiment. There is just one application for receiving messages either from AC2DM or Mobile Host. When the application receives the message, it parses the payload (in JSON format) and extracts the server Timestamp (when the notification was sent) and stores it in a Sqlite databases along with the device Timestamp (when the notification was received) and the messages for archiving purposes.

The harness software for IBM MQTT was developed by the team in New Zealand. The architecture includes a small broker provided by IBM, an Android application and a Java application that runs in the server which acts as the service provider, both developed by the University of Auckland.

The experiments regarding latency, reliability and bandwidth consumption were run five times for a period of two hours each. Moreover, the experiments were run at different times in the day. It tries to improve the fairness of the experiment by not stressing the service provider's networks at a specific time every day. The push notification messages were sent at a rate of three messages per minute. Each message contains a payload of 256 bytes which includes the server Timestamp of when the notification was sent. In addition, the experiments were run through Wi-Fi and 3G networks. The payload is fixed at 256 bytes, as it is the lowest common denominator of the allowed message sizes of the considered approaches. The maximum allowed payloads of APNS, AC2DM, MQTT and Mobile Host are 256 bytes, 1024 bytes, 256 bytes and "unlimited size", respectively.

The experiments regarding the battery life required a different approach due to the nature of the resource usage of the approaches. The experiments ran for a period of two-hours sending notifications period and had one-hour relaxing-period after that. The one-hour-relaxing-period aims to avoid overloading the push notification service provider. For example, it has been observed that AC2DM stops sending messages after a long period of continuous activity. Third-party applications were used to measure the battery level during the experiments. The experiments were run for a 24-hour period or until the battery was totally drained.

The table 5.1 summarizes the results. As can be seen from the graphic in figure 5.4 the IBM MQTT mechanism seems to be the most efficient one in the battery consumption under Wi-Fi networks. Similarly, Mobile Host also shows to perform well under Wi-Fi networks, draining all its battery only after 20 hours of conducting continuous experiments. However, Mobile Host suffers some performance penalties when the connection happens through 3G networks, draining the battery to zero after 16 hours of continuously receiving notifications. In contrast, iOS shows the poorest performance in Wi-Fi since after 12 hours of receiving notifications, the device runs out of the battery. Also, iOS presents the poorest performance under 3G networks since, as shown in the table 5.1, iOS devices drains all the energy after 8 hours of receiving notifications. This is by reason of the keep-alive mechanism that keeps a constant secure connection between the service and the device. AC2DM is a special case, since it is not possible to send notification for long periods of time. We have observed that after one hour of sending the notifications, the AC2DM delivery time grows up exponentially, even taking hours to receive the notification in certain cases. Moreover significant percentages of the messages are also lost after the first hour of sending the messages. Therefore, it is not possible to perform such experiments in AC2DM case. However, a two hours-experiment was performed using AC2DM and the results are shown in the figure 5.5. From the picture it can be observed that the performance of AC2DM is similar to Mobile Host and IBM for short periods of time.

**Table 5.1:** Battery Consumption - Push Notification Mechanism

| Hours | MH / 3G | MH/Wi-Fi | MQTT/Wi-Fi | iOS/Wi-Fi | iOS/3G | AC2DM |
|---|---|---|---|---|---|---|
| 0 | 99 | 99 | 99 | 99 | 99 | 99 |
| 2 | 87 | 89 | 90 | 81 | 70 | 93 |
| 4 | 75 | 84 | 85 | 62 | 39 | - |
| 6 | 62 | 76 | 78 | 43 | 9 | - |
| 8 | 51 | 69 | 72 | 29 | 0 | - |
| 10 | 38 | 61 | 65 | 2 | - | - |
| 12 | 25 | 53 | 60 | 0 | - | - |
| 14 | 12 | 41 | 53 | - | - | - |
| 15 | 2 | 33 | 45 | - | - | - |
| 16 | 0 | 22 | 37 | - | - | - |
| 18 | - | 10 | 30 | - | - | - |
| 20 | - | 0 | 26 | - | - | - |
| 22 | - | - | 15 | - | - | - |
| 24 | - | - | 6 | - | - | - |

