Context-Aware Discovery of Relevant Web Services in Ad-hoc Mobile Cloud

Thesis · November 2016

DOI: 10.13140/RG.2.2.10714.44483

1 author:

Dominic Afuro
Queensland University of Technology
11 PUBLICATIONS **35** CITATIONS

SEE PROFILE

Context-Aware Discovery of Relevant Web Services in Ad-hoc Mobile Cloud

A dissertation submitted

By

Egbe Dominic Afuro

(201454855)

in fulfillment of the requirements for the degree of

Master of Science in Computer Science

To

The Department of Computer Science, Faculty of Science and Agriculture,

University of Zululand

Supervisor: Professor Matthew. O. Adigun

Co-supervisor: Mr. Bethel M. Mutanga

2017

Declaration

I declare that this dissertation is my original work, conducted under the supervision of Prof. M. O. Adigun and Mr B. M. Mutanga. It is submitted for the degree of Master of Science in the Faculty of Science and Agriculture at the University of Zululand, Kwadlangezwa. I further declare that this dissertation has not been presented or submitted to any institution for the award of any degree except this submission. All sources used in the dissertation have been duly acknowledged.

Signed:…… Date:……

Dedication

To the Trinity: God the Father, Son, and Holy Spirit.

Acknowledgment

I would like to express my wholehearted appreciation to Prof. M. O. Adigun, the Head of the Centre of Excellence in Mobile e-Services, Computer Science Department, University of Zululand. Without your role, particularly with the financial support from Telkom, Thrip and Huawei that helped in funding my research and other needs, this project would not have become a reality.

I would also like to thank my co-supervisor, Mr B. M. Mutanga, for being there for me. Your immense contributions are highly appreciated. Your wonderful and genuine support will always be remembered.

My deep appreciation also goes to my beloved Wife, Mrs. Queenneth Dominic. Despite your being far away from home, you left the impression in my heart that you were always with me throughout the difficult journey.

My appreciation also goes to Mr. Noel Afuro for his financial support - you are a blessed brother.

My heartfelt gratitude is also extended to the humble and generous family of Barr, Sunny and Mrs. Jane Ochaje. You willingly accepted the burden of caring for me even though you had every reason to decline – you are my Shunammite!

Finally, CAP cluster members – you are wonderful people! You guys turned a furious journey into fun. Thank you all.

Table of Contents

Table of Figures

List of Tables

Abstract

The growth and popularity of Mobile Cloud Computing coupled with advancements in the capabilities of mobile devices has led to the emergence of Ad-hoc Mobile Cloud. This emerging Cloud paradigm presents a huge opportunity to build a platform for inexpensive and collaborative resource provisioning similar to the traditional Cloud setting. Apart from complementing the Cloud, the Ad-hoc Mobile Cloud can significantly minimise monetary cost for small computing communities like Small, Medium and Micro-sized Enterprises (SMMEs) by avoiding Internet connection charges. An infrastructure-less platform is another benefit of the Ad-hoc Mobile Cloud with regards to cost reduction. However, effectively discovering relevant services in the Ad-hoc Mobile Cloud is still an open research issue. This challenge centres around the problem of inherent resource constraint and dynamic context of the Ad-hoc Mobile Cloud environment, which significantly impacts on service discovery. To address these issues, this research formulates a context-aware service discovery mechanism (CaSDiM) for the Ad-hoc Mobile Cloud. Resource-friendly context-aware service matching and relevancy ranking algorithms are proposed for the Ad-hoc Mobile Cloud. In order to validate the solution approach, the developed mechanism was prototyped and evaluated based on parameters that justify the impact of context in service discovery. The prototype was developed in Android SDK version 21 integrated with Eclipse Juno 4.2, and coded in Java language for ICS 4.1 and above. The results from the test-bed experiments showed that CaSDiM was relatively promising, as demonstrated by its high precision rate and the capability to retrieve relevant services in a resource-efficient manner.

CHAPTER ONE

INTRODUCTION AND BACKGROUND

1.1 Preamble

The widespread use of mobile devices in recent decades coupled with the increasing availability of reliable and fast wireless Internet connectivity for mobile devices has motivated huge interest in integrating mobile devices with the Cloud Computing paradigm. In this paradigm, most of the processing occurs on the server-side (Papakos et al, 2010). However, even though mobile devices continue to evolve and improve with respect to hardware and functionality they are still subject to the common constraints of limited processor speed, memory size, disk capacity, small bandwidth and battery dependence. In fact, the issue of resource limitation in mobile devices is described in Satyanarayanan & Bahl, (2009) as a state of resource poverty. This peculiarity of mobile devices recommend Cloud Computing architecture as an ideal infrastructure to support the mobile application scenario (Papakos et al, 2010).

The computing platform that integrates mobile devices into the traditional Cloud architecture has come to be known as *Mobile Cloud Computing* (MCC). Marinelli (2009) defined MCC as an extension of Cloud Computing in which the foundational hardware consists at least partially of mobile devices. MCC has overcome obstacles associated with mobile computing such as performance (battery life, storage, and processing speed), environment (scalability, and availability) and security (Dinh et al, 2013). However, being a service provisioning environment, the requirements for relevant service discovery and invocation in addition to cost efficiency still prove to be significant challenges to mobile clients. These challenges are the main reasons behind the emergent domain of the Ad-hoc Mobile Cloud.

1.2 Background

Although MCC seeks to address the challenge of resource poverty in mobile devices, several other issues associated with the inherent nature of mobile clients are not eliminated. For instance, as shown in Figure 1.1, a mobile device will require an intermediary infrastructure like an access point or satellite as well as a stable and strong Internet connection to be able to sufficiently discover and access any Cloud resource. Such a requirement has various implications on the client-side: the wireless connection from the mobile device to the Cloud leads to high energy consumption, weakness or absence of Internet connection cause inaccessibility of the Cloud, and the hop distance between the mobile device and Cloudlet (viewed as a "data center in a box" whose goal is to bring the cloud closer) or Cloud being accessed can introduce high latency (Dinh et al., 2013; Marinelli, 2009; Palmer, Kemp, & Kielmann, 2009; Satyanarayanant et al., 2014).

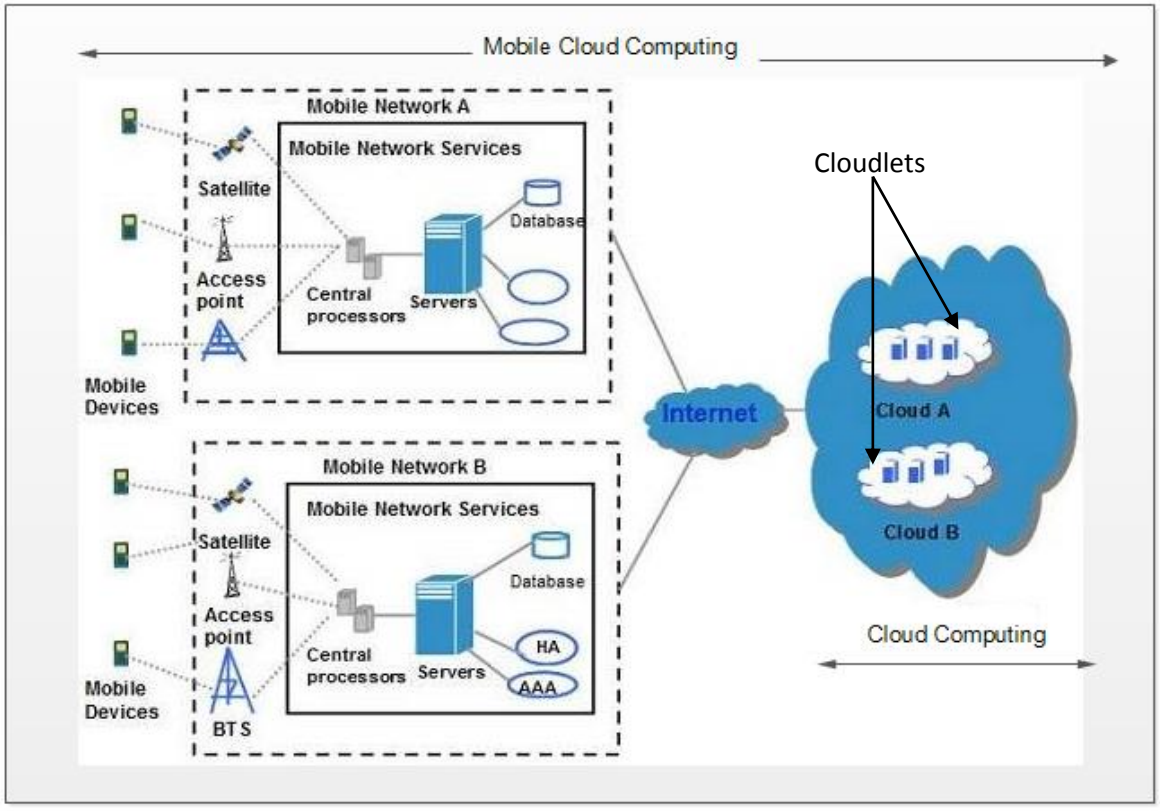


Figure 1.1: Mobile Cloud Computing Architecture
(Tutorialspoint, 2014)

Furthermore, from a service discovery and consumption perspective, service relevance is influenced by the dynamic context of mobile devices while discovery efficiency hinges on the capabilities of the device consuming the service. These aforementioned challenges still remain open issues in the Mobile Cloud computing domain.

To tackle the challenges highlighted above, current research is exploring a complementary approach to MCC. This emergent approach focuses on enabling direct service provisioning between mobile devices. These devices exploit self-organising networks to support direct communication between each other to form an Ad-hoc Mobile Cloud (AMC). Therefore AMC is a network of mobile devices that act as Cloud providers by enabling each device to expose its resources to others within the network (Dejan et al, 2011). With this approach, the AMC can function without need for infrastructure support both at the network level and service level (Mascitti et al, 2014). In the AMC scenario, a mobile node can host Web Services that are discovered and accessed by other mobile nodes within the Ad-hoc network. First, this kind of opportunistic computing platform is essential in scenarios with weak or no Internet connection to the central Cloud, such as in the rural areas. Second, although in the general large-scale Web Service hosting on mobile devices is infeasible, the AMC setting is envisioned to offer services to small computing communities (Renzel et al, 2010). For example, Small Medium and Micro Enterprises (SMMEs) that have common needs. This can raise the prospect of offering a relatively small set of Web Services (when compared to services hosted on Cloud servers) that are tailored to the needs of such computing communities from mobile devices.

Fundamentally, delivering software and or processing capability as services over the network remains the central goal of Cloud Computing, MCC and AMC computing (Renzel et al, 2010; Dejan et al, 2011). Fulfilling this goal in any Cloud platform therefore, requires a service discovery mechanism that enables consumers to find suitable services (Le et al,

2007). The suitability of a given Web Service to its consumer defines the issue of service relevance, which has put context-awareness at the centre stage of research in service discovery in pervasive environments. Context simply refers to any information that defines the state of an entity at any point in time. A Web Service is considered relevant if it satisfies the requirements of a client (Rathinam et al, 2014).

In AMC, discovering relevant services is challenging due to resource constraints and the dynamic context of mobile devices – their battery level and available memory, amongst other context parameters can change. Such changes in a device's context can have significant impact on relevant service discovery (Papakos et al, 2010). This is because in the context of this work, service relevance implies meeting both user requirements and matching the clients device context (current resource capability).

1.3 Statement of the Problem

Service discovery generally requires the mechanism to return relevant services according to users' requirements. Nevertheless, in the Ad-hoc Mobile Cloud, resources are limited and dynamic. This implies that fulfilling a user's requirements alone is insufficient to guarantee discovering relevant services because the device's context plays a critical role in determining the relevance of discovered services. This limitation and dynamism in resources require that the discovery process be resource-efficient and proactive to context change. Therefore, there is a need for discovery mechanisms in AMC to utilise device context to improve service discovery and resource efficiency.

The challenge is how to discover services in a resource efficient way in the Ad-hoc Mobile Cloud while utilising device context (battery and memory). This requires that Ad-hoc mobile nodes be monitored for context change and services discovered based on their current context. Consequently, this work proposes a mechanism for discovering Web Services in the

Ad-hoc Mobile Cloud based on device context. Towards this goal, this research will attempt to answer the following question:

How can context-aware service discovery be achieved in the Ad-hoc Mobile Cloud in a manner that minimises device resource consumption?

i. How is context-aware service discovery achieved in MCC?
ii. Why are the mechanisms for achieving context-aware service discovery in MCC not adequate in the Ad-hoc Mobile Cloud?
iii. How can device context be utilised to enhance service discovery in AMC?
iv. How can Ad-hoc mobile nodes be monitored as a strategy to minimise resource consumption during service discovery?

1.4 Rationale of the Study

Rural areas are often characterised by lack of ICT infrastructure, therefore providing Cloud e-services might be too expensive to accomplish in the short term. Also, weakness or absence of Internet connection in some cases makes it necessary to explore a complementary e-service provisioning platform like the Ad-hoc Mobile Cloud. In addition to complementing the traditional Cloud with respect to resource accessibility, the Ad-hoc Mobile Cloud would also reduce the cost associated with Internet connection charges and capital infrastructure. Therefore, the emergence of AMC computing architecture designed to harness a collection of mobile devices to support collaborative resource sharing offers a rich opportunity to service SMMEs in the rural communities. But the challenge of the limited resources associated with participating devices in the AMC coupled with their dynamic context requires research to address issues related to discovering relevant Web Services.

1.5 Research Goal and Objectives

The goal of this study was to achieve relevant service discovery in the Ad-hoc Mobile Cloud based on device context. In achieving this goal and offering answers to the research questions, the following objectives were set:

i. To investigate context-aware service discovery mechanisms in Mobile Cloud Computing.
ii. To investigate how context-aware service discovery can be achieved in the Ad-hoc Mobile Cloud.
iii. To formulate a device-context model, resource-friendly service matching, and device context-based service relevancy ranking algorithms.
iv. To formulate an architectural model for context-aware service discovery in the Ad-hoc Mobile Cloud.
v. To develop a proof-of-concept prototype to validate the properties of the proposed model.
vi. To evaluate the model in a manner to demonstrate the possible impact of device context in discovering relevant services in the Ad-hoc Mobile Cloud.

1.6 Research Methodology

This work adopts a two dimensional approach of research methodology. This consists of literature survey and the design science research methodology:

1.6.1 Literature Survey

This part of the research involved an in-depth study of scholarly and relevant literature in context-aware Web Service discovery. In order to analyse existing approaches, a framework that captures the goal of this study was formulated. The framework was then used as the basis

to evaluate similar literature with the aim to hone our focus and develop the relevant knowledge needed to achieve our overall goal.

1.6.2 Design Science

The design science research methodology (DSRM) is important for conducting research in disciplines oriented to creating artifacts that serve as solutions to defined problems. This study adopted the DSRM as illustrated in Figure 2.

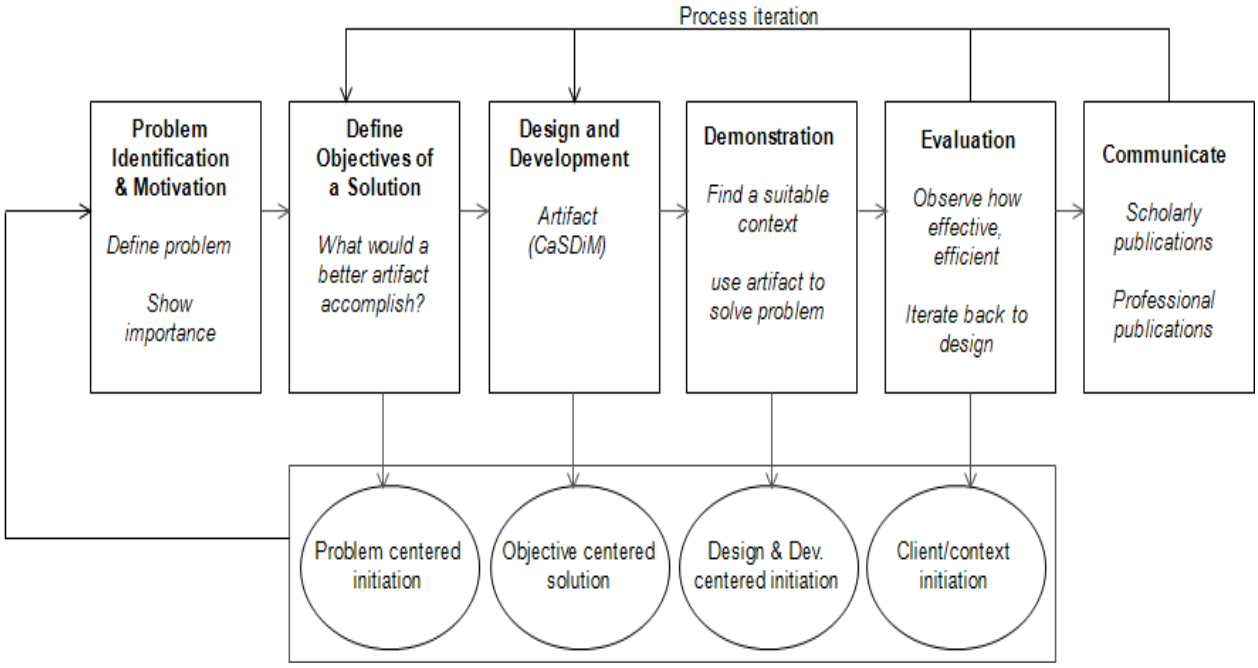


Figure 1.2: The Design Science Research Methodology Process Model
(Peffers et al, 2014)

i. **Problem Definition and Motivation**: this step took the input of the problem statement for this study and used the information gathered from the literature survey to motivate the relevance of the study. The survey was also a tool to acquire useful knowledge for developing the proposed solution approach.

ii. **Definition of Objectives**: the outlined objectives for this study were utilised at the coarse level because the core of this work is not system development.

iii. **Design and Development**: in this step design criteria were formulated, the required model components were defined, and the context-aware service discovery mechanism (CaSDiM) was offered as a solution approach.

iv. **Demonstration**: for the purpose of experimental demonstration, a proof-of-concept prototype was developed. The development was based on Android SDK version 21 integrated with Eclipse Juno 4.2 and coded in the Java language. Other technologies used included:

   a. Wi-Fi Direct (utilising native Java Sockets: java.net.ServerSocket and java.net.Socket) to create a P2P ah-hoc network and
   b. SQLite relational database, which is embedded in Android platforms.

v. **Evaluation**: the model developed in this work was then evaluated using the following performance parameters:

   a. Precision and Recall rates
   b. Quality of service discovery (QoSD)
   c. Context overhead
   d. Resource utilisation

1.7 Dissertation Outline

The rest of the dissertation is organised as follows: Chapter 2 explores fundamental Web Service concepts and their implications for the Ad-hoc Mobile Cloud. Chapter 3 analyses relevant literature of other scholars using a framework formulated in the same chapter. An architectural design and concept formulation is presented in Chapter 4 based on the conclusions reached in chapter 3, which clarifies the existing gap in the literature and recommends a possible solution. Chapter 5 provides the design and implementation details of the proposed model. Using an application scenario and UML diagrams, an attempt is made to

mimic the working of the model. The chapter also discusses the assumptions of this study and presents the experimental setup and results. Chapter 6 offers a conclusion to the study: a detailed discussion that links the achievements of this study to their corresponding objective or research question was rendered. The chapter then discusses the limitations of this study and concludes by suggesting possible future directions.

CHAPTER TWO

WEB SERVICES: THE AMC PERSPECTIVE

2.1 Introduction

Any piece of software system designed to support interoperable machine-to-machine interaction over a network is known as a Web Service. The hosting and discovery of such services in AMC has unique requirements compared to other traditional service oriented environments. These distinctive requirements are due to the inherent nature of AMC with regards to resource limitation and changing context. As indicated in chapter one, these requirements are open research challenges in AMC service discovery. These requirements of Web Service discovery in AMC are also impacted by approaches adopted in the Web Service development process (description and implementation). Such approaches, therefore, may be considered as factors that influence the suitability of service discovery mechanisms intended for AMC.

As stated in the previous chapter, the aim of this study is to achieve service discovery in AMC by utilising device context as a way to enhance the discovery of relevant services and minimise resource consumption. Hence, it becomes imperative to investigate the factors that underline the suitability of Web Services' approaches for AMC.

This chapter discusses the implications of conventional Web Services' approaches on AMC service discovery. Context awareness, its role in service discovery, and context modelling are discussed in sections 2.2 and 2.3 respectively. The Web Service development process - consisting of Web Service description approaches and implementation frameworks is discussed in sections 2.4. A discussion on the features of service discovery is then presented

in section 2.5; describing the impact of techniques predominantly adopted at the different levels of service discovery mechanisms, followed by the chapter's summary in section 2.6.

2.2 Context-awareness in Web Service Discovery

In order to improve service discovery performance with respect to accuracy and efficiency, the issue of context has gained significant research attention in recent years. Such investigations have focused on using context-awareness as a technique for developing pervasive computing's autonomous applications that are flexible and adaptable (Bettini et al, 2010). In Web Services, context is generally associated with non-functional properties, which can be interpreted in a variety of ways, depending on the area of application. For instance, the definitions of context such as the one offered by Dey, (2000) do not apply to this work due to their broad scope of context.

Therefore, this dissertation adopts the definition by Doulkeridis & Vazirgiannis, (2008) that, "context is the implicit information related both to the requesting user and service provider that can affect the usefulness of the returned results (services)". With regards to this work, it is argued that the usefulness of discovered services is based on whether the services match the current context of the client's device and fulfill a user's requirement.

Context plays an important role in relevant service discovery (Al-Masri & Mahmoud, 2006). This is because by using context information, service discovery mechanisms are able to support automated and dynamic discovery, which makes it possible to discover the most suitable service based on a client's current context.

Context can be divided into three categories according to Schilit et al, (1994): 1) user context, which consists of user's profile, people nearby, location, and current social situation; 2) computing context that is, network connectivity, communication costs, bandwidth, and

nearby resources like printers and workstations; 3) physical context such as lighting, noise levels, traffic conditions, and temperature. However, existing knowledge in the area of pervasive computing points to the existence of another context category, namely device context. This context type refers to the resources state of a providing or consuming device such as battery life and memory. Based on the context definition adopted in this work, device context is being explored to enhance service discovery in AMC. Such enhancements can be achieved by designing mechanisms that automatically monitor a device's context and in which the monitored outcomes are used in the service request as constraint parameters to filter Web Services.

2.3 Context Representation (Modeling)

To effectively utilise context information for the discovery of services, appropriate modeling or representation of the context information is desirable. This is due to the fact that to be useful, context must be in a form that can be processed by a service discovery mechanism. There are several modeling approaches in the literature (Bettini et al, 2010). These approaches adopt varying techniques to represent context for different domains. This variation in modeling approaches is based on the fact that context modeling is domain-dependent and each approach has its weaknesses and strengths. Generally, modeling approaches are chosen based on factors such as scalability, easy updating mechanism, low overhead of context manipulation, efficient search, and query access (Chihani et al, 2013). Based on the level of semantic constraints, Chihani et al, (2013) classified existing context modeling approaches into two categories: models with strong semantic constraints or models with weak semantic constraints. The first category is based on Resource Description Framework (RDF), a language for describing tagged oriented graphs. With RDF, context is modeled as a triple: (subject, predicate, object). Where subject describes a resource, the

predicate describes a property type that can be applied to the resource, and the object represents data or another resource. This formalism can aid the construction of context graphs with the predicate being a relation, while subject or object can be entities, for example (SheratonHotel, IsLocatedIn, Pretoria). More advanced works based on RDF employ Web Ontology Language (OWL) (Ziafati et al, 2011). The second category of context modeling approach is based on Model Driven Architecture (MDA), a model-centric and flexible approach for system development. Some examples of MDA-based context models include ContextUML, UML, and ContextML. ContextUML is used to model context-aware Web Services and provisioning. The UML context model is particularly used to facilitate the extension of Business Process Management (BPM) while ContextML is an XML-based model for representing and exchanging context (Sheng & Benatallah, 2005; Wieland et al, 2011).

The context models mentioned in this section either represent context data in a hierarchical or entity-centric manner. These styles of context representation account for the disadvantages associated with the two categories of context models (Chihani et al, 2013). For instance, while the first category is less flexible and may create high resource overhead for mobile devices, approaches under the second category are said to lack the required expressiveness to model more non-functional properties.

In contrast to hierarchical or entity-centric- based context representation models, Chihani et al, (2013) presented a graph-based approach. The graph-based context model utilised the intuitive nature of conventional graphs as a way of representing connected data. This approach to context modeling offers more flexibility and supports scalability.

2.4 The Web Service Development Process

Web Services facilitate the sharing of data between different software applications so that the heterogeneity in platform or programming language is not an impediment. Such programming techniques basically implement the Service Oriented Architecture (SOA) paradigm (Erik et al, 2001). The Web Service Description Language (WSDL) is the de facto standard for specifying Web Services. To describe services entails explicating their metadata in order to facilitate their discovery from a service directory or repository. Service providers use a WSDL to describe their service offerings so that a potential consumer can make an up-front decision on whether and how to consume the functionality offered by the service (Elgazzar et al, 2014b).

SOA is an "architectural style whose goal is to achieve loose coupling among interacting software agents" (OpenGroup, 2013). Meaning, with SOA, software can be delivered over the network in the form of services. The SOA paradigm defines a Web Service by a web interface that supports interoperable operations between diverse software applications using a standard messaging protocol. There are different ways to describe and implement Web Services. However, the choice of a suitable Web Service description and implementation technique for AMC must take into account the constraints of mobile devices, as stated in chapter one. In subsections 2.4.1 Web Service description and its different components are discussed, while subsections 2.4.2 and 2.4.3 present the approaches to Web Service description and implementation frameworks respectively.

2.4.1 Web Service Description

Web Services are described so that they can be discovered. This implies that the success of the service discovery process is to a larger extent dependent on how well a service is described.

This subsection discusses the four components of a service description namely, the Information model, the Functional capabilities, the Non-functional parameters, and the Technical model (Omrana et al, 2013):

A. Information Model

In the information model the provider defines the data model which is made of input/output messages as well as other data relevant to the service operation. Basically, Web Services are designed to support interoperability in order to accommodate the heterogeneity of middleware platforms, programming languages, consumers and other technologies in which these applications are realised (IBM, 2014). This design goal is achieved using the information model, which facilitated intercommunication.

B. Functional Capabilities

The operations that are offered by services and how potential consumers can interact with the provided services are determined by functional capabilities specification. The purpose of functional descriptions is to provide a black box description of what a Web Service does, disregarding other detailed technical information. This is often used as the primary way of differentiating Web Services in traditional service discovery mechanisms.

C. Non-functional Parameters

Non-functional parameters specify the running and environmental parameters such as reliability, availability and quality of service (QoS). With the vital role played by context in mobile environments, non-functional parameters specification includes user context (device capabilities, user preferences, user profile, location, time, temporal constraints), Web Service

context (name, business category, provider identity, location of Web Service, cost, and payments method.) and quality of Web Service (QoWS) (Saadon & Mohamad, 2011).

D. Technical Specifications

The technical specifications are primarily concerned with details about implementation of such message structures, transport protocols, access information, and service location. In the traditional Web Service discovery framework, technical specifications are contained in the "green pages" of the Universal Description, Discovery, and Integration (UDDI) service registry.

Service description contributes to the effective discovery of relevant Web Services because for services to be discovered they have to be well described and published (Sibiya, 2010). Following the lack of expressiveness in the UDDI approach to service description as revealed by (El-Bitar et al, 2013), several enhanced service discovery mechanisms have been proposed. Some of these enhanced Web Service discovery mechanisms as discussed in (Bai & Liu, 2008; Paliwal et al, 2012) support different service description formalisms. A discussion on Web Service description approaches is important because they are one of the factors that influence the suitability of a Web Service discovery mechanism for the Ad-hoc Mobile Cloud environment.

2.4.2 Web Service Description Approaches

Web Service description is an essential component of the Web Services development cycle. The reason for this importance is that when capabilities of a Web Service are described it becomes possible to classify, discover and use the service. Therefore, achieving effective Web Service description is critical to the success of Web Service discovery.

Generally, Web Service description approaches are classified into two categories namely, non-semantic and semantic description.

A. Non-semantic Description

Non-semantic Web Services are described using the Web Service Description Language (WSDL). The WSDL is an XML format for describing services that are rendered over networks as a set of endpoints operating on messages (Erik et al, 2001). With WSDL, service providers are able to describe their services and explain to customers how the functionalities offered by such services can be consumed. Non-semantic description is concerned with describing a service at a syntactic level by using XML schemas and WSDL interface to specify the information model and functional capabilities respectively. Non-functional service parameters are represented using standard Web Service specifications such as WS-policy, WS-agreement, and WS-reliability, while technical details of Web Services are defined as service bindings and endpoint information (Fensel et al, 2010).

