UNIVERSITY OF LONDON THESIS

Degree phd Year 2007 Name of Author KUN YANG

University College London

University of London

Department of Electronic and Electrical Engineering

Policy-based Model-driven Engineering of QoS-aware Pervasive Services for Wireless Networks

By

Kun Yang

Supervisor: Professor Chris Todd

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering of the University of London, and for the Diploma of the University College London.

London, United Kingdom

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Declaration

I, Kun Yang, confirm that the work presented in this Thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the Thesis.

Signature:

Print Name: Kun Yang

Abstract

While the Internet has evolved into an essential service delivery infrastructure, the future success of mobile wireless networks also largely relies on their ability to provide customisable and self-adaptable services (named as pervasive services) in a cost-effective manner. The traditional static service engineering approach has become inadequate due to their problems related to adaptability, scalability and quality-of-service. This Thesis proposes to tackle these problems from the internal logic of pervasive services themselves by employing a combination of policies and models.

This Thesis investigates how a pervasive service itself is represented as a policy-based management (PBM) system and how its creation is (semi-)automated by models of OMG Model-driven Architecture (MDA). A novel integration of PBM and MDA for the benefit of pervasive service engineering is proposed and validated. The output of the above creation is an executable sketch of a pervasive service with a generic policy decision engine reasoning over policies as its core. Service (re-)composition, together with loosely coupled policies for service description, provides a flexible and comprehensive mechanism to achieve context-awareness.

The constituent components of a composed service can be discovered on the fly, using two complementary ad hoc service discovery algorithms based on the pull method and the push method respectively. The service discovery in this Thesis reuses the MDA models that are generated during static service creation stage and makes them available to the dynamic stages of the service lifecycle such as service discovery and service re-composition. Another contribution of this Thesis is the investigation of a comprehensive Quality of Pervasive Service (QoPS) model that integrates the QoS parameters perceived by end users and network QoS parameters. This QoPS model has been utilized for service component selection.

The system is evaluated and validated both analytically and experimentally illustrating the methodology's feasibility, the system's performance effectiveness and efficiency.

Acknowledgement

My first and foremost thanks go to my thesis supervisor, Professor Chris Todd, the head of the Networks and Services Group in E&EE-UCL. I feel extremely luck to have Prof. Todd as my supervisor, who is always dedicated himself to the success of his students. I would like to sincerely thank him for his consistent and invaluable guidance and support during my entire PhD research and study. I would especially thank him for his inspiration and encouragement when I experienced the most devastating thing in my life. Without these encouragements and support this Thesis would not have been possible. Prof. Todd has provided me with a perfect balance between guidance and freedom, which not only allows me to pursue my own ideas along the right direction but also enables my personal career flourish. I will always be grateful for his consistent encouragement and his confidence in me.

I would like to thank Professor Izzat Darwazeh of E&EE-UCL for his helpful discussions on the ideas in my transfer thesis. His comments during my transfer viva have greatly helped the improvement of my work. I would also like to extend my special thanks to Professor Alex Galis of the Networks and Services Group in E&EE-UCL. He was amazingly managing several big European Union (EU) projects in which I had the opportunity to work. I am grateful to him for the inspiring discussions that, both technically and spiritually, helped me in generating some of the ideas of this Thesis.

I would like to extend my special thanks to Professor George Pavlou of the Network Research Group in University of Surrey, from whom I have learned a lot. I also had the opportunities to work with Professor Joan Serrat (Universitat Politècnica de Catalunya (UPC) - Spain), Dr. Laurent Levefre (The French Institute for Research in Computer Science and Control (INRIA) - France), Dr. Ekhard Muller (FhG-Focus, Germany), Arto Juhola (Technical Research Centre of Finland (VTT) - Finland), amongst many other experts in the networks and services areas via several EU IST projects or project proposals. I was grateful to them for the insightful discussions and joyful research experiences.

My gratitude goes to all the current and previous members of the Networks and Services Group of E&EE-UCL, especially Dr. David Griffin, Dr. Walter Eaves, Dr. Richard Lewis, Zahra Norris, Ali Hassani, Lawrence Cheng, Wilson Lim, Dr. Alvin Tan, Dr. Nikos Vardalachos, Ghadah Aldabbagh, Kerry Jean. I thank them for their

support, assistance and insightful discussions. I also thank them for their moral support during my PhD study.

Last but not the least, my most profound gratitude goes to my family. I am indebted to my always beloved parents for their greatest love, support and encouragement. Great thanks go to my dear wife, Xin Guo, who has sacrificed so much of her own time to support me in pursuit of the success in my research and career, and to my lovely son, Halvin Yang, whose happy face always brings me joy and relaxation. I owe my today's achievement to them and this dissertation is dedicated to them.

To my parents, Xin and Halvin

List of Abbreviations

AHBP	Ad-hoc Broadcast Protocol
AP	Access Point
BRG	Broadcast Relay Grid
BRL	Broadcast Relay List
BRN	Broadcast Relay Node
BS	Base Station
CCST	Conjunctive Condition Sub-Tree
CLD	Cross-layer Design
CIM	Computation Independent Model
CST	Condition Sub-Tree
CUST	Compare Unit Sub-Tree
DHCP	Dynamic Host Configuration Protocol
DiffServ	Differentiated Services
DMTF	Distributed Management Task Force
DSL	Domain Specific Language
DVB	Digital Video Broadcasting
EAOT	Extended AND-OR Tree
EMF	Eclipse Meta-model Facility
GMF	Graphical Modelling Framework
HCI	Human Computer Interface
iCAR	integrated Cellular and Ad Hoc Relaying System
idPSIM	implementation-dependent PSIM
IETF	Internet Engineering Task Force
iiPSIM	implementation-independent PSIM
IntServ	Integrated Services
JVM	Java Virtual Machine
MANET	Mobile Ad-hoc Networks
MBSD	Model-based Service Discovery
MDA	Model-driven Architecture
MDT	Message Duplication Table
MH	Mobile Host

MOF	Meta-object Facility
MSB	Module Sub-Tree
OCL	Object Constraint Language
OMG	Object Management Group
OPES	Open Pluggable Edge Services
OSS	Operational Support Systems
PAST	Policy Action Sub-Tree
PBM	Policy-based Management
PCST	Policy Condition Sub-Tree
PDA	Personal Digital Assistant
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIM	Platform Independent Model
PS	Pervasive Service
PSB	Policy Sub-Tree
PSIM	Pervasive Service Information Model
PSM	Platform Specific Model
PV	Policy Variable
QoPS	Quality of Pervasive Service
QoS	Quality of Service
SA	Service Advertisement
SAM	Service Advertisement Message
SCE	Service Creation Environment
SDA	Service Discovery Agent
SEE	Service Execution Environment
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SREQ	Service REQuest
SUT	Service Update Table
TSIT	Transmitted Service Information Table
TTL	Time To Live
UMTS	Universal Mobile Telecommunications System
UPnP	Universal Plug and Play
VOHE	Virtual Office and Home Environment

WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Networks
WPAN	Wireless Personal Area Networks
WSDL	Web Service Description Language
XMI	XML Metadata Interchange
XML	eXtensible Mark-up Language
XOCL	Executable Object Constraint Language
XSD	XML-based Service Description

Table of Contents

Table of Figures

Chapter 1. Introduction

This Thesis focuses on how to engineer pervasive services that run in mobile and wireless environment, including issues related to creation, discovery, selection, composition, adaptation and QoS assurance of pervasive services. The resulted work is collectively named POETRY - a QoS-aware pervasive service engineering framework. The design and implementation of the POETRY framework is motivated by users' need for mobile and customizable application services which are further driven by the increasingly ubiquitous presence of mobile wireless networks and devices on one hand, and service developers or providers' need to fulfil the above requirements from the customers in a fast and cost-effective manner on the other hand. The objective of the POETRY framework is to provide an easy way for service providers to develop services that are adaptable to their executing environments and customers' needs.