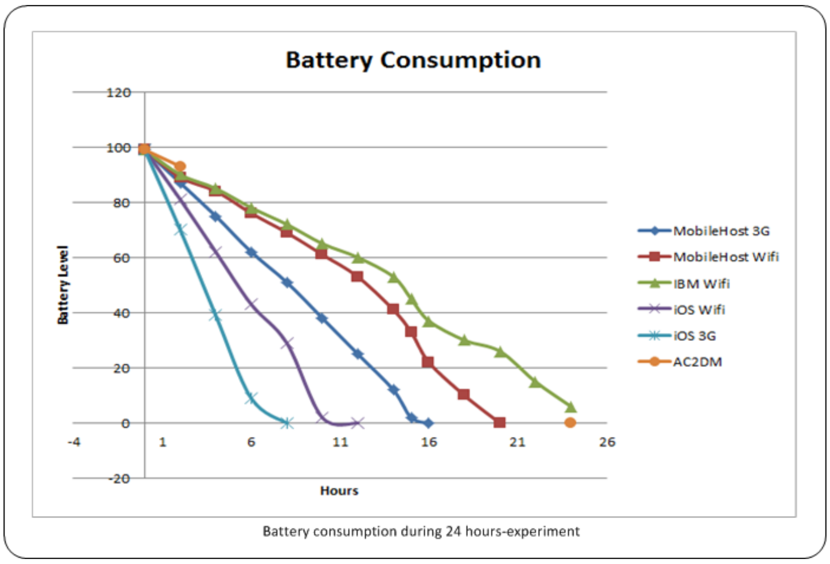


**Figure 5.4:** Battery Consumption - Push Notification Services

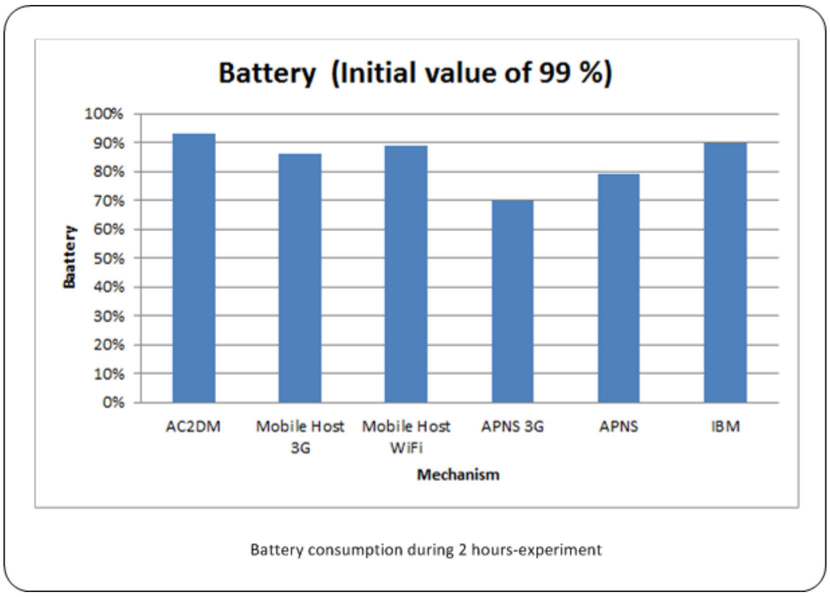


**Figure 5.5:** Battery Consumption of Push Notification Services for 2 hour duration

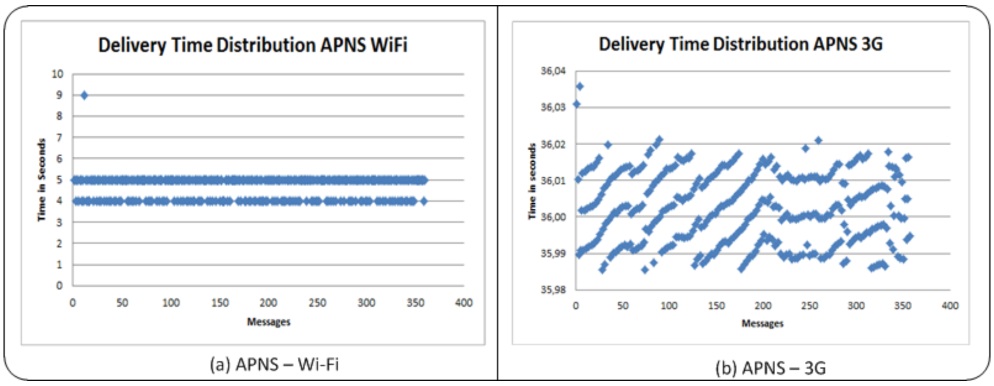


**Figure 5.6:** Delivery Times and Patters - Apple Push Notification Services

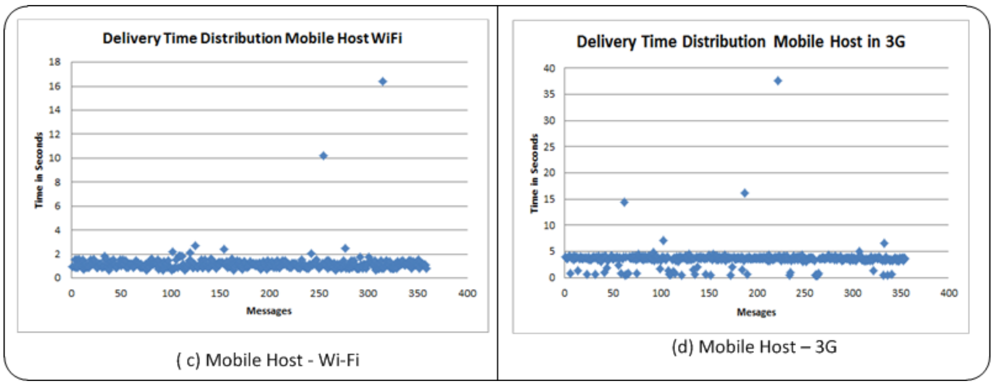


**Figure 5.7:** Delivery Times and Patters - Mobile Host

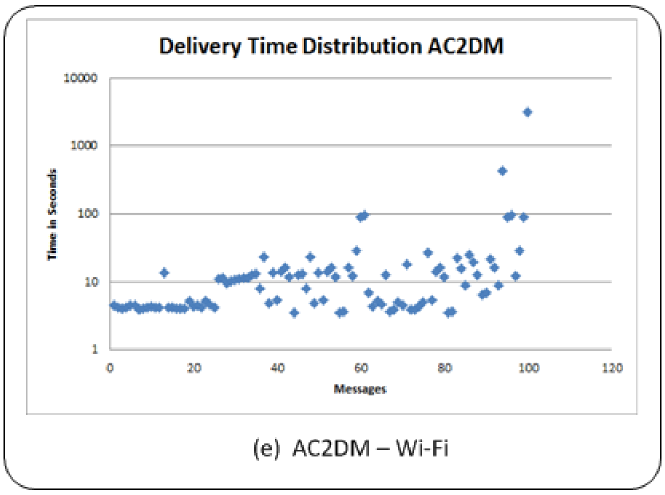


**Figure 5.8:** Delivery Times and Patters - AC2DM

Nevertheless, APNS suffers serious penalties in the battery consumption, it is the most reliable of the services with rate of 100% of messages received. From the graphic in the figure 5.6 (a) it can be observed that the delivery times and patterns when the notifications are sent to iOS devices through Wi-Fi network. The delivery times oscillate between 4 and 5 seconds. Moreover, figure 5.6 (b) illustrates the delivery times and pattern when the notifications are sent through 3G networks to iOS devices. From the graphic, we assume that the queue APNS mechanism prioritizes the notifications based on the number of notifications sent to the device. The more notifications the device receives the lowest priority it gets in the queue. When the device reaches the lowest priority, its priority is reset and the device gets a high priority in the queue for receiving notifications. On the other hand, Mobile Host and IBM MQTT show similar performance in terms of reliability, delivering 98.6% (3G)/99.72%(Wi-Fi) and 97.5%(Wi-Fi) of the notifications respectively. From the graphic in the figure 5.7(c) it can be observed the delivery patterns of Mobile Host in 3G(a) and Wi-Fi(b) networks. The delivery times for Mobile Host oscillate between 1 and 5 seconds in Wi-Fi and up to 10 seconds in 3G networks, performing better than all other mechanisms in terms of delivery times. In terms of bandwidth all the approaches have similar performance. Mobile Host is the mechanism with the highest bandwidth consumption.