The WSDL is the default standard for describing and publishing Web Services in the Universal Description, Discovery, and Integration (UDDI) registry used in the traditional Web Service architecture (Omrana et al, 2013). However, WSDL standard suffers some limitations in that it only operates at the syntactic level and lacks the semantic expressivity needed to unambiguously represent the requirements and capabilities of a Web Service (Dong et al, 2013). The following are the weaknesses of WSDL as highlighted Dong et al (2013): (a) lack of semantic supports that help to indicate the meaning and semantic constraints of data involved in Web Services, which may lead to high probability of generating ambiguities during the service discovery process; (b) the capabilities of a Web Service are not covered in the description, which can make the matchmaking process incapable of recognising the similarity between the capabilities of a provided Web Service and the functionalities of a

Web Service being requested for. The effect of these shortcomings of WSDL on service discovery is that as more Web Services are published, there would be multiple services with similar functionality, making it difficult to discover relevant services that match consumers' requirements. On the other hand, it is possible for two services to have the same syntactic definition but perform significantly different functions, while another two syntactically dissimilar services can perform the same function (Akkiraju et al, 2005). Extensions to WSDL such as SAWSDL, WS-Policy, and WSDL-M have been proposed (Vedamuthu et al, 2007; Al-Masri & Mahmoud, 2010).

B. Semantic Description

In contrast to non-semantic Web Service description, the semantic approach relies on ontologies to describe Web Services (Omrana et al, 2013). An ontology is a formal explicit specification of shared conceptualisation (Gruber, 1993). In the Web Services domain, essential components of ontology are extracted to form individual ontologies and by providing concepts, and relationships between concepts, these individual ontologies help define an agreed common (in terms of quality of service) between service providers and potential clients (Elgazzar et al, 2014a). The essence of semantic description is to enrich service description through enhanced expressivity. Such enhancements can help to support automation in service discovery and improve the efficiency of discovering relevant Web Services. Also, by semantically describing a service, it becomes possible for the service to be linked to abstract and cross-platform concepts. This sematic-enabled linking facilitates the possibility of resolving and matching different semantic representations and the mapping between services from different providers (Martino et al, 2014).

Fundamentally, the semantic description approach differs from the non-semantic approach in its rich expressiveness which allows a broader perspective to service description that

adequately covers both functional and non-functional properties like availability, scalability, and reliability. In the context of service discovery, a major advantage of semantic description is that it allows for unambiguous definition of description terms, which addresses the lack of understanding of the semantic meaning of messages and data.

There are several Semantic Description Languages, for example, Web Ontology Language for Services (WOL-S), Semantic Web Services Ontology (SWSO), Web Service Modeling Ontology (WSMO), and WSMO-Lite. (Martin et al, 2004). Although the semantic approach augments Web Service description capabilities, it is resource-intensive, especially when is it to be used in a mobile environment, which is made up of resource-constrained devices (Elgazzar, 2013).

2.4.3 Web Service Implementation Frameworks

Web Services achieve their goal by implementing the SOA paradigm of "software-as-a-service" in a technology-neutral fashion independent of hardware platform, programming languages and even operating systems (Hamad et al, 2010). Such technology independence is realised by providing well-defined interfaces for distributed functionalities (services). The advantage of this technology neutrality is that the complexity with regards to the heterogeneity that characterises the Web Services provisioning space is bridged; distributed functionalities otherwise services, which may be running on dissimilar hardware and software platforms can still communicate effectively through Web Service interfaces.

To achieve the interoperability described above, Web Services use the Extensible Markup Language (XML) technology to define and implement message exchange or intercommunication protocol, which is referred to as an implementation framework. Although there are several frameworks for developing Web Services, nowadays, the main

approaches are SOAP and REST (Castillo et al, 2011). The SOAP and REST frameworks are discussed below.

A. SOAP-based Implementation

Simple Object Access Protocol (SOAP) is a messaging framework for Web Services realisation of SOA. At a fundamental level, the core idea in the SOAP-based framework revolves around the exchange of XML encoded messages over Hyper-Text Transmission Protocol (HTTP). This simply means that the SOAP protocol works by exchanging messages over HTTP using GET/POST operations. The SOAP-based framework follows the Remote Procedure Call (RPC) style of Web Service interaction in which service providers and consumers are required to establish a common understanding with regards to the service syntax and operations in order to be able to communicate with each other. This RPC-based framework is widely adopted because, among other reasons, it is supported by massive development tools. In addition to being extensively used in enterprise software like Google, SOAP-based mobile Web Services have also been demonstrated by scholarly works such as (Wagh & Thool, 2012; Castillo et al, 2011). However, the SOAP framework was tailored for fixed networks which make it heavy in nature. Hence, it is not suitable for resource constraint devices because it leads to heavy discharge of battery power (Wagh & Thool, 2012). As stated in (SnapLogic, 2011), "SOAP is all about servers talking to servers, with rigid standards, extensive design, serious programming, and heavyweight infrastructure all essential parts of the equation". Also, since SOAP was originally designed for fixed networks, performance factors such as heavy payload, tight coupling and mobility, which now constitute a huge resource burden to mobile devices, were not considered. Therefore, SOAP-based Web Services are not the best candidates for resource-constrained devices.

B. REST-based Implementation

The Representational State Transfer (REST) implementation differs from its SOAP counterpart in very fundamental aspects. For example, while SOAP adheres to the RPC model, REST employs the concept of resources and focuses on using the intrinsic power of HTTP to retrieve representations of these resources in their varying states. That is, with the REST framework, Web Services functionality is exposed as resources and signified by a distinctive Uniform Resource Identifier (URL). These web resources are operated on by a set of well-known, standard operations – GET, POST, PUT, and DELETE - in attempt to emulate HTTP and similar protocols. Unlike SOAP, the REST messaging framework has a lightweight payload which makes it suitable for mobile networks. REST-based Web Services have gained significant popularity in the web community owing to their scalability and simplicity (Elgazzar et al, 2014b). Although both REST and SOAP frameworks can be used to implement mobile Web Services each has its own strengths and weaknesses. In the next subsection selected key performance parameters are presented. These parameters are used to compare both frameworks with regards to their suitability for AMC.

2.4.4 Performance Parameters for Evaluating Web Services Frameworks

Web Services are tending towards the mobile wireless world as an emerging technology for mobile environments. This tendency is driving research efforts to focus on how mobile devices can operate both as Web Service providers and clients (Guido Gehlen, 2005; Cao et al, 2009). An imperative requirement of this research direction is to provide uninterrupted, lightweight, Web Services to resource-constrained devices operating in a dynamic environment. To guarantee the above need, performance evaluation research has used different parameters to evaluate the suitability of conventional Web Services standards with regards to accommodating the intrinsic device and network constraints in mobile

environments. In this section, some key performance evaluation parameters are discussed. These parameters have greater implications for mobile devices' resources and quality of service, as stated in (Hamad et al, 2010; Castillo et al, 2011). The purpose of this discussion is to form the bases for comparison between SOAP and REST frameworks with respect to their suitability for adoption to AMC.

i. Payload

In SOA, payload refers to the actual message content that is exchanged between applications. Since SOAP and REST adopt different Web Services information model implementation their message sizes (payload) are equally different. Considering the limited nature of mobile devices' resources, employing a communication model with heavy payload may result in unacceptable performance overheads. Such performance overhead is mainly due to the encoding and decoding of messages, which contradicts the resource requirements of mobile devices with limited processing speed and short battery life. Furthermore, in an AMC scenario, resource usage optimisation is a key requirement because resource-constrained devices are also expected to function as Web Service providers.

ii. Flexibility

Generally sense, flexibility is an attribute or design principle of a software that makes it possible for the software to adapt to different requirements. In our context, flexibility is considered with regards to the ability to allow varying types of return data. Because AMC is characterised by dynamic context and consists of heterogeneous mobile nodes, a flexible information model would be a more ideal choice. Therefore, constraining the acceptable return data to a particular data type may affect the overall essence of the AMC setting.

iii. Response Time

Response time is the time it takes to get feedback for a service request. The effect of response time can be considered from two perspectives: device resources and quality of service. Generally, response time is directly proportional to energy (battery) consumption. That is, less response time corresponds to less transmission time, which results in lower energy consumption. On the other hand, more response time implies more transmission time, leading to more power consumption (Hamad et al, 2010). On the aspect of quality of service, long response time leads to client dissatisfaction. Therefore, a good Web Service development framework for AMC should guarantee the design of Web Services with short response time to be able to achieve optimal performance in mobile devices with limited battery power.

iv. Power Consumption

As pointed out in Ravi & Peddoju, (2013), energy consumption (the rate of battery depletion) is one of the major challenges currently faced by mobile devices, especially mobile phones. Since AMC relies on mobile devices that are dependent on batteries with a short life span, it is imperative to take into consideration the power consumption rate of Web Services. Several factors can be responsible for the power consumption rate of Web Services. Such determinants include, amongst others, response time, payload or message size, and bandwidth requirement. Interestingly, these factors that influence the power consumption rate of a Web Service are characteristic of the development framework employed (AlShahwan & Moessner, 2010; Wagh & Thool, 2012).

v. Memory Footprint

The amount of memory consumed in processing a Web Service is known as memory footprint. With mobile devices memory or processing capability is a scarce resource. And in

AMC the principal operation is about sending requests and receiving responses, which requires sufficient memory. Consequently, deploying Web Services that require larger memory footprints in AMC implies placing increasing demand for memory, which could create a resource burden for a mobile device. SOAP-based and REST-based Web Services have been shown to have different footprints because the differences in their development framework mean that they have different payload sizes (AlShahwan & Moessner, 2010).

vi. Caching

Basically, caching is the process of temporally storing data so that future requests for such data can be served faster. In a Web Service provisioning environment, the caching technique is an import tool that can boost performance by minimising response time. Another important aspect of caching is the ability to reduce the amount of processing actually done on the server-side or client-side by storing the results of Web Services requests or invocations. The above-mentioned benefits of caching can greatly save the limited resources of mobile devices because each request sent or received translates to energy and resource consumption.

2.4.5 Comparative Analysis of SOAP and REST Web Services

This subsection compares SOAP and REST frameworks to determine their suitability for use in in AMC. For the sake of clarity the selected parameters for comparison are further categorised into two groups: design flexibility and performance overheads, as shown in Table 2.1.

Outside the parameters used in the comparison, each framework has its own distinctive features and shortcomings that make it more or less suitable for certain types of application environment (Castillo et al, 2011). Therefore, since resource limitation is a central challenge in AMC, this comparison focused on parameters that directly impact on mobile resource.

From Table 2.1 it is concluded that the REST approach is more suitable for AMC compared to SOAP.

Table 2.1: Comparison of SOAP and REST Frameworks

Evaluation Parameters		SOAP-based	REST-based
Design flexibility	Flexibility	Less flexible • Requires binary attachment parsing • Web Service return only XML data	More flexible • Support all data types • Returns all data types
	Caching	Not supported	supported
Performance overheads	Bandwidth requirement	High	Low
	Power consumption	High	Low
	Response time	Long	Short
	Payload	Heavy	Light
	Memory footprint	Large	Small

Suitability for Ad-hoc Mobile Cloud		
Evaluation Parameters	SOAP	REST
Flexibility	X	✓✓
Power consumption	X	✓✓
Response time	✓	✓✓
Payload	✓	✓✓
Memory footprint	✓	✓✓

2.5 Features of Service Discovery Mechanisms

Service discovery mechanisms have gained research interest especially in distributed computing infrastructures like Cloud and Mobile Cloud Computing in recent years (Kousiouris et al, 2012). In these domains. there are different taxonomies of service discovery mechanisms based on a number of characteristics such as functionality, implementation differentiation, network type, and performance (Kousiouris et al, 2012; Perianu et al, 2005). These categorisation parameters have different features, based on which the impact on service discovery can be analysed. In this section, selected features of service discovery mechanisms are introduced in relation to the requirements of service discovery in the Ad-hoc Mobile Cloud as was mentioned in chapter one and emphasised in earlier sections of this chapter. This work classifies the features of service discovery mechanisms according to Figure 2.1, which is adapted from (Kousiouris et al, 2012).

With regards to functionality, a service discovery mechanism can be described based on its specification, knowledge modeling, and knowledge processing features.

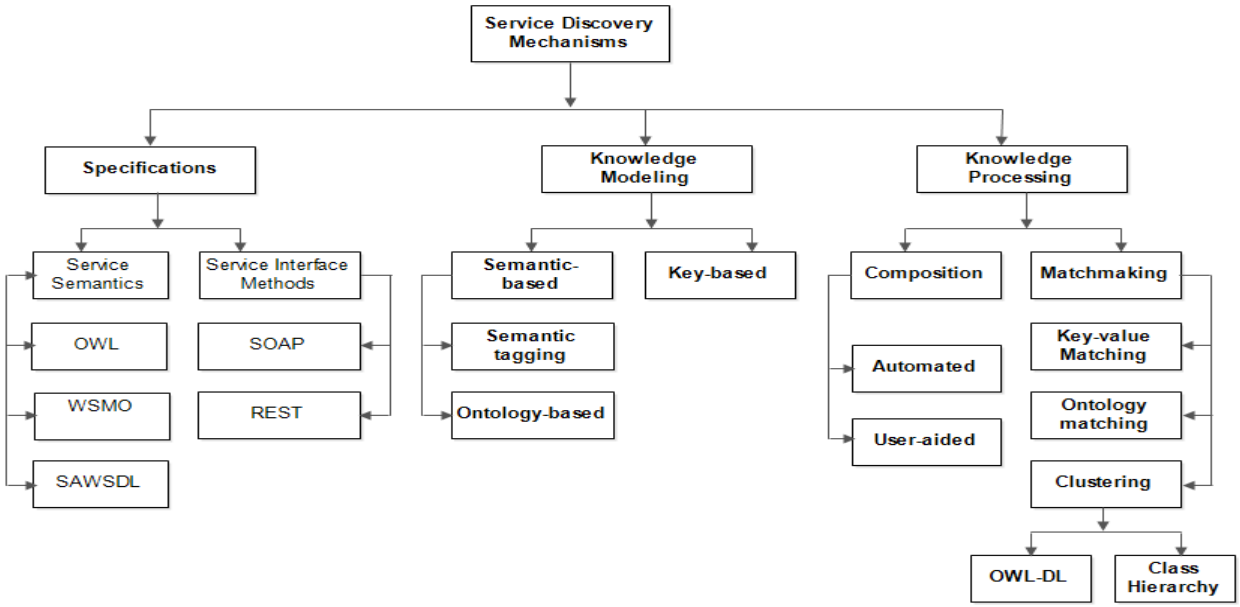


Figure 2.1: Cloud/MC Service Discovery Features, adapted from (Kousiouris et al, 2012; Perianu et al, 2005).

These features depicted in Figure 2.1 are directly linked to the two phases in the Web Services process, namely Web Service development (description, implementation) and Web Service consumption (matchmaking/discovery). Specification and knowledge modeling fall under the first phase and knowledge process under the second. With respect to resource limitation, which is part of the core requirements for service discovery mechanisms in AMC, the above-mentioned features are used as the basis to discuss the suitability of Mobile Cloud service discovery mechanisms for use in AMC.

2.5.1 Specification Feature

In order to provide a mechanism for the efficient implementation of service-relevant operations in the Mobile Cloud environment, such as service provision, service discovery, and service management, a comprehensive and precise service specification model is required. Specification is the aspect of service discovery that deals with how network services are described. There exist a number of specifications and protocols that can meet various needs of implementing a service discovery mechanism. Such needs may vary depending on factors such as the type of registry implementation, the kind of information model used, the type of interfaces design, and whether the mechanism involves the use of semantics (Perianu et al, 2005) amongst others. These specifications have technological dependencies that impact on the performance of a discovery mechanism or make it suitable in a given environment (Omrana et al, 2013). The two dimensions of specifications are service semantics and service interface methods.

i. Service Semantics

Service semantics deals with how to create service abstractions that make meaning for discovery agents or mechanisms. That is, the service semantic aspect addresses the issue of what exactly a service is and how it can be described. This level of specification refers to how

a Web Service is described or defined using its non-functional and functional parameters as well as other service features as discussed in subsection 2.4.1. In an earlier discussion of in subsection 2.4.1.1 that introduces the two approaches to Web Service description (specification), it was concluded that semantic approaches are resource-intensive for mobile devices. However, from Figure 3, the three approaches for realising service semantics are Web Ontology Language (OWL), Web Service Modeling Ontology (WSMO), and Semantic Annotation for Web Service Description Language (SAWSDL). These approaches are all based on semantics and ontology. The indication is that current trends in Cloud and Mobile Cloud service discovery are tending towards total support for semantic service description in order to guarantee the efficient discovery of relevant services and enhance interoperability (Venticinque et al, 2011; Rodríguez-García, 2014). The reason for this reliance on semantics is that the large volume of services in the Cloud, in addition to their specific features, present a number of challenges associated with service delivery, service consumption and management (Sun et al, 2012). Therefore, one way to address these challenges is to adopt a comprehensive and precise specification language with which to: 1) describe the unique features of Cloud services, 2) improve the ability of Cloud services discovery and selection mechanisms, and 3) ensure reliable service provisioning (Dong et al., 2013).

ii. Semantic Interface Methods

This feature of service discovery mechanisms deals with how to specify information exchange mechanisms that enables Web Services to communicate and inter-operate with each other. From our discussion in subsection 2.4.2, the sematic interface method feature represents the Web Service implementation platform.

Considering the technology used, Cloud resource interface specifications are of three kinds (Prodan & Ostermann, 2009): 1) Web Service-based, using SOAP, 2) HTTP query-based,

using REST, or 3) simple command line-based. The SOAP approach is the de facto standard approach widely adopted by the industry and supported by almost all development tools (Elgazzar et al, 2014b). However, according to Castillo et al, (2011), the majority of Cloud services with public API support REST interfaces, some support both REST and SOAP, while few offer just SOAP. Although a majority of the major Cloud service providers offer RESTful Web Services (Castillo et al, 2011), in the context of Mobile Cloud, the RESTful services are described semantically as shown in Figure 2.1, which introduces a resource burden in service discovery for mobile devices.

A comprehensive discussion on SOAP and REST frameworks was presented in subsection 2.4.2.

2.5.2 Knowledge Modeling Feature of Service Discovery Mechanisms

Providing services over the Internet, like the case of Cloud or Mobile Cloud Computing and other SOC environments are practical instances of automating real-life business processes. Such automation requires eliciting and representing knowledge for the purpose of processing. In order to represent knowledge with regard to the context of the services or resources provided, there are a number of ways to represent this information. In Figure 2.1 knowledge can be modeled using a semantics-enabled or key-based approach. Apart from aiding the storage and processing of data or resources in the Cloud these advanced formalisms for representing knowledge also help to hierarchically classify the numerous concepts that may be used to describe a service or resource, thereby enhancing fast, effective, and efficient service discovery (Kousiouris et al, 2012).

i. Semantic-based

Semantic modeling is either achieved by using sematic annotation and tagging or ontology-based methods. The latter requires structured semantics such as ontologies to store and

correlate information. In contrast, the first approach requires the use of a semantic Annotation Service Description Language to describe service functionality and other features. Semantic modeling using either of the above approaches is widely supported by Cloud and Mobile Cloud service discovery mechanisms. Generally, support for semantics, whether at the specification or modeling levels, enables service discovery mechanisms to adopt sophisticated techniques that help to improve the efficiency and accuracy of discovering relevant services (Kousiouris et al., 2012). But such advanced techniques may only be practicable in the Cloud and Mobile Cloud scenarios, where computational resource is not a challenge.

ii. Key-based

The simplest approach to model knowledge for pervasive computing is by using key-value pairs. This key-based approach is less popular in current literature on Cloud service discovery because of scalability and flexibility issues. However, this modeling approach still finds rich application in context modeling and in frameworks where services are described by a list of simple attributes in a key-value manner (Bettini et al, 2010). When services are modeled as key-value pairs or attributes, the employed service discovery procedure must execute an exact matching algorithm on the attributes. The major advantage of this approach is that it requires less computational resources as against the former approach with high level matching of semantic information (Nadoveza & Kiritsis, 2014).

2.5.3 Knowledge Processing Feature of Service Discovery Mechanisms

Service discovery entails processing knowledge in the matchmaking process, which involves matching a user's requirement against the available services. Notwithstanding the way the information is represented or stored, processing of this information for the purpose of performing matchmaking between service requests and service offers is of immense

importance in the discovery of relevant services. However, the matchmaking, especially when it involves processing semantically represented knowledge requires significant computational resources (Steller, 2010; Elgazzar, Hassanein, et al, 2014). An important feature of a service discovery mechanism with regards to functionality is the manner in which it processes knowledge. Some discovery mechanisms use a composition technique while others employ the matchmaking technique.

i. Composition Approach

A composite service is created between different services in order to meet a client's requirements in case a single service is not found that satisfies the client. This process of assembling services may be automated or user-aided, depending on the service description approach adopted. Although composition can be user-aided, in the Cloud and Mobile Cloud environment with a large number of services it is difficult and less efficient to compose services manually. This makes automated composition, which largely depends on semantics and ontology technologies, more ideal for the Cloud or Mobile Cloud. However, as motivated in chapter one, AMC is an opportunistic service offering platform to complement the Cloud and meant to service the needs of a small computing community, say, an SMME. With this characteristic of the AMC, there may be no need for such complex services that require composition techniques.

ii. Matchmaking Approach

Matchmaking contributes directly to the discovery of relevant services by a service discovery mechanism and determines whether the mechanism will be suitable for a particular environment. In the first instance, the efficiency of the matchmaking algorithm is critical to the discovery of relevant services. More importantly, the suitability of such algorithms depend on the technique used by the matching process such as key-value based, reasoning or

the clustering technique. As stated earlier, matchmaking is generally a resource-intensive process in a service discovery mechanism. For this reason, the service discovery mechanism may be rendered unsuitable for a particular environment due to resource considerations (Steller, 2010; Kousiouris et al, 2012).

2.6 Chapter Summary

This chapter discussed issues related to AMC service discovery requirements. The role of context-awareness in service discovery and the concept of device context were introduced in sections 2.2 and 2.3. Web Service development was discussed in section 2.4 with a focus on how certain Web Service techniques and approaches may not support the resource requirement of mobile devices. Based on existing knowledge, it is concluded that non-semantic descriptions and RESTful Web Services are more suitable for AMC. This conclusion is drawn based on previous discussions in subsections 2.4.2 and 2.4.3, which revealed that these two approaches are, among other factors, less resource-intensive.

Furthermore, it was learnt from Kousiouris et al, (2012) that the performance of a service discovery mechanism and its suitability for use in a particular environment depends on, among other things, its knowledge modeling and processing features. Therefore, section 2.5 discussed and analysed various features of discovery mechanisms with regard to how they support the requirements for relevant service discovery in AMC. In the next chapter, relevant literature on Mobile Cloud Computing was reviewed. In the review, special attention was paid to how the concepts discussed in this chapter are adopted. The state-of-the-art on mobile Web Service discovery was also be explored with a view to ascertain the gap that this research work seeks to address.

CHAPTER THREE

LITERATURE REVIEW

3.1 Introduction

It was concluded in the previous chapter that non-semantic approaches to Web Service description and implementation are more suitable for service discovery in AMC. Concluding thus was based on the fact that semantic approaches are more resource-intensive (Steller, 2010). Meanwhile service discovery is crucial to any service centric system and is significantly impacted by the aforementioned problem. Efficient discovery of relevant Web Services in the Mobile Cloud is often achieved through advanced discovery mechanisms which mostly rely on semantic concepts, as revealed in the previous chapter. This adoption of advanced techniques by Mobile Cloud service discovery mechanisms raises the question of whether the same mechanisms can be adopted in AMC.

In this chapter, related scholarly works in Mobile Cloud and AMC service discovery are reviewed with a view to answering the above question. A formulated framework based on the goal of this study is used to analyse existing works. The rest of the chapter is structured as follows. A framework for analysing existing works is presented in section 3.2. The framework highlights the impact of core requirements such as resource-intensiveness, context-awareness, request adaptation, and relevancy factor in AMC service discovery. Section 3.3 discusses service discovery in the Mobile Cloud, focusing on the implications of its reliance on semantics techniques. The state-of-the-art in AMC service discovery is explored in section 3.4, where there is an attempt to categorise scholarly works in the domain into three schools of thought. These categories are then comprehensively analysed within the context of the proposed framework for analysis, which emphasises the goal of this research work. Then section 3.5 summarises the chapter.

3.2 Framework for Literature Analysis

In order to critically analyse scholarly works in the Mobile Cloud service discovery domain, a framework has been formulated to examine discovery mechanisms proposed in the literature. The framework depicted in Table 3.1 is based on the goal of this study which is to achieve relevant service discovery in AMC in a resource-aware manner based on device context.

Table 3.1: Framework for Analysing Existing Work

Resource-intensiveness	The level of computing resources required for processing. A good service discovery mechanism for AMC should be lightweight i.e., adopt techniques that do not weigh down the constrained resources of mobile devices. Resource-intensiveness is mainly caused by the use of semantic or ontology at two levels in a service discovery process: i) Service description level: requires semantic description languages ii) Matchmaking level: requires a semantic reasoner (computationally complex and resource-intensive activity that does not scale well to mobile devices)
Context-awareness	This refers to the ability of a service discovery mechanism to take into cognisance the changes in its environment (context change) and respond proactively to discover relevant services that fit the current context. This work distinguishes two kinds of context that can be used in service discovery: i. Static context: device profiles ii. Dynamic context: user ratings, user preferences, environmental parameters, and device context (battery level, available memory). In AMC combining device context with other contexts data will both enhance relevant service discovery and resource utilisation.
Retrieval approach	Services are either retrieved before filtering or the service request is first adapted based on certain parameters so that only services that meet such conditions are retrieved for ranking. The second approach returns a smaller number of relevant services and is therefore more suitable for AMC due to limited resources (screen, memory)
Relevancy factor	The factor(s) that determine the relevance of a retrieved service. Relevance is either user-centred or user and device-centred. It is better to consider a service to be relevant if it fulfills the required functionality and matches the resource capability of the client device. Considering relevance in such a dimension helps to create a balance between user's satisfaction and resource optimisation, which is key to the success of AMC.

This framework is intended for two purposes: first, to argue the suitability of Mobile Cloud service discovery mechanisms for AMC and second, to justify the gap in the state-of-the-art in AMC service discovery as highlighted in the statement of the problem of this work in chapter one.

3.3 Service Discovery Approaches in Mobile Cloud

Generally, service discovery in Mobile Cloud extends across multiple levels ranging from infrastructure service discovery such as physical computing resources like printers, and storage devices, to application-level service discovery, in which potential clients search for network functions that satisfy their requirements. This section focuses on the application level service discovery and related works are reviewed. However, whether viewed from the perspective of infrastructure services or application level services, discovery in Mobile Cloud is driven by either of two approaches – semantic or syntactic (non-semantic).

3.3.1 Semantic-based Service Discovery

Earlier findings in chapter two revealed the predominant use of semantics and ontology concepts in Mobile Cloud service discovery. For example, Cortazar et al, (2012) proposed a Cloud Computing ontology to facilitate semantic identification, discovery and invocation of Cloud services. This Cloud ontology uses a semi-automatic tool to annotate Cloud services and store their semantic descriptions. Rodríguez-García, (2014) also presented a similar proposal but with the addition of an advanced feature called ontology evolution. According to the framework in Table 3.1, the aforementioned service discovery approach is resource-intensive. Such resource-intensiveness is caused by the requirement to semantically describe Web Services and therefore involves the use of semantic matchmakers. With respect to context-awareness, although semantics can enhance the use of context information in service

discovery, the work under consideration did not deal with context-awareness. Therefore, the last two parameters in the framework do not apply.

In a similar approach, Nagireddi & Mishra, (2013) proposed a framework for constructing a Cloud ontology based on Cloud services and their attributes. Their ontology was designed to incorporate a service discovery mechanism for searching services using a generic search engine approach. Using the Cloud ontology, service requests are ordered and relevant services discovered based on their associated attributes. Then discovered services are ranked based on the service attributes using the Analytic Hierarchy Process (AHP). Nagireddi & Mishra, (2013) dealt with some issues considered in the framework of Table 3.1 such as context-awareness and relevancy ranking. However, the heart of their solution approach hinges on the use of ontology, which is strongly discouraged for AMC service discovery mechanisms.