This chapter introduces the research motivations, reviews in a high level the existing solutions, outlines the POETRY framework, and summarizes the major contributions of the Thesis. The chapter finishes with an outline of the rest of the Thesis.

1.1. Motivations

Recent years have seen a significant increase in research and development of mobile and wireless networks, including 3/4G cellular networks, IEEE 802.11 Wireless LAN (WLAN or WiFi), IEEE 802.15 Wireless PAN (Personal Area Networks), IEEE 802.16 WiMAX (Worldwide Interoperability for Microwave Access), IEEE 802.20 (also referred to as MobileFi), mobile ad hoc networks, sensor networks, etc. DVB (Digital Video Broadcasting) is also increasingly regarded as a carrier for IP data. The future success of these networks lie in its ability to provide users with cost-effective *services* that have the potential to run anywhere, anytime and on any device without (or with little) user attention across multiple access networks. Services of these features are termed as *pervasive services* (PS), which is an active branch of pervasive (or ubiquitous) computing [Satyanarayanan01], [Banavar00]. A pervasive service can be a simple one such as helping a user on a mobile device (such as a PDA (Personal Digital Assistant) or a smart phone) find his/her favourite restaurant in vicinity based on his/her current

location or simply find a nearby printer to print out his/her presentation slides. However, it is more of a composite of the abovementioned small services (the latter are called component services), for example, a Virtual Office and Home Environment (VOHE) service that could enable Julie on her PDA to continue her normal office work such as dealing with emails, editing meeting memos left on her office desktop machine even when she is at the Heathrow Airport waiting for boarding.

Obviously, seamless and cost-effective access to network resources, in an open network neutral way, is needed to build revenue generating pervasive services. To attain this goal, industrial and academic communities are required to address two major and complementary infrastructural challenges: networks and services. The first one is more concerned about the networks where the technological integration of different types of the emerging networks constitutes its main challenge. The second one is focused on the design and implementation of a technology-neutral open service framework or infrastructure that supports easy creation, composition and autonomic execution (and adaptation) of the services operating over these networks. The work of this Thesis focuses mainly on the second challenge, i.e., services. However, the pervasive service engineering is carried out in this Thesis in a *cross-layer manner* that considers also the wireless network characteristics. Moreover, the Thesis is concerned about the methodology and enabling techniques/frameworks that enable the creation of pervasive services in an easy and effective manner and the execution of these services in an *adaptive* way.

Service *adaptability* is an important characteristic of pervasive services and it is highly coupled with *mobility*. For instance, as users move from one place to another, they will expect their tasks to logically "follow" them around – as in the above VOHE scenario. This requires the tasks to be able to adapt themselves to the new environment. Such adaptability requires a shift of service development from large, monolithic applications with fixed user interface mode and task logic (no matter how comprehensive it may be) to a new methodology where the service logic is constructed on the fly from collections of loosely coupled small components as driven by its context changes. This is because in a mobile environment it is hard for service developers to know at the service development stage which kind of environment the service user will be in and when. In other words, a pervasive service should grant its users as much flexibility as possible. An ideal pervasive service would be able to seamlessly support the following extended VOHE scenario where when Julie arrives at Barcelona and needs to fill in a Spanish-written form she could be able to use the Spanish-to-English

translator available within the public hot-spot area or a fellow visitor John's PDA which happens to have this kind of translator and locates within her PDA's one-hop wireless transmission range.

How to provide adaptability to services is still a largely unsolved problem despite the enormous research efforts in this area. Most researches are concerned with a powerful middleware infrastructure sensing the interesting contextual changes and adapting accordingly, such as One.world [Grimm04], Semantic Space [Wang04], PICO middleware for pervasive computing [Kumar03]. Big projects such as the MIT Oxygen project [Dertouzos99] and the Portolano Project [Elser99] also require strong support from a middleware-like infrastructure. Usually, these middleware and their underlying context-gathering mechanisms are proprietary, making the services offered by different service providers hardly interoperate, and consequently leading to duplicated development of software component (and sometimes hardware such as sensors). Furthermore, the development of a middleware that operates on a variety of pervasive service operating environments can be cumbersome or in some cases impractical or unaffordable.

While the necessity of the presence of a middleware infrastructure is appreciated, this Thesis attempts to tackle the adaptability requirement of pervasive services from another angle, i.e., the internal logic of pervasive service itself, which is largely neglected in the related research work. The fundamental concept in this Thesis is to grant the adaptability to services themselves rather than their execution environment aiming to leave the latter as clean and simple as possible (ideally only standard JVM – Java Virtual Machine, plus some Java-based service discovery mechanism). It is believed that pervasiveness comes from simplicity. A simpler environment stands a much better chance of being more widely deployed – a principle that has been proved true in practice. A simple service execution environment (SEE) will also leave service developers more space to design their services without being too much confined by implementing technologies or network heterogeneity. The time for new services to be deployed on market will be significantly reduced if there is very little or no needs of building a new SEE to support these new services. The work presented in this Thesis strives to complement the state-of-the-art research works in pervasive service engineering by proposing, for the first time (to be best of the author's knowledge), to shift the focus of pervasive service engineering from the external service execution

environment to the internal service logic itself. A prototype system that fulfils the above objective is designed, implemented and evaluated.

1.1.1. Technical Motivation

The next question then is how to grant the high-level adaptability (or intelligence as it can be regarded as) to the service logic. The Thesis proposes to utilize *policies* (in the form of rule: IF <conditions> THEN <actions>) to describe the pervasive service logic at a higher level and to use *models* to enhance the information model (providing implementation) of the pervasive service. Policy employed here is a concept derived from Policy-based Management (PBM), a successful technique widely researched for distributed systems and IP network management [Yang02]. A policy is employed to define a choice in the behaviour of a pervasive service. Models utilized here specifically refer to a modern software engineering method led by OMG (Object Management Group) Model-Driven Architecture (MDA) [MDA]. MDA's promising platform-, language- and middleware-neutral features is making a big impact on the current methods and techniques applied to manage software development process [Orriens03], [Georgalas04].

Given the resource-constrained nature of mobile devices (e.g., battery, bandwidth and computation, storage) when compared with their desktop PC siblings, it is impossible for mobile devices to maintain as many services as stationary desktop PCs do. However, service users would still like their services following them wherever they are and whatever mobile devices they are on. As a result, mobile services are deemed to borrow or buy services from others (mostly mobile devices as well). These newly imported services will need to be integrated seamlessly with their existing services to provide consistent services to user's satisfaction. Service discovery is an essential mechanism to find component services needed by a pervasive service on the fly. In POETRY, a novel methodology for service discovery, i.e., *model-based service discovery* (MBSD), is introduced. This model-based service discovery approach matches with the abovementioned model-driven service creation and also simplifies the service description procedure.