Each of the notification mechanisms has strengths and weaknesses and thus are suitable for different scenarios. For example, in scenarios where high reliability is re-

**Table 5.2:** Reliability - Push Notification Mechanism

| Mechanism | Messages Sent | Messages Received | Rate |
|---|---|---|---|
| AC2DM | 360 | 200 | 0.555555556 |
| Mobile Host 3G | 360 | 355 | 0.986111111 |
| Mobile Host Wi-Fi | 360 | 359 | 0.997222222 |
| APNS 3G | 360 | 360 | 1 |
| APNS | 360 | 360 | 1 |
| IBM MQTT | 360 | 351 | 0.975 |

quired APNS emerges as the most suitable solution. However, APNS is not a suitable for devices with high power consumption constraints. Moreover, APNS software development model is quite restrictive and complex which, from the developer's perspective, makes it unfeasible in most of the cases. In contrast, Mobile Host provides reasonable delivery rate (99.72%), battery consumption performance and not so aggressive usage of the network bandwidth. Even though, IBM MQTT presents similar performance characteristics as that of Mobile Host its programming model lacks the Mobile Host's simplicity. IBM MQTT requires including in the mobile application the libraries and jar files that enable the notification handling, thus increasing the footprint of the mobile application. On the other hand, the Mobile Host mechanism is rather light and only requires the creation of one class for handling the notifications. Finally, AC2DM performs well in terms of battery and bandwidth consumption; however, it lacks the reliability. As mentioned before, it has been observed that after longer periods of activity the delivery times increase exponentially and it stops sending the notifications. In addition, AC2DM latencies are bigger than 50 minutes, which are unacceptable under scenarios where soft-real time responses are required. The tables 5.2 and 5.3 summarize the results concerning to latency, bandwidth consumption and reliability.

#### 5.2.1.1 Summary of the push notification mechanisms

To summarize the study, most of the mobile OS providers have their own push notification mechanisms. Most often they rely on the Push Access Protocol (PAP) from Open Mobile Alliance (OMA). The providers also maintain standard set of servers and services, helping the mobile applications in receiving the push notification messages.

**Table 5.3:** Battery Consumption and Latency - Push Notification Mechanism

| Mechanism | Bandwidth(MB) | Avg. TT | Max TT | Min TT | Battery |
|---|---|---|---|---|---|
| AC2DM | 3.80 | 49.38185 | 3152.683 | 3.474 | 93% |
| Mobile Host 3G | | 3.564935211 | 37.579 | 0.406 | 86% |
| Mobile Host WiFi | 5.28 | 1.21435376 | 16.398 | 0.631 | 89% |
| APNS 3G | | 36.00320381 | 36.37824769 | 35.9854941 | 70% |
| APNS | 4.48 | 0.501263574 | 0.66 | 0.436 | 79% |
| IBM | | | | | 90% |

While most of the relevant mobile platforms are providing asynchronous mechanisms (aka notification services) for dealing with remote executions, the mechanisms have certain constraints and limitations such as being platform specific (e.g. AC2DM for Android, APNS for iOS etc.), the size of the message that can be pushed into the device (e.g. 1024 bytes for Android, 256 bytes for iOS, etc.) and the number of the messages that can be sent to a single handset (e.g. 200,000 for AC2DM). Moreover, such mechanisms are considered to be moderately reliable, and thus are not recommended in scenarios that require high scalability and quality of service. For example: AC2DM simply stops retrying after some delivery attempts.

Moreover, by following these approaches the applications are actually competing with the community. Thus the scalability of the applications is at trouble in certain cases. This is what is exhibited by the behavior of AC2DM as the community relying on these services is significantly huge. Alternatively one can rely on his own infrastructure and can establish his own Push Proxy Gateway (PPG), with the IBM MQTT approach. This way, the mobile applications delivered by an enterprise can rely on their own push notification infrastructure. This is certainly possible in most of the cases, depending on the popularity of the applications. Moreover the enterprise can rely on the elasticity of the cloud to vary the scale of the notification services provided for the applications, vertically or horizontally scaling up the PPG when required and scaling down when the revenue from the applications decreases.

Alternatively, this thesis proposes the usage of Mobile Host concept for the push notification services. Mobile Host concept and the Mobile Host aided push notification service have been explained in detail. The push notification mechanism is reliable, fast and reasonable in terms of battery usage. Moreover, this approach pushes the cost of the push notification to the service provider and the client rather than the application provider. For example; envision a mobile application is developed and deployed which uses the push notification and has become very popular. Since both the mobile phone users (user of Mobile Host and client) are aware of the resource (load on battery) and monitory (price of network transmission) costs they are incurring, they can always decide how long to use the application and when to turn down the application. It would also be ideal approach to market a beta applications and applications from startup companies, with the best possible reliability and speed, as one do not have to rely on third party severs and loose reliability or start their own PPG and start incurring costs from the beginning.

## 5.3 Mobile cloud service invocation aided by Mobile Host

Mobile Host and Mobile Host aided push notification are also of significant use in designing architectures for proper invocation of cloud services from mobile devices. The approach can also be applied in several mobile domain related middleware technologies. This section describes how the approach is applied for the Mobile Cloud Middleware (MCM).