Also, Kang & Sim, (2011) proposed the Cloudle, which is an ontology-enhanced service search engine. The Cloudle consults a Cloud ontology to reason about the relations between Cloud services and performs inferences that are used to make informed decisions via a three tier similarity reasoning model comprising i) concept similarity reasoning, ii) object property similarity reasoning, and (3) datatype property similarity reasoning. Again, referring to the framework in Table 3.1, the Cloudle, apart from being silent on other issues in the framework will not be adequate for AMC because of its use of complex and resource-demanding reasoners for matchmaking.

Another ontology enhanced service discovery mechanism is discussed by Ngan & Kanagasabai, (2012). The authors formulated a Cloud service brokering system that performs an OWL-S based service discovery and selection. The OWL-S broker supports dynamic semantic matching of Cloud services that are described based on complex constraints. The

broker helps to handle changes in service context. However, the description and matchmaking levels involve semantic technique and the service discovery process is mediated by a broker. As explained in the framework in Table 3.1, discovery approaches involving semantics are often resource-intensive for AMC. Furthermore, the presence of a broker implies that the discovery mechanism will be suitable for platforms with infrastructural support, unlike AMC, which is infrastructure-less.

In other efforts, semantically enhancing service categorisation and service requests has equally been explored to achieve relevant service discovery. For instance, Paliwal et al, (2012) proposed a solution for achieving functional level service categorisation based on an ontology framework. Their solution approach employs a clustering technique to accurately classify Web Services based on service functionality. Two features enable this discovery mechanism to achieve efficient matching and retrieval of relevant services: i) the semantic enhancement of service requests, that is, allowing requests to be expanded with additional terms retrieved from ontology that may be deemed relevant for the requested functionality, ii) the use of Latent Semantic Indexing (LSI). With respect to Table 3.1, Paliwal et al, (2012) addressed the issues of context-awareness, relevancy ranking and service request adaptation. However, the approach adopted by the authors is not lightweight. In addition, the ontology framework proposed by Paliwal et al, (2012) relies on the UDDI registry to perform offline semantic-based categorisation. This dependence on the UDDI registry does not conform to the envisioned architecture of AMC.

There are several other notable research efforts with diverse approaches. However, such approaches are still centred around the use of semantics and ontology as reported in the literature (Zeng et al, 2009; Gutierrez-Garcia & Sim, 2010; Hatzi et al, 2012; Liu et al, 2014).

The trend in the works reviewed so far indicates that service discovery mechanisms in the Mobile Cloud are highly dependent on semantics. This reliance on semantics is because semantics greatly improves service discovery (Bener et al, 2009) and the elastic nature of Mobile Cloud resources allows it to accommodate heavy and complex semantic computations. However, as explained in the framework in Table 3.1, semantics techniques impose a huge resource burden on resource-constrained devices. Therefore, it is inferred that the Mobile Cloud service discovery mechanisms discussed thus far are not suitable for AMC.

3.3.2 Non-semantic-based Service Discovery

Two challenging issues face service discovery in the mobile environment. These are dynamic context and resource constraints. Although Mobile Cloud computing was originally introduced to, among other issues, address the second problem (Dinh et al, 2013), current research has also focused on addressing the issue of context change, which is inherent in the nature of mobile environments.

In line with the above research direction, diverse discovery approaches have been proposed without semantic enhancements. Such discovery mechanisms have laid more emphasis on how to adapt to changes in context in order to discover relevant services. Notable among them is the work of Capra et al, (2003), where the authors developed context-aware reflective middleware system for mobile application – CARISMA. The idea is to build a middleware to enhance and support the construction of context-aware mobile applications rather than a platform for facilitating context-aware service discovery.

In a similar effort, Keeney & Cahill, (2003) presented an open framework for dynamic adaptation of services called Chisel. The Chisel framework uses the concept of reflection in a policy-based manner to drive dynamic context-aware adaptation of service objects. Also, in (Rouvoy et al, 2009) a middleware to support self-adaptation in ubiquitous and service-

oriented environments (MUSIC) was proposed. The middleware uses the planning-based adaptation approach which makes use of a QoS-aware model to support dynamic service discovery.

Keeney & Cahill, (2003) and Rouvoy et al, (2009) achieved dynamic discovery of relevant services by adapting services to changes in execution environment, user context, and application context as highlighted in Table 3.1. However, device context was not considered and this leaves a gap to be filled in this research. Also, the architecture of MUSIC relies on a central repository, which does not support the concept of AMC.

In contrast, Papakos et al, (2010) provided a context-aware service discovery middleware for mobile systems "VOLARE". The middleware enables dynamic adaptation of Cloud service discovery based on the concept of reflection. By leveraging on the resource rich Cloud infrastructure, Papakos et al, (2010) comprehensively addressed the issue of context-aware services discovery in MCC. Their approach dealt with service request adaptation as well as using context to determine the relevance of discovered services as explained in the framework in Table 3.1.

However, the requirements for context-aware service discovery in MCC may not all be the same for AMC. For instance, in MCC services are hosted in a Cloud server, not on resource-constrained devices as it is envisaged in the AMC scenario. Hence, in AMC, changes in a device's contexts, that is, battery level and available memory may cause the device to disappear from the network or become incapable. Also, VOLARE is designed to adapt service request for streaming Cloud services at runtime. Deploying streaming Web Services in an AMC scenario as envisioned in this work is not practicable for reasons such as limited bandwidth, absence of a central infrastructure, and resource limitation.

Since consumers find and consume services from different devices, some scholars have utilised device profile (static context) to drive the process of discovering relevant services. Al-Masri & Mahmoud, (2010) worked in this direction by proposing "MobiEureka" - a device-aware Web Service discovery mechanism that integrates device capabilities into Web Service descriptions. The goal of their mechanism is to discover services that not only satisfy users' requirements but also function within the client's device constraints. To improve the expressivity of traditional WSDL to be able to incorporate device capabilities in a service description the authors proposed the WSDL-M. Using a device-aware ranking algorithm, retrieved services are ranked according to their fit to the device features.

However, with regard to addressing the open issues identified and explained in the proposed framework of Table 2.1, Al-Masri & Mahmoud, (2010) did not consider dynamic device context (battery level, and available memory). Also, in their approach, service requests are not first adapted before services are discovered. Contrary to the above approach, this study defines relevance of discovered services as matching both the static and dynamic device context of the consuming device.

3.4 Service Discovery Approaches in the Ad-hoc Mobile Cloud

Mobile Web Service discovery faces a unique challenge due to, among other factors, the stringent constraints of mobile devices. Ideally, existing Mobile Cloud service discovery mechanisms should be adopted into AMC since the fundamental techniques of service discovery are the same. But generally speaking, current service discovery techniques are essentially designed for static and wired environments.

Therefore, the inference drawn in section 3.3 agrees with the claim of Elgazzar et al, (2011) that existing discovery approaches are incapable of understanding the limitations of mobile devices and wireless networks. Following this inference, which is based on existing

knowledge, this study argues that existing Mobile Cloud service approaches are not the best candidates to deliver efficient and reliable discovery in mobile scenarios. Towards evolving discovery mechanisms that take into account the unique characteristics of mobile environments, especially with regards to resource constraints, literature is still scanty. However, this study identified and classified existing scholarly works into three schools of thought: i) using light-weight semantic reasoners, ii) offloading computational intensive tasks to the Cloud, and iii) the pure Ad-hoc approach. Before discussing these schools of thought, works that investigated and validated the feasibility of mobile Web Service hosting as the foundation of AMC service provisioning are discussed first.

3.4.1 The Concept of Mobile Web Service Provisioning

Current research is paving the way for Mobile Web Service provisioning as a major step towards the realisation of pervasive and ubiquitous computing. Such research efforts are primarily aimed at easing the various challenges faced by mobile devices in accessing remote Web Services, by introducing mobile Web Services.

Since the inception of the idea of "Personal Server" as proposed by Want et al, (2002), the topic of mobile Web Services has continuously gained research attention. Subsequent researches have leveraged on the advancements in mobile device hardware and software capabilities to experiment with mobile Web Service hosting.

For instance, El-Masri & Suleiman, (2005) presented a "Mobile Web Server" framework. In their framework, the authors addressed the challenge of enabling hosting capabilities in mobile devices. The idea is to allow both mobile providers and clients to communicate with each other and the mobile web server via their individual network operators. The architecture incorporates a public UDDI registry to advertise Web Services hosted on mobile devices.

A purely decentralised approach to mobile Web Service hosting was first reported in (Kim & Lee , 2007). In a similar effort, Guido Gehlen, (2005) introduced the P2P Web Services Provisioning platform for mobile environments. This approach uses mobile peer nodes that act as both server and client and provide services to each other in an Ad-hoc manner.

Recent efforts in this domain include the works reported in (Wagh & Thool, 2013; Srirama et al, 2006), which implement various approaches to enhance successful hosting and discovery of Web Services on mobile devices. The authors use technologies like a Lightweight SOAP server and SQLite server respectively to implement a Mobile host. Our work takes a lead from the works of Guido Gehlen, (2005) and Wagh & Thool, (2013).

3.4.2 The Light-weight Technology Approach in AMC Service Discovery

While several technologies and approaches for Web Service discovery are considered too resource-intensive for mobile Web Services, a number of optimised technologies have been developed over the years to cater for the unique limitations of mobile devices. For instance, despite the fact that semantic reasoning is a computationally complex and resource-intensive activity to mobile devices, Steller, (2010) advocates the use of highly optimised semantic reasoners. The work aims at bringing the efficiency and accuracy introduced by the use of semantics into relevant service discovery in mobile environments. To realise this aim, the author presents a light-weight and adaptive approach for on-board semantic mobile matching. The light-weight approach proposed involves the development of an optimised semantic reasoning algorithm called “mTableaux”.

However, Steller, (2010) only concentrated on semantic reasoning optimisation for mobile devices not service discovery. Hence, other issues described in the framework of Table 3.1 were not discussed.

In another research effort, Bianchini et al, (2007) presented a light-weight ontology-based service discovery approach. Their work formulated a semantic-enriched framework to describe services and an ontology-based discovery mechanism. In order to cope with the limitations of mobile devices, their solution focuses on simplifying service descriptions by only specifying a few elements. The framework uses a central registry (UDDI) to store the ontology database for matchmaking and discovery. The discovery process utilises functional parameters to discover and rank services based on device type and location of requesters.

Another light-weight solution to mobile service discovery is proposed by Saadon & Mohamad, (2014b). By using WSMO-Lite, the authors provide an enhanced description of Non-Functional Properties (NFPs) of services. This enhancement to service description generates small payload as against traditional semantic descriptions. The actual discovery mechanism employs a semantic matchmaking algorithm based on WSMO-Lite while Quality of Web Services (QoWS), Non-Functional Properties (NFPs), and various dynamic and static contexts are used to discover and rank services.

Both Bianchini et al, (2007) and Saadon & Mohamad, (2014) failed to address the issues of device context, request adaptation, and semantic matchmaking in the context of the framework offered in Table 3.1. In addition, the approach of Bianchini et al, (2007) relies on a central registry. This dependence on a centralised registry makes the approach unsuitable for a pure Ad-hoc scenario. Based on the literature reviewed so far, this work agrees that optimising service discovery mechanisms has rich benefits for mobile environments. But it is also argued that such optimisation does not eliminate the resource burden on limited resource devices. More so, in the Ad-hoc Mobile Cloud scenario, which is basically a small computing community made up of mobile devices with limited individual hosting capacities, the

number of offered Web Services is expected to be relatively moderate. Therefore, the use of high-level (semantic) matchmakers will not be necessary.

3.4.3 Offloading of Execution to the Cloud Technique in AMC Service Discovery

Resource constraint is a major challenge facing service discovery in mobile environments (Elgazzar et al, 2014a), principally due to their inherent nature. Consequently, to bridge the gap between resource-constrained environments and resource-intensive Web Service discovery, the second school of thought has explored the option of pushing resource-intensive tasks of service discovery to the Cloud. The goal, then, is to develop service discovery mechanisms that can achieve relevant service discovery while minimising the resource burden placed on resource-constrained devices.

One notable contribution in the above direction is the Cloud-based Mobile Web Service Discovery "CMDis" proposed by Saadon & Mohamad, (2014a). In CMDis, the authors formulated a service discovery mechanism that uses two strategies to addresses the challenge of resource limitation in mobile devices:

i. Adoption of a semantic light-weight Web Service description using hREST semantic description language on REST-based architecture. This is meant to handle the issue of unacceptable overhead messages generated by semantic Web Services and SOAP-based architecture.
ii. Running of a complex semantic-based service discovery process mediated by a Cloud broker. By this design, the computational intensive processing of ontology concepts during service discovery is performed in the Cloud.

Saadon & Mohamad, (2014) implemented a Degree-of-Match (DoM) algorithm that utilises environmental context, and device profile to rank relevant Web Services.

In the same vein, but with a more flexible approach, Elgazzar et al, (2013a) presents a framework for Cloud-assisted mobile service provisioning and discovery. The idea is to provide a framework to support dynamic offloading of computational-intensive tasks to the Cloud based on the resource status at the mobile end. However, the authors did not focus on service discovery but on how to determine through an intelligent decision making process when to offload.

Also, in a closely related work, Elgazzar et al, (2014a) presented another Cloud-based context-aware service discovery framework. The idea is to offer Discovery-as-a-Service (DaaS) in the Cloud for mobile devices. They argued that with such an approach the Cloud can build bridges between mobile devices as a convenient ubiquitous interface to bootstrap Web Service discovery. In DaaS, resource-intensive processes run on the Cloud server and are only offered as service to mobile clients on demand, as depicted in Figure 3.1.

Figure 3.1: Service Discovery as a Service
(Elgazzar et al, 2014)

Leveraging on the unlimited Cloud Computing resources, Elgazzar et al, (2014) provided a robust service discovery mechanism capable of discovery and ranking non-semantic and semantic Web Services using a keyword-based information retrieval technique and a high-level matchmaking approach respectively. This discovery mechanism also uses various contexts for Web Service filtering and ranking.

Although in CMDis and DaaS the issues of resource-intensiveness and context-awareness have been variously dealt with, the other issues of service retrieval and relevancy factor were not addressed in the context of the framework in Table 3.1. Also, the core idea presented in (Saadon & Mohamad, 2014a; Elgazzar et al, 2014a) is not feasible in the AMC scenario where there is either limited or no access to the Cloud.

Another significant effort was made by Ravi & Peddoju, (2013) in their work which presents a service provisioning and discovery framework that uses Cloud and Mobile Ad-hoc Cloud together. The main goal of their proposal is to achieve energy efficiency by utilising the computational power of heterogeneous mobile devices within a cluster of devices. The authors realised this goal by providing a platform for load sharing among peer devices as well as a mechanism that facilitates the offload of intensive computations to the Cloud.

Ravi & Peddoju, (2013) employed an Offloading Decision and Discovery Algorithm to determine when it is beneficial to offload to the cloud server (when the required computation is large) and discover the right service. Their algorithm utilises several parameters, such as distance (hop count), computation capability, network characteristics, and available energy. Although Ravi & Peddoju, (2013) addressed the fundamental challenge of computational burden, other pertinent issues of context-awarenes, device context, and request adaptation were not looked into, as highlighted in Table 3.1. In addition, there was no experimental evaluation.

3.4.4 The Pure Ad-hoc Approach to AMC Service Discovery

The last category of works in mobile Web Service discovery is based on a pure Ad-hoc or decentralised framework where the Web Service discovery process takes place between mobile peers. Service discovery in AMC requires a decentralised approach because of the dynamic nature of participating nodes.

Generally, not much literature is available on Ad-hoc Mobile Cloud service discovery because it is an emergent field. But recent years have witnessed significant research interest in mobile Web Service provisioning, which is the motivation for AMC. An early work in this domain is reported in (Srirama et al, 2008). In this work the authors presented a P2P mechanism for publishing and discovering Web Services based on JXTA technology. The JXTA advertisement feature was employed to describe and publish Web Services as JXTA advertisements. Additionally, a keyword-based search mechanism was utilised to discover services that are then ordered according to their relevance using Apache Lucene tool. Nonetheless, the issues of context-awareness, request adaptation, and the use device context to determine service relevance as were highlighted in Table 3.1 were not taken into consideration by Srirama et al, (2008).

In an attempt to address all the fundamental aspects that account for efficiency and effectiveness in mobile Web Service discovery, Elgazzar et al, (2011) presented a generic Web Service discovery framework for mobile environments. Using three-layer architecture, the authors highlighted the core requirements of any sound and reliable discovery mechanism that can facilitate efficient service discovery in resource-limited environments. However, being a generic framework, there was no experimental evaluation and specific approaches were not discussed.

In another effort, Srirama & Paniagua, (2013) extended the idea of Mobile Host to provide a new framework for mobile Web Service discovery. Mobile Host was redesigned based on the OSGi (Open Services Gateway initiative) framework implemented on RESTful architecture. A P2P-based Web Service discovery solution based on the basic keyword-based search mechanism provided by JXTA was utilised. The work did not consider device context or request adaptation.

Device profile and other forms of context information have been richly explored in mobile Web Services discovery. Following the requirements of sound and reliable service discovery mechanisms as suggested in the framework of (Elgazzar et al, 2011), AL-Masri & Mahmoud, (2010) developed a dynamic service discovery mechanism, "MobiEureka". Their approach employs a device-aware technique, which relies on expanded service descriptions based on WSDL-M. With WSDL-M, supported device features are incorporated into Web Service descriptions.

Along the same line of thought, Elgazzar et al, (2013b) presented a services discovery approach that utilises user preferences and other context information such as user rating and device profile. The idea aims at (1) making sure that discovered services fit the user's device constraints and (2) accommodating user preferences in order to improve the quality of user experience.

Although Al-Masri & Mahmoud, (2010) and Elgazzar et al, (2013b) similarly attended to the issues of context-awareness and ranking of relevant Web Services, device context was not considered. Also, in their work, services are first retrieved before filtering as against adapting service request in order to reduce the number of returned services as advocated in the framework of Table 3.1. While several service discovery mechanisms have been proposed for supporting service discovery from mobile devices, there is need to pay specific attention to using device context (as defined in this work) as the basis to determine the relevance of Web Services in AMC in addition to other context information.

3.5 Chapter Summary

This chapter first formulated a framework to analyse existing literature on Mobile Cloud and AMC service discovery. The current trend in Mobile Cloud service discovery with regards to the major approaches used was discussed. The discussion showed that semantic techniques

are predominantly used to enhance service discovery in Mobile Cloud. It was also revealed that such enhancements basically address the limited expressivity of conventional WSDLs as described in chapter 2. The overall goal of improving expressiveness of WSDL is to boost accuracy and efficiency in service discovery as well as to support service discovery automation. Also, the state-of-the-art in Ad-hoc Mobile Cloud service discovery was explored. Based on the framework developed in this chapter, it was argued that, first, current Mobile Cloud service discovery mechanisms are too resource-intensive for resource-constrained devices and are therefore not suitable to be adopted for AMC. Second, the state-of-the-art in AMC service discovery does not consider the relevance of a Web Service as a function of fulfilling both users' requirements and device resource capabilities. Taking the above findings into account the goal of the next chapter will be to formulate a service discovery model that employs techniques that address the heighted gap in AMC service discovery.

CHAPTER FOUR

CONCEPTUALISATION OF THE SOLUTION APPROACH

4.1 Introduction

The previous chapter revealed that traditional WSDLs lack expressiveness, while their semantic enhancement and the associated discovery mechanism create resource overhead. These weaknesses directly impact on the requirements of service discovery in AMC, that is, discovering services that fit clients' resource capabilities in addition to having the right functionality. There are two dimensions to the aforementioned problem: i) not having service descriptions that comprehensively capture functional and non-functional properties and ii) not considering resource capabilities as part of device context. However, as learned in chapter three, resource limitation makes it difficult for mobile devices to process semantically enhanced service descriptions without breaching their resource requirements. Furthermore, discovering services in AMC is challenging due to its dynamic context. In this chapter, a context-aware service discovery mechanism (CaSDiM) that utilises a resource-friendly but expressive Web Service description approach that incorporates device context is proposed.

Section 4.2 discusses the issues of resource-friendliness and expressiveness as the design criteria for this work's solution approach; the effect of these criteria on service discovery is analysed. The idea of device context is then presented in section 4.3 as an essential requirement for service discovery in AMC. Based on the device context model, a service description approach is formulated. In line with the proposed design criteria, section 4.4 presents the proposed service discovery and relevancy ranking algorithms. As one of the major objectives of this study an integrated architecture of the proposed context-aware service discovery mechanism (CaSDiM) and its components description are provided in section 4.5. The chapter summary is offered in section 4.6.

4.2 Design Criteria for Context-aware Service Discovery

This study has identified resource-intensiveness and lack of expressiveness as the major shortcomings with conventional Web Service discovery techniques in relation to the requirements of AMC.

In the same vein, due to the characteristics of the Ad-hoc Mobile Cloud environment, which include resource scarcity and dynamic context, it becomes imperative to consider the aforementioned weaknesses in the design and implementation of service discovery mechanisms in order to achieve optimal results.

Following findings from the previous chapter, the design criteria of resource-friendliness and expressiveness were identified as vital requirements for achieving efficient and effective service discovery in AMC.

4.2.1 Resource-friendliness

Resource limitation is one of the core challenges in mobile environments and is widely considered as the yardstick to evaluate the adoptability of a service description language. Moreover, service discovery efficiency is impacted by the quality or nature of the service description (Omrana et al, 2013).

Although certain techniques can boost, among other things, service discovery efficiency, they may have overwhelming resource implications on mobile devices. Such techniques have been found to be too resource-intensive (Steller, 2010) for mobile use.

Therefore, in the context of AMC, resource-friendly or lightweight approaches are highly desirable.

4.2.2 Expressiveness

Lack of expressiveness refers to the inability of a Web Service description language to fully represent a Web Service's functional and nonfunctional properties (Al-Masri & Mahmoud, 2010). This weakness is strongly associated with WSDL W3C standard, which only describes services at the syntactic level, without covering non-functional parameters (Elgazzar et al, 2014b). Meanwhile, the inherent context variation in AMC requires a expressive service description approach in order to guarantee the relevance of a discovered service to the client.

Therefore, it is desirable that service discovery mechanisms in AMC pay attention to the use of various contexts' information as a means to enhance relevant service discovery. One way of achieving such a requirement is to adopt description approaches that are expressive.

4.3 The Proposed Device Context Model for CaSDiM

The design criteria discussed in section 4.2 emphasise the need to utilise approaches that are both resource-friendly and expressive in the design and implementation of AMC service discovery mechanisms. In order to implement a discovery mechanism that fulfils the highlighted design criteria, a device context model is formulated in this section as a tool to support resource-friendliness and expressiveness in AMC service discovery. This work defines device context as comprising two components, namely, device profile (specific supported device features) and resource profile (state of battery and memory). The goal of this approach is to achieve a resource-aware service discovery process suitable for the dynamic nature of AMC.

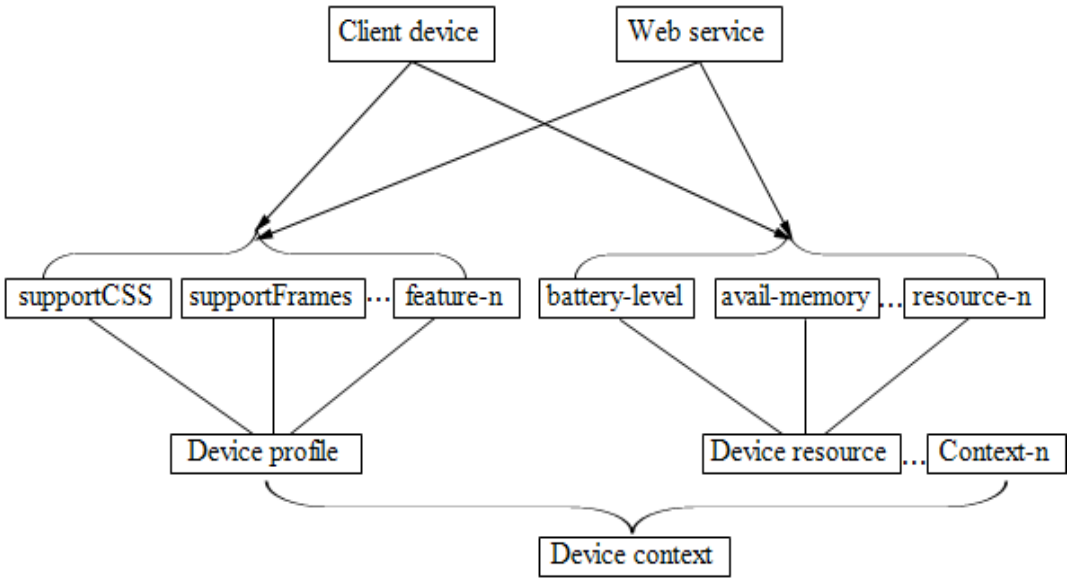


Figure 4.1: CaSDiM Device Context Model

The same design criteria discussed in section 4.2 is applied in the formulation of the device context model, using the context modeling approach introduced in subsection section 2.9.1.

A context model is a systematic way of representing contextual information in context-aware applications in order to facilitate easy manipulation of the represented information. Following the solution approach, which is based on the device context idea, an abstract graph-based context model is formulated as shown in Figure 4.1. This context model was adapted from (Chihani et al, 2013). There are other approaches to modeling context, but most of such models are domain dependent (Bettini et al, 2010).

The graph context model approach is adopted in this study because of its flexibility; more nodes and edges can easily be added to the graph without having to write expressive schemas. Generally, context data are interrelated or define a form of connection between an entity and a state, location or activity. Taking advantage of the graph as an intuitive way of representing connected data, device context represented here as a graph made up of atomic and non-atomic nodes connected by arcs. Where atomic and non-atomic nodes are entities (mobile

device, service) and context information (battery level, available memory) respectively and maps each entity to its context data.

From Figure 4.1, a mobile device (entity) is always associated with a mobile profile (which are the unique features it supports) and resource context, which captures the state of its battery and memory at any point in time. There are a number of standards for representing device capabilities, for example, Composite Capabilities/Preference Profiles (CC/PP) as defined by the World Wide Web Consortium (W3C), User Agent Profile (UAProf), and the Microsoft-based Mobile Device Profile (MDP). All these standards simply represent the same specific device features in different ways.

For a relevant Web Service to be discovered, Web Service capabilities are matched with device context during filtering or matchmaking. In order to facilitate this kind of discovery process, each Web Service is described in a manner that indicates a number of supported device features as well as resource usage implication of the Web Service.

Considering the fact that advances in mobile device technology might introduce new capabilities, the context model described in Figure 4.1 is non-exhaustive. That is, the list of device features and resources is extendable.

4.3.2 Device Context-based Service Description

To achieve a balance between expressiveness and resource-intensiveness in service description, WSDL-M (Al-Masri & Mahmoud, 2010) is adopted in this study. The choice of WSDL-M was based on the fact that it provides rich expressivity in a manner that meets the lightweight resource requirements of mobile devices without involving semantics. This feature of WSDL-M satisfies this work's design criteria. Most importantly, WSDL-M

specifies a service description in a way that does not only specify how to interface with a service, but also includes any recommended to-have device features.

The approach proposed in this study, therefore, takes advantage of the capability of WSDL-M to incorporate device context into service descriptions. Utilising device context has the advantage of offering more expanded service descriptions that cover more non-functional parameters.

The CaSDiM service description, which is based on the design criteria for this study, is presented in this section. Based on the device context model in Figure 4.1, the CaSDiM service description is basically structured into two blocks or sections:

i. Service functionality (SF): describes the basic functionality of a Web Service (the Web Service name, what the Web Service does, and how to invoke it) as well as the operation which implements the specific functionality. Technically, the functionality of a Web Service is represented by a pair of its Action and the Object of the action (Shin et al, 2009): SF = <Action, Object>.
ii. Service capabilities (SC): specifies the explicit device features that are supported by the Web Service (the recommended to-have features) and the Web Service's resource capability (which indicates the resource consumption level of the Web Service) as discussed in subsection 4.3.1.

Some examples of commonly used device features as described by Al-Masri & Mahmoud, (2010) and the resources profile as considered in the proposed context model of Figure 4.1 are given in Table 4.1.