These pervasive services typically involve accessing information available in another machine across networks. For instance, the following enhanced VOHE scenario is not uncommonly expected in real life. When Julie is walking through a park on her way to work, she suddenly receives a text message from her son saying that a video clip of his piano performance is available on their home server. Then Julie would like to

watch it on her PDA or 3G mobile phone right in the park or whatever she is in. A flexible VOHE service, in connection with its underlying networks, will have to be geared to provide this kind of services to the user's satisfaction (e.g., Julie apparently would like to see a reasonable picture of her son performing and good sound). The dominant services to be requested by the future mobile users will involve transmission of high-speed, content-rich and burst-type multimedia traffic across wireless networks. An essential mechanism to support these services is Quality of Service (QoS). While much effort has been made on the network QoS and there is emerging effort studying application-level QoS, little research has been seen on the seamless integration of these two efforts into a unique QoS model for more complex and dynamic pervasive services. This Thesis attempts to make an effort towards this combined model to express the *quality of pervasive services (QoPS)*. A comprehensive QoPS model facilitates the QoS-aware service selection algorithm utilized in POETRY.

1.2. Research Challenges

G. Banavar *et al.* presented a vision of pervasive computing along with a new application model that supports this vision and a set of challenges that must be met in order to bring the vision to reality [Banavar00]. Though more than six years have passed since these challenges were presented, most of them are still largely unsolved. At the same time, new challenges are emerging as new types of wireless networks makes their appearance and new service requirements arise from end users. The specific research challenges this Thesis has been focused on are described as follows.

- **Pervasive service adaptability**. Service adaptation is a necessity for any adaptive service such as pervasive services. Fulfilling this mission solely via service middleware is hardly a universal solution given the wide range of diversities imposed by users, devices/networks, service content and software techniques. The first challenge then is what the best way is to maximize the adaptability of pervasive services while not causing significant change or imposing much requirement to service supporting environments. Furthermore, this service adaptation mechanism should be able to accommodate various constraints or requirement imposed to a service, some of which might be conflicting amongst themselves, while still trying to best serve the service users at a cost acceptable to network operators. A methodology and the corresponding mechanism to fulfil this vision need to be investigated.

- **Service lifecycle management**. A pervasive service will typically go through the following lifecycle: service creation, service deployment, service execution, service discovery, service dynamic composition, and service adaptation. In addition to these, service description is also critical for service discovery, selection and composition. An effective method needs to be investigated that can thread all these service engineering stages together to provide a consistent and efficient mechanism for pervasive service lifecycle management. Some questions that need to be answered within this challenge include: how to describe a pervasive service in a universal manner; how to design an efficient service discovery mechanism that considers also network features and conditions; is it possible to design and to create a service in such a manner that it would make easier the service re-composition and integration in a later stage. In addition, a service creation and execution environment that fulfils the above missions will also be welcomed by service providers and developers.

- **Ad hoc service discovery**. Many service discovery mechanisms have been proposed, both for wired networks and for wireless networks. While service discovery mechanisms for wired networks have been mature and widely utilized, service discovery for wireless mobile networks, especially in an ad hoc manner, is still a largely unexploited field. Furthermore, the existing service discovery mechanisms have adopted various service description languages, thus limiting the scope of global service discovery. Most importantly, the current service discovery mechanisms all fail to provide incentive for service composition. In contrast, the discovered service components as a result of any service discovery will typically be used to compose new services (including integrating with the current user services).

- **Quality model of pervasive services.** In order to satisfy user requirements, QoS mechanisms have to be in place. Much research work regarding QoS has been conducted from the network's perspective. But the quality of pervasive services will ultimately need to reflect end users' perception of the service. As such a comprehensive quality model for pervasive services (QoPS) should take into consideration the parameters perceived by both end users and wireless networks. QoPS model is fundamental to almost all aspects of service lifecycle. A proper QoPS can provide a more accurate service differentiation and as such beneficial to service discovery, service selection, service composition, etc. QoS-aware

service selection and composition is carried out at the application layer, which imposes new challenges that differ from the traditional network QoS problems. QoS criteria for pervasive services and the corresponding QoS-aware service selection algorithms pose another challenge.

1.3. Setting the Scope

As its title (Policy-based Model-driven Engineering of QoS-aware Pervasive Services for Wireless Networks) indicates, the Thesis involves using two major techniques, namely, *policy-based management* and *model-driven architecture*, to engineer *pervasive services* that are *QoS-aware* and operate in *wireless network* environment. Each topic itself, as marked in italic, makes a separate and stand alone research area. It is not this Thesis's intention to delve into each topic but rather to utilise PBM and MDA as enabling tools to facilitate the engineering of pervasive services. In this Thesis particular emphasis is placed on pervasive services that operate in wireless networks; such services will usually need to provide quality of service to satisfy users' requirements. How to engineer *adaptive* pervasive services is the core of this Thesis. All the other materials which might relate to another separate research field are threaded around this primary objective and are researched to different depths, as driven by the needs of pervasive service engineering.

Related work on these topics is to be discussed in the next chapter. The purpose of this section is to set the scope in terms of what is to be investigated in this Thesis for each of the above key technologies.

Pervasive Services: this Thesis embarks on a specific and active branch of pervasive computing, i.e., pervasive service engineering. The main focus is on *service adaptation* and this service adaptation is achieved by considering not only the dynamic stage of the pervasive service lifecycle, namely, during service execution, but also the static stage such as *service creation*. The speciality of the service creation mechanism of POETRY (particularly the use of MDA) leads to the introduction of model-driven *service discovery*. Another factor that makes the research on service discovery a necessity in this Thesis is wireless networks whose service discovery mechanisms are relatively less exploited. A model for describing the quality of a pervasive service is also investigated, using which a *service selection* algorithm is proposed for the purpose of service composition.

Wireless Networks: the research on wireless networks themselves falls outside the scope of this Thesis. However, since the pervasive services run over wireless networks

the effectiveness and efficiency of a pervasive service can hardly be independent of its underlying networks. As such some existing work on wireless network algorithms is extended, when necessary as driven by service engineering, such as the efficiency of *service* request broadcasting in Chapter 4, the efficiency of *service* advertising in Chapter 5, and QoS modelling of pervasive *services* in Chapter 6. As far as the type of wireless networks is concerned, this Thesis is mainly based on mobile ad hoc networks (MANETs). However, the fundamental principles apply to other types of wireless networks.

Policy-based Management: since a pervasive service in this Thesis is designed and developed as a PBM system some fundamental changes have been made to the current literature of PBM, which lie in the following three aspects. Firstly, a new policy language is to be proposed that is designed specifically for describing behaviours of pervasive services. Secondly, a practical information model that takes advantage of MDA methodology is to be investigated. Thirdly, a reasoning engine carrying out policy decision making is to be designed. Refer to Section 2.1 for related work discussion and the motivation behind these changes.