### 5.3.1 Mobile Host and Mobile Cloud Middleware

Middleware solutions such as the Mobile Cloud Middleware (MCM) benefit a lot from asynchronous notification capabilities of Mobile Host. In short, MCM tries to counter the problems concerned with the interoperability across multiple clouds, to perform data-intensive processing invocation from the handset and to introduce the platform independence feature for the mobile cloud applications. The middleware provides a unique interface for mobile connection and multiple internal interfaces and adapters, which manage the connection and communication between different clouds. Due to the time and resource consuming nature of the services provided by MCM, it requires the asynchronous notification features. To have a generalized solution, one can make the mobile check for the status of the service regularly. However, this puts a lot of load on the mobile networks. Alternatively, we can make the mobile device a service provider, which was studied in mobile web service provisioning project and this thesis. The study

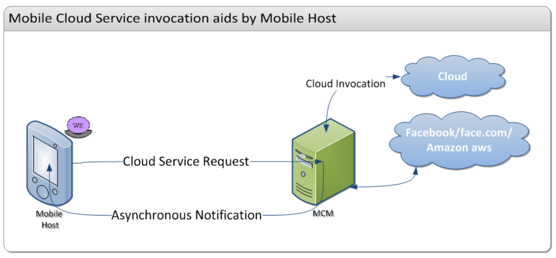

Mobile Cloud Service invocation aids by Mobile Host
WS
Cloud Invocation
Cloud
Facebook/face.com/ Amazon aws
Cloud Service Request
Asynchronous Notification
Mobile Host
MCM

**Figure 5.9:** Mobile Host and MCM

developed a Mobile Host which can be used in providing web services from the smart phones. When the smart phone acts as a server, the mobile cloud service response can be sent directly to the device. Mobile Host for Android is based on REST protocol, where the services are considered as resources that can be accessed through HTTP requests. The services exposed by Mobile Host can be categorized in OSGI Services and Messaging Services for notifications. The Messaging Services interact with the applications installed in the device, passing data between the Mobile Host and the applications. MCM utilizes Messaging Services for reactivating the application as it is waiting for the cloud results.

Figure 5.9 illustrates the invocation process. When a mobile application sends a request to the middleware (MCM), the handset immediately gets a response that the transaction has been delegated to remote execution in the cloud, while the status of the mobile application is sent to local background. Now the mobile device can continue with other activities. Once the process is finished at the cloud, MCM sends a HTTP POST request for the resource http://mobilehost/notification (Messaging Service), with two parameters, application and message, where application corresponds to the mobile application that needs to be notified about the results and message is a JSON string with the results of the invocation. Mobile Host resolves the request, reactivates the application running in the background by sending a message through a Broadcast Intent, and thus the user can continue the activity. The approach can also be used to concurrently execute several tasks in multiple clouds.

MCM and the resource intensive tasks performed on the cloud can easily be envisioned in several scenarios. For example, CroudSTag (34), consists of the formation of a social group by recognizing people in a set of media files stored in the cloud by means of facial recognition technologies and Hadoop MapReduce (47). The application is explained in detail in the next section.

### 5.3.2 Social Group Formation with Facial Recognition and Mobile Cloud Services

CroudSTag facilitates a very interesting scenario as it tries to form social groups of people with common interest. Social networks (48) have become quite popular these days and the creation of social groups, results in sharing and collaboration relationships among the members. CroudSTag is a mobile application developed for Android devices, with the aim of aiding in the social group formation by means of facial recognition technologies and MapReduce (47) video processing. CroudSTag recognizes the people who appear in media content such as pictures or videos and joins them together into a social group. For example, consider a researcher who attends conferences around the world and has a set of media content (pictures and video files) of the people with whom he/she had interacted at the event. The media files are probably taken from his/her mobile itself and are stored on the cloud. The researcher later wants to create and keep connections with his acquaintances on the social network. He/She also wants to group them according to specific interests and would like to follow the groups directly from his/her mobile phone. The scenario can also be envisioned with any other type of the event or community that wants to keep its members in contact, something like alumni.

Figure 5.10 shows the screenshots of the CroudSTag usage scenario which is an extension of the original CroudSTag application presented in our previous work (34). With the help of CroudSTag application the user can upload pictures and videos to the cloud and store them in Amazon Simple Storage Services (S3) (25), as shown in the top cycle of the figure. When the user starts the application, three buttons are shown: facebook Login, Take Picture and Take Video, as shown in screenshot 1. The facebook login goes directly to facebook and authenticates the user getting an authorization token which is used for accessing the facebook graph and also for accessing the user's tags during the facial recognition process. This authentication token is also needed to send the invitation to people, to join the social group. The Take Picture and Take Video buttons are used to take a picture/video respectively, and upload the file to S3. The camera embedded in the device is used to take the media content (screenshot 2) and when the user is finished and taps the save button, he/she is asked to select the cloud provider where the files are going to be stored (screenshot 3). Multiple clouds are considered for keeping the files, as the researcher may store the pictures in his private cloud (something like our SciCloud (49), a private cloud based on Eucalyptus technology with Walrus storage) when in Europe and while traveling through US or Japan he/she may upload them to public clouds like S3 or GoGrid (50). Once the image/video is saved in the cloud, the application comes back to the main menu from where the user can take a new video/picture or start the social group formation process.

The social group formation process starts with facebook login, as explained in above paragraph (screenshot 4). Once the user is logged-in, the application enables a new button "CroudSTag" on the main screen (screenshot 5). Once the user taps the button, the social group formation process is initiated and he/she is asked to select the cloud provider where the media files to be processed are stored (screenshot 6). The media content stored in the cloud is used to detect and recognize user's acquaintances and contact them in facebook. The device sends to the cloud, this time via the Mobile Cloud Middleware (MCM), a request for the social group formation service along with the authorization token from facebook, the mobile host address and port for notifications, and the folder and cloud where the media files are. Moreover, face recognition is performed based on SaaS from face.com. Right after the request is received by MCM, it sends an acknowledgement back to the mobile, notifying that the process has been started. Then the device is free to start other applications or a new process at the cloud. When MCM finishes with the facial recognition process, it sends the list of people recognized, asynchronously to the device via Mobile Host. Once this list is received, the application displays it to the user (screenshot 7), who decides whom to send the invitation to join the social group in facebook. Each invitation is sent in the form of a private message containing a link to accept the invitation and join the social group in the facebook (screenshot 8). The invitees will later join the social group based on their interests. The entire scenario is illustrated in detail in figure 5.10. More details of the application are available at (7).