Table 4.1: Common Device Profile Capabilities and resources, adapted from (Al-Masri & Mahmoud, 2010)

Device profile		
#	Device Feature	Attribute Value
1	ScreenPixelsWidth	320dp
2	MaximumHrefLength	2083 Char
3	ScreenPixelsHeight	470dp
4	Supported browser	Explorer, Ericsson, Chrome
5	InputChaSet	ISO-8859-1, US-ASCII
6	InputType	TextInput, VoiceInput
7	CanInitiateVoiceCall	Yes/no
8	SupportsImageSubmit	Yes/no
9	HasBackButton	Yes/no
10	SupportsSelectMultiple	Yes/no
11	SupportsCSS	√
12	SupportsBold	√
13	IsMobileDevice	√
14	SupportsFontColor	√
15	Frames	√
16	SupportsItalic	√
17	SupportsBodyColor	√
18	Cookies	√
19	SupportsCallback	√
20	BackgroundSounds	√
21	SupportsXmlHttp	√
22	Tables	√
23	MaximumRenderedPageSize	Integer value
Resource profile		
#	Device resource	Attribute value
1	Battery	High (2), Moderate (1), Low (0)
2	Memory	High (2), Moderate (1), Low (0)

The above service description approach enables service providers to create expanded Web Service descriptions that allow them to not only define the functionality of their services but

also to specify the device features supported. Providers can also include the impact each service will have on limited resources. For example, they can indicate the battery consumption level and the memory requirement of the Web Service. From the device context section of Table 4.1, numeric scales (0, 1, and 2) are used to indicate the impact a particular Web Service may have on the battery and memory of a client device. For instance, a Web Service "A" with minimal battery consumption effect and high memory requirement will have the values 0 and 2 in the "battery effect" and "memory requirement" tag of its service description. With this information built into the Web Service description it becomes possible for Web Service capabilities to be matched with the requested capabilities during matchmaking in order to discover relevant services.

Using the resource-friendly design criteria discussed in section 4.2 that employ device context as illustrated in Figure 4.1, the CaSDiM service description is realised as depicted in Table 4.2.

Table 4.2: The CaSDiM Service Description

<wsdl: description

Xmlns: wsdl="http://sampleWeather.com/ns/wsdl"

targetNamespace= "http://sampleWeather.com/WeatherInfor.wsdl"

xmlns:tns= "http://sampleWeather.com/ns/WeatherInfor/wsdl"

xmlns:whttp= "http://sampleWeather.com/ns/wsdl/http"

xmlns:wsdlx="http://sampleWeather.com/ns/wsdl-extensions"

xmlns:xs="http://sampleWeather.com/2001/XMLSchema-extensions"

xmlns:msg="http://sampleWeather.com/2001/WeatherInfor/xsd">

<wsdl: documentation>

Web Service functionality section :

</wsdl: documentation>

<wsdl: types>

```
        <xs: import
namespace= “http://sampleWeather.com/WeatherInfor/xsd”
        schemaLocation= “WeatherInfor.xsd”>
    </wsdl: types>
    <wsdl: interface name= “WeatherInforInterface”
        <wsdl: operation name=“getWeatherInfor”
                pattern=“sampleWeather.com/ns/wsdl/in-out”
                style=“sampleWeather.com/ns/wsdl/style/iri”
                Wsdlx: safe=“true”
        <wsdl: input element=“msg:getWeatherInfor”/>
        <wsdl: output=“WeatherInfor”/>
    </wsdl: operation>
    <wsdl: binding name=“WeatherInforHTTPBinding”
        type=“http://sampleWeather.com/ns/wsdl/http”
        interface=“tns:WeathetInforInterface”>
        <wsdl: operation ref=“tns:getWeatherInfor”
           whtt: methos=“GET”/>
    </wsdl:binding>
    <wsdl: service name=“WeatherInfor”
        interface=“tns.WeatherInforInterface”>
    <wsdl: endpoint name=“WeatherInforHTTPEndpoint”
        binding=“WeatherInforHTTPBinding”
        address=“sampleWeather.com”
    </wsdl: endpoint>
    <wsdl: documentation
        Web Service Capabilities section
    </wsdl: documentation>
    <wsdl service name=“WeatherInfor”>
        <wsdl-m>
```

```
            <wsdl-m: hardware>
                <ScreenSize>
                    <max>176x220</max>
                    <min>101x80</min>
                </ScreenSize>
                <input CharSet>
                    <i>US-ASCII</i>
                    <ii>ISO-8859-6</ii>
                </input CharSet>
        </wsdl-m: hardware>
        <wsdl-m: browser>
                <supported browsers>
                    <i>navigator</i>
                    <ii>firefox</ii>
                    <ii>ericsson</iii>
                </supported browsers>
                <supportTables>No</supportTables>
                <supportFrames>Yes</supportFrames>
                <maximumHrefLength>2083</maximumHrefLength>
        </wsdl-m: browser>
        <wsdl-m: resource capability>
                <batteryEffect>0</ batteryEffect>
                <memoryRequirement>2</memoryRequirement>
        </wsdl-m: resource capability>
      </wsdl-m: service>
  </wsdl: service>
</wsdl: description>
```

4.4 Device Context-based Service Discovery and Ranking Algorithm

In an earlier discussion in section 3.2, it was argued that resource-intensiveness can be introduced either at the service description or matchmaking levels or both in a service discovery process, and that such resource-intensiveness is largely attributed to the use of semantics. This research work addresses the above issue at the service description level by adopting the WSDL-M in order to avoid the use of semantics. While at the discovery or matchmaking level, this study formulates a discovery mechanism that incorporates device context as described in section 4.3. The mechanism is composed of two operations: i) keyword-based matching and ii) Web Service ranking.

4.4.1 Proposed Keyword-based Matchmaking Algorithm

Generally, service requests are processed through a matchmaking process that determines which service or services to return to the requester. In this work the keyword-based matching is adopted to minimise resource consumption as against resource-intensive semantic methods. Also, device context is used in the discovery process to introduce proactive capability to the discovery mechanism. Utilising device context is meant to help adapt service requests in response to context change. More significantly, as highlighted in chapter 3, with device context information, the discovery mechanism is able to adapt service requests in response to context change. Effectively, adapting service requests based on context helps to constrain retrieved services to only relevant ones. Consequently, the number of returned services is reduced, thereby reducing processing time and, by implication, resource burden.

The keyword matching algorithm proposed in this study is given in Table 4.3 as adapted from (Zhao et al, 2012). This algorithm is based on the assumption that Web Services are named using English words or phrases.

It is also assumed that mobile providers constitute small computing communities like SMMEs with common needs.

Table 4.3: Keyword Matching Algorithm, originally from (Zhao et al, 2012)

Algorithm 1:

Input: a keyword (w), device profile (battery level, available memory)

Output: list of Web Services matching the keyword (L)

Take a keyword input w Ɛ W, where W is a set of available Web Services provided

Form a set N of the synonyms of the keyword entered

Foreach n_i Ɛ N

    Retrieve n if (name match keyword or phrase)
    Store the retrieved Web Services into set R

End

Foreach r Ɛ R

    Filter out Web Services that do not meet battery & memory requirement “x”, “y” respectively
    Store remaining Web Services into set T

End

//Calculate the semantic distance of the relation between n and r

Foreach n_i Ɛ N

    Foreach r_i Ɛ R

$$d(n_i, r_j) = \sum_{k=1}^{|T|} \left(p\left(t_k \mid s_i \right) x \log \frac{p(t_k \mid s_i)}{p(t_k \mid r_j)} \right) \quad \ldots\ldots\ldots\ldots (1)$$

    Fnd

    add r to list of returned Web Services (L) if r is closely related to n

End

Therefore, each mobile device provides Web Services that are related to the common needs of the community to which they belong. With such an assumption, the number of Web Services provided is expected not to be too large, hence a simple keyword matching can suffice, as against using complex semantic techniques.

The matchmaking process of the algorithm in Table 4.3 relies on equation 1, which measures the semantic distance between the synonyms of an input keyword in a service request and the associated Web Service name. The idea is that the smaller the distance is the closer the relation between the inputted keyword to the corresponding Web Service. Therefore, the algorithm will return all Web Services that are closely related to the keyword used in the service request.

4.4.2 Proposed Web Service Relevancy Ranking Algorithm

The purpose of ranking retrieved Web Services is to measure the degree of their relevance to a client. From the findings in section 3.4, it was reported that current AMC service discovery mechanisms mostly adopt the user-centred approach to determining the relevance of Web Services. Such an approach disregards or only partly considers the device-centred dimension while focusing mainly on the Web Service functionality. However, it is better to consider a service to be relevant if it both satisfies the required functionality and matches the resource capability of the client device. The problem with the first approach is that it is possible for a Web Service to satisfy a requester's requirement but not match the resource capability of the client's device. If a service does not match the resource capability of a client device then, in the context of AMC with constrained resources, the service is not relevant.

The service ranking procedure implemented in this study, as depicted in Table 4.4 also utilises device context. The idea is to rank retrieved Web Services according to the order of their relevance based on the current device context. To realise this ranking, this study introduces two relevancy weighting parameters:

i. Resource weight (R_w) the value of which depends on the resource profile (battery level, available memory) and
ii. Features weight (F_w), which is the total number of supported device features.

Let there be:

a. a set of Web Services W = {w_1, w_2, w_3, …, w_i}, where w_i Ɛ L is generated by the keyword matching algorithm in Table 4.3;

b. a set of Web Service supported features C = {c_1, c_2, c_3, …, c_k};

c. a set of device resource profile P = {x, y}, and

d. a set of client's device supported features D = {d_1, d_2, d_3, …, d_i}. Then based on the context model offered in subsection 4.3.1, for each retrieved Web Service w_i there are corresponding Web Service supported features d_i as illustrated by the matrix in equation (2):

$$W_i = \begin{vmatrix} w_1c_1 & w_1c_2 & w_1c_3 & \cdots & w_1c_k \\ w_2c_1 & w_2c_2 & w_2c_3 & \cdots & w_2c_k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_ic_1 & w_ic_2 & w_ic_3 & \cdots & w_ic_k \end{vmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

where rows represent individual Web Services and columns represent the set of corresponding distinct device features. The same matrix can also be derived for the device contexts.

The proposed ranking algorithm relies on a resource weight function $f_{rw}(\ldots)$ and a features weight $f_{fw}(\ldots)$ of equations (3) and (4) respectively to manipulate the aforementioned parameters extracted from the mobile client device and Web Service descriptions.

$$R_wW_i = f_{rw}(W, P) = \sum_{i=1}^{n} p_i \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

$$F_wW_i = f_{fw}(W, C, D, M) = M * \sum_{i=1}^{n} f(c_i, d_i) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Equation (3) returns a maximum score of 2 against a Web Service with a resource profile pair of {0, 0} and a score of zero if {1, 1} and so forth. This rating is meant to allow for the ranking of Web Services based on device context. From equation (4) the features weight function (f_{fw}) calls an Object Relationship Function, f in equation (5), which computes the relation between two objects using a an arithmetic operation and the Normalized Google Distance – NGD (Cilibrasi & Vitanyi, 2007).

$$\int(x,y) = \begin{cases} \frac{y}{x+y} & if\ x,y: numbers \\ sim(x,y)\ if\ x,y: strings \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

where the function f(a, b) generates values in the range k Ɛ [0...1], where k ≥ 0.5 implies that x and y are related (the closer k to 1 the stronger the relationship) while k < 0.5 means they are not related. Another distinctive idea of the proposed ranking approach is the introduction of a Minimising Factor (M) in equation (4). The Minimising Factor is meant to retain the resource weight as a dominant score in the rating processes, that is, to avoid the chances of a less resource efficient Web Service but with more supported features being ranked best at the expense of a more resource efficient service. To achieve this, a default value of 0.05 is assigned to the Minimising Factor. This implies that a less resource efficient Web Service must support at least 20 more features for it to be ranked better than a resource efficient one. However, it is assumed that no single Web Service will support up to 20 device features, which makes the above condition infeasible.

The features weight (F_w) of each Web Service is then calculated from equation (4).

Table 4.4: Context-aware Relevancy Ranking Algorithm

Algorithm 2:

Input: Set of Web Services, W (generated from equation 1);
Set of resource profiles, P;
Set of Web Service supported features, C;
Set of device supported features, D; and Minimising factor, M

Output: Resource-aware ranked list of Web Services (RaRnk)

1 Initialize RaRnK

2 // Compute resource and features weight for each Web Service using equation (3) and (4)

3 **foreach** w_i in W do

4 Compute resource weight ($R_w w_i$): f_{rw} (W, P) ……(4)

5 Compute features weight: f_{fw} (W, C, D, M)……...……(5)

6 **End**

7 // Rank Web Services by device context but prioritising resource profile

8 Initialize RelScore, RankedList

8 **foreach** w_i Ɛ W do

9 $RelScore_{wi} = (R_w w_i + F_W w_i)$……..… (6)

10 // Sort Web Services in descending order of their resource score

11 RankedList = SortedList (w_i)

12 **End**

13 //Display sorted list of retrieved Web Services

14 Return RankedList

4.5 The CaSDiM Architecture and Components Description

This research work proposes a service discovery model that facilitates the usage of device context in relevant Web Service discovery in Ad-hoc Mobile Cloud depicted in Figure 4.2. The architecture is divided into two layers: the context generation layer and the interaction and processing layer. These two layers consist of three main components with modules that run in them or within the layer. The three main components are the node monitor, a lightweight database, and the discovery engine.

4.5.1 Context Generation and Storage Layer

As discussed in section 4.3, an important consideration for the proposed model is the use of static context (device profile) and dynamic context (resource profile). The context generation and storage layer is made up of components that provide or store relevant context data. There are two components in this layer – node monitor and lightweight database.

4.5.1.1 Node Monitor (NoM) Component

Due to the constrained and dynamic nature of mobile device resources, which can impede relevant service discovery in AMC, the need to monitor mobile nodes during service discovery was considered in section 1.2 as an imperative issue to be addressed by this research work. The node monitor component is responsible for monitoring device context and providing the relevant data used for the service discovery process.

There are two modules that run within the node monitor component:

a. **Device Resource Monitor Module**: this module monitors the dynamic context (which is referred to as resource profile in this work) of a mobile device. For example,

whenever a service request is launched the device resource monitor collects information about the current battery level and the available memory of the requesting device. This is achieved by using suitable Android utility functions to extract relevant device information. This work utilises the public static long FreeMemory() and public static float getBatteryLevel() methods respectively, manipulated to generate resource context (battery level, and available memory). The context data extracted by this module are used in the Web Service discovery and ranking processes as discussed in subsection 4.4.

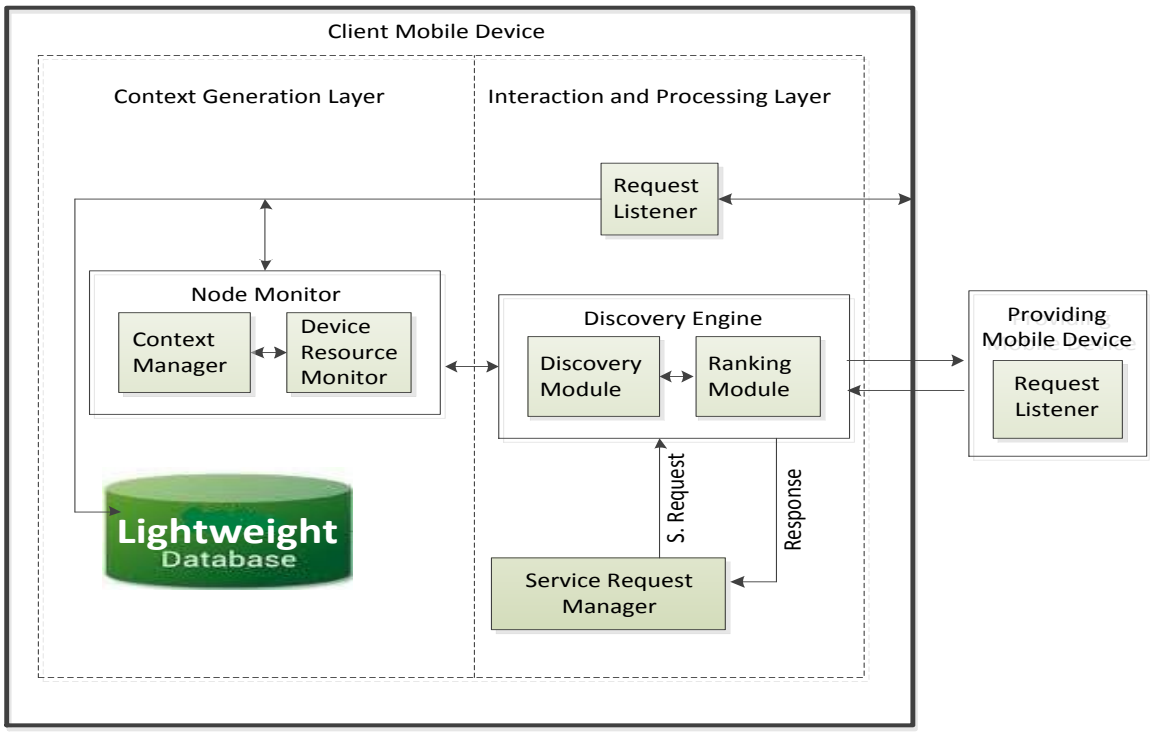


Figure 4.2: Conceptual CaSDiM Architecture

b. Context Manager Module: This is designed to make context data available for use. To achieve this, the context manager module interacts with the device resource monitor and the lightweight database. During this interaction the context manager retrieves dynamic context and static device context (device profile) from the device resource monitor and lightweight database respectively.

4.5.1.2 Lightweight Database Component

A lightweight database is an embedded, self-contained library with no server component, no need for administration, a small code footprint, and limited resource requirements**.** Such databases fit resource-constrained environments like AMC. There are several lightweight databases, such as BerkeleyDB, Couchbase Lite, LevelDB, SQLite, UnQLite. This work adopts SQLite because it supports a wider range of mobile phones than the rest (Kreibich, 2010). SQLite is an open-source C-library for managing relational databases that can be stored both on memory and disk and supports dynamic typing. Each database created in SQLite is stored as a single disk file in a cross-platform format. That is, any SQLite database created on one machine can be used on another machine with an entirely different architecture. Static context (e.g. device profile or any other relevant context information that can be useful for service discovery) is stored in the SQLite database. Most importantly, the RESTful Web Services deployed in AMC will be hosted on this lightweight Server, as explored in the work of Wagh & Thool, (2013) and Srirama et al, (2006).

Because this work focuses on mobile Web Service discovery not implemetation, the SQLite database holds Web Service description documents as references to their actual implementations. This is in conformity with the conventional Web Service registry, which maintains pointers to Web Service descriptions and to the actual service.

4.5.2 Interaction and Processing Layer

This layer consists of the central component, i.e. – the discovery engine and modules that facilitate both internal and external interactions. Internal interaction happens between a services requester and their mobile device, while external interaction refers to the communication between a client device and a providing device. The first interaction occurs

when a mobile user initiates a service request, but the process of a client device requesting a service from a provider is termed external interaction.

4.5.2.1 Service Request Manager Module

With the help of the service request manager, a service request is constructed and subsequently used by the client device to query a nearby AMC node for a required Web Service. Basically, the service request manager performs three essential roles. First it provides user interface that allows users to enter a word or phrase representing the name of a desired Web Service as discussed in sub section 4.4. Second, it constructs an XML or JSON string/message using the word or phrase entered by requesters and forwards the request to the discovery engine for processing. (JSON string is the data format supported by Wi-Fi Direct, the technology employed in this work for inter-node connection and communication). Third, the service request manager displays a list of relevant services returned by the discovery engine.

4.5.2.2 Discovery Engine (DiEn) Component

The discovery engine plays the central role in the proposed service discovery mechanism. Basically, DiEn performs the functions of discovery and ranking of relevant Web Services through the adaptation/discovery and ranking of modules. In order to function, the adaptation engine interacts with the node monitor component to obtain context data.

a. **Service Discovery Module**: this module is responsible for adapting service requests and discovering relevant Web Services. It receives Web Service requests from the request manager and incorporates device context data (battery level, available memory) extracted from the node monitor component as additional request parameters. The process of expanding the original service request by adding device

context is termed service request adaptation. The adapted service request is used to retrieve relevant Web Services from a provider device.

The discovery module relies on the keyword Web Service matching algorithm discussed in subsection 4.4.

b. **Service Ranking Module**: with the CaSDiM model, Web Services are first discovered or retrieved and then ranked according to their relevance to the current context of the client device. This ranking is done by the ranking module. The ranking algorithm described in subsection 4.4 is implemented by the ranking module. Then, the ranked list of Web Services is returned to the service request manager.

4.5.2.3 Service Request Listener Module

With a network through which mobile peers in AMC communicate, the service request listener module listens and responds to incoming service requests from clients. Such interaction, which represents sending and receiving service requests, is mediated by the service request listener module. In addition to intercepting service requests from potential clients, the request listener directly communicates with the lightweight database to access hosted Web Services. Each AMC node has a request listener module. However, the module is only in active mode from the service provider's side. This design is aimed at avoiding redundancy, which can lead to energy waste. With the envisioned AMC scenario being an opportunistic setup as described in chapter one, it is assumed that participating devices will not have to stay running continuously. Hence, the additional resource burden can be minimal. More so, in a practical scenario, providing devices can take advantage of the Opportunistic Power Save protocol (OPS) and the Notice of Absence protocol (NOA) offered by Wi-Fi Direct technology. These protocols are energy-saving mechanisms.

4.6 Chapter Summary

This chapter conceptualised the solution approach of this study. Overall, the chapter aimed to formulate a service discovery mechanism or model for discovering relevant Web Services in AMC. The mechanism was formulated in a manner that utilises device context while taking into account resource constraints. First, a resource-friendly Web Service description approach was proposed, leading to the formulation of the device context model, which illustrates how device context may be incorporated into Web Service descriptions. To support a lightweight approach to discover relevant service in AMC based on device context, the chapter also presented the keyword-based matching and Web Service ranking algorithms. The proposed integrated architecture of CaSDiM was also presented, showing the various components that implement the concepts modeled in this chapter.

CHAPTER FIVE

PROTOTYPE IMPLEMENTATION AND EVALUATION

5.1 Introduction

The previous chapter presented the conceptual framework of CaSDiM and its integrated architecture. The idea centres around using device context to enhance the discovery of services in a manner that minimises resource consumption in the Ad-hoc Mobile Cloud. Based on the conceptual framework formulated in chapter 4, a proof-of-concept prototype is needed to evaluate the proposed solution approach. Therefore, this chapter focuses on the design, implementation, and evaluation of the proposed model. For the most part, technical procedures and concepts that support the realisation of the proposed solution are shown. These procedures and concepts help to clarify issues pertaining to the choice of appropriate implementation platform and other technologies that may be required. Furthermore, the actual implementation of the proposed model utilises information exposed by these techniques and concepts in the form of functional systems requirements. Most importantly this chapter seeks to produce empirical evidence that is used to measure the performance of the proposed model.

Section 5.2 presents the design of a CaSDiM prototype, consisting of a usage scenario and some UML diagrams. CaSDiM implementation details and the assumptions that were made in this study are briefly highlighted in section 5.3 and 5.4 respectively. The CaSDiM prototype is then evaluated in section 4.5 and the chapter summarised in section 5.6.

5.2 The Design of a CaSDiM Prototype

This section presents a theoretical application scenario of the solution approach of this study followed by the technical design of the CaSDiM implementation in the form of UML diagrams.

5.2.1 Application Scenario of CaSDiM

For the purpose of clearer understanding, this section paints an application scenario of the proposed model. First, it is considered that there is a traditional Mobile Cloud Scenario where John, a local trader has a mobile device "A" which he uses to access Cloud services. However, because of erratic Internet connectivity in the rural area where John does business, he cannot connect to the Cloud server in the morning to conduct the day's business.

The above scenario assumes that there are some local traders in a small trading community that have mobile devices which have the capacity to host Web Services that are tailored for the needs of the computing community. Assuming also that such devices can utilise Wi-Fi Direct technology or any other wireless network platform with the capability to create a P2P without going through an access point to form Ad-hoc Mobile Cloud within the vicinity of John, and that his device has the required configuration to connect to it, then John's challenge is partly solved. Now John's device can discover a nearby node in the Ad-hoc Mobile Cloud and query it for the service it needs. We also assume that the nearby Ad-hoc node hosts multiple Web Services, $\{W_1, W_2, W_3, \ldots, W_i\}$ of the same functionality but having different resource requirements (battery and memory) and supporting different device features. For example, W_2 may be efficient in battery and memory (i.e., consumes less battery and memory) and supports most of John's device's features compared to W_1. On the other hand, W_1 is only efficient in memory but less efficient in battery and supports fewer of John's

device's features. The other Web Services are neither efficient in battery nor memory. In terms of relevancy ranking, W_1 is more relevant than W_2.

(a) Scenario 1: For the day's business, John needs an advertisement service for his local craft. His device battery is 80% charged and has 75% available memory. John opens the CaSDiM search engine installed on his device which helps him to automatically discover provider devices within the vicinity. Through the service request interface John sends a service request to a selected host. As soon as John initiates the service request, CaSDiM extracts John's device's context (battery level and available memory) and uses it as the "search-and-filter" condition for the request. By this operation CaSDiM's Discovery Engine retrieves discovered services and returns them to John, sorted in the order of their relevance to John's device's current capabilities (i.e., W_2 first followed by W_1). The rest of the services are left out though they may match John's search keyword because they are neither efficient in battery nor memory. For the purpose of flexibility, John is allowed at this point to invoke either of the services based on his preference because his device's resources are not yet depleted.

(b) Scenario 2: While John is using his device for other routine tasks, the device's battery level and available memory drops to 50% and 60% respectively (meaning that its device resource contexts has changed). Now, device "A" needs to optimise its resource usage to be able to provide services to others and perhaps still enable John to invoke another Web Service. As John sends another service request, the new state of his device resource context is used as the priority filtering parameter to discover services. This time, device "A" is low in battery and memory. Now, assuming John still wants to invoke the same advertisement service, this time his service request only returns W_2 (the optimally efficient service) since WS_1 is only efficient in memory and not in battery. Ultimately, at any service request, John

gets a Web Service or services that both meet his need and match the current resource capabilities of his device.

The discussion on the AMC computing paradigm as offered in chapter 1 provides the framework upon which the above application scenario works. Therefore, Figure 5.1 depicts an architectural demonstration of AMC, which consist of mobile devices that run the CaSDiM prototype.

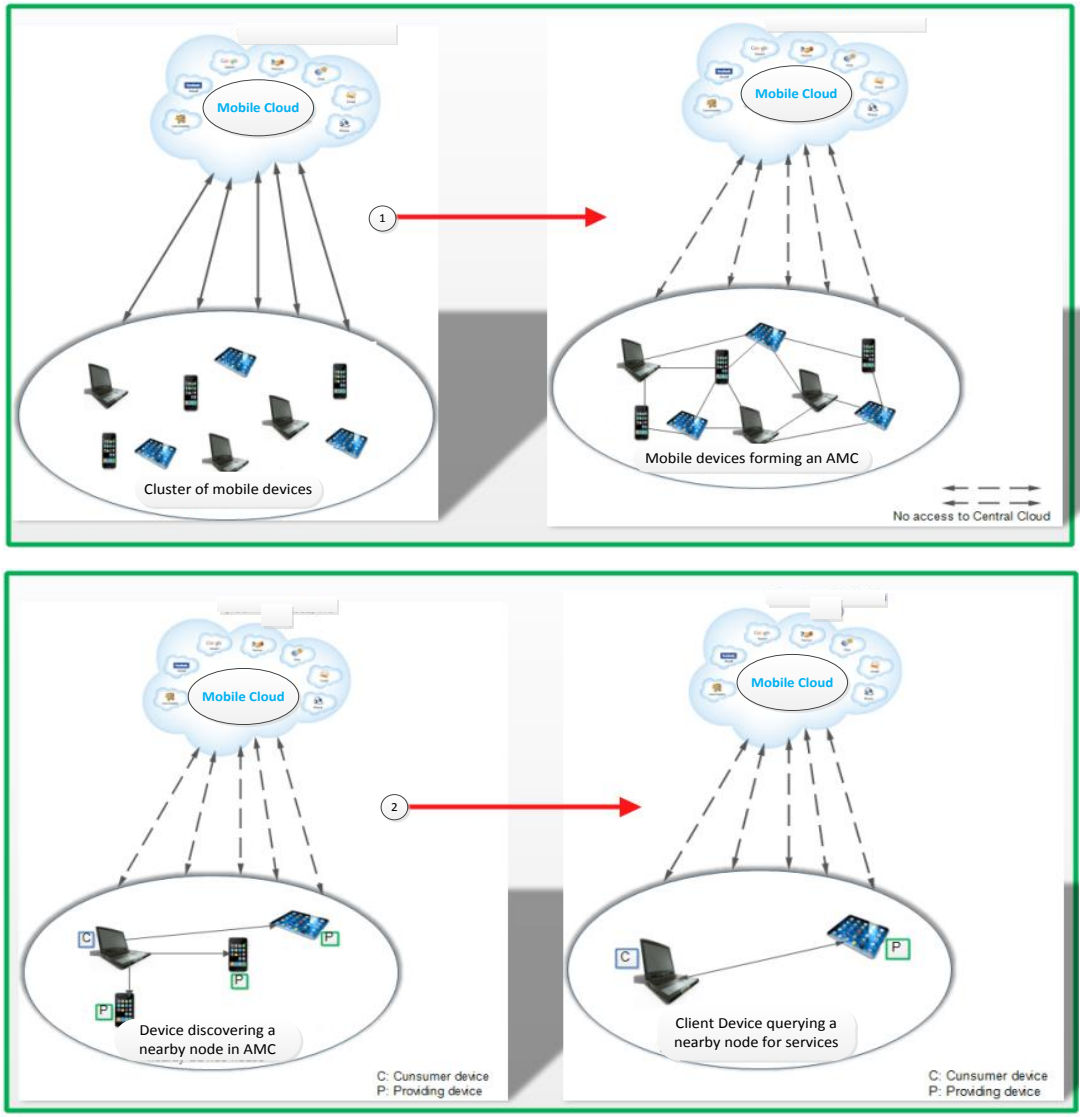


Figure 5.1: AMC Formation, Node and Service Discovery

5.2.2 The CaSDiM Use Case

The use case of CaSDiM prototype as implemented in the work is shown Figure 5.2. There are four actors in this use case, namely, the Services Request Manager (SeRM), the Discovery Engine (DiEn), the Node Monitor (NoM), and the Request Listener (ReLn).