Model-driven Architecture: The researches on MDA can be roughly divided into two schools. One researches on MDA itself, such as meta-modelling, constraints using object constraint language, semantic mapping between different models, and MDA toolkits, etc. The other is more concerned with the application of MDA technique to certain application domains. This Thesis falls into the second category and is particularly interested in how to utilize MDA to implement pervasive services. In particular, this Thesis is more concerned with the higher-level models such as how to create a meta-model for pervasive service domains and how to create a platform independent model for a given service scenario whereas leaving the modelling mapping and its formal validation work to be carried out by a particular MDA toolkit. Furthermore, these high-level models are derived in a more engineering manner rather than investigating their semantics using formal methods. Therefore ontology-related issues in MDA are not of concern in this Thesis. However, the development of ontology and semantic web researches, especially these related to pervasive services, could equally benefit the POETRY platform by e.g., providing a universal terminology or naming set in the course of service description. Living with the lack of such ontology orientated for pervasive services in MDA communities, this Thesis derives a simple meta-model for pervasive service domain (also called DSL or domain specific language) based on service scenario analysis.

1.4. POETRY Overall Functions and Major Contribution

POETRY is a QoS-aware framework that performs the lifecycle management of pervasive services. POETRY is composed of two independent parts (refer to the POETRY system architecture in Chapter 2): one is to deal with the static aspects of the service engineering and the other is to cope with the dynamic features of service engineering such as service adaptation. The former is implemented as an off-line Service Creation and deployment Environment (SCE or SCE-P to specifically denote a SCE for POETRY), and this part is developed for service developers or providers. The latter is a much simplified distributed middleware framework for ad hoc service discovery – the purpose of service discovery is for service adaptation.

The major functions provided by POETRY include: (1) *service creation*, which can take as input the service description following a given format and automatically generate an executable service skeleton; (2) *pull-based service discovery*, which utilizes client-initiated queries (keyword-based) to discover component services in wireless ad hoc network environments; (3) *push-based service discovery*, which, as a complementary to the above pull-based service discovery method, employs various service advertisement mechanisms to reduce the service discovery time but at a cost of an increased traffic overhead; (4) *service selection*, which can select component services needed to fulfil certain functions arising during the service execution as a result of context change (including a user initiating a new task) - and the selected component services are to integrate with the current pervasive service. The generated services as a result of SCE-P have the following functions embedded inside: a service execution engine, the client side of the POETRY service discovery mechanism, a POETRY service selection module, etc. The POETRY service discovery and adaptation operate over mobile wireless networks and make use of the existing wireless network protocols (such as ad hoc network routing) to fulfil the application-layer tasks.

To assist the above major functions, POETRY also provides a set of enabling functions: (1) *service description*, which allows service developers to easily specify the service behaviours in both normal environment and exceptional circumstance such as battery running out, and to specify administration requirements related to such as security and pricing; (2) *service deployment*, which provides a web-based means for service customers to subscribe to the pervasive services created and to download necessary classes; (3) *monitoring* of the corresponding QoS parameters and context information.

Other more fundamental functions that underpin the above functions include: (1) the compiler of the service description language, which takes service description as input and filters out all the context information and action components used by the service; (2) the policy-based service decision making engine for service execution, which reasons over policies to select the suitable policies whose actions are to be triggered; (3) the modelling of the quality of pervasive services. The POETRY system architecture implementing the above functions is to be presented in Chapter 2.

1.4.1. Major Contributions

In summary, the major contributions of POETRY are as follows:

- **Inherent adaptability for pervasive services.** POETRY advocates a shift of the focus of pervasive service engineering from the *external service execution and supporting environment* to the *internal service logic* itself. In this light it proposes to utilize policies to describe pervasive service logics and for the first time creates a service as a PBM system. In this manner inherent adaptability can be granted to pervasive services aiming to maximizing the service adaptability.
- **Pervasive Service Information Model (PSIM) based on MDA.** In order for a pervasive service to run as a PBM system, the corresponding class hierarchy and its implementation, or so called PSIM, need to be available. Instead of following the legacy IETF policy information models which were developed for network management, this Thesis proposes to take advantage of the emerging software engineering technique, namely, OMG (Object Management Group) MDA (Model Driven Architecture) to develop the PSIM. Doing so will bring into PSIM not only MDA's promising platform-, language- and middleware-neutral features but also the layered modelling approach which makes service implementation more flexible and compatible. In POETRY, a PSIM that reflects the policy nature of pervasive services while utilizing the MDA approach is proposed, which, as to be evidenced by a scenario service development, simplifies the pervasive service development procedure.
- **Model-based ad hoc service discovery.** The most salient point of this mechanism is the fact that it reuses the models that are generated during static service creation/development stage and makes them available to the dynamic stages of the service lifecycle such as service discovery and service re-composition. This is reflected in the following two main aspects. Firstly, rather than manually creating service description as it is in almost all other service discovery mechanisms, the service description in POETRY can be automatically

generated from these models and as such providing a more precise description of services. Secondly, the model information carried in POETRY service description can be used to regenerate original models that can be used, either entirely or partially, by other mobile applications locating in different places. In association with the above model-based method, this Thesis also presents two complementary service discovery algorithms based on the pull method and the push method respectively. The wireless mobile ad hoc nature of the underlying networks is considered when designing the POETRY pull and push algorithms.

- **A comprehensive quality model for pervasive services.** This model integrates the QoS parameters perceived by end users and network QoS parameters. The reason why network parameters are also considered when describing quality of application-level pervasive services (QoPS) is because the locations, mobility and error-prone transmission channel of wireless nodes all make significant impact on the availability and reliability of a service running in wireless mobile networks. Both network QoS parameters and user-perceived QoS parameters are quantified and integrated together for both elementary services that are the building blocks of pervasive services and composite services (pervasive services considered in this Thesis are more complex composite services). The creation of this QoPS model can be used for two main purposes. On one hand, it can be used to differentiate multiple services with overlapping or identical functionalities; and on the other hand and more importantly, it is essential for service composition process to compose a service that satisfies the QoS requirements from users.
- **A pervasive service development toolkit**. Using this toolkit, service developers can easily create an executable pervasive service skeleton. This skeleton bears a generic but intelligent decision making engine that underpins the adaptability of the pervasive service via an event-based context sensing mechanism. The corresponding service discovery algorithms are also embedded inside this skeleton and are triggered automatically in response to certain context-event change.

1.5. Outline of Thesis

The remainder of the Thesis is organized as follows. Chapter 2 starts with an analysis of the related work – both in terms of the challenges to be tackled themselves and their enabling techniques. Then it introduces the features of the wireless networks

for which POETRY is developed. Finally it discusses the essences of POETRY and presents the POETRY system architecture.

Chapter 3, Chapter 4, Chapter 5 and Chapter 6 detail the POETRY system. Chapter 3 presents the POETRY service creation environment. It firstly analyzes the features of pervasive services using a typical pervasive service scenario and models pervasive services as a domain-specific language using the MDA techniques. Then a policy-based pervasive service description language called PoPSiDL is presented, which is followed by a presentation of the POETRY pervasive service information model that facilitates the implementation of the described services. As the unique "intelligent" part of a pervasive service, the policy-based pervasive service decision making algorithm is also designed in this chapter. Chapter 3 finishes with a comprehensive presentation of the POETRY service creation environment and its work flow.