## 5.4 Summary

The chapter starts introducing the concept of push notifications and its relevance in pervasive mobile applications. In such applications changes occur asynchronously and require the mobile users to be notified in a timely fashion when the changes happen.

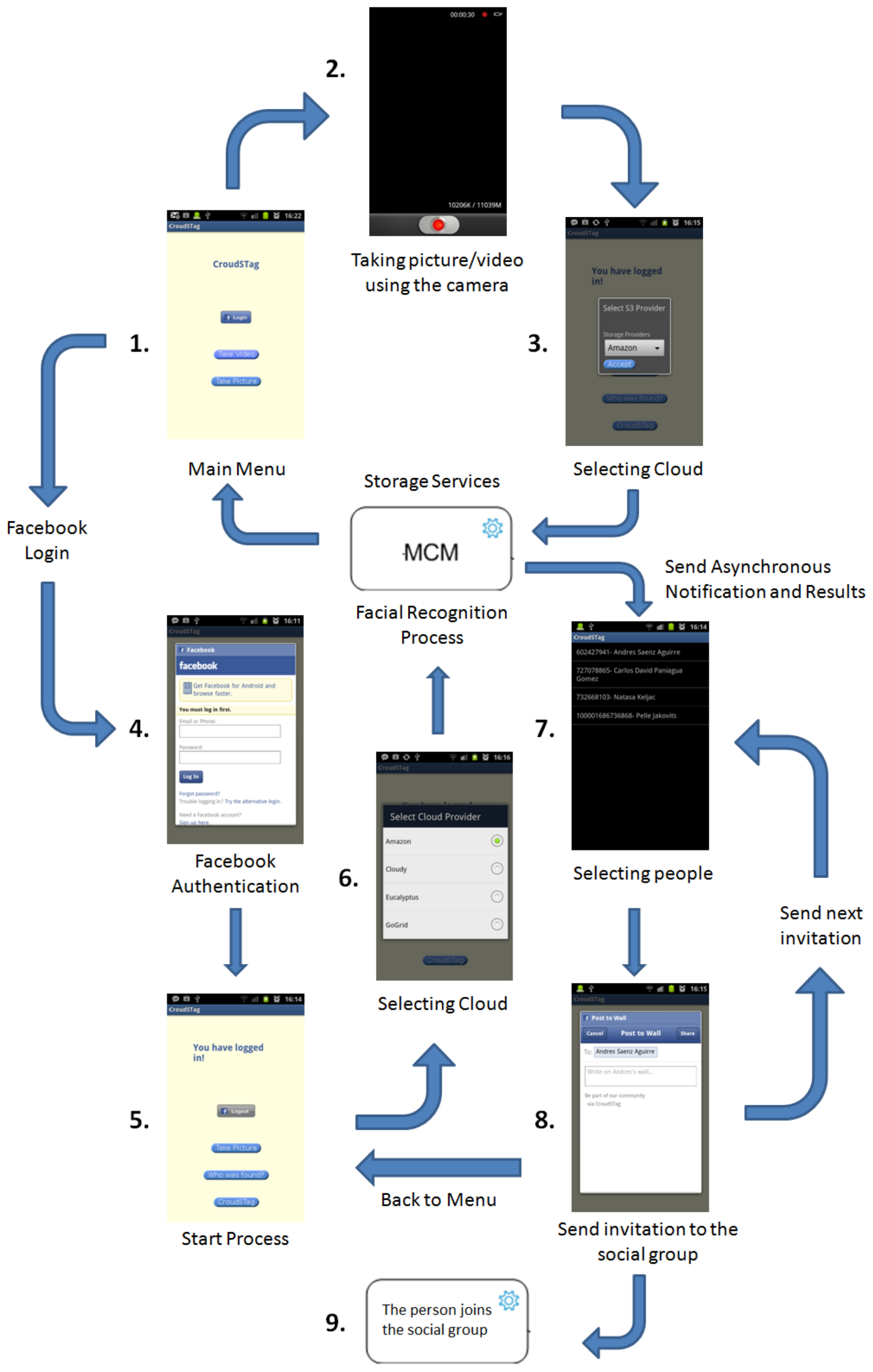


**Figure 5.10:** Screenshots of the CroudSTag usage scenario

Later the chapter describes two popular notification mechanisms which have high presence in the industry (APNS and AC2DM) and two alternative notification mechanisms for resource constraint devices (IBM MQTT and Mobile Host). The four notification mechanisms were extensively tested considering quantitative metrics such as battery consumption, latency and bandwidth consumption. The chapter describes the experiments and summarizes the results. The study identified that the Mobile Host aided push notification mechanism is reliable, fast and reasonable in terms of battery usage. Middleware solutions such as the Mobile Cloud Middleware (MCM) also benefit a lot from the asynchronous notification capabilities of Mobile Host. The scenario is demonstrated with an application.

# 6

# Conclusions

The emerging mobile computing domain has enabled the next generation of rich mobile applications that benefit from the mobile and ubiquitous nature of the mobile devices but at the same time also benefit from the vast amount of resources available in the clouds. This synergy among clouds and mobiles opens the possibility for creating more powerful and rich applications fostering context-awareness and ubiquitous scenarios. However, for the cloud computing domain to penetrate the mobile computing domain it requires proper mechanism of communication between the clouds and the smart phones. Middleware approaches such as the MWSMF and MCM support the provisioning of web services and cloud services, respectively, to the mobile devices.