Figure 5.2: CaSDiM Use Case Diagram

CaSDiM has one major use case which is a request for Web Service. This request is forwarded to the discovery engine. In this use case, there are two main sub-tasks, namely, a) get device context – interacts with the node monitor to extract the battery level and available memory of the client device to be used for request adaptation, and b) execute service request – the discovery process in CaSDiM is aimed at minimising resource consumption and

enhancing retrieval of relevant service. So, the "execute service request" task has two sub-tasks. The first sub-task adapts service requests with the obtained device context, while the second performs the actual service request, and retrieves and ranks relevant services.

A. The Service Request Manager Actor

In the interaction and processing layer, the service request manager (SeRM) offers an interface for clients to perform service requests. SeRM is also responsible for displaying the final list of filtered and ranked services returned by the discovery engine. Basically, SeRM uses the keyword or phrase entered by a client to construct a service request message (JSON string) which is then forwarded to the discovery engine.

B. The Discovery Engine Actor

Another use case actor is the discovery engine (DiEn), which is a Java method that invokes native Java Sockets (java.net.ServerSocket and java.net.Socket) to initiate interaction between peer nodes for the purpose of performing a search for provided services. As a central component in the CaSDiM model, the discovery engine is responsible for adapting initial service requests using device context information obtained from the node monitor. The adapted service request is then forwarded to the providing Ad-hoc node.

After receiving a response from the provider, DiEn also performs the ranking of relevant services before returning the list of services back to the service request manager. Therefore, the keyword-based service matching and relevancy ranking algorithms proposed in section 4.4 are executed by the discovery engine.

C. The Node Monitor Actor

The duty of this system actor is to monitor the battery and memory of the client device and make such device context data available to the discovery engine whenever a service request is

made. This role is critical to the proactive service discovery approach adopted for the CaSDiM prototype which requires the discovery mechanism to keep track of changing device context in order to discover relevant services.

The node monitor utilises Java utility methods offered by Android resource manager under the class “public class util()”. For example, public static long FreeMemory() and public static float getBatteryLevel() methods are respectively used to exstract resource context (battery level, and available memory) from the client device.

D. Request Listener Actor

This actor only becomes active when a device is serving as a provider. Request listener listens to incoming services requests and interacts with the lightweight database to retrieve requested services for the client’s device. It also performs the role of interpreting service requests from clients encoded in the form of JSON string/message. The Wi-Fi Direct technology utilises JSON as the data format for transmitting messages between Android devices. This format produces relatively small message size suitable for web socket protocols, intended to be used with small payloads. Fundamentally, the function of this system actor ensures that communication between the client and host device is sustained throughout the discovery process by utilising native Java Sockets.

5.2.3 The CaSDiM Sequence Diagram

The diagrammatic illustration in Figure 5.3 shows the sequence of the flow of messages among the actors and components of CaSDiM. This process is initiated when a client launches a service request through the service request manager.

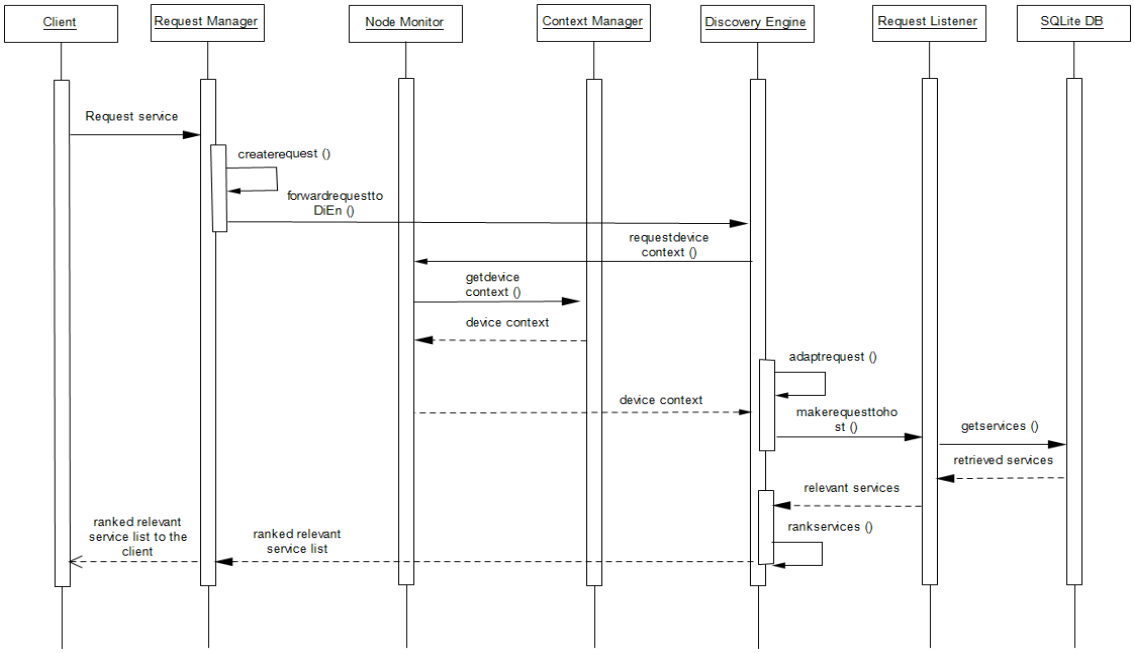


Figure 5.3: The CaSDiM Sequence Diagram

When a client enters a keyword or phrase (representing a Web Service name) into the CASDiM request interface and clicks send, the request manager transforms the request to a JSON string which is then forwarded to the discovery engine. The flow of messages that involves the rest of the system actors begins from here. When the discovery engine receives a request string, it queries the node monitor for the device's context while the node monitor extracts the requested context from the context manager. Because device context is dynamic, the node monitor does not hold context information but only queries for it at service request time in order to avoid the need to perform regular updates which may have resource implications. With this information, the discovery engine performs a service request adaptation before placing an actual request to the host device via the request listener. The request listener maintains a communication link with the client device while it queries the host's SQLite database to retrieve and return services to the discovery engine. Returned services are then ranked by the discovery engine and the ranked list is returned to the client through the request manager.

5.2.4 The CaSDiM Activity Diagram

The bulk of the activities that take place in the CaSDiM relevant service discovery process are centred around the decision-making component, also called the discovery engine. Figure 5.4 shows CaSDiM's activity diagram with the discovery engine acting as the dominant actor.

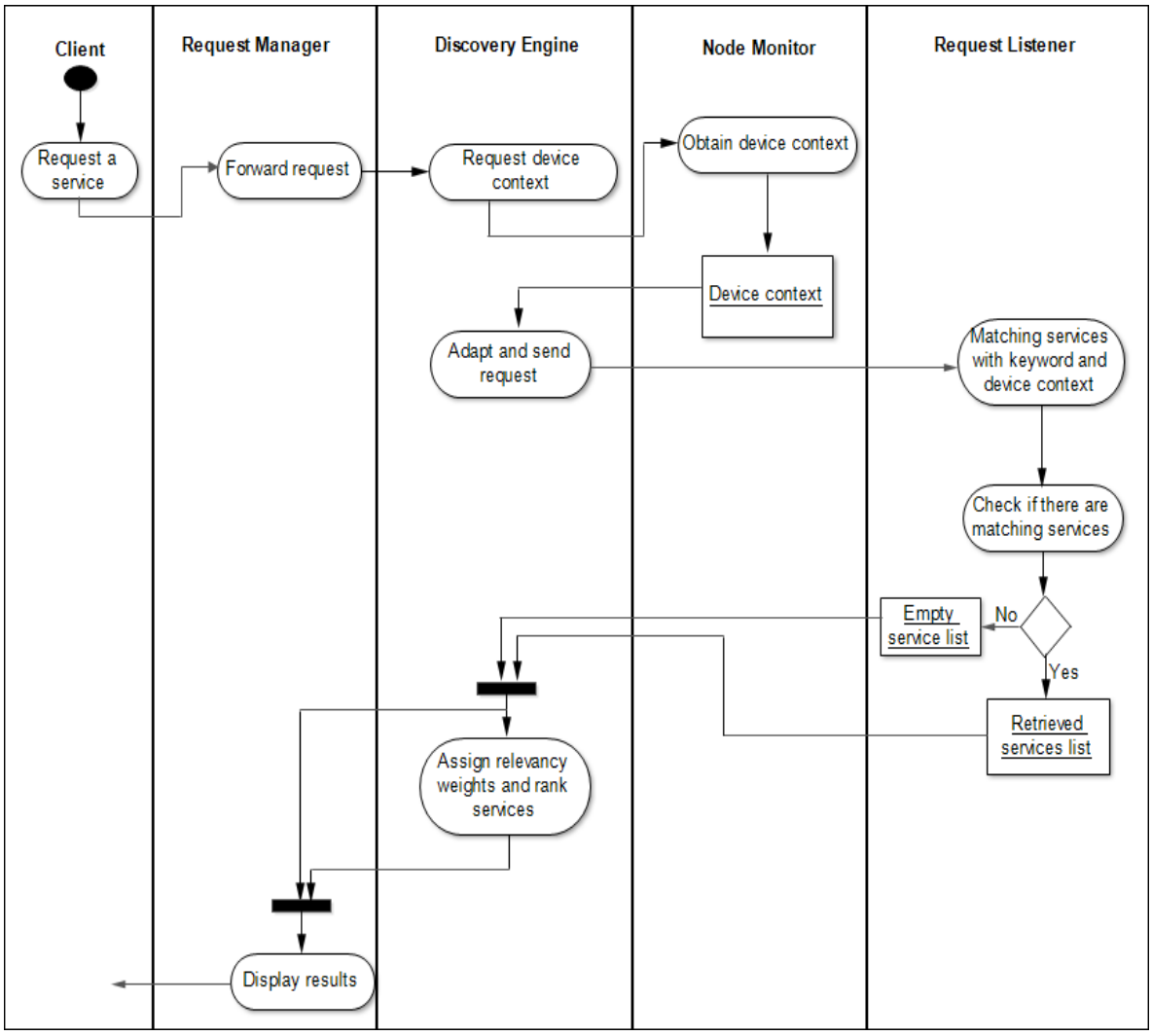


Figure 5.4: The CaSDiM Activity Diagram

The process starts with the discovery engine receiving a client request message from the request manager. This triggers the discovery engine to prompt the node monitor to provide it with the device context information of the client device. With the context information, the

discovery engine performs service request adaptation and then proceeds to place an actual service request to the providing node through the request listener. The request listener queries the SQLite database for services based on the keyword and the context information it received. If there awere services that match the request conditions the request listener returns either a list of relevant services or an empty list. The discovery engine executes a relevancy weighting algorithm to determine the relevancy score of each retrieved service and ranks the services according to their relevance. This ranked list of relevant services is finally returned to the client through the request manager.

5.3 CaSDiM Implementation Description

The CaSDiM Engine (prototype) was developed for Android platforms using Android SDK version 21 integrated with Eclipse Juno 4.2 and coded in Java language for ICS 4.1 and above. This choice of Android was informed mainly by the fact that Android OS has much larger market share of consumer mobile devices today, followed by iOS (Butler, 2011).

By design, CaSDiM runs as a service at the background of the providing device in order for the device to be discoverable to other client devices within its vicinity. For inter-node connection and communication CaSDiM utilises the Wi-Fi Direct technology with the capability to create a P2P ah-hoc network without having to go through an access point that is, a router is not required.

With this technology, the hosting device automatically becomes a sort of hotspot provider. Utilising native Java Sockets (java.net.ServerSocket and java.net.Socket), the hosting device can receive, process, and sustain an incoming connection instance (an instance of java.net.Socket pointing to the url of the server device) that is initiated by a client device.

Also, to facilitate CaSDiM operation, a Web Service description document directory is created on SQLite database, which is embedded in Android platforms. This directory is used to hold Web Service description documents obtained from online Web Service directories, for example WebserviceList, WebserviceX and XMethods. In order to effectively test the proposed service discovery approach, these web description files are first extended based on the WSDL-M standard as discussed in section 4.3. This modification of Web Service description documents follows the work of Al-Masri & Mahmoud, (2010) and Elgazzar, Martin, & Hassanein, (2013b).

CASDiM's matching module measures the similarity between the request and Web Service description using a keyword match scheme by first extracting keywords (Web Service names) and their hardcoded capabilities from service description. In the Web Service description file each Web Service is marked with the tag <wsdl: documentation>.

SQLite is a lightweight relational database that normally sits in a file; does not have the client-server architecture. Interestingly too, this relational database also offers helper libraries for connecting to the database. Its lightweight nature and flexibility make it an ideal database for resource-constrained devices.

In the experimental setup, five android mobile devices (the client discovering four providers) with CaSDiM engine installed as shown in Figure 5.5 could discover themselves as peer nodes forming an Ad-hoc Cloud as depicted in Figure 5.6. The third and fourth screenshots depicted in Figure 5.7 shows a host device with the services it provides as well as a client device's service request interface.

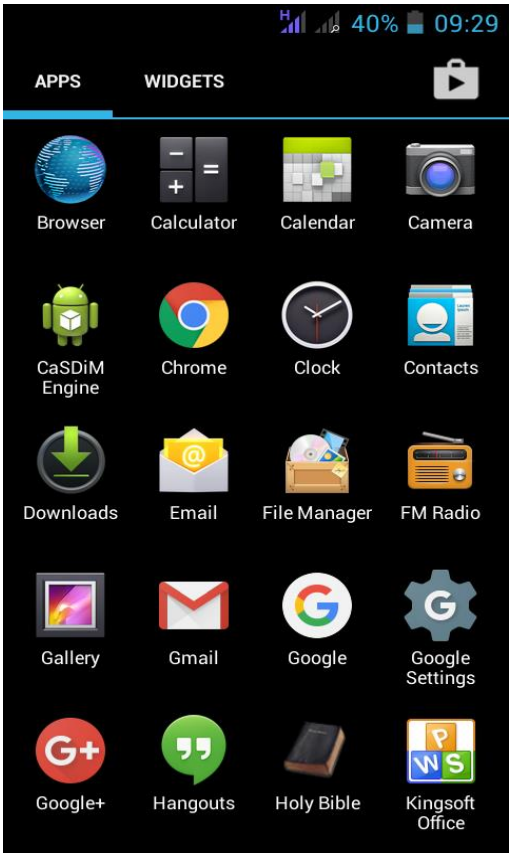


Figure 5.5: CaSDiM Prototype Installation Screenshot

For evaluation purposes, the service discovery was performed from the client device to any of the discovered providers: 1) Hisense HS-U939 cell phone, with internal and external memory of 1.24GB and 1.27GB respectively and running Android 4.2.2, and 2) HP-Slat 7 HD tablet, with an internal memory of 12.15GB. The Hisense HS=U939 device served as the providing device while the HP-Slat 7 HD device acted as a client. The preference for the HP-Slat 7 HD device as the client was based purely on its screen size, which provides a better view of the experimental results.

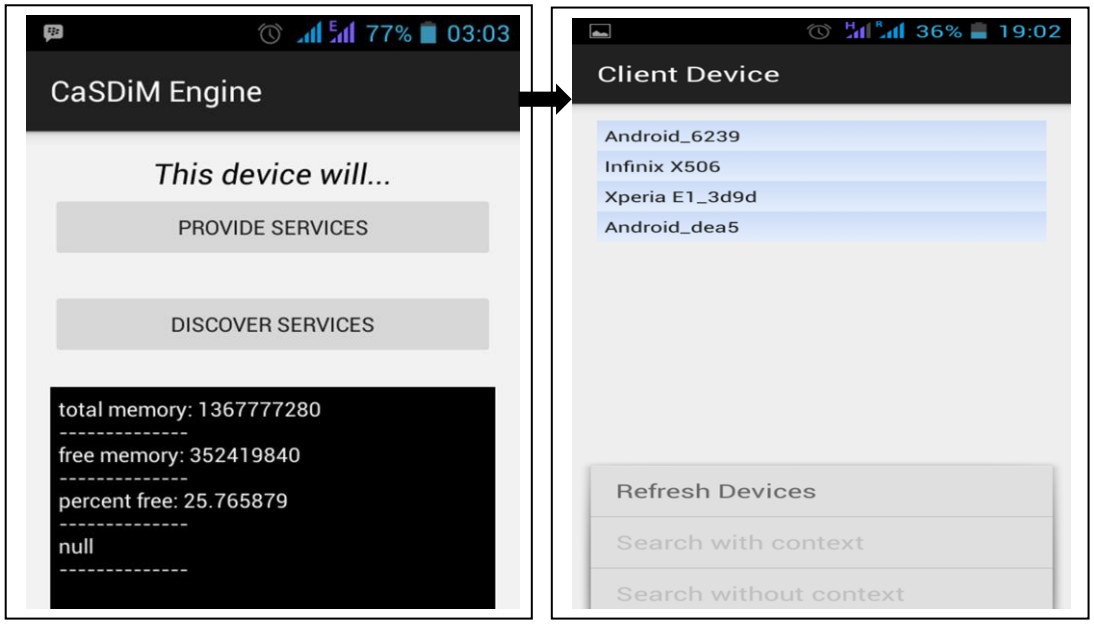


Figure 5.6: CaSDiM Implementation Snapshots – Device Role Selection and Client Discovery Windows

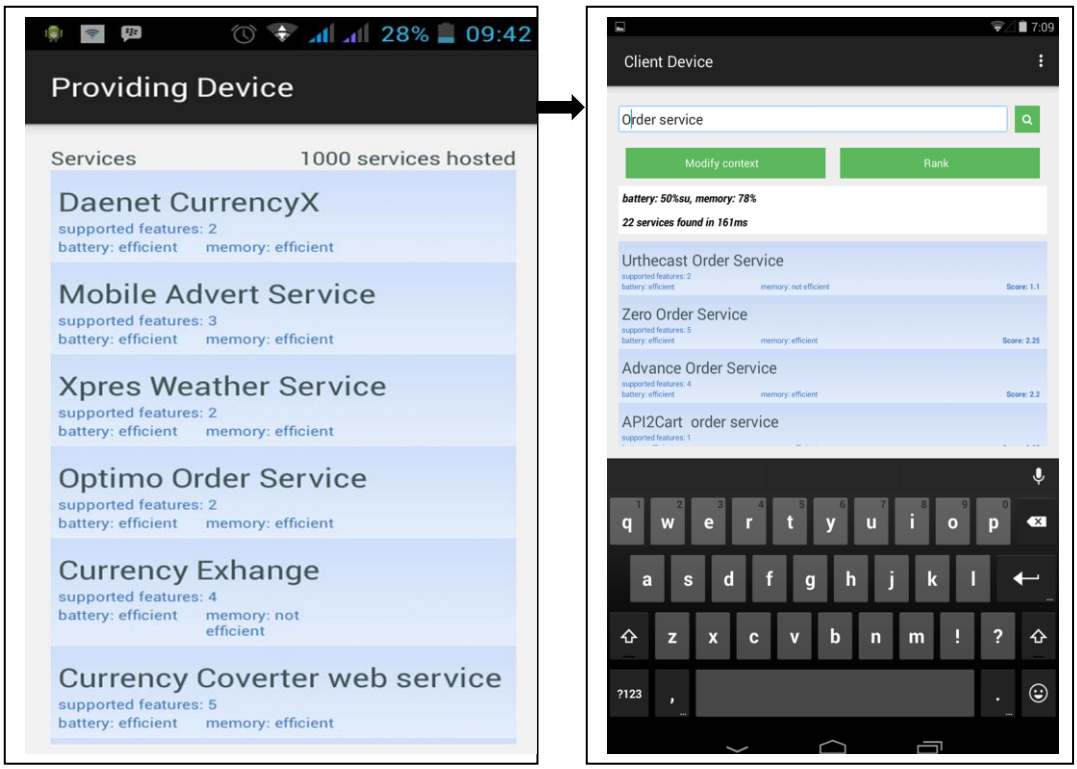


Figure 5.7: CaSDiM Implementation Snapshots - Providing Node and Client Discovering Services

For the purpose of easy deployment of test WSDL files, a web interface (hosted on a free web server) was created with Java and QSL. The WebSocket technology (part of Java EE 7) was

employed, which defines an API for establishing socket connections between a web browser and a server and creates a persistent connection for bi-directional and real-time client (Android device hosting services)/server communications. Since WebSocket creates a single persistent connection it can be efficient with regards to resource usage, which is clearly important to mobile devices where battery and network data usage are valuable resources. The web interface enable remote pushing of Web Service description documents to the hosting device.

5.4 Assumptions Made for CaSDiM Design and Implementation

Being research, the formulation, design, and implementation of CaSDiM was meant to provide a platform to validate the feasibility of this study's proposal through laboratory experimentation. Therefore, the following assumptions were made by necessity to delimit the scope of this study:

1. That there is negligible node mobility, because this work does not investigate mobility management. Such investigation would cater for situations where a providing device may move away from the reach of the client device while providing a service.
2. That every service-providing device has the right context with regards to capability to offer a service and that a device will only willingly enlists as a provider based on its resource capabilities. It is also envisioned that future innovations may create a scenario where some providers may not necessarily be resource- constrained devices. Based on this assumption, this study only took into account the context of client devices and not the context of provider/host devices. This implies that instances where the provider may get drained of its resources were not covered in this work.
3. That all participating nodes are authentic members of an Ad-hoc computing community although such assumption cannot be realistic. No security mechanisms

were employed but this can, in real life lead to being open to malicious attacks such as wireless denial of service DoS and air sniffing that are linked with the Wi-Fi Direct technology (Seokung Yoon et al, 2012).

4. That the Wi-Fi signal does not suffer from attenuation due to interference and other environmental factors.
5. That a client device is only able to access the Web Services offered by the providing device and no other personal directories or confidential information in order to avoid internal vulnerability. For instance, a client device, though consuming a service from the host, is not able to access, copy or modify the provider's phone log, contacts, text messages or emails.

5.5 Evaluation of the CaSDiM Prototype

The section presents the experimental setup and results based on which CaSDiM is evaluated. Broadly speaking, CaSDiM evaluation focuses on five parameters namely, recall and precision, relative quality of relevant service discovery, context overhead and processing time, service relevance, and resource-awareness.

Fundamentally, recall and precision are the most prominent parameters widely used to evaluate the effectiveness of information retrieval systems (Selvi & Raj, 2014). The effectiveness of service discovery is measured based on the proportion of relevant services retrieved accurately. Most importantly, the goal of this study focuses on using device context to enhance service discovery and optimise resource utilisation in AMC. To measure the achievement of this goal requires evaluation parameters in the service discovery domain that deal with context. This argument informed the choice of the other four parameters used in evaluating CaSDiM prototype.

5.5.1 Experiment 1: Recall and Precision

In this experiment, the performance of CaSDiM is evaluated using two well recognized metrics, namely service recall rate, and precision rate. Recall rate is defined as the proportion of relevant services that are retrieved or returned by a service discovery mechanism, while precision rate refers to the proportion of retrieved services that are precisely matched (Deng et al, 2010). These two parameters can be derived mathematically using the equations (7) and (8) respectively as given by Hernández del Olmo & Gaudioso, (2008) and Gunawardana & Shani, (2009):

$$\text{Precision} = \frac{Number\ of\ Retrieved\ Relevant\ Services}{Number\ of\ Retrieved\ Services} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (7)$$

$$\text{Recall} = \frac{Number\ of\ Retrieved\ Relevant\ Services}{Number\ of\ Published\ Relevant\ Services} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8)$$

In this experiment, a service is considered to be either relevant or not relevant depending on its resource requirement and supported device features. A total of one thousand Web Service description documents (representing 1000 Web Services) were used in this experiment: the number of published relevant services is kept constant while varying the number of published services by 200 services. Published relevant service and published services are represented by their respective service descriptions in the Web Service description document directory.

The reason for varying the number of published services is to influence the independent variables (retrieved services and retrieved relevant services) in the above equation upon which precision and recall depend. Five service requests were performed with the CaSDiM engine under two scenarios: 1) with the context module disabled and 2) with the context module enabled. The aim was to study the performance of CaSDiM with regards to recall and

precision rates under the aforementioned scenarios. Data obtained in the recall and precision experiments is as shown in Table 5.1 and 5.2 respectively.

Table 5.1: CASDiM Recall Data

Published Services	Recall Without Device Context	Recall With Device Context
0	0	0
200	0.39	0.19
400	0.51	0.21
600	0.57	0.29
800	0.59	0.29
1000	0.72	0.35

Table 5.2: CASDiM Precision Data

Published Services	Precision Without Device Context	Precision With Device Context
0	0	0
200	0.25	0.56
400	0.21	0.52
600	0.18	0.53
800	0.16	0.16
1000	0.14	0.49

a. The Effect of Device Context on Recall

The results of recall investigation in Table 5.1 as presented in Figure 5.8 shows a 20% and 37% difference between the startup and cutoff points of the recall rate without device context and recall rate with device context respectively. That is, when device context is not applied the recall rate of CaSDiM starts from 39% and cuts off at 72%, but the recall rate drops to a startup and cutoff of 19% and 35% respective when device context is applied. Therefore, a higher recall rate is observed when CaSDiM service requests are executed without device

context compared to when device context is used. This means that more relevant services are retrieved when device context is not applied.

CaSDiM tends to show higher recall rate when device context is not applied because a service request becomes broad (just based on a keyword or phrase) when device context is not used as a filterand a broad service request leads to the retrieval of many matching services. In contrast, the drop in CaSDiM's recall rate when device context is applied signifies that a smaller number of relevant services is retrieved. CaSDiM is deliberately designed to ensure that only really relevant services are retrieved by using the technique of service request adaptation. This methodology is in accordance with the goal of optimising usage and improving the quality of relevant service discovery.

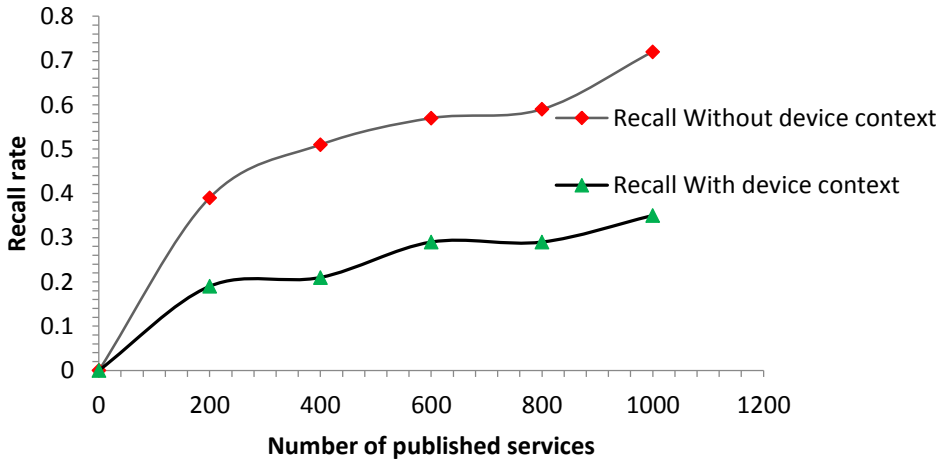


Figure 5.8: CaSDiM Recall

b. The Effect of Device Context on Precision

In the case of the precision rate depicted in Figure 5.9, which is derived from Table 5.2, a direct reverse of the trend seen in the recall analysis is observed: CaSDiM records a higher precision rate of 56% when device context is applied as against 25% when device context is not used. Precision also means exactness. Therefore, the above finding indicates that

CaSDiM is more accurate in discovering relevant services when device context is applied. This strength of CaSDiM with regard to precision rate is contributed to by the request adaptation technique implemented by the discovery engine component. With this technique the numbers of retrieved services and retrieved relevant services are reduced, thereby increasing precision rate according to the equation (7). Another observable trend is the variation in the precision range (the highest precision value minus the lowest) between the two scenarios: 11%, and 7% respectively for precision without device context and precision with device context.