Chapter 4 and Chapter 5 present two complementary service discovery algorithms in wireless mobile ad hoc networks. Chapter 4 focuses on a reactive, pull-based service discovery algorithm, i.e., model-based service discovery (MBSD). Chapter 5 describes a proactive, push-based service discovery algorithm, which utilizes service advertisement to reduce the service discovery time. If there is more than one service providing the similar or identical function to the requested task then service selection is needed, which is the focus of Chapter 6. After proposing a quality model for pervasive services, Chapter 6 discusses a QoS-aware service selection algorithm for service composition. Finally, Chapter 7 concludes the Thesis and suggests some future work.

Chapter 2.

Related Work and the POETRY System Architecture

The work within POETRY is carried out under the umbrella of pervasive computing. So this Chapter firstly introduces the related background knowledge and the related work. The related work is discussed both in terms of the challenges focused by this Thesis, i.e., QoS-aware pervasive service engineering in wireless networks, and the enabling techniques adopted by this Thesis. Secondly, this Chapter presents an example pervasive network on which and for which the POETRY system is developed. Thirdly, the novelty of POETRY is discussed, which constitutes the essence of POETRY. Finally, the POETRY overall system architecture is presented, which also draws out the landscape for the following chapters.

2.1. Related Work

This section gives a high-level view of the related work and their comparison with the work proposed in this Thesis. Detailed discussion is conducted in the corresponding chapters in a closer comparison with the proposed technical details of POETRY.

2.1.1. Pervasive Computing and Service Engineering

The seminal project in pervasive computing is the ParcTab effort at Xerox PARC [ParcTab]. Recently, other efforts have begun under the designation of "Invisible Computing", which appears to pick up where the ParcTab effort left off. Big projects such as Oxygen [Dertouzos99] and Portolano [Esler99] from USA and the Ambient Networks [Niebert04] based in Europe are among them. Most of the researches in this area are more focused on building up a pervasive execution or supporting environment.

As far as middleware for pervasive computing is concerned, the Spectra project [Flinin02] proposes a remote execution system for mobile devices used in pervasive computing. Chen et al. proposed an offloading framework [Chen04] that focuses on a Java-based environment and dynamically decides whether to execute locally or remotely based on the computational complexity and communication channel conditions. [Grimm04] proposes a new pervasive computing architecture, one.world, that promises to provide an integrated framework for building adaptive applications. The adaptive applications developed by one.world bear similar features as pervasive

services discussed in this Thesis. Issues related to fault-tolerance in pervasive computing are discussed in [Chetan05]. A variety of technologies are used for building up pervasive computing environment and for implementing services and applications. For instance, software agents are utilized in [Genco06], whereas [Lee03] and [Pashtan04] embark on OSGi and web technology respectively. This Thesis adopts Java, which is low level in comparison with these more advance technologies, to exemplify the proposed principles for creating and adapting pervasive services.

A pervasive service will typically go through the following lifecycle: service creation, service deployment, service discovery, service composition, service adaptation and service management. Here *service creation* means the design and implementation of a service at compile time and not at the runtime, i.e. to generate the executable code for this service. After a service is created, it is deployed to the service user side (and probably to the supporting middleware if some special functions are needed) – and this procedure is usually called *service deployment. Service composition* refers to a dynamic procedure that composes a new service (or precisely a new part of a running service) from the components that are selected at run time. If the components needed by the new requirements of the service are not available at the design time then these components will have to be found at run time – and this is the function of *service discovery. Service adaptation*, while implemented by certain mechanism, usually describes the most important feature of pervasive services, i.e., the ability to adapt itself to the change of the environment (or context) in a dynamic manner. *Service management* usually means a collection of some or all of the above procedures.

One.world framework [Grimm04] also emphasizes that a pervasive computing framework must embrace contextual changes and not hide it from applications. *Context-awareness* is one of the most important characteristics of pervasive services. It is also regarded as a mechanism to fulfil service adaptation. Context-aware services need to consider the whole service lifecycle. The initial researches on context-aware computing focused mainly on the human-computer interface (HCI) [Dey00a]. Currently there is an increasing interest in the creation and development of context-aware services themselves which can be sold out by service providers and benefit a broader population [GuT04]. Furthermore, considering the researches in pervasive computing area as a whole, there has not been an open standard for any stage of pervasive service development lifecycle, let alone the whole lifecycle. This Thesis aims to contributing to some extent to an open pervasive service platform by adopting emerging technologies for creation, composition and adaptation of pervasive services. Such technologies

include IETF (Internet Engineering Task Force) policy-based management (PBM) method, OMG (Object Management Group) Model-driven Architecture (MDA), open source technique, among others. Furthermore, instead of being tightly confined by these technologies which are still under development, this Thesis proposes a novel way of integrating PBM and MDA, which is to be detailed in Chapter 3. The state-of-the-art of PBM and MDA are to be surveyed in the following sub-sections respectively.

Service Oriented Architecture (SOA) [Erl05] is an architectural style whose goal is to achieve loose coupling among interacting software components. A service in SOA is a unit of work done by a service provider to achieve desired end results for a service consumer. The loose coupling principle of SOA inspires the methodology of constructing pervasive service internal logic, i.e., describing a pervasive service as a set of loosely-coupled behaviour-describing policies.

2.1.2. Service Adaptation

Heterogeneous users and organisations in pervasive computing environment require a service to be presented in a variety of ways. Appropriate adaptation is therefore necessary for providing personalised services to users. Incorporating specific user or application information to service provision has drawn more attention from researchers in recent years. The work so far has been concentrated more on the interaction between middleware and applications. How to gather a variety of service-related information and synthesize them in service provision is still a new area requiring more investigations.

In this Thesis, service adaptation is to be carried out by two complementary mechanisms: *at the service creation stage*, policies are employed to describe the service behaviours (including user preferences about the service) in a user-friendly and non-technical manner, and models of different abstraction level are used to describe the implementation of the service in a platform- and technology-neutral way; *at the service execution stage*, dynamic service discovery and composition are in place to instantiate the service components missing at the service creation stage.

In the last few years, several initiatives have appeared that all have in common the use of polices or rule-decision based approaches to tackle the problem of fast and customisable service adaptation. Amongst them the most representative ones are OPES and e-Services. The Open Pluggable Edge Services (OPES) model [IETF-OPES], proposed by the IETF OPES Working Group, is a form of content services overlay network. The OPES defines the surrogate and the call-out server intermediaries in the

content path. The e-Services concept encapsulates the overall business activity behind the delivery of a service to a customer. The e-service model requires the representation for the whole interaction process between service provider and service consumer to be in a format automatically tractable. Harroud et al. [Harroud03] propose an agent-based service provisioning system for mobile users. It utilizes a set of cooperative agents distributed over different sites that work together to provide customized personal services for mobile users. The network concerned in this work is the Internet.

2.1.3. Policy-based Management (PBM)

The concept of PBM is not new. PBM reference architectures that cleanly isolate management logic from physical management operation have made their appearance in both the distributed system community [Sloman94] and the network community [IETF-policy] more than a decade ago.