While invocation of cloud and web services is feasible from mobiles, provisioning of services from mobiles is also of significant interest in cases such as extracting the context and preferences from mobiles, mobile P2P, direct invocation of services on devices, push notifications to send alerts to mobiles etc. The Mobile Host realized by Srirama et al. addressed this issue of service provisioning from mobiles and further provided proper discovery and QoS mechanisms. This work has developed a Mobile Host for Android, updating the earlier implementations of Mobile Host to the current technologies and platforms. Moreover, it was extended to be easy to integrate with other middleware solutions such as MCM for clouds. In terms of web service provisioning from smart phones, this work also proposed a new Mobile Host architecture which relies in Apache Felix, OSGi Framework for Android, and REST protocol. The proposed architecture is based on REST protocol instead of the traditional SOAP, as it improves the performance of Mobile Host by reducing the overloads in the traffic, removing the necessity for compressing algorithms, and replacing the necessity of encryption mechanisms for security with the utilization of HTTPS connections. Moreover, the new developments bring the concept of Mobile Host to the nowadays mobile applications market since it has been developed for Android; the most widely used mobile operating system.

The popularity of web services, mobile web services and cloud services results in a massive pool of services available for mobile devices. This massive pool of services demands a proper discovery mechanism which is efficient in terms of resource consumption from the mobile devices and provides the most relevant services available to the user taking into account his/her preferences and current context. This work extended the previous discovery mechanism based on JXTA peer-to-peer networks and Lucene Framework (as the service search engine) to a Semantic Mobile Web Services Discovery also over JXTA networks for the discovery but also over ZeroConf networks for the local discovery and global addressing. The current discovery mechanism considers challenges such as the mobility of the networks and the size of the networks. The thesis addresses these issues by supporting the service discovery in small and medium size networks with low and medium mobility, with the aid of ZeroConf and by supporting the service discovery in large networks such as the wide area network with high mobility of devices, with the aid of JXTA and ZeroConf technologies. The thesis proves that it is feasible to provide such service discovery mechanism with the aid of cloud services, peer-to-peer networks and OWL-S standards.

The work also developed a Push Notification service for mobiles, using the Mobile Host feature of the device. The Mobile Host based Push Notification service was extensively tested and compared with other approaches for push notifications such as APNS and AC2DM. The study realized that the Push Notification services are a key requirement of pervasive mobile applications where changes occur asynchronously and it is important that when they occur mobile users are to be notified in a timely fashion. Similarly, when developing rich mobile applications that require offloading of information to be processed by more power appliances, as in the case of the mobile cloud services where mobile devices offload, most often sensor data, for data mining and pattern discovery. Moreover, the push notification services are also a critical component for context-awareness applications where the mobility of the devices demands to update the user's context frequently. Therefore, we decided to perform a quality analysis of the most widely used push notification mechanisms in the market. The study considered the Apple Notification Services (APNS), Android Cloud to Device Messaging Framework (AC2DM), Mobile Host based Push Notification and IBM MQTT. The thesis summarized the results of the analysis highlighting strengths and weaknesses of each mechanism in terms of resource consumption, reliability and latency. The study identified that the Mobile Host aided push notification mechanism is reliable, fast and reasonable in terms of battery usage. Middleware solutions such as the Mobile Cloud Middleware (MCM) also benefit a lot from the asynchronous notification capabilities of Mobile Host. The scenario is demonstrated with the CroudSTag application.

# 7

# Future Research Directions

This work has developed a Mobile Host for Android, updating the earlier implementations of Mobile Host to the recent technologies and platforms. As part of the future research directions, we are interested in extending the architecture to other common mobile platforms in the market like the Apple iOS and Windows Phone7. We are also interested in developing an eclipse plugin for easing the development of new services for the Mobile Host. The plugin should ease the adoption of Mobile Host and services in different application domains and by wider community.

The thesis also has shown the feasibility of a P2P based service discovery mechanism for mobile web services that can also be applied for mobile cloud services. The study leaves significant scope for improvements in several areas and offers new research opportunities. For example, the current service discovery takes into account the users' context and preferences. However, the service discovery and provisioning can be extended to dynamically adapt itself to the dynamics in the context due to the mobile nature of the devices. Similarly, the emerging mobile cloud computing domain demands that web services provided from smart phones and cloud services provided in the clouds can be abstracted under the same WSDL description, fostering the integration and composition of cloud based services with mobile web services. This composition aids in scenarios such as the Mobile Enterprise envisioned by Srirama (8) where the easy integration among enterprise services, cloud services and mobile web services enables the creation of rich and meaningful applications to the users. We are interested in integrating the updates and new developments of Mobile Host to the Mobile Enterprise proposed by Srirama. This integration also includes the adaptation of the service discovery mechanisms proposed in this work.

However, enterprise networks deploy disparate applications, platforms, and business processes that need to communicate or exchange data with each other or with the Mobile Hosts. The applications, platforms and processes of enterprise networks generally have non-compatible data formats and non-compatible communications protocols. Besides, the discovery approach of the mobile cloud and web services offered solutions in disparate technologies such as JXTA. This leads to serious integration problems within the networks. The integration problem extends further if two or more of such enterprise networks have to communicate among themselves. The mobile web services mediation framework (MWSMF), introduced by Srirama et al. (51) tried to address this enterprise service integration problem. ESB is used as the background technology in realizing the mediation framework. We are interested in adding the Mobile Host based push notification technology and upgrading the integration of the discovery mechanism to the mediation framework.

Moreover, the composition of mobile services and cloud services fosters the realization of context-aware applications and technologies that benefit different domains such as wearable computing, smart environments, smart cities, location based services, analysis of data gathered by body sensors, among others. We are interested in realizing some of these applications, demonstrating the feasibility and applicability of Mobile Host and Enterprise.

# 8

# Sisukokkuvõte

Viimase viie aasta jooksul on mobiilsed seadmed nagu sülearvutid, pihuarvutid, nutitelefonid jmt. tunginud peaaegu kõigisse inimeste igapäevaelu tegevustesse. Samuti on põhjalik teadus- ja arendustegevus mobiilsete tehnoloogiate vallas viinud märkimisväärsete täiustusteni riistvara, tarkvara ja andmeedastuse alal. Tänapäeval on mobiilsed seadmed varustatud sisseehitatud sensorite, kaamera, puutetundliku ekraani, suurema hulga mäluga, kuid ka tõhusamate energiatarbemehhanismidega. Lisaks on iOS ja Android operatsioonisüsteemide väljalaske tõttu suurenenud nii mobiilirakenduste arv kui keerukus, pakkudes arvukamalt kõrgetasemelisi rakendusi.