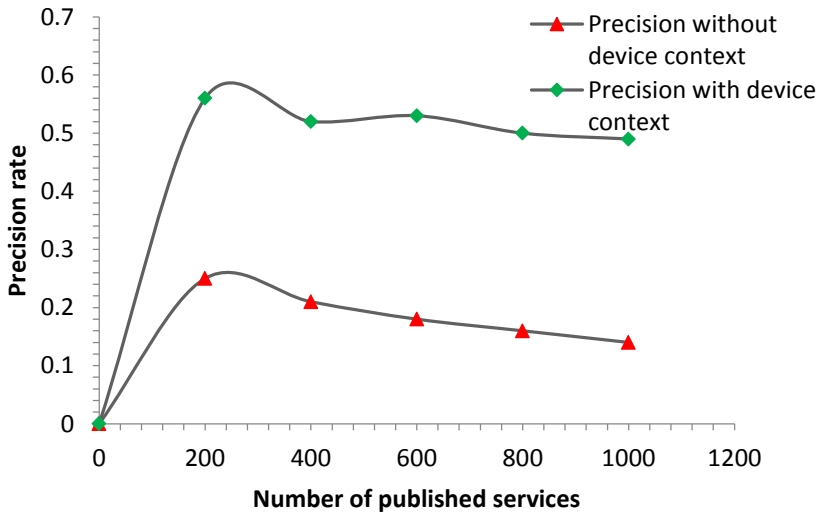


Figure 5.9: CaSDiM Precision

High precision range translates to a fast drop in precision rate as the number of published services grows and vice versa. For service requests executed without device context, the above finding suggests that the number of slightly similar service will grow as the number of published services grows. Such a tendency will most likely increase the likelihood of retrieving services that are only similar but not actually relevant.

In contrast, CaSDiM has a low precision range in the difference of 4. This can be attributed to cases where requesters type an incomplete (wrong) phrase or word as keyword. When a

wrong phrase or word is used as a search key it increases the chances of retrieving more services, which impacts on the value of precision.

In practice, any service discovery mechanism can achieve either high recall or high precision, but rarely both simultaneously. An effort to enhance the performance of one variable generally causes a drop in the performance of the other. This is often referred to as the "Recall-Precision tradeoff", which is outside the scope of this study. In summary, it is understandable that having high precision and low recall has the consequence of possibly leaving out some relevant services. On the other hand, this study contends that although some relevant services may be left out, they may not necessarily be relevant with regards to meeting the resource requirement of the client device – which is the core idea behind the filtering approach adopted in this study.

5.5.2 Experiment 2: Relative Quality of Relevant Service Discovery

The essence of incorporating device context in CaSDiM is to enhance service discovery. Such enhancement in the context of this work is measured with respect to the proportion of relevant services retrieved, that is, services that match the resource capabilities of a client device at the point of request. This section evaluates the relative quality of relevant service discovery of CaSDiM. Relative quality of relevant service discovery (RQoRSD) is defined as the percentage of the ratio of services filtered out to the total number of services retrieved (Elgazzar et al, 2013). That is:

$$\text{RQoRSD} = \left(\frac{\text{Number of Filtered-out Service}}{\text{Number of Retrieved Relevant Services}}\right) * 100 \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \;(9)$$

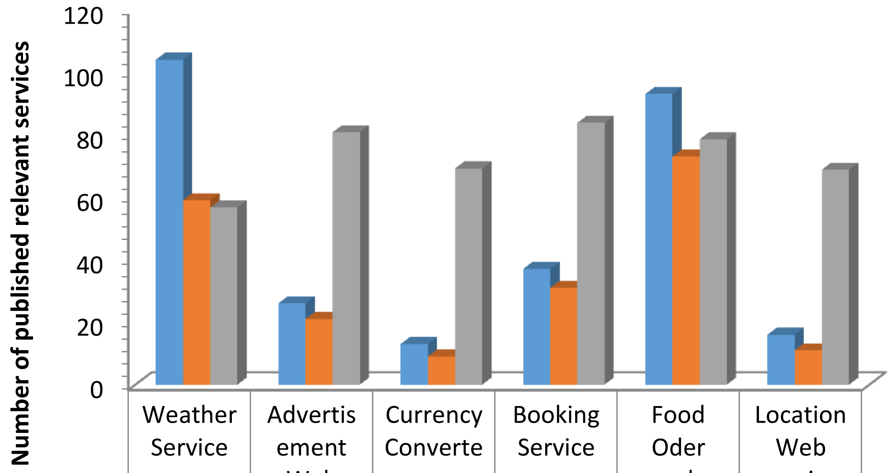
Where: Filtered-out Services = $\left(\text{Retrieved Relevant Services} - \text{Filteredin Services}\right)$….. (10)

In this experiment, six different service requests are made at random with the number of relevant services associated with each request also determined at random. Three types of

services namely, retrieved relevant services, filtered-in services, and filtered-out services were then considered. The total number of published services was kept constant. Varying the number of relevant services for the different service requests changes the number of retrieved relevant services; changing alters the number of filtered-in and filtered-out services as well. The aim is to help determine the filtering capability of CaSDiM. To achieve the above aim, each service request was executed and the number of relevant services retrieved and the number of services filtered-in (that is the services that finally retained after filtering) determined. Knowing these two sets of services, the number of filtered-out services is computed based on equation (10). These two parameters were then used to evaluate the relative quality of relevant service discovery of CaSDiM.

a. The Effect of Device Context on Relative Quality of Relevant Service Discovery

The performance of CaSDiM with regards to RQoSD is demonstrated in Figure 5.10. Overall, CaSDiM shows an average RQoRSD performance of 73%. The peak of this performance was recorded in the "Booking service" request with 84% RQoRSD. This high RQoRSD performance was contributed to by CaSDiM's context-aware discovery strategy, which reduced the number of retrieved relevant services and, by implication, increases the number of services filteredout. RQoRSD was positively affected here because it is the percentage of the ratio of filtered-out services to retrieved relevant services. There was also an observed irregularity in the RQoRSD across the six services requests. The lack of a defined pattern or sequence revealed that the discovery process is context dependent, and changes in device context are irregular. This result is achieved through the Node monitor feature, which extracts current context of the client device whenever a service request is initiated.



	Weather Service	Advertisement Web service	Currency Converter	Booking Service	Food Oder web service	Location Web service
Retrieved Relevant Services	104	26	13	37	93	16
Filtered-out Services	59	21	9	31	73	11
RQoSD (%)	56.7	80.8	69.2	83.8	78.5	68.8

Figure 5.10: CaSDiM Quality of Service Discovery

5.5.3 Experiment 3: Context Overhead

The CaSDiM approach aims at facilitating resource efficient service discovery based on the utilisation of device context. However, the addition of context information can create an added resource burden or overhead. Therefore, this section evaluates the effects of using device context with regards to generating context overhead. Context overhead is a function of context matching time and total response time. In this study, context overhead is defined as the percentage of the ratio of context matching time to overall response time (Elgazzar et al, 2013). That is:

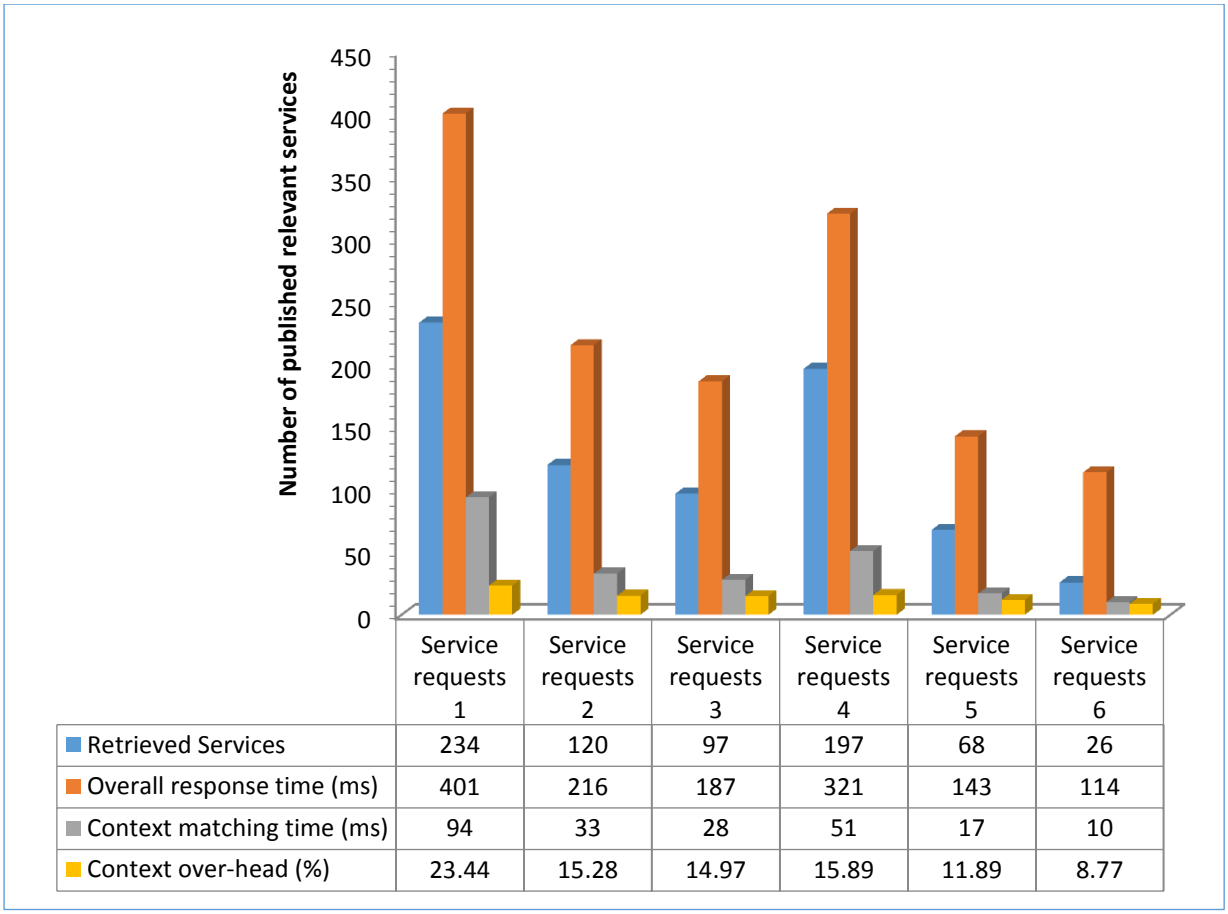
$$\text{Context Overhead} = \left(\frac{Context\ Matching\ Time}{Total\ Response\ Time}\right) * 100 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(11)}$$

The setup of this experiment is similar to the previous one in subsection 5.5.2. The only difference is the structuring of the discovery process. In this experiment, each service request was launched in two stages with the same number of published relevant services. First, the services request was launched to determine the number of relevant services retrieved and the

overall response time, which accounted for communication, service matching, and context matching time. Second, the same service request was re-launched, this time in two steps: a) searching and retrieval and b) filtering and ranking. The first operation in the second stage involves initiating a service request from the CaSDiM request interface with the context module disabled. With this operation, Web Services matching the entered keyword were retrieved and the time taken to complete the processes (i.e., the service matching time without context) is reported by the CaSDiM engine. The second operation then utilises device context to filter the services retrieved in the first stage. In this step, the experiment is only concerned with determining the context matching time.

a. The Effect of Device Context information on Overhead

In Figure 5.11 CaSDiM's performance was appraised with respect to the overhead generated by the use of device context. The major observable trend here was a low context overhead recorded in all the six service requests executed with an overall average of 15%. Two factors were responsible for this low context overhead: 1) CaSDiM's service description and matching processes did not utilise semantic techniques and 2) the context matching task was shared between the client and service-providing device (server-side). Consequently, the context overhead recorded by CaSDiM was accounted for by the client side matching which involved only the number of retrieved services. This result also explains why context overhead is more dependent on the context matching time than the overall response time of each service request. For instance, even with the highest response time of 401ms, context matching time still remains as low as 23%.



	Service requests 1	Service requests 2	Service requests 3	Service requests 4	Service requests 5	Service requests 6
Retrieved Services	234	120	97	197	68	26
Overall response time (ms)	401	216	187	321	143	114
Context matching time (ms)	94	33	28	51	17	10
Context over-head (%)	23.44	15.28	14.97	15.89	11.89	8.77

Figure 5.11: CaSDiM Context Overhead

Low context overhead means CaSDiM's discovery process creates a smaller resource burden, which fulfills the critical requirement for AMC as emphasised in the statement of problem and research goal of this study.

b. The Effect of Device Context information on Processing Time

Context overhead can impact on processing time and processing time has a phenomenal implication for computing resources (battery, memory etc.) especially in the context of the Ad-hoc Mobile Cloud. Due to its importance, processing time forms one of the bases for the evaluation of CaSDiM. From the graphical illustration of processing time in Figure 5.12, it is inferred that there is an observable proportional relationship between the number of retrieved services and the context matching time. For instance, 234 and 26 retrieved services require

94ms and 10ms of context matching time respectively. In addition to this inference, the processing time is longer in CaSDiM when the service request is executed without device context compared to when device context is utilised. This difference in processing time between the two can be seen in their minimum and maximum values: 104, 307 and 10, 94 for service request without device context and service request with device context respectively. A short processing time is observed in CaSDiM with device context because the total number of retrieved services is always smaller compared to when device context is not utilised. To realise a reduction in the number of retrieved service, this work adopted the techniques of service request and resource adaptation in CaSDiM.

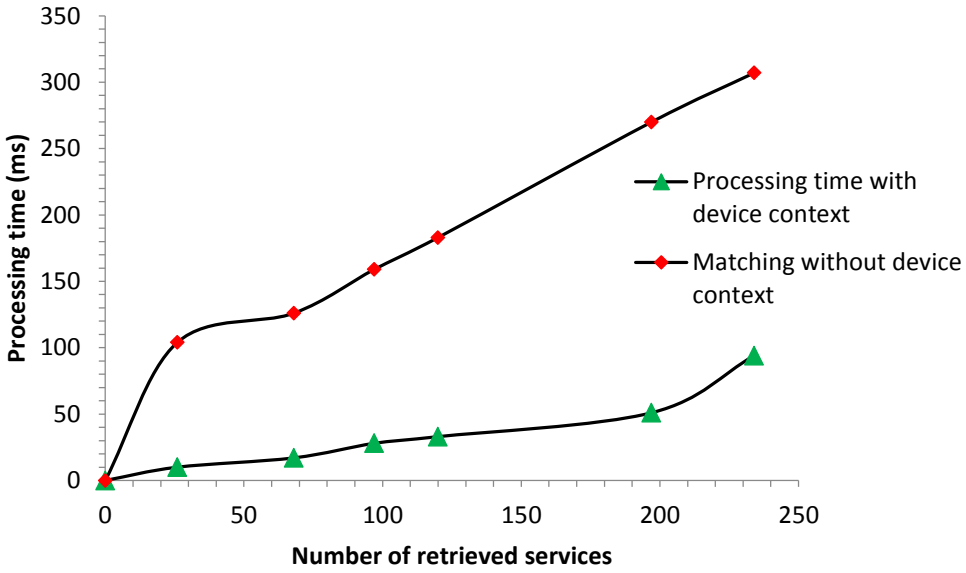


Figure 5.12: CaSDiM Processing Time

From the graph in Figure 5.12, a sharp rise in processing time corresponding to the number of retrieved services is connected with CaSDiM when device context is not used. Therefore, it is reasoned that there is a rise in processing time because preconditions (device context) were not attached to the search keyword. Without preconditions, more services will be returned, thereby taking more time to process. However, the reverse was the case when device context was used, as CaSDiM produced a curve that only rises minimally. This slight rise in

processing time suggests that the number of retrieved services (which accounts for processing time) does not increase linearly when device context is applied since it depends on the number of relevant published services. Having short processing time implies that CaSDiM will minimise resource consumption since long processing time has adverse implications for both battery and memory.

5.5.4 Experiment 4: Resource-aware Service Ranking

In this study, the relevance of a service was considered with respect to the service matching a client device's current context and, in addition, having the required functionality. This dimension makes it imperative to evaluate CaSDiM's performance with respect to how service relevance is determined. Consequently, two different service requests were performed with the context module enabled. The number of published services is kept constant while the number of published relevant services was varied at random in each of the service request. By varying the number of published relevant services the number of retrieved relevant services changes in each service request. Also, each service request was executed twice with the same number of published relevant services: first, with the relevancy ranking module disabled – that is, retrieved relevant services are returned in no specific order; second, with the relevancy ranking module enabled. When the ranking module is enabled, the discovery engine computes the relevancy score of each retrieved service according to equation (6) and returns services sorted in the order of their relevance.

Data obtained from experiment 4 are reported as follows: tables 5.3 and 5.4 contain data generated from service rests 1 while data obtained from service request 2 are shown in tables 5.5 and 5.6.

Table 5.3: Unranked Service Retrieval (Service Request 1)

Order of retrieved services (unranked)	Relevancy score
Service 1	2.2
Service 2	2.15
Service 3	1
Service 4	1.1
Service 5	1
Service 6	2.1
Service 7	2
Service 8	2
Service 9	1.05
Service 10	2.25
Service 11	2.18
Service 12	2.05
Service 13	2.1
Service 14	2.15
Service 15	1
Service 16	2.25
Service 17	2.1
Service 18	2.05
Service 19	2.15

Table 5.4: Ranked Service Retrieval (Service Request 1)

Order of retrieved services (ranked)	Relevancy score
Service 10	2.25
Service 16	2.25
Service 1	2.2
Service 11	2.18
Service 2	2.15
Service 14	2.15
Service 19	2.15
Service 6	2.1
Service 13	2.1
Service 17	2.1
Service 12	2.05
Service 18	2.05
Service 7	2
Service 8	2
Service 4	1.1
Service 9	1.05
Service 3	1
Service 5	1
Service 15	1

Table 5.5: Unranked Service Retrieval (Service Request 2)

Order of retrieved services (unranked)	Relevancy score
Service 1	2.15
Service 2	1.15
Service 3	2.05
Service 4	1.2
Service 5	2.1
Service 6	2
Service 7	2.2
Service 8	2.25
Service 9	1.1
Service 10	2.1
Service 11	1.1
Service 12	2
Service 13	2.2
Service 14	2.15
Service 15	1.05
Service 16	2.15

Table 5.6: Ranked Service Retrieval (Service Request 2)

Order of retrieved services (ranked)	Relevancy score
Service 8	2.25
Service 7	2.2
Service 13	2.2
Service 1	2.15
Service 14	2.15
Service 16	2.15
Service 5	2.1
Service 10	2.1
Service 3	2.05
Service 6	2
Service 12	2
Service 4	1.2
Service 2	1.15
Service 9	1.1
Service 11	1.1
Service 15	1.05

a. The Impact of Ranking on Service Discovery: Request 1

The order of services retrieved by CaSDiM is shown in Figure 5.13 – 5.14. These are representations of the data in tables 5.3 and 5.4. From Figure 5.13, retrieved services appear in the order that they are returned to the client device (unranked), starting from service 1 (the first relevant service to be matched) to service 19. Since the retrieved services are not ranked according to their relevance, it was observed that the two best or most relevant services (i.e., service 10 and 14) appeared in the 10^{th} and 14^{th} positions of the retrieved service list. This arrangement is based on the time a service is matched. Unfortunately, based on the unranked retrieved service list, out of the nineteen retrieved services, only three (service 1, 2, and 10) of the most relevant (best) services appeared among the first ten services listed. This finding means that except a requester has pre-knowledge of the service requested, there is every tendency to invoke the wrong service. This tendency can be seen in the result depicted in Figure 5.13, which shows that returned services are displayed in no specific order. Another shortfall of this unranked retrieval is that it is time consuming since a requester will have to scroll through the list of services, trying to identify the right service to invoke.

However, when relevancy ranking is applied as illustrated in the Figure 5.14 the retrieved relevant services are automatically ranked (sorted) in the order of their relevance. For example, service 10 and service 15 with the highest and lowest relevancy score are listed first and last respectively. When there is a tie in relevancy score, the first service to be matched is listed first, as in the case of services 10, 16 and service 2, 14, and 19. In CaSDiM relevancy ranking was achieved through the ranking module of the service discovery engine as described in sections 4.4 and 4.5.7. The outcome depicted in Figure 5.14 shows that the adopted approach provides an easy and fast way of invocating relevant services without necessarily scrolling through the entire list of services returned, since the displayed services are sorted according to their relevance.

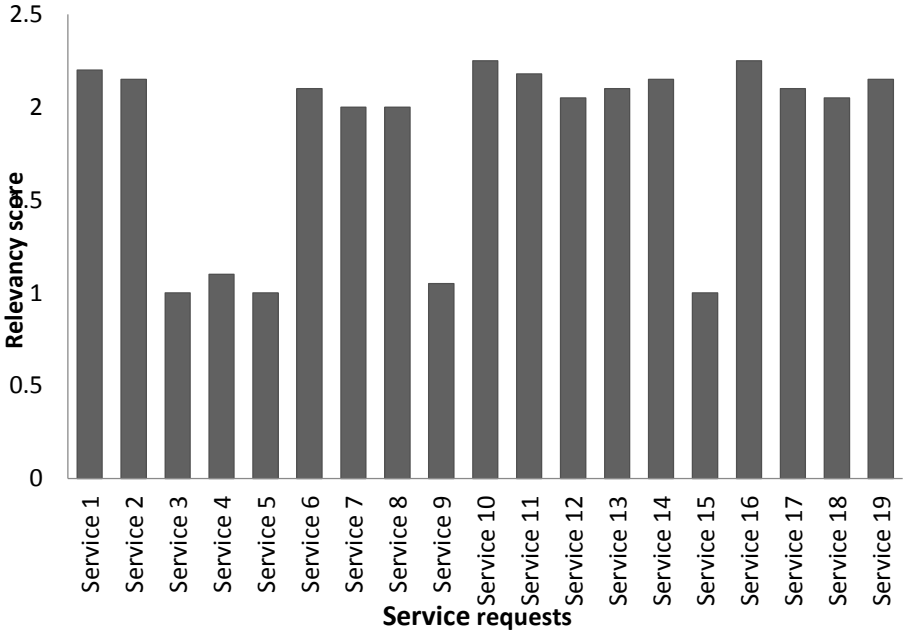


Figure 5.13: Request 1: Retrieved Unranked Services

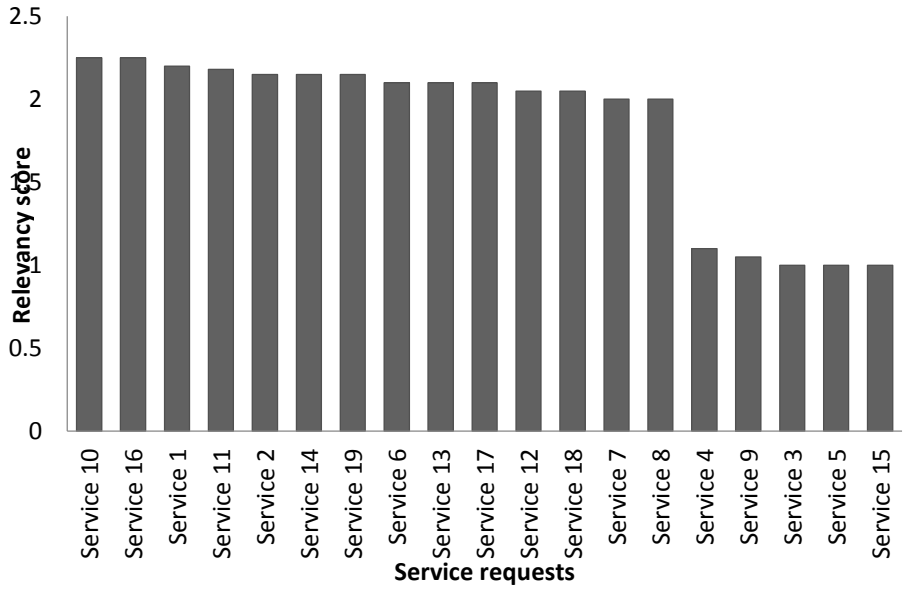


Figure 5.14: Request 1: Retrieved Ranked Services

b. The Effect of Relevancy Ranking on Service Discovering: Request 2

A similar trend is observed in the second service request with sixteen retrieved services as reported in tables 5.5 and 5.5 and represented in Figures 5.15 – 5.16. For instance, in the unranked list of retrieved services depicted in Figure 5.15, the best service appeared in 8th

position and the second best service in 7th. Although four out of the best ten services came under the first ten listed relevant services in the unranked list it can be seen that only service 4 appeared before other less relevant services among the first 10 retrieved services.

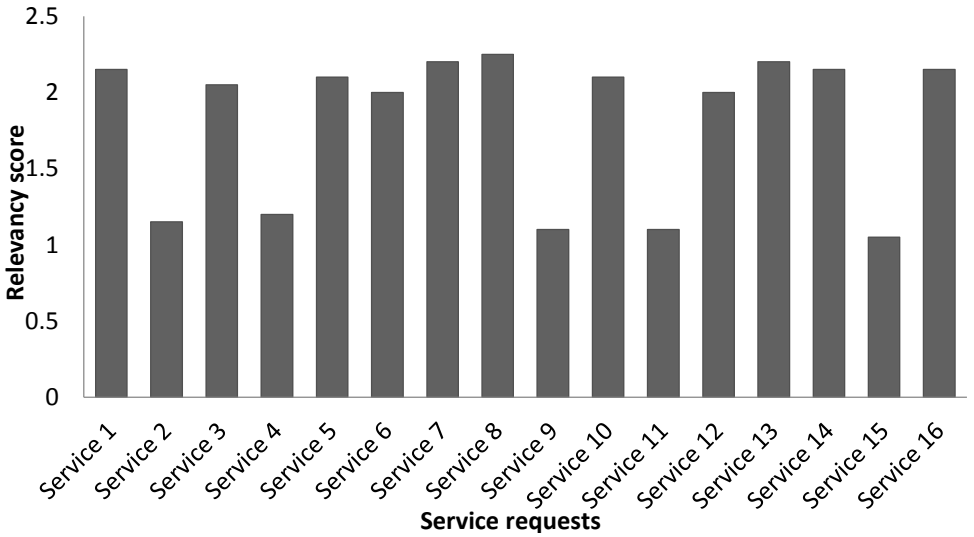


Figure 5.15: Request 2: Retrieved Unranked Services

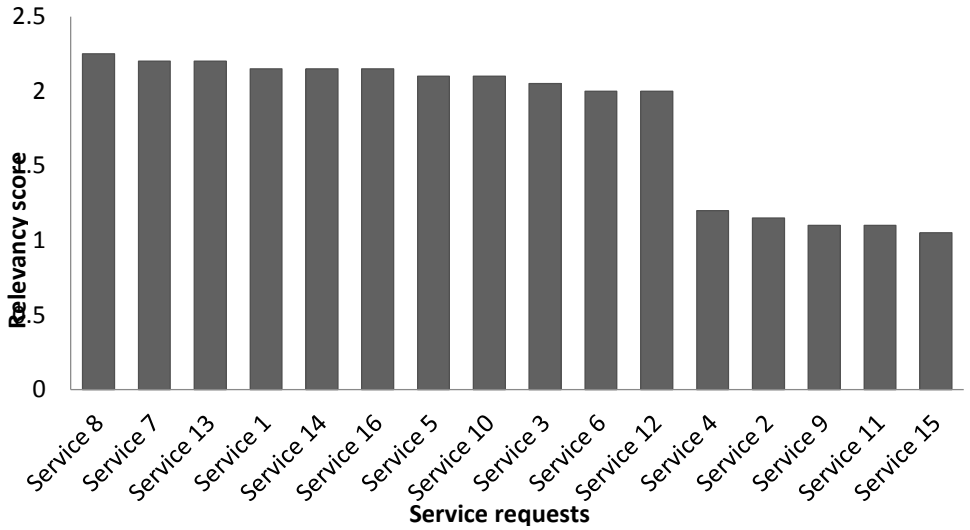


Figure 5.16: Request 2: Retrieved Ranked Services

5.5.5 Experiment 5: Resource Awareness

Due to resource scarcity in the Ad-hoc Mobile Cloud, one key aspect of this work is to minimise resource utilisation as a way to enhance effective service discovery. CaSDiM adopts a proactive approach in addressing the issue of limited resources by first adapting service requests to the current context of the client device before discovering relevant services. This section presents an evaluation of CaSDiM based on resource awareness. The setup of this experiment is similar to that of experiment 4 with regards to the number of published services. However, in this case, a service request is executed twice (with the same keyword) with a constant number of published relevant services. In this experiment device resource context is categorised into two levels: critical and normal. Critical level is when a client device is low in both battery and memory. On the other hand, when a client device is either low in memory or battery but not in both, it is said to be at normal device resource context level. Using the Android resource manager tool this device resource (battery and memory) categorisation was simulated to create three scenarios:

a) Critical resource - battery low, memory low
b) Normal resource level 1 – low in battery only
c) Normal resource level 2 – low in memory only

For each service request executed, the device resource context is varied from normal to critical. The purpose of altering the resource context is to evaluate the effect of resource-aware service discovery on resource usage optimisation. From the implementation point of view, CaSDiM proactively responds to changes in device context by discovering only services that best fit the current context of the client device at the time of service request. Table 5.7 shows the data obtained from the experiment:

Table 5.7: CASDiM Resource-based Service retrieval Data

	Retrieved Services Based on Resource Context				
Resource context	Battery and memory effecicient	Only battery efficient	Only memory efficient	Battery and memory inefficient	total
Critical:9%, 11%	19	0	0	0	19
Normal 1:74%, 36%	19	5	0	0	24
Normal 2:41%, 69	19	0	2	0	21

a. The Effect of Resource-awareness on Service Discovery and Resource Usage Optimisation

The way CaSDiM adapts in response to changes in device context during service discovery is demonstrated in Figure 5.17 using data from Table 5.7. First, it was observed that although the number of the same set of published relevant services was kept constant at 30 services through the three executions of the same service request, the number of retrieved relevant services varied in all three scenarios. At critical resource level, fourteen relevant services were returned. By manually verifying the list of retrieved relevant services, it was discovered that all the retrieved services were both efficient in battery and memory (according to our description in section 4.3.2). Also, compared to the other two scenarios, CaSDiM retrieved the smallest number of relevant services (14 services) at critical resource. This number of services amounts to 63% cumulative quality of relevant service discovery. This result showed that CaSDiM was able to proactively respond to device context change by constraining the discovery process to only retrieve the most optimal services with regards to resource requirement.