Ponder is a language for specifying management and security policies for distributed systems and network management systems. It has been developed as part of ongoing research being carried out by the Policy Research Group in Imperial College London [Ponder]. Ponder includes authorisation, filter, refrain and delegation policies for specifying access control and obligation policies to specify management actions. Ponder thus provides a uniform means of specifying policies relating to a wide range of management applications – network, storage, systems, application. In addition, it supports a common means of specifying enterprise-wide security policy that can then be translated onto various security implementation mechanisms. Ponder is declarative, strongly-typed and object-oriented which makes the language flexible, extensible and adaptable to a wide range of management requirements [Damianou01]. As part of the Ponder development effort, a complete toolkit has been developed to support the users of the language. Ponder is a proprietary language/platform that is more concerned with security policies. No context information and service adaptation feature are considered in Ponder. Furthermore, it is not clear how the policy decision making is carried out. But many of its initial ideas will serve as the guidelines for the design of a context-aware policy definition language.

Other related work regarding policy definition languages is as follows. Policy Definition Notation (PDN) [Meyer94]: this approach is based on a distributed computing platform, and the authors chose the area of performance management for the application of policies. Their notation looks very much like a programming language as such details have to be defined by its users. Policy Template Definition [Weis94,

Weis95]: apart from the policy classification and the policy hierarchy, R. Weis also derived a policy template definition, which is then transformed into policy objects/management scripts. Policy Definition Language (PDL) [Koch96]: the work defines a three level policy hierarchy (or management hierarchy). This allows a stepwise refinement of policies from an informal strategic level to a formalised operational level: Requirements level, Goal-oriented level, Operational level. They use different language constructs to specify policies for each abstraction level. Their work is based on the early proposals for policy notation in Ponder.

However, the current PBM solutions are still not appropriate for service oriented computing though researchers have started to evolve PBM towards this direction. For instance, Yew *et. al.* [Yew05] propose a policy-based middleware for context-aware service. It makes policies go beyond the network management by employing policies to describe user preferences. However, the operational PBM components (PDP and PEP) in [Yew05] are still within the control of network operators, and the information model for policies is still confined by the IETF's policy information model. Note that IETF policy information model was designed mainly for network management, such as QoS control or IP VPN (Virtual Private Networks) of core IP networks, which does not have the diversity of wireless access networks. And in Yew's work, policies are used only to describe user preferences, whereas in this Thesis policies are applied to the full description of service logic. Kagal *et al.* [Kagal03] propose a policy language for pervasive computing applications. While this work tackles a different computing environment (i.e., a more dynamic mobile environment) from that of Sloman's policy system [Sloman94] [Damianou01], the objectives are the same, namely, it still focuses on the security aspects of the computing environment. Furthermore, its policy engine [Kagal03] is based on traditional Prolog, a logic programming language, which makes its syntax rigid (the syntax of this policy language is similar to that of Prolog). Because of this Prolog-related nature, the whole policy language is concerned about only reasoning (and for security purpose). There is no discussion as to how actions are carried out, namely no policy information model is mentioned.

As far as the PBM itself is concerned, this Thesis is not just to apply existing PBM techniques to the network aspect of pervasive services or to concentrate on the building up or extending the standard-based information model (e.g., Policy Core Information Model by IETF [IETF-policy], Core Information Model by DMTF (Distributed Management Task Force), Parlay policy management APIs). While being aware of the

importance of these two aspects of the PBM research, this Thesis gives specific emphasis on the policy-based description of pervasive services, and a practically functioning pervasive service information model. With regard to pervasive service description, a specific policy-based language is designed. A functioning pervasive service information model is built by using an emerging new software engineering technique called MDA (Model Driven Architecture).

This Thesis argues that rather than defining a standard and fixed class hierarchy that has to be followed by all service providers or network operators, which is the path IETF Policy Working Group is following, a more flexible way of defining information model and consequently implementing the information model is more appealing to the fast changing network and user environments. The fact that there is not any reference implementation for IETF policy core information model or DMTF Core Information Model, while making the PBM hard to walk into practice, also indicates to some extent the infeasibility of this path. Applying MDA for policy information model is an attempt towards a more flexible way of defining and implementing policy information model. In such a manner, information model (either for service or network) can easily take advantage of any new development in MDA-related research areas such as meta-data/ontology without having to resort to long-time standardization. At the same time and most importantly, the inherent platform-, language- and middleware-neutral features of MDA grants the most possible flexibility and openness to the policy information model.

2.1.4. Model-Driven Architecture (MDA)

MDA is an approach to IT (Information Technology) system development fostered by the OMG (Object Management Group). Its essence is the concept of separation between the specification of a system's essential functions as Computation Independent Model (CIM), Platform Independent Models (PIM) and the realisation of the system using more specific and detailed platform as Platform Specific Models (PSM). Its promising platform-, language- and middleware-neutral features is affecting the current methods and techniques applied to manage the software development process [Yang05a, Yang05b].

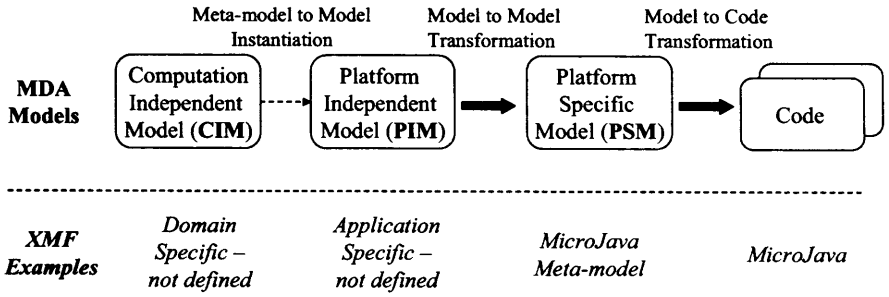


Figure 1: MDA (Model-Driven Architecture) Concept

Figure 1 illustrates the main concepts of MDA. Most MDA toolkits such as XMF [XMF] provide a PSM (namely the meta-model of the programming language to be generated) and the transformation from the PSM to code that implements the logic defined by PIM. However, since PIM is dependent on the service or application to be designed and implemented by MDA it is open to service developers but the corresponding design facilities are available in MDA toolkits. Figure 1 also illustrates the developing procedure using MDA. Typically, service developers start with the design of a CIM for the domain their applications will fall into. CIM is used to capture the requirement of a system in a specific domain [Miller03] and as such sometimes called Domain Specific Language (DSL). The design of a DSL is assisted by an analysis of domain features. Then specific applications can be designed by utilizing the entities (or classes) defined by the DSL. Since a DSL defines the meta-information of an application-oriented PIM, the DSL is also called the meta-model of this PIM. Then the transformation from PIM to PSM needs to be defined – again MDA toolkits such as XMF provide facilities for this.

MDA is regarded as a very promising future software development methodology. The software engineering community has seen an increasing number of software components being developed in MDA manner, including many activities in applying MDA for service engineering [Orriens03], and trialling MDA in OSS (Operational Support Systems) operated by network and service providers such as British Telecom [Georgalas04]. As far as model driven service composition is concerned, the majority of existing researches are based on web services [Orriens03]. This Thesis explores how the MDA approach can be used to facilitate the creation and composition of pervasive services. Here pervasive services are not in the form of web services.

In terms of MDA supporting software or platform, apart from the IBM Rational Rose and its UML (Unified Modelling Language), other more specific MDA tools have made their appearance in the market, such as XMF toolkit from Xactium Ltd. [XMF].