Sarnaselt on toimunud olulised arengud ja standardiseerimisele suunatud jõupingutused veebiteenusete valdkonnas ja elementaarsetele veebiteenuste ligipääsu kasutatakse laialdaselt nutitelefonidest. See on viinud loogilise järgmise sammuna veebiteenuste pakkumiseni nutitelefonidest. Telefonidest veebiteenuste pakkumise kontseptsioon ei ole uus ning seda on põhjalikult uurinud Srirama, kes pakkus välja Mobile Host (Mobiilne Veebiteenuse Pakkuja) kontseptsiooni. Algne realisatsioon kasutas aga aegunud tehnoloogiaid nagu JMEE, PersonalJava, SOAP arhitektuur jne. See töö uuendab Mobile Host'i kasutades uusimaid tehnoloogiad, nagu Android OS ja REST arhitektuur, ning pakub välja teenusemootori, mis põhineb Apache Felix'il - OSGi platvormi realisatsioonil piiratud ressurssidega seadmetele.

Hämmastava kiirusega toimunud arengud mobiilsete arvutuste vallas võimaldavad uue põlvkonna veebirakenduste loomist valdkondades nagu keskkonnateadlikkus, sotsiaalvõrgustikud, koostöövahendid, asukohapõhised teenused jne. Sellised rakendused saavad ära kasutada Mobile Host'i võimalusi. Selle tulemusena on klientidel ligipääs väga suurele hulgale teenustele, mistõttu tekib vajadus efektiivse teenuste avastamise mehhanismi järele. See töö pakub välja kataloogipõhise avastusmehhanismi võrgu ülekatte toega suurtele, kõrge liikuvusega võrgustikele. See mehhanism toetub OWL-S'le, mis on ontoloogia veebiteenuseid pakkuvate ressursside avastamiseks, väljakutseks, koostamiseks ja jälgimiseks. Töö kirjeldab ka Srirama välja pakutud algupärast teenuste avastamise mehhanismi, mis toetub peer-to-peer võrkudele ja Apache Lucene võtmesõna otsingumootorile. Uurimuse käigus uuendatakse teenuseotsing kasutama Apache Solr'i, Apache Lucene'i viimast versiooni. Teenuste avastust testiti põhjalikult ja tulemused on töös kokkuvõtvalt välja toodud.

Mobiilsete tehnoloogiate vallas uuritakse ka võimalust kasutada pilvetehnolologiat laiendamaks mobiilseadmete salvestusmahtu ja töökoormust edastades pilve andme- ja arvutusmahukad ülesanded. See soodustab keerulisemate ja võimalusrohkemate mobiilirakenduste arendust. Pilve delegeeritavate toimingute aeganõudva iseloomu tõttu aga on vajalik asünkroonne mehhanism teavitamaks kasutajat, millal töömahukad tegevused on lõpetatud. Mobiilsete pilveteenuste pakkujad ja vahevara lahendused võivad kasu saada Mobile Host'ist ja selle asünkroonsete teavituste võimekusest. Uurimus esitleb nelja teavitusmehhanismi: AC2DM, APNS, IBM MQTT ja Mobile Host'i põhine teavitus. Töö võtab kokku kvantitatiivse analüüsi tulemused ja toob välja nelja teavitamise lähenemise tugevused ja nõrkused. Lisaks kirjeldatakse CroudSTag rakenduse realisatsiooni - CroudSTag on mobiilirakendus, mille eesmärgiks on sotsiaalsete gruppide moodustamine kasutades näotuvastustehnoloogiat. CroudSTag-i realisatsioon kasutab mobiilseid pilveteenuseid ja Mobile Host'i, et pakkuda oma funktsionaalsust kasutajale.

# Bibliography

[1] S. N. Srirama, M. Jarke, W. Prinz, Mobile web service provisioning, in: AICT-ICIW '06: Advanced Int. Conf. on Telecommunications and Int. Conf. on Internet and Web Applications and Services, IEEE Computer Society, 2006, p. 120. 1, 2, 10

[2] S. Srirama, M. Jarke, W. Prinz, Mobile host: A feasibility analysis of mobile web service provisioning, in: 4th International Workshop on Ubiquitous Mobile Information and Collaboration Systems, UMICS, Citeseer, 2006, pp. 942–953. 2, 10, 12

[3] J. Schiller, A. Voisard, Location-based services, Morgan Kaufmann, 2004. 2

[4] M. Armbrust et al., Above the clouds, a berkeley view of cloud computing, Tech. rep., University of California (Feb 2009). 2

[5] R. Fielding, Architectural styles and the design of network-based software architectures, Ph.D. thesis (2000). 2

[6] H. Flores, S. Srirama, C. Paniagua, A generic middleware framework for handling process intensive hybrid cloud services from mobiles, in: Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, ACM, 2011, pp. 87–94. 4

[7] H. F. S. N. Srirama, C. Paniagua, Social group formation with mobile cloud services, Service Oriented Computing and Applications Journaldoi:10.1007/s11761-012-0111-5. 4, 70

[8] S. N. Srirama, M. Jarke, Mobile hosts in enterprise service integration, International Journal of Web Engineering and Technology (IJWET) 5 (2) (2009) 187–213. 4, 21, 77

[9] S. N. Srirama, M. Jarke, W. Prinz, Mobile web services mediation framework, in: Middleware for Service Oriented Computing (MW4SOC) Workshop @ 8th Int. Middleware Conf. 2007, ACM Press, 2007. 6, 15

[10] S. Srirama, M. Jarke, W. Prinz, K. Pendyala, Security aware mobile web service provisioning. 8