The Normal 1 and Normal 2 scenarios returned sixteen (16) and nineteen (19) relevant services respectively. Following a manual check, it was revealed that only services that are most efficient in the depleted resource or both were retrieved in each case. These results were achieved using a combination of the node monitor and the discovery engine to monitor device resource and adapt service requests to suit a client device's current context.

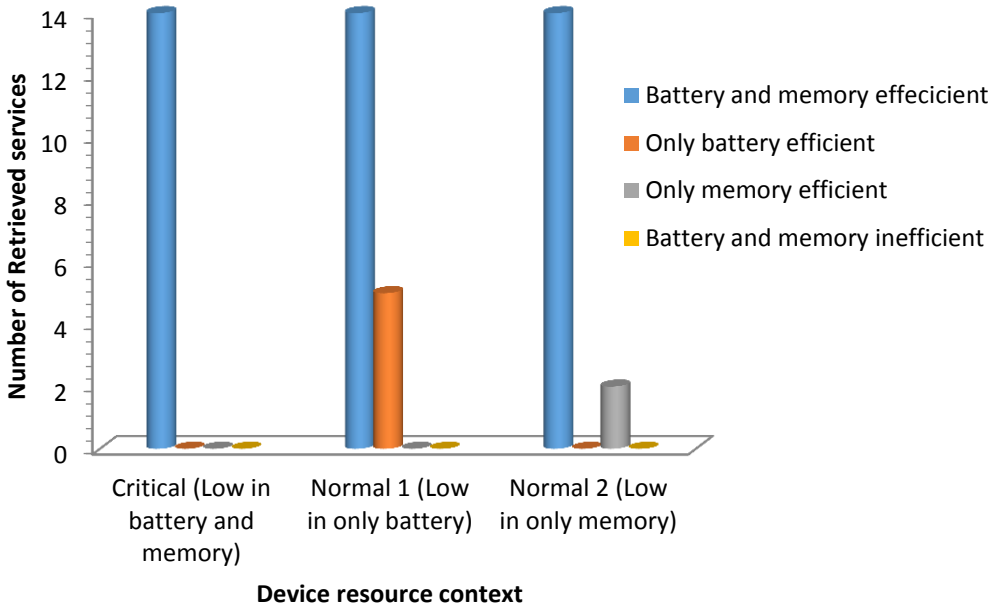


Figure 5.17: CaSDiM Resource-awareness

5.6 Chapter Summary

This chapter presented the results of a CaSDiM prototype test-bed experiment. Using lightweight technologies the CaSDiM prototype was developed and evaluated based on two scenarios. One scenario involved execution without device context, while the other employed device context. Experiments on recall and precision, relative quality of relevant service discovery, context overhead, service relevance, and resource-awareness were carried out. The goal of the CaSDiM solution approach was to achieve relevant service discovery in the Ad-hoc Mobile cloud while minimising resource consumption. This goal was realised by

incorporating a node monitoring module as a component of the discovery mechanism. The component extracts the battery level and available memory (device context) of a client device at the point of service request. This device context is then used by a discovery component to search, discover and retrieve only services that match the client's current resource capability. Experimental results show that using device context improves relevant service discovery and reduce resource burden.

The implications of the results obtained in this chapter with regards to the goal of this study were analysed in the next chapter. These analyses consist of how the outlined research questions of this study were answered by meeting the specified objectives. The next chapter also acknowledges the limitations of this study and defined this work's future direction.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Introduction

This study is a positive attempt to investigate the utilisation of device context in the discovery of services in the Ad-hoc Mobile Cloud. Utilising device context in AMC service discovery is imperative due to the inherent resource scarcity and dynamic context associated with the environment. These limitations imply that proactive service discovery techniques that incorporate resource usage intelligence are highly desirable. Using device context will therefore enhance service discovery and resource usage optimisation in AMC through supporting resource-aware, pre-emptive service discovery operations.

In an attempt to achieve the goal of this study, a solution approach was designed based on the knowledge gained from existing service discovery techniques in Mobile Cloud and the-state-of-the-art in AMC. This knowledge served as an antecedent to develop prerequisite ideas to evaluate and meet the requirements of AMC service discovery.

This chapter gives an overview of the work reported in this dissertation. Section 6.2 offers a discussion that attempts to substantiate how the proposed solution approach utilises the outlined objectives to answer the research question of this study. The discussion focuses on analysing the implications of the results obtained in the previous chapter with regards to the goal of this research work. The limitations of this study are discussed in section 6.3 with respect to the assumptions made in the previous chapter. The future direction of this study is also defined in this section.

6.2 Discussion and Conclusion

The initial investigation conducted in this study revealed the abundance of literature on service discovery techniques that are traditionally designed for Cloud and Mobile Cloud. But the investigation suggested that such techniques do not adequately consider the unique needs of resource-constrained devices.

On the other hand, approaches that are adopted for AMC do not consider device resources as a component of device context. To this end, this study identified the need to utilise an expanded device context model that comprises device profile and resource profile. This expanded context model was intended to help enrich service descriptions with more non-functional parameters in order to support proactive service discovery in the Ad-hoc Mobile Cloud.

Therefore, the research question that informed the aim of this study was answered as follows:

How can context-aware relevant service discovery be achieved in AMC in a manner that minimises device resource consumption?

i. How is context-aware service discovery achieved in MCC?
ii. Why are the mechanisms for achieving context-aware service discovery in MCC not adequate in Ad-hoc Mobile Cloud?

In answering sub questions (i) and (ii) this study first carried out an in-depth background study of Web Service standards. The study was used to gain an understanding of how different Web Service standards impact on mechanisms of service discovery. Based on this background knowledge, a thorough literature review of scholarly works in the domain of this research was conducted. The literature review was geared towards learning the techniques employed in Mobile Cloud service discovery mechanisms and to investigate whether such

techniques can be adopted in AMC. To achieve the above, an evaluation framework was formulated in section 3.1, which attempts to align Mobile Cloud service discovery approaches with the requirements of service discovery in AMC.

iii. How can device context be utilised to enhance service discovery in AMC?

First, this sub research question was answered by offering an architectural design of CaSDiM, which has components equipped with functionalities to generate and use device context to enhance service discovery in AMC. Second, by designing an expanded device context model in subsection 4.3, which was then used to formulate an expressive, resource-friendly service description, and service discovery and ranking algorithms in subsection 4.3.2 and section 4.4 respectively. These algorithms were implemented by the CaSDiM prototype as described in section 5.3. Also, the architectural design of CaSDiM includes components equipped with functionalities to generate and use device context to enhance service discovery in AMC.

iv. How can Ad-hoc mobile nodes be monitored as a strategy to minimise resource consumption during service discovery?

The overall design of CaSDiM as depicted in Figure 4.2 also helps to contribute to answering the research question through its key components. For example, the dynamic context of AMC means that discovered services will not always match the resource capability (battery, memory) of the client device. In solving this challenge, the Node Monitor (NoM) component as described in subsection 4.5.1.1, was designed to monitor the battery and memory levels of a client node. By utilising the utility methods offered by Android resource manager, NoM provides device context data at any time a service request is launched, which addresses our fourth (iv) sub research question.

Second, the proposed solution approach with regards to service description, service matchmaking and relevancy ranking algorithms as developed in sections 4.3.2 and 4.4 were all lightweight, aimed at minimising resource consumption. The Discovery Engine (DiEn) component, which implements these algorithms, as described in subsection 4.5.2.2, employs the concept of service request adaptation as demonstrated in experiments 2-5. This concept allows DiEn to first incorporate the current device context information extracted from the node monitor into a service request before the request is launched. This discovery technique enables CaSDiM to:

a) Only retrieve services that match the current resource capability of the client device
b) Reduce the total number of retrieved services and, by implication, reduce the processing time
c) Proactively adapt service request to change in context

These specifically answered the second part of the fourth (iv) sub research question in addition to supporting answers to sub question (iii).

The experimental results of this study validate this study's proposal that the utilisation of device context will enhance service discovery in AMC and facilitate proactive service discovery that is capable of optimising resource usage.

The results obtained in the experimentation have interesting implications for the overall goal of this research as analysed:

A. Implication for the Effectiveness of Service Discovery: Recall and Precision

First, the effect of utilising device context on recall and precision rates was investigated. As reported in Figure 5.6 of experiment 1, CaSDiM has higher performance with a recall rate of 72% for service requests launch without using device context, compared to when device

context is utilised. Higher recall is observed here because the request has no filter conditions or is not adapted (only considers the keyword), hence more services are retrieved even though they might not all be relevant.

In contrast, in precision evaluation, CaSDiM produced a higher precision rate of 56% when device context was applied. The higher precision rate is attributed to the fact that the service request is first adapted to current context thereby trimming down the number of retrieved relevant services. In summary, CaSDiM has low recall and high precision.

This study understands the fact that having high precision and low recall has the inevitable implication of possibly leaving out some relevant services. However, this work establishes the stand that although some relevant services may be left out, they may not necessarily be relevant within the context of service relevance as maintained in section 2.2 – which is the idea behind our filtering approach.

Therefore, the CaSDiM solution approach is capable of delivering an effective service discovery system for the Ad-hoc Mobile Cloud.

B. Implication for the Quality of Service Discovery: RQoRSD

Evaluating the impact of utilising device context on the quality of service discovery showed an overall average improvement of 73% relative to the service discovery performance without device context. This outcome demonstrates the fact that device context plays a significant role in AMC service discovery.

The implication, therefore, is that to realise efficient service discovery in AMC it is highly desirable to incorporate device context into the discovery process.

C. Implication for Overhead and Processing Time

The investigation of the effect of employing device context information on overhead showed a low overall average of 15%. When the same investigation was carried out on the processing time, it was revealed that with device context CaSDiM performed at a peak processing time of 94ms as against 307ms otherwise. Context overhead and processing time were always low with CaSDiM because context matching was performed on the number of retrieved services, which are relatively small due to the effect of request adaptation. Low overhead and processing time are good indications that CaSDiM supports minimal resource consumption.

These results lend strong support to the idea of this study, which emphasises the adoption of resource-friendly approaches to minimise resource burden.

D. Implication for Efficient Service Discovery

The CaSDiM approach adopts relevancy ranking techniques that also utilise device context. An analysis of the impact of this approach on service discovery efficiency shows a discovery process that is more appealing, time saving and less stressful to clients. As illustrated in Figures 5.14 and 5.16, the most relevant services are guaranteed to be on top of the list of retrieved services. With relevancy ranking, therefore, a client can easily and quickly invoke the best services without having to scroll through several services. In contrast, the approach that does not utilise relevancy ranking cannot guarantee such benefit because the best service might be last on the list of retrieved services as shown in Figures 5.13 and 5.15.

In this study it is argued that a Web Service is relevant if it both matches the client's device capability and meets the user's required functionality. The result achieved through the relevancy ranking capability supports the solution approach adopted in this work and addresses the issue in the above argument.

E. Implication for Resource usage optimisation

On resource-aware service discovery, investigation showed that CaSDiM was proactive to context change. Using device context to drive proactive service discovery was of special interest because of the observable impact on resource optimisation. For instance, the number of retrieved services depended on the current state of the client device's resources (battery, memory). As illustrated in Figure 5.17, the number of retrieved services was trimmed down to the minimum compared to all other service requests. This reduction in the number of retrieved services is beneficial to resource-constrained devices because it translates to shorter processing time, leading to less battery consumption.

However, from the technical point of view, reducing the number of retrieved services means increasing the number of filtered-out services. Taking this relationship into account implies that the proactive response of CaSDiM to context change also impacts significantly on the relative quality of relevant service discovery (RQoRSD) as defined in equation (9).

These findings demonstrate the capability of the proposed solution approach to enhance AMC service discovery.

6.3 Limitations and Future Work

The major shortcomings of this study are outlined in this section. Although the concerns raised here do not invalidate the statement of the problem and research question of this study, their relevance to the domain of AMC service discovery cannot be overlooked, especially as the assumptions made for this study as stated in section 5.4 imply that the issues raised in the limitations fall outside the scope of the study.

Although evaluation has shown that CaSDiM will improve relevant service discovery in AMC, the limitations of this work are acknowledged based on the following grey areas:

i. Limited Web Service description parameters: for demonstration purposes, the WSDL file used for the experiments was only made of three parameters – battery requirement, memory requirement, and supported device features. This limited description may impact on processing time and filtering capability. For successful application, active Web Services with more functional and non-functional parameters will be required.

ii. Security, inter-node-connectivity, and communication: CaSDiM relies on the Wi-Fi Direct technology to achieve peer-peer connection. However, Wi-Fi Direct was recently shown to be vulnerable to security attacks capable of triggering a reboot or wireless denial of service (Seokung Yoon et al, 2012). Also, for two nodes to communicate, they must be on the same Wi-Fi network. Most importantly, since Wi-Fi signals are within the bandwidth range of 2.4GHz to 5GHz frequency, its signal suffers from attenuation due to interference. But in this work it is assumed that there is a vigorous security stage that verifies and authenticates participating nodes and that the Wi-Fi Direct signal is not subject to attenuation or other signal distortion.

However, such ideal situations are not feasible in real life so there is need for a more robust technology to connect devices. Also, security techniques that can secure clients and hosts from external and internal malicious intents are highly desirable.

iii. Single node search: CaSDiM only enables a client node to search and discover services from a single provider at a time. But in real life there may be need for simultaneous querying of more than one provider. Hence, there is a need to investigate how to achieve simultaneous service discovery and composition in AMC.

iv. No consideration for provider's context: in spite of the fact that the contexts of the providing and client nodes are vital, this work just considered the client's context alone. But in a practical scenario, there is the chance that a providing device may run out of battery or memory while providing a service. Nevertheless, the assumption was that to activate the provider status, the providing device must be in the right context and that there is the likelihood of having providing devices that are not constrained in resources.

As a result of these limitations, the major future focus of this study will consist in formulating a more elaborate and robust AMC service discovery framework capable of addressing the fundamental issues of: 1) security 2) simultaneous service discovery, and taking into consideration the context of the providing node. Furthermore, owing to the fact that node mobility can disrupt service provisioning in the Ad-hoc Mobile Cloud, mobility management will be another interesting future focus that will look into mobility models for the Ad-hoc Mobile Cloud.

BIBLIOGRAPHY

Akkiraju, R., Farrell, J., & Miller, J. (2005). Web Service Semantics - WSDL-S. *Kno.e.sis Publications*. Retrieved from http://corescholar.libraries.wright.edu/knoesis/69

Al-Masri, E., & Mahmoud, Q. (2006). A context-aware mobile service discovery and selection mechanism using artificial neural networks. *In Pro- Ceedings of the 8th International Conference on Electronic Commerce: The New E-Commerce: Innovations for Conquering Current Barriers, Obstacles and Limita- Tions to Conducting Successful Business on the Internet*, 594–598. Retrieved from http://dl.acm.org/citation.cfm?id=1151467

Al-Masri, E., & Mahmoud, Q. H. (2010). MobiEureka: An approach for enhancing the discovery of mobile Web Services. *Personal and Ubiquitous Computing*, *14*(7), 609–620. http://doi.org/10.1007/s00779-009-0252-5

AlShahwan, F., & Moessner, K. (2010). Providing SOAP Web Services and RESTful Web Services from Mobile Hosts. In *Fifth International Conference on Internet and Web Applications and Services* (pp. 174–179). IEEE. http://doi.org/10.1109/ICIW.2010.33

Bai, L., & Liu, M. (2008). A Fuzzy-set based Semantic Similarity Matching Algorithm for Web Service. In *IEEE International Conference on Services Computing* (Vol. 2, pp. 529–532). IEEE. http://doi.org/10.1109/SCC.2008.147

Bener, A. B., Ozadali, V., & Ilhan, E. S. (2009). Semantic matchmaker with precondition and effect matching using SWRL. *Expert Systems with Applications*, *36*(5), 9371–9377. http://doi.org/10.1016/j.eswa.2009.01.010

Bettini, C., Brdiczka, O., & Henricksen, K. (2010). A Survey of Context Modelling and Reasoning Techniques. *Pervasive and Mobile Computing*, *6*(2), 161–180. Retrieved from http://www.sciencedirect.com/science/article/pii/S1574119209000510

Bianchini, D., De Antonellis, V., & Melchiori, M. (2007). Lightweight Ontology-Based Service Discovery in Mobile Environments. In *17th International Conference on Database and Expert Systems Applications (DEXA'06)* (pp. 359–364). IEEE. http://doi.org/10.1109/DEXA.2007.83

Butler, M. (2011). Android: Changing the Mobile Landscape. *IEEE Pervasive Computing*, *10*(1), 4–7. http://doi.org/10.1109/MPRV.2011.1

Cao, Y., Jarke, M., & Klamma, R. (2009). Mobile Access to MPEG-7 Based Multimedia Services. *Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, 102–111. http://doi.org/10.1109/MDM.2009.21

Capra, L., Emmerich, W., & Mascolo, C. (2003). CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, *29*(10), 929–944. http://doi.org/10.1109/TSE.2003.1237173

Castillo, P., Bernier, J., & Arenas, M. (2011). SOAP vs REST: Comparing a master-slave GA implementation. *arXiv Preprint arXiv*. Retrieved from http://arxiv.org/abs/1105.4978

Chihani, B., Bertin, E., & Crespi, N. (2013). A graph-based context modeling approach. In *2013 International Conference on Smart Communications in Network Technologies (SaCoNeT)* (Vol. 1, pp. 1–6). IEEE. http://doi.org/10.1109/SaCoNeT.2013.6654570

Cilibrasi, R. L., & Vitanyi, P. M. B. (2007). The Google Similarity Distance. *IEEE*

Transactions on Knowledge and Data Engineering, *19*(3), 370–383. http://doi.org/10.1109/TKDE.2007.48

Cortazar, G. O., Zapater, J. J. S., & Sanchez, F. G. (2012). Adding semantics to cloud computing to enhance service discovery and access.

Dejan, K., Cao, Y., & Klamma, R. (2011). Mobile cloud computing: A comparison of application models. *arXiv Preprint arXiv:1107.4940*. Retrieved from http://arxiv.org/abs/1107.4940

Deng, S., Wu, Z., & Wu, J. (2010). An Efficient Service Discovery Method and its Application. *International Journal of Web Services Research (IJWSR)*, *6*(4), 93–115. http://doi.org/10.4018/jwsr.2009071305

Dinh, H., Lee, C., & Niyato, D. (2013). A survey of mobile cloud computing: Architecture, Applications, and Approaches. *Wireless Communications and Mobile Computing*, *13*(18), 1587–1611. http://doi.org/10.1002/wcm.1203

Dong, H., Hussain, F., & Chang, E. (2013). Semantic Web Service matchmakers: state of the art and challenges. *Concurrency and Computation: Practice and Experience*, *25*(7), 961–988. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/cpe.2886/full

El-Bitar, I., Belouadha, F.-Z., & Roudies, O. (2013). Review of Web Services description approaches. In *2013 8th International Conference on Intelligent Systems: Theories and Applications (SITA)* (pp. 1–5). IEEE. http://doi.org/10.1109/SITA.2013.6560813

El-Masri, S., & Suleiman, B. (2005). A Framework for providing mobile Web Services. *The Second International Conference on Innovations in Information Technology*. Retrieved from http://www.it-innovations.ae/iit005/proceedings/articles/D_1_IIT05_Elmasri-1.pdf

Elgazzar, K. (2013). Discovery , Personalization and Resource Provisioning of Mobile Services, Ph.D Thesis. *Queen's University Kingston, Ontario, Canada*, (August).

Elgazzar, K., Hassanein, H., & Martin, P. (2011). Effective Web Service discovery in mobile environments. In *2011 IEEE 36th Conference on Local Computer Networks* (pp. 697–705). IEEE. http://doi.org/10.1109/LCN.2011.6115537

Elgazzar, K., Hassanein, H., & Martin, P. (2014a). DaaS: Cloud-based Mobile Web Service Discovery. *Pervasive and Mobile Computing*, *13*, 67–84. http://doi.org/10.1016/j.pmcj.2013.10.015

Elgazzar, K., Hassanein, H., & Martin, P. (2014b). Mobile Web Services : State of the Art and Challenges. *International Journal of Advanced Computer Science and Applications(IJACSA)*, *5*(3).

Elgazzar, K., Martin, P., & Hassanein, H. S. (2013a). Empowering mobile service provisioning through cloud assistance. *Proceedings - 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013*, 9–16. http://doi.org/10.1109/UCC.2013.20

Elgazzar, K., Martin, P., & Hassanein, H. S. (2013b). Personalized Mobile Web Service Discovery. In *IEEE Ninth World Congress on Services* (pp. 170–174). IEEE. http://doi.org/10.1109/SERVICES.2013.18

Erik, C., Francisco, C., & Greg, M. (2001). Web Service Definition Language (WSDL). Retrieved June 9, 2015, from http://www.w3.org/TR/wsdl

Fensel, D., Fischer, F., & Kopecký, J. (2010). WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. Retrieved June 10, 2015, from http://www.w3.org/Submission/WSMO-Lite/

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*. Retrieved from http://www.sciencedirect.com/science/article/pii/S1042814383710083

Guido Gehlen, L. P. (2005). Mobile Web Services for Peer-to-Peer Applications. In *Second IEEE Consumer Communications and Networking Conference, 2005. CCNC. 2005* (pp. 427–433). IEEE. http://doi.org/10.1109/CCNC.2005.1405210

Gunawardana, A., & Shani, G. (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *The Journal of Machine Learning Research*, *10*, 2935–2962. Retrieved from http://dl.acm.org/citation.cfm?id=1577069.1755883

Gutierrez-Garcia, J. O., & Sim, K.-M. (2010). Self-Organizing Agents for Service Composition in Cloud Computing. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science* (pp. 59–66). IEEE. http://doi.org/10.1109/CloudCom.2010.10

Hamad, H., Saad, M., & Abed, R. (2010). Performance Evaluation of RESTful Web Services for Mobile Devices. *International Arab Journal of E-Technology*, *1*(3), 72–78. Retrieved from http://www.iajet.org/iajet_files/vol.1/no.3/Performance Evaluation of RESTful Web Services for Mobile Devices.pdf%5Cnhttp://dblp.uni-trier.de/db/journals/iajet/iajet1.html#HamadSA10

Hatzi, O., Vrakas, D., Nikolaidou, M., & Bassiliades, N. (2012). An Integrated Approach to Automated Semantic Web Service Composition through Planning. *IEEE Transactions on Services Computing*, *5*(3), 319–332. http://doi.org/10.1109/TSC.2011.20

Hernández del Olmo, F., & Gaudioso, E. (2008). Evaluation of recommender systems: A new approach. *Expert Systems with Applications*, *35*(3), 790–804. http://doi.org/10.1016/j.eswa.2007.07.047

IBM. (2014, February 1). Understanding Web Service interoperability. Retrieved from http://www.ibm.com/developerworks/library/ws-inter/

Kang, J., & Sim, K. (2011). Cloudle: An Ontology-Enhanced Cloud Service Search Engine. *Web Information Systems Engineering–WISE 2010 Workshops*, *6724*, 416–427. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-24396-7_33

Keeney, J., & Cahill, V. (2003). Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework. In *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks* (pp. 3–14). IEEE Comput. Soc. http://doi.org/10.1109/POLICY.2003.1206953

Kim, Y. S., & Lee, K. H. (2007). A light-weight framework for hosting Web Services on mobile devices. *Proceedings of the 5th IEEE European Conference on Web Services, ECOWS 07*, 255–263. http://doi.org/10.1109/ECOWS.2007.21

Kousiouris, G., Kyriazis, D., Varvarigou, T., Oliveros, E., & Mandic, P. (2012). *Achieving Real-Time in Distributed Computing*. IGI Global. http://doi.org/10.4018/978-1-60960-827-9

Kreibich, J. (2010). *Using SQLite*. "O'Reilly Media, Inc." Retrieved from

https://books.google.com/books?hl=en&lr=&id=HFIM47wp0X0C&pgis=1

Le, D. N., Angela, G., & Tru, C. H. (2009). A Survey of Web Service Discovery Systems. Retrieved May 16, 2015, from http://www.igi-global.com/chapter/survey-web-service-discovery-systems/5037

Liu, L., Yao, X., & Qin, L. (2014). Ontology-based service matching in cloud computing. In *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (pp. 2544–2550). IEEE. http://doi.org/10.1109/FUZZ-IEEE.2014.6891698

Marinelli, E. E. (2009). Hyrax : Cloud Computing on Mobile Devices using MapReduce. *M.Sc. Thesis, School of Computer Science Carnegie Mellon University Pittsburgh*, *389*(September).

Martin, D., Burstein, M., & Hobbs, J. (2004). OWL-S: Semantic markup for Web Services. *W3C Member Submission*. Retrieved from http://www.ai.sri.com/~daml/services/owl-s/1.2/overview/

Martino, B. Di, Cretella, G., & Esposito, A. (2014). Semantic Representation of Cloud Services: A Case Study for Microsoft Windows Azure. In *2014 International Conference on Intelligent Networking and Collaborative Systems* (pp. 647–652). IEEE. http://doi.org/10.1109/INCoS.2014.76

Mascitti, D., Conti, M., & Passarella, A. (2014). Service Provisioning through Opportunistic Computing in Mobile Clouds. *Procedia Computer Science*, *40*, 143–150. http://doi.org/10.1016/j.procs.2014.10.042

MSDN. (2009). SOAP, REST, and More. Retrieved June 16, 2015, from https://msdn.microsoft.com/en-us/magazine/dd942839.aspx

Nadoveza, D., & Kiritsis, D. (2014). Ontology-based Approach for Context Modeling in Enterprise Applications. *Special Issue on The Role of Ontologies in Future Web-Based Industrial Enterprises*, *65*(9), 1218–1231. Retrieved from http://www.sciencedirect.com/science/article/pii/S0166361514001420

Nagireddi, V. S. K., & Mishra, S. (2013). An ontology based cloud service generic search engine. In *2013 8th International Conference on Computer Science & Education* (pp. 335–340). IEEE. http://doi.org/10.1109/ICCSE.2013.6553934

Ngan, L. D., & Kanagasabai, R. (2012). OWL-S Based Semantic Cloud Service Broker. In *2012 IEEE 19th International Conference on Web Services* (pp. 560–567). IEEE. http://doi.org/10.1109/ICWS.2012.103

Omrana, H., El Bitar, I., & Belouadha, F.-Z. (2013). A Comparative Evaluation of Web Services Description Approaches. In *2013 10th International Conference on Information Technology: New Generations* (pp. 60–64). IEEE. http://doi.org/10.1109/ITNG.2013.17

OpenGroup. (2013). Service-Oriented Architecture Defined. Retrieved June 9, 2015, from https://www.opengroup.org/soa/source-book/togaf/soadef.htm

Paliwal, A. V., Shafiq, B., & Vaidya, J. (2012). Semantics-Based Automated Service Discovery. *IEEE Transactions on Services Computing*, *5*(2), 260–275. http://doi.org/10.1109/TSC.2011.19

Palmer, N., Kemp, R., & Kielmann, T. (2009). Ibis for mobility. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications - HotMobile '09* (pp. 1–6). New York, New York, USA: ACM Press. http://doi.org/10.1145/1514411.1514426

Papakos, P., Licia, C., & Rosenblum, D. S. (2010). VOLARE: Context-Aware Adaptive Cloud Service Discovery for Mobile Systems. In *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware* (pp. 32–38). New York, New York, USA: ACM Press. http://doi.org/10.1145/1891701.1891706

Peffers, K., Tuunanen, T., Rothenberger, M., & Chatterjee, S. (2014). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77.