Open Source Alliance Eclipse also have several initiatives targeting MDA toolkit, such as EMF (Eclipse Meta-model Facility). Amongst a number of MDA tools currently available, this Thesis adopts XMF toolkit from Xactium Ltd. to illustrate the building-up of meta-models and models. XMF is a generic meta-programming environment that aims to support a wide variety of MDA development scenarios. It implements a MicroJava meta-model as a PSM (Platform Specific Model) language and its transformation to the corresponding Java source code. A PIM conformant to XMF PIM format can also be mapped to MicroJava PSM by XMF. Some open source MDA platforms in Eclipse are alternatives to XMF, such as, Eclipse Modelling Framework – EMF [EMF] and Graphical Modelling Framework – GMF [GMF], etc. These Eclipse modelling tools tend to be more general and flexible. Their further development will provide enormous momentum to model-driven pervasive services.

The POETRY is independent of the MDA tools but assumes that the adopted tool supports a Java PSM and its transformation to Java source code. Both XMF and EMF satisfy this requirement. This assumption enables the Thesis to focus on the high-level service information model (i.e., the PIM) without being troubled too much by its class-level implementation.

2.1.5. Web Services

Web service is a standards-based, service-oriented technology that integrates PCs, other devices, databases, and networks into one virtual computing fabric to be used by users via browsers [WS]. The services themselves usually run on Web-based servers (instead of PCs), therefore moving functions from the desktop to the Internet. Cross-platform capabilities (or interoperability) are one of web service's key attractions. Users could work with the services over any web service-enabled machine with Internet access, including handheld devices. Web services would thus change the Internet into a computing platform, rather than a medium in which users primarily just view and download content.

The majority of researches on service composition are concerned with web services [Zeng04] that typically run over the Internet. The platform neural nature of web services technology [WS] provides a good platform for enterprisers to develop their business process by directly using or combining existing web services that are offered by different service providers. Web services work fine on resource-rich wired networks. However, the performance of web services operating over resource-constrained mobile devices and wireless networks is a concern. The POETRY system complements this

service composition development by basing the service components directly on light-weight Java objects. The finer level of granularity can increase the performance of composed services. Instead of generating web services-oriented WSDL (Web Service Description Language) or even executable BPEL4WS (Business Process Execution Language for Web Services), Java bytecode is to be generated as a result of the POETRY pervasive service creation.

Another active research branch that is highly coupled with web services is semantic web [Decker00]. Its main aim is to provide computer-processable meanings to ordinary words used by web pages that are usually designed for human beings. Though there is no semantic web outcome dedicated to pervasive services, its general development is expected to provide guidance to the design of pervasive DSL.

2.1.6. Service Discovery

Service discovery is an essential mechanism to find on the fly the component services needed by a pervasive service. It has been broadly utilized by researchers and competing industries to pursue dynamic and automatic software system composition, configuration and adaptation [Helal02]. It impacts on the cost-effectiveness and efficiency of the creation, operation and maintenance of services of service providers. Many service discovery mechanisms have been proposed, firstly for wired networks such as UPnP, Jini, Solute [Helal02], and then for wireless networks such as DEAPspace [Nidd01], Konark [Lee03]. However, the current situation of service discovery to some extent hinders the services or the service supporting systems from pursuing their maximum effectiveness. Existing service discovery mechanisms have adopted various service description languages, thus limiting the scope of global service discovery. Though each of them has its advantages and disadvantages in terms of expressiveness and query efficiency, they are primarily bound to the architecture of the individual discovery mechanism. Most importantly, the current service discovery mechanisms all fail to provide incentive for service composition. In contrast, the discovered service components as a result of any service discovery will typically be used to compose new services (including integrating with the current user services). It is natural to have a service discovery mechanism that inherently provides such insight so as to ease the difficulty of service composition.

Motivated primarily by the above considerations, this Thesis introduces a novel methodology for service discovery, i.e., model-based service discovery (MBSD). A model-based service discovery approach matches with the emerging trend of modern

software engineering, i.e., model-driven computing such as OMG MDA, which is also utilized by pervasive service creation. The most salient point of this methodology is the fact that it reuses the models that are generated during static service creation/development stage and makes them available to the dynamic stages of the service lifecycle such as service discovery and service re-composition.

2.1.7. Quality of Service (QoS)

Service composition mechanisms enable *individual services* to be composed together to generate a more powerful service (*composite service*). In this Thesis, pervasive services are composite services. *QoS-aware service composition* refers to a problem as to how to select a set of appropriate services to instantiate composition logic while satisfying user's QoS requirements. Some researches have been conducted in the web services field [Zeng04]. However, not too much effort has been given to the service selection and composition for pervasive services that run over wireless mobile networks.

As far as QoS is concerned, much work has been carried out at the network layer both in wired network (i.e., the Internet) [QoSForum] and in wireless mobile ad hoc networks (MANETs) [Chakrabarti01]. For instance, originated for the best-effort wired Internet, Integrated Service (IntServ) provides guaranteed bandwidth for each flow whereas Differentiated Services (DiffServ) offers guarantees on a service class basis [QoSForum]. Recent years have also seen investigation on user-perceived service quality. For instance, O'Sullivan *et. al* [O'Sullivan02] and Zeng *et. al* [Zeng04] describe service quality as a broad concept that encompasses a number of non-functional properties such as availability, reliability, reputation and price. However, a comprehensive quality model for pervasive services that integrates *both* network conditions and user perception is missing. The reason why network parameters are also considered when describing the quality of application-level pervasive services is because the locations, mobility and error-prone transmission channel of wireless networks all make significant impact on the availability and reliability of a service running in wireless mobile networks. As such a flavour of cross-layer design is to be beneficial to the overall performance of not only services but also networks over which the services perform.

2.2. Wireless Networks

Recent years have seen a rapid development of telecommunication and wireless communication systems, UMTS (Universal Mobile Telecommunications System) networks, mobile ad hoc networks and sensor networks. Diverse network structures and

protocols cause lots of problems in network integration and service integration. For a more efficient utilization of variable network materials, two or more network structures are integrated, such as iCAR (Integrated Cellular and Ad Hoc Relaying System) [Wu01]. iCAR tries to introduce mobile ad-hoc networks into cellular networks to improve the latter's call block rate in hot spots such as sporting venues or on the scene of emergency events. The above heterogeneous network can be complicated be adding WiMAX or DVB to construct a more flexible and robust multi-service network platform. The heterogeneity in network structures is a typical feature of 4G mobile networks. In recent years, small portable devices have been increasingly equipped with multiple communication interfaces (e.g., both cellular air interface and WiFi interface). Some new multiple-interface servers are also under design and development for the future use in heterogeneous networks. These networks and devices are penetrating people's everyday life and becoming pervasive. These pervasive network facilities have formed a firm foundation of offering people with more flexible services such as pervasive services. These networks, in connection with the Internet which provides global IP connection, gradually become the main platforms over which pervasive services run.

One type of wireless networks is of particular interest to this Thesis: mobile wireless ad hoc networks.