[11] S. Srirama, E. Vainikko, V. Sor, M. Jarke, Scalable mobile web services mediation framework, in: Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on, IEEE, 2010, pp. 315–320. 8, 17

[12] P. Farley, M. Capp, Mobile web services, BT Technology Journal 23 (3) (2005) 202–213. 10

[13] N. Ravi, P. Stern, N. Desai, L. Iftode, Accessing ubiquitous services using smart phones, in: Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on, IEEE, 2005, pp. 383–393. 10

[14] D. Schall, M. Aiello, S. Dustdar, Web services on embedded devices, International Journal of Web Information Systems 2 (1) (2006) 45–50. 10

[15] M. Asif, S. Majumdar, R. Dragnea, Hosting web services on resource constrained devices, in: Web Services, 2007. ICWS 2007. IEEE International Conference on, IEEE, 2007, pp. 583–590. 10

[16] Y. Kim, K. Lee, A lightweight framework for mobile web services, Computer Science-Research and Development 24 (4) (2009) 199–209. 11

[17] S. Berger, S. McFaddin, C. Narayanaswami, M. Raghunath, Web services on mobile devices-implementation and experience, in: Mobile Computing Systems and Applications, 2003. Proceedings. Fifth IEEE Workshop on, IEEE, 2003, pp. 100–109. 13

[18] R. Steele, K. Khankan, T. Dillon, Mobile web services discovery and invocation through auto-generation of abstract multimodal interface, in: Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on, Vol. 2, IEEE, 2005, pp. 35–41. 13

[19] L. Capra, S. Zachariadis, C. Mascolo, Q-cad: Qos and context aware discovery framework for mobile systems, in: Pervasive Services, 2005. ICPS'05. Proceedings. International Conference on, IEEE, 2005, pp. 453–456. 13

[20] G. Lecture Notes in Informatics (Ed.), Publishing and Discovery of Mobile Web Services in Peer to Peer Networks, Proceedings of First International Workshop on Mobile Services and Personalized Environments (MSPE'06), November 16-17, 2006. 14

[21] A. S. Mix, http://servicemix.apache.org/. 15

[22] O. I. Java Business Integration, http://java.sun.com/developer/earlyAccess/jbi/. 15

[23] P. Papakos, L. Capra, D. Rosenblum, Volare: context-aware adaptive cloud service discovery for mobile systems, in: Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware, ACM, 2010, pp. 32–38. 16

[24] M. Armbrust et al., Above the clouds, a berkeley view of cloud computing, Tech. rep., University of California (Feb 2009). 17

[25] Amazon, Inc, Amazon - Amazon Web Services, http://aws.amazon.com/. 17, 69

[26] Google Inc., Google Code - google application engine, http://code.google.com/appengine/ (2011). 17

[27] D. Nurmi, R. Wolski, C. G. G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The Eucalyptus Open-source Cloud-computing System, 2011. 18

[28] S. McIlraith, T. Son, H. Zeng, Semantic web services, Intelligent Systems, IEEE 16 (2) (2001) 46–53. 18

[29] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, et al., Daml-s: Web service description for the semantic web, The Semantic WebISWC 2002 (2002) 348–363. 19

[30] K. Sivashanmugam, K. Verma, A. Sheth, J. Miller, Adding semantics to web services standards, in: Proceedings of the International Conference on Web Services, 2003, pp. 395–401. 19

[31] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, J. Miller, Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services, Information Technology and Management 6 (1) (2005) 17–39. 19

[32] A. Patil, S. Oundhakar, A. Sheth, K. Verma, Meteor-s web service annotation framework, in: Proceedings of the 13th international conference on World Wide Web, ACM, 2004, pp. 553–562. 19

[33] foursquare Inc., https://foursquare.com/. 26

[34] S. Srirama, C. Paniagua, H. Flores, Croudstag: Social group formation with facial recognition and mobile cloud services, Procedia Computer Science 5 (2011) 633–640. 27, 68, 69

[35] S. E. C. LTD., Samsung galaxy sii, http://www.samsung.com/global/ microsite/galaxys2/html/feature.html. 27

[36] A. Mian, R. Baldoni, R. Beraldi, A survey of service discovery protocols in multihop mobile ad hoc networks, Pervasive Computing, IEEE 8 (1) (2009) 66–74. 31

[37] S. Srirama, M. Jarke, H. Zhu, W. Prinz, Scalable mobile web service discovery in peer to peer networks, in: Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on, IEEE, 2008, pp. 668–674. 32, 38

[38] A. Gulbrandsen, A dns rr for specifying the location of services (dns srv). 34

[39] A. Solr, Apache solr, http://lucene.apache.org/solr/. 39

[40] A. Lucene, Apache lucene, http://lucene.apache.org/core/. 39

[41] W.-S. Interoperability, http://www.ws-i.org/. 41

[42] S. Srirama, M. Jarke, W. Prinz., Mobile web service discovery in peer to peer networks. 41

[43] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al., Owl-s: Semantic markup for web services, W3C Member submission 22 (2004) 2007–04. 42

[44] B. McBride, Jena: A semantic web toolkit, Internet Computing, IEEE 6 (6) (2002) 55–59. 44

[45] J. Pozdena, An apple push notification service gem, https://github.com/jpoz/APNS. 60

[46] . S. Inc., Ruby and rails, http://rubyonrails.org/. 60

[47] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113. 68, 69

[48] D. M. Boyd, N. B. Ellison, Social network sites: Definition, history, and scholarship, Journal of Computer-Mediated Communication 13 (2008) 210–230. doi:10.1111/j.1083-6101.2007.00393.x. 69

[49] S. N. Srirama, O. Batrashev, E. Vainikko, SciCloud: Scientific Computing on the Cloud, in: The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGrid 2010), 2010, p. 579. 70

[50] GoGrid, GoGrid - Complex Infrastructure Made Easy, http://www.gogrid.com/. 70

[51] S. Srirama, Mwsmf: a mediation framework for mobile hosts and enterprise on cloud, International Journal of Pervasive Computing and Communications 7 (4) (2011) 316–338. 78