Perianu, R. M., Hartel, P., & Scholten, H. (2005). A classification of service discovery protocols. *Internal Report, Dept. Electrical Eng., Mathematics, and Computer Science, Univ. of Twente, Netherlands, June 2005*. Retrieved from http://eprints.eemcs.utwente.nl/735/01/0000012d.pdf

Prodan, R., & Ostermann, S. (2009). A survey and taxonomy of infrastructure as a service and web hosting cloud providers. *10th IEEE/ACM International Conference on Grid Computing*, 17–25. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5353074

Rathinam, N. E., Parthiban, L., & Mariakalavathy, G. (2014). Automatic Discovery of Relevant Web Services with Semantic Ranking. *Journal of Applied Sciences, Engineering and Technology*, *8*(22), 2240–2247.

Ravi, A., & Peddoju, S. K. (2013). Energy Efficient Seamless Service Provisioning in Mobile Cloud Computing. *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, 463–471. http://doi.org/10.1109/SOSE.2013.67

Renzel, D., Kovachev, D., & Klamma, R. (2010). Mobile Community Cloud Computing: Emerges and Evolves. *2010 Eleventh International Conference on Mobile Data Management*, 393–395. http://doi.org/10.1109/MDM.2010.78

Rodríguez-García, M. (2014). Creating a semantically-enhanced cloud services environment through ontology evolution. *Future Generation Computer Systems*, (32), 295–306. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167739X13001684

Rouvoy, R., Barone, P., & Ding, Y. (2009). Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. *Software Engineering for Self-Adaptive Systems*, *5525*, 164–182. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-02161-9_9

Saadon, N. A., & Mohamad, R. (2011). A Comparative Evaluation of Web Service Discovery Approaches for Mobile Computing. *International Conference of Informatics Engineering and Information Science (ICEIS)*, pp. 238–252.

Saadon, N. A., & Mohamad, R. (2014a). Cloud-based Mobile Web Service Discovery framework with semantic matchmaking approach. In *2014 8th. Malaysian Software Engineering Conference (MySEC)* (pp. 113–118). IEEE. http://doi.org/10.1109/MySec.2014.6985999

Saadon, N. A., & Mohamad, R. (2014b). WSMO-lite based Web Service discovery algorithm for mobile environment. *International Journal of Advances in Soft Computing and Its Applications*, *5*(SPECIALISSUE.3), 75–90.

Satyanarayanan, M., & Bahl, P. (2009). The case for vm-based cloudlets in mobile computing. IEEE *Pervasive Computing*, 8(4), PP. 1536-1268. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5280678

Satyanarayanant, M., Chent, Z., Hat, K., Hut, W., Richtert, W., & Pillai, P. (2014). Cloudlets : at the Leading Edge of Mobile-Cloud Convergence (Invited Paper). 2014 *6th International Conference on Mobile Computing, Applications and Services (MobiCASE).*

Selvi, R. T., & Raj, E. G. D. P. (2014). An Approach to Improve Precision and Recall for Ad-hoc Information Retrieval Using SBIR Algorithm. In *2014 World Congress on Computing and Communication Technologies* (pp. 137–141). IEEE. http://doi.org/10.1109/WCCCT.2014.68

Seokung Yoon, SoonTai Park, & Haeryoung Park. (2012). Security Analysis of Vulnerable Wi-Fi Direct. *Computing and Networking Technology (ICCNT), 2*012 8th International Conference on Computing and Networking Technology (INC, ICCIS and ICMIC), pp. 340-343.

Sheng, Q. Z., & Benatallah, B. (2005). ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development. In *International Conference on Mobile Business (ICMB'05)* (pp. 206–212). IEEE. http://doi.org/10.1109/ICMB.2005.33

Shin, D., Lee, K., & Suda, T. (2009). Automated generation of composite Web Services based on functional semantics. Web Semantics: Science, Services and Agents on the World Wide Web, 7(4), pp. 332–343. Retrieved from http://www.sciencedirect.com/science/article/pii/S1570826809000171

Sibiya, M. G. (2010). Context-Aware Service Discovery for Dynamic Grid Environments. *M.Sc Thesis, Computer Science Department, University of Zululan - SA*, (20000658).

SnapLogic. (2011). Give SOAP a REST. Retrieved July 5, 2015, from http://www.snaplogic.com/blog/give-soap-a-rest/

Srirama, S. N., Jarke, M., & Prinz, W. (2006). Mobile Web Service Provisioning. *Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06)* (pp. 19–25). http://doi.org/10.1109/AICT-ICIW.2006.215

Srirama, S. N., Jarke, M., & Zhu, H. (2008). Scalable Mobile Web Service Discovery in Peer to Peer Networks. In *2008 Third International Conference on Internet and Web Applications and Services* (pp. 668–674). IEEE. http://doi.org/10.1109/ICIW.2008.52

Srirama, S., & Paniagua, C. (2013). MS '13 Proceedings of the 2013 IEEE Second International Conference on Mobile Services (pp. 15-22). Retrieved from http://lepo.it.da.ut.ee/~srirama/publications/ms13.pdf

Steller, L. A. (2010). Light-weight and Adaptive reasoning for mobile Web Services, Ph.D thesis, Monash University, Australia. Retrieved from http://arrow.monash.edu/vital/access/manager/Repository/monash:63104;jsessionid=3C0431CC58C8C3EC99AD730C5892C447?exact=sm_subject%253A%2522Tableaux%2522

Sun, L., Dong, H., & Ashraf, J. (2012). Survey of Service Description Languages and Their Issues in Cloud Computing. In *2012 Eighth International Conference on Semantics, Knowledge and Grids* (pp. 128–135). IEEE. http://doi.org/10.1109/SKG.2012.49

Vedamuthu, A., Orchard, D., & Hirsch, F. (2007). Web Services policy 1.5-framework. *W3C* Retrieved from http://www.w3.org/TR/2007/PR-ws-policy-20070706/ws-policy-

framework.pdf

Venticinque, S., Aversa, R., & Martino, B. Di. (2011). A cloud agency for SLA negotiation and management. *Euro-Par 2010 Parallel Processing Workshops*, *6586*(587–594). Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-21878-1_72

Wagh, K. S., & Thool, R. (2013). Web Service Provisioning on Android Mobile Host. *International Journal of Computer Application*, *81*(14), 1–7.

Wagh, K., & Thool, R. (2012). A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host. *Journal of Information Engineering and Applications*, *2(5)*, *12-16*. Retrieved from http://www.iiste.org/Journals/index.php/JIEA/article/view/2063

Want, R., Pering, T., & Danneels, G. (2002). The Personal Server: Changing the way we Think about Ubiquitous Computing. *International Conference on Ubiquitous Computing*, 194–209. Retrieved from http://link.springer.com/chapter/10.1007/3-540-45809-3_15

Wieland, M., Nicklas, D., & Leymann, F. (2011). Context model for representation of business process management artifacts. 2011 *International Conference on Economics and Business Information*, *9*, 46–51. Retrieved from http://www.researchgate.net/profile/Frank_Leymann/publication/228422430_Context_Model_for_Representation_of_Business_Process_Management_Artifacts/links/0c96052a4716a7feed000000.pdf

Zeng, C., Guo, X., & Ou, W. (2009). Cloud Computing Service Composition and Search Based on Semantic. *Cloud Computing Letures*, *5931*, 290–300. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-10665-1_26

Zhao, Z., Huang, X., & Crespi, N. (2012). A system for web widget discovery using semantic distance between user intent and social tags. *Social Informatics: 4th International Conference SocInfo2012*, 1–14. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-35386-4_1

Ziafati, P., Mastrogiovanni, F., & Sgorbissa, A. (2011). Fast Prototyping and Deployment of Context-Aware Smart Outdoor Environments. In *2011 Seventh International Conference on Intelligent Environments* (pp. 206–213). IEEE. http://doi.org/10.1109/IE.2011.73

APPENDIX A: CaSDiM DEVICE DISCOVERY MODULE

```
package wifi.direct.wd;

import wifi.direct.wd.adapter.DeviceAdapter;
import wifi.direct.wd.adapter.WdServiceAdapter;
import wifi.direct.wd.db.ServiceDbHelper;
import wifi.direct.wd.model.ClientMessengerAsyncTask;
import wifi.direct.wd.model.WdService;
import android.content.IntentFilter;
import android.net.wifi.WpsInfo;
import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pDeviceList;
import android.net.wifi.p2p.WifiP2pInfo;
import android.net.wifi.p2p.WifiP2pManager;
import android.net.wifi.p2p.WifiP2pManager.ActionListener;
import android.net.wifi.p2p.WifiP2pManager.Channel;
import android.net.wifi.p2p.WifiP2pManager.ChannelListener;
import android.net.wifi.p2p.WifiP2pManager.ConnectionInfoListener;
import android.net.wifi.p2p.WifiP2pManager.PeerListListener;
import android.os.Bundle;
import android.os.Handler;


deviceListAdapter = new DeviceAdapter(this,
                                        /**
                                         * When this device tries to discover other devices, and it successfully does so,
                                         * those devices are added to a list for the user to choose to connect to

                                        new GenericClickListener<WifiP2pDevice>() {
                                                @Override
                                                public void onClick(View v, WifiP2pDevice device) {
                                                        WifiP2pConfig config = new WifiP2pConfig();
                                                        config.deviceAddress = device.deviceAddress;
                                                        config.wps.setup = WpsInfo.PBC;

                                                        DeviceListActivity.this.connect(config);
```

```
                                        }

                                });
                deviceListView = (RecyclerView) findViewById(R.id.device_list);
                deviceListView.setLayoutManager(new LinearLayoutManager(this));
                deviceListView.setHasFixedSize(true);
                deviceListView.setAdapter(deviceListAdapter);
                intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
                intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
                intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
                intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
                manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
                channel = manager.initialize(this, getMainLooper(), null);
                clientBroadcastReceiver = new ClientBroadcastReceiver(manager, channel,
                                this);
                registerReceiver(clientBroadcastReceiver, intentFilter);
        }
public void discoverPeers() {
                manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
                        @Override
                        public void onSuccess() {
                                Toast.makeText(DeviceListActivity.this, "Discovery initiated successfully",
                                                Toast.LENGTH_SHORT).show();
                        }
                        @Override
        public void onFailure(int reasonCode) {
                                Toast.makeText(DeviceListActivity.this,
                                                "Discovery Failed : " + reasonCode, Toast.LENGTH_SHORT)
                                                .show();
                        }
                });
        }
@Override
        protected void onDestroy() {
                // TODO Auto-generated method stub
                super.onDestroy();
```

```
                manager.stopPeerDiscovery(channel, null);
                if (ClientMessengerAsyncTask.clientMessengers.size() > 0) {
                        for (ClientMessengerAsyncTask clientMessenger : ClientMessengerAsyncTask.clientMessengers) {
                                try {
                                        clientMessenger.socket.close();
                                } catch (Exception e) {
                                        // TODO: handle exception
                                }
                        }
                }
                manager.removeGroup(channel, new ActionListener() {
                        @Override
                        public void onFailure(int reasonCode) {
                                toast("removeGroup fail: " + reasonCode);
                        }
                        @Override
                        public void onSuccess() {
                                toast("removeGroup success");
                        }
                });
        }
public void log(String log) {
                this.log.append(log + "\n----------\n");
        }
        @Override
        public void showDetails(WifiP2pDevice device) {
        }
        @Override
public void cancelDisconnect() {
                if (manager != null) {
                        if (thisDevice != null
                                        || (thisDevice.status == WifiP2pDevice.CONNECTED)) {
                                disconnect();
                        } else if (thisDevice.status == WifiP2pDevice.AVAILABLE
                                        || thisDevice.status == WifiP2pDevice.INVITED) {
                                manager.cancelConnect(channel, null);
```

```
                }
            }
        }
        @Override
public void connect(WifiP2pConfig config) {
            manager.connect(channel, config, new ActionListener() {

                @Override
                public void onSuccess() {
                    toast("connect success");
                }
                @Override
                public void onFailure(int reason) {
                    toast("connect failed: " + reason);
                }
            });
        }
```

APPENDIX B: CaSDiM SERVICE SEARCH AND RETRIEVAL MODULE

```
package wifi.direct.wd.model;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.net.Socket;

import wifi.direct.wd.ClientMessengerRunnable;

import android.content.Context;

import android.widget.Toast;

import com.google.gson.Gson;

public class ClientMessengerAsyncTask extends

                        AsyncTask<Map<String, String>, Void, String> {

        Context context;

        Handler handler;

        String host;

        String action;

        ClientMessengerRunnable onPostExecute;

        public Socket socket;

        private static final int SOCKET_TIMEOUT = 5000;


        long startTime, endTime;

        public static List<ClientMessengerAsyncTask> clientMessengers = new ArrayList<ClientMessengerAsyncTask>();

        public ClientMessengerAsyncTask(Context context, String host,

                                String action, ClientMessengerRunnable onPostExecute) {

                this.context = context;

                this.host = host;

                this.action = action;

                this.onPostExecute = onPostExecute;

                handler = new Handler();

        }

public class ServiceDbHelper {


        private static Context appContext;

        private static DbHelper dbHelper;
```

```
	public static void init(Context appContext) {
		if(ServiceDbHelper.appContext == null) {
			ServiceDbHelper.appContext = appContext;
		}
		dbHelper = new DbHelper(appContext);
	}
public static Map<String, Integer> saveWdServices(List<WdService> services) {
		Map<String, Integer> savedStats = new HashMap<>();
		savedStats.put("savedCount", 0);
		savedStats.put("notSavedCount", 0);
		int saved = 0;
		int notSaved = 0;
		if(services != null && services.size() > 0) {
			for(WdService service: services) {
				if(saveWdService(service)) {
					saved ++;
				} else {
					notSaved ++;
				}
			}
			savedStats.put("savedCount", saved);
			savedStats.put("notSavedCount", notSaved);
		}
		return savedStats;
	}
public static Map<String, Integer> deleteWdServices(List<Integer> serviceIds) {
		Map<String, Integer> savedStats = new HashMap<>();
		savedStats.put("deletedCount", 0);
		savedStats.put("notDeletedCount", 0);
		int deletedCount = 0;
		int notDeletedCount = 0;
		if(serviceIds != null && serviceIds.size() > 0) {
			for(int id: serviceIds) {

				if(deleteWdService(id)) {
					deletedCount ++;
```

```
                        } else {
                                notDeletedCount ++;
                        }
                }
                savedStats.put("deletedCount", deletedCount);
                savedStats.put("notDeletedCount", notDeletedCount);
        }
        return savedStats;
    }
public static boolean saveWdService(WdService service) {
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(ServiceDbEntry.COLUMN_WEB_SERVICE_ID, service.webServiceId);
        values.put(ServiceDbEntry.COLUMN_SERVICE_NAME, service.getServiceName());
        values.put(ServiceDbEntry.COLUMN_BATTERY_EFFICIENCY,
                        service.getBatteryEff());
        values.put(ServiceDbEntry.COLUMN_MEMORY_EFFICIENCY,
                        service.getMemoryEff());
        values.put(ServiceDbEntry.COLUMN_FRAMES_SUPPORT,
                        service.getFramesSupport());
        values.put(ServiceDbEntry.COLUMN_COOKIES_SUPPORT,
                        service.getCookiesSupport());
        values.put(ServiceDbEntry.COLUMN_CALLBACK_SUPPORT,
                        service.getCallbackSupport());
        values.put(ServiceDbEntry.COLUMN_XMLHTTP_SUPPORT,
                        service.getXmlHttpSupport());
        values.put(ServiceDbEntry.COLUMN_TABLES_SUPPORT,
                        service.getTableSupport());
        long insertId = db.insertWithOnConflict(ServiceDbEntry.TABLE_NAME,
                        null, values, SQLiteDatabase.CONFLICT_REPLACE);
        db.close();
        return (insertId != -1);
    }
public static boolean deleteWdService(WdService service) {
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        try {
```

```
                db.delete(ServiceDbEntry.TABLE_NAME, ServiceDbEntry._ID + "="
                                + service.getId(), null);
                deleteWdService(service.getId());
            } catch (Exception e) {
                return false;
            }
            return true;
        }
public static boolean deleteWdService(int id) {
            SQLiteDatabase db = dbHelper.getWritableDatabase();
            try {
                db.delete(ServiceDbEntry.TABLE_NAME, ServiceDbEntry._ID + "="
                                + id, null);
            } catch (Exception e) {
                return false;
            }
            return true;
        }
public static List<WdService> getAllServices() {
            List<WdService> services = new ArrayList<>();
            SQLiteDatabase db = dbHelper.getReadableDatabase();
            String columns[] = serviceColumns();
            Cursor cursor = db.query(ServiceDbEntry.TABLE_NAME, columns, null,
            null, null, null, ServiceDbEntry._ID + " DESC");
            while (cursor.moveToNext()) {
            services.add(cursorToService(cursor));
            }
            db.close();
            return services;
        }
        /**
         * This method resides on the device offering services(device 2), it is called whenever
         * device 1 searches, the code below does an SQL query of services with similar names
public static List<WdService> searchServices(String keyword) {
            List<WdService> services = new ArrayList<>();
            SQLiteDatabase db = dbHelper.getReadableDatabase();
```

```
                String columns[] = serviceColumns();
                String querySql = ServiceDbEntry.COLUMN_SERVICE_NAME + " LIKE '%"
                                + keyword + "%'";
                String orderSql = ServiceDbEntry.COLUMN_BATTERY_EFFICIENCY + " ASC, "
                                + ServiceDbEntry.COLUMN_MEMORY_EFFICIENCY + " ASC";
                Cursor cursor = db.query(ServiceDbEntry.TABLE_NAME, columns, querySql,
                                null, null, null, null);
                while (cursor.moveToNext()) {
                        services.add(cursorToService(cursor));
                }
                db.close();
                return services;
        }
public static List<WdService> searchServicesByProfile(String keyword,
                        int batteryEff, int memoryEff) {
                List<WdService> services = new ArrayList<>();
                SQLiteDatabase db = dbHelper.getReadableDatabase();
                String columns[] = serviceColumns();
                String keywordSql = "";
                String querySql = ServiceDbEntry.COLUMN_SERVICE_NAME + " LIKE '%"
                                + keyword + "%'";
                querySql += " AND " + ServiceDbEntry.COLUMN_BATTERY_EFFICIENCY + " <= "
                                + batteryEff;
                querySql += " AND " + ServiceDbEntry.COLUMN_MEMORY_EFFICIENCY + " <= "
                                + memoryEff;
                Cursor cursor = db.query(ServiceDbEntry.TABLE_NAME, columns, querySql,
                                null, null, null, null);
                while (cursor.moveToNext()) {
                        services.add(cursorToService(cursor));
                }
                db.close();
                return services;
        }
public static void rankServices(List<WdService> services) {
                if (services != null && services.size() > 1) {
```

```
                        Collections.sort(services, new Comparator<WdService>() {
                                @Override
                                public int compare(WdService ws1, WdService ws2) {
                                        return Double.compare(ws2.getScore(), ws1.getScore());
                                }
                        });
                }
        }
        private static String[] serviceColumns() {
                return new String[] { ServiceDbEntry._ID,
                                ServiceDbEntry.COLUMN_SERVICE_NAME,
                                ServiceDbEntry.COLUMN_BATTERY_EFFICIENCY,
                                ServiceDbEntry.COLUMN_MEMORY_EFFICIENCY,
                                ServiceDbEntry.COLUMN_FRAMES_SUPPORT,
                                ServiceDbEntry.COLUMN_COOKIES_SUPPORT,
                                ServiceDbEntry.COLUMN_CALLBACK_SUPPORT,
                                ServiceDbEntry.COLUMN_XMLHTTP_SUPPORT,
                                ServiceDbEntry.COLUMN_TABLES_SUPPORT,
                                ServiceDbEntry.COLUMN_WEB_SERVICE_ID};
        }
public static WdService cursorToService(Cursor cursor) {
                int idIndex = cursor.getColumnIndex(ServiceDbEntry._ID);
                int webServiceIdIndex = cursor.getColumnIndex(ServiceDbEntry.COLUMN_WEB_SERVICE_ID);
                int index1 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_SERVICE_NAME);
                int index2 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_BATTERY_EFFICIENCY);
                int index3 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_MEMORY_EFFICIENCY);
                int indexS1 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_FRAMES_SUPPORT);
                int indexS2 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_COOKIES_SUPPORT);
                int indexS3 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_CALLBACK_SUPPORT);
                int indexS4 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_XMLHTTP_SUPPORT);
                int indexS5 = cursor.getColumnIndex(ServiceDbEntry.COLUMN_TABLES_SUPPORT);
                WdService service = new WdService(cursor.getString(index1),
                                cursor.getInt(index2), cursor.getInt(index3),
                                cursor.getInt(indexS1), cursor.getInt(indexS2),
                                cursor.getInt(indexS3), cursor.getInt(indexS4),
                                cursor.getInt(indexS5));
```

```
                service.setId(cursor.getInt(idIndex));
                service.webServiceId = cursor.getInt(webServiceIdIndex);
                return service;
        }
public static boolean putAction(String action) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                ContentValues values = new ContentValues();
                values.put(ActionDbEntry.COLUMN_ACTION_NAME, action);
                values.put(ActionDbEntry.COLUMN_STATUS, 0);
                long insertId = db.insert(ActionDbEntry.TABLE_NAME, null, values);
                db.close();
                return (insertId != -1);
        }
public static Map<Integer, String> vals;
public static void getActions(Future<Boolean> futurable) {
                SQLiteDatabase db = dbHelper.getReadableDatabase();
                String columns[] = { ActionDbEntry._ID,
                                ActionDbEntry.COLUMN_ACTION_NAME, ActionDbEntry.COLUMN_STATUS };
                Cursor cursor = db.query(ActionDbEntry.TABLE_NAME, columns, null, null,
                                null, null, null);
                vals = new HashMap<>();
                while (cursor.moveToNext()) {
                        int idIndex = cursor.getColumnIndex(ActionDbEntry._ID);
                        int actionIndex = cursor.getColumnIndex(ActionDbEntry.COLUMN_ACTION_NAME);
                        vals.put(cursor.getInt(idIndex), cursor.getString(actionIndex));
                }
                try {
                        boolean isProceed = futurable.get();
                        if (isProceed) {
                                // Remove those rows
                                try {
                                        String delimIds = "", delim = "";
                                        for (Map.Entry<Integer, String> entry : vals.entrySet()) {
                                                delimIds += delim + "" + entry.getKey();
                                                delim = ", ";
                                        }
```

```
                                        db.delete(ActionDbEntry.TABLE_NAME, ActionDbEntry._ID
                                                        + " IN (" + delimIds + ")", null);
                                } catch (Exception e) {
                                }
                        }
                } catch (Exception e) {
                }
                vals = null;
                db.close();
}
/**
 * This method will filter & rank services according to battery and memory level of the device
 *
 * @param services
 * @param batteryLevel
 * @param memoryLevel
 * @return
 */
public static Rt filterAndRankServices(List<WdService> services, int batteryLevel, int memoryLevel) {
                Rt rt = new Rt();
                List<WdService> loose = new ArrayList<>();
                if (services != null && services.size() > 1) {

                        long startTime = System.currentTimeMillis();

                        List<WdService> list = new ArrayList<>(services);

                        for (Iterator<WdService> iterator = list.iterator(); iterator.hasNext();) {
                                WdService thatService = iterator.next();
                           if (thatService.getBatteryEff() <= batteryLevel
                                        && thatService.getMemoryEff() <= memoryLevel) {
                                loose.add(thatService);
                           }
                        }
                                Collections.sort(loose, new Comparator<WdService>() {
```

```
                @Override
                public int compare(WdService ws1, WdService ws2) {
                    return Double.compare(ws2.getScore(), ws1.getScore());
                }
            });
            long endTime = System.currentTimeMillis();
            rt.elapsedTime = (endTime - startTime);
        }
        rt.serviceList = loose;
        return rt;
    }

}
```

```
window.onload = init;
var socket = new WebSocket("ws://localhost:8080/WebsocketHome/actions");
socket.onmessage = onMessage;

function onMessage(event) {
    var device = JSON.parse(event.data);
    if (device.action === "add") {
        printDeviceElement(device);
    }
    if (device.action === "remove") {
        document.getElementById(device.id).remove();
        //device.parentNode.removeChild(device);
    }
    if (device.action === "toggle") {
        var node = document.getElementById(device.id);
        var statusText = node.children[2];
        if (device.status === "On") {
            statusText.innerHTML = "Status: " + device.status + " (<a href=\"#\" OnClick=toggleDevice(" + device.id + ")>Turn
off</a>)";
        } else if (device.status === "Off") {
            statusText.innerHTML = "Status: " + device.status + " (<a href=\"#\" OnClick=toggleDevice(" + device.id + ")>Turn
on</a>)";
        }
    }
}
function addDevice(name, type, description) {
    var DeviceAction = {
        action: "add",
        name: name,
        type: type,
        description: description
    };
    socket.send(JSON.stringify(DeviceAction));
}
function removeDevice(element) {
    var id = element;
    var DeviceAction = {
        action: "remove",
        id: id
    };
    socket.send(JSON.stringify(DeviceAction));
}
function toggleDevice(element) {
    var id = element;
    var DeviceAction = {
        action: "toggle",
        id: id
    };
    socket.send(JSON.stringify(DeviceAction));
}
function printDeviceElement(device) {
    var content = document.getElementById("content");
        var deviceDiv = document.createElement("div");
    deviceDiv.setAttribute("id", device.id);
    deviceDiv.setAttribute("class", "device " + device.type);
    content.appendChild(deviceDiv);
    var deviceName = document.createElement("span");
    deviceName.setAttribute("class", "deviceName");
    deviceName.innerHTML = device.name;
    deviceDiv.appendChild(deviceName);
    var deviceType = document.createElement("span");
    deviceType.innerHTML = "<b>Type:</b> " + device.type;
    deviceDiv.appendChild(deviceType);
    var deviceStatus = document.createElement("span");
```

```
    if (device.status === "On") {
        deviceStatus.innerHTML = "<b>Status:</b> " + device.status + " (<a href=\"#\" OnClick=toggleDevice(" + device.id +
")>Turn off</a>)";
    } else if (device.status === "Off") {
        deviceStatus.innerHTML = "<b>Status:</b> " + device.status + " (<a href=\"#\" OnClick=toggleDevice(" + device.id +
")>Turn on</a>)";
        //deviceDiv.setAttribute("class", "device off");
    }
    deviceDiv.appendChild(deviceStatus);
    var deviceDescription = document.createElement("span");
    deviceDescription.innerHTML = "<b>Comments:</b> " + device.description;
    deviceDiv.appendChild(deviceDescription);
    var removeDevice = document.createElement("span");
    removeDevice.setAttribute("class", "removeDevice");
    removeDevice.innerHTML = "<a href=\"#\" OnClick=removeDevice(" + device.id + ")>Remove device</a>";
    deviceDiv.appendChild(removeDevice);
}
function showForm() {
    document.getElementById("addDeviceForm").style.display = '';
}
function hideForm() {
    document.getElementById("addDeviceForm").style.display = "none";
}
function formSubmit() {
    var form = document.getElementById("addDeviceForm");
    var name = form.elements["device_name"].value;
    var type = form.elements["device_type"].value;
    var description = form.elements["device_description"].value;
    hideForm();
    document.getElementById("addDeviceForm").reset();
    addDevice(name, type, description);
}
function init() {
    hideForm();
}
```