2.2.1. Mobile Ad-hoc Networks

Mobile ad-hoc networks (MANETs) consist of a set of wireless mobile nodes that cooperatively form a network without a fixed infrastructure. In such a wireless network, a message sent by a node usually reaches all its neighbouring nodes that are located within the transmission radius of the sender. Because of the limited transmission radius (which is affected by the transmission power), the routes between the original sender node and the intended final receiver node normally consist of several hops. As such, each node in a MANET also serves as a router to route information between neighbours, thus contributing to the end-to-end message delivery across the whole network. Due to their mobility nature, nodes in mobile ad hoc networks usually have limited resources (e.g. limited process power, battery life, memory, and bandwidth), and use unreliable wireless channel.

Mobile devices (such as IP-supporting mobile phones, PDAs, etc), on which pervasive services run, constitute such an ad hoc network where each mobile device is a node in the network. For example, the Virtual Office and Home Environment (VOHE) service scenario mentioned in Chapter 1 could enable a user on a PDA to continue his/her normal office work even when he/she is at the Heathrow Airport waiting for

boarding. This exemplifier pervasive service demonstrates the highly dynamic nature of pervasive services, which is mainly caused by the mobility of service customers. Relevant parties required to satisfy a customer's need will need to form a community in an ad hoc and loosely-coupled manner. Figure 2 illustrates such a service-oriented mobile wireless ad hoc network. For instance, MH6 provides services *A* and *C* but requires services *X*, *Y*, *Z*. While services *X* and *Z* can be provided by MH1 and MH7 respectively, service *Y* is available in both MH1 and MH7. Furthermore, services provided by MH7 can be discovered and used by MH1 via a multi-hop route.

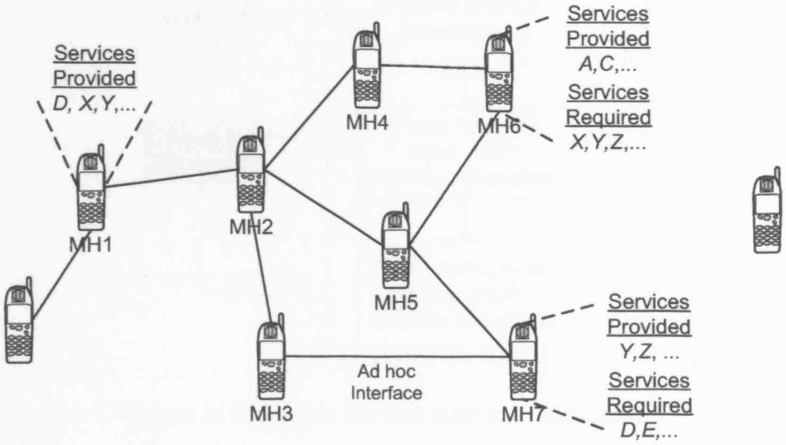


Figure 2: An Example of Service-oriented Mobile Wireless Ad hoc Network

2.3. POETRY Essences

In Chapter 1, POETRY was briefly introduced in terms of its major functions. Before embarking on the presentation of the POETRY architecture, this sub-section describes the technical core, or the essences, of POETRY, aiming to further elaborating the novelty or characteristics of POETRY.

Essence #1: A Pervasive Service as a PBM System

As discussed in Chapter 1, one of the contributions of the Thesis is to advocate the shift of the pervasive service engineering attention from the *external* service execution environment to the *internal* service logic itself. Policies are proposed to describe the service logic at a higher level. The policies whose conditions satisfy the current environment will be activated automatically by an event-driven policy decision engine. In this context, the internal logic of a pervasive service is in the form of a set of loosely-coupled policies of no particular order. Then a pervasive service itself, as a PBM system, is a collection of the following components, as illustrated in Figure 3:

- policies that define the service behaviours or guidance under different conditions and are defined by service developers;
- a service decision engine that reasons over these policies (called the Policy Decision Point or PDP in IETF terminology [IETF-policy]); and
- a service execution engine (called Policy Enforcement Point or PEP) that takes real action as a result of service decision making.

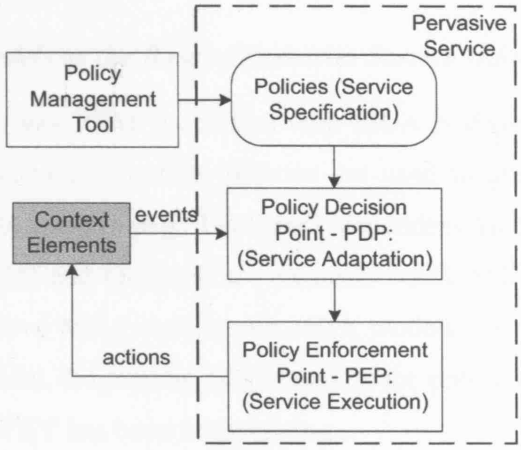


Figure 3: Pervasive Service Internal Architecture

There have been research activities that apply PBM for service engineering. However, the concept of regarding the whole PBM system as a pervasive service has not been found in the literature. The adaptability of the pervasive service comes from its awareness of the operation environment, i.e. the context. Service adaptation occurs when there is an interesting change in the context. This context change is reported to the service decision engine (PDP) through an event. Upon receiving the event, the PDP then makes the proper decision by reasoning over the policies. The reasoning process might involve in conflict resolution if the new contextual changes result in incompatibility with user or system requirements.

So far the problem of pervasive service creation has been converted into the problem of creating a functioning PBM system that fulfils service requirements. To this end, policies reflecting the service behaviours, user preferences, etc need to be defined via a *policy management tool* (refer to Figure 3). Policies of a service are loosely coupled and the activation of certain policies depends on context changes (such as a user changing his/her preference or the network connection suddenly gets worse). PDPs that reason over the service's policies to select which actions to take also need to be designed and developed. A powerful PDP can grant significant adaptability to a pervasive service.

Pervasive services created in such a manner have inherent adaptability. However, this born adaptability has to be fulfilled via service execution, and that is the work of PEP, as illustrated in Figure 3. Underpinning the PEP is a functioning pervasive service information model (PSIM) that provides implementations of policy operations. Since a pervasive service is implemented as a PBM system, this PSIM is then equivalent to the IETF policy information model. POETRY adopts OMG MDA models to construct PSIM.

Essence #2: MDA Models as the Base of Pervasive Service Information Model (PSIM)

In POETRY, the way PBM cooperates with MDA is different from these in the current literature. In current literature, policies are used to guide the transformation between different MDA models (e.g., Platform Independent Model or PIM to Platform Specific Model or PSM) and therefore are elements *inside* MDA. In POETRY, PBM locates *outside* and above MDA models and MDA models are used for describing the policy information model. Employing MDA models for policy information model is a new thought that POETRY has been investigating.

A hierarchical PSIM is proposed in POETRY. This PSIM makes use of MDA *model* concept to separate the system design part of the PSIM from its real implementation part (classes, code). Figure 4 illustrates how the MDA components are utilized to construct PSIM. By employing MDA for PSIM a layered and more flexible PSIM can be constructed. For instance, any change made internally to the implementation-independent part of the PSIM (iiPSIM) will not affect the implementation. In the same vein, any change made to the implementation-dependent part of the PSIM (idPSIM) (e.g., change to a new implementation technique or language) will be kept transparent to iiPSIM. In this way, MDA's platform-, language- and middleware-neutral features can be integrated into pervasive services. Furthermore, with the further development of MDA itself, there is a great potential to automate the whole procedure of pervasive service creation and code generation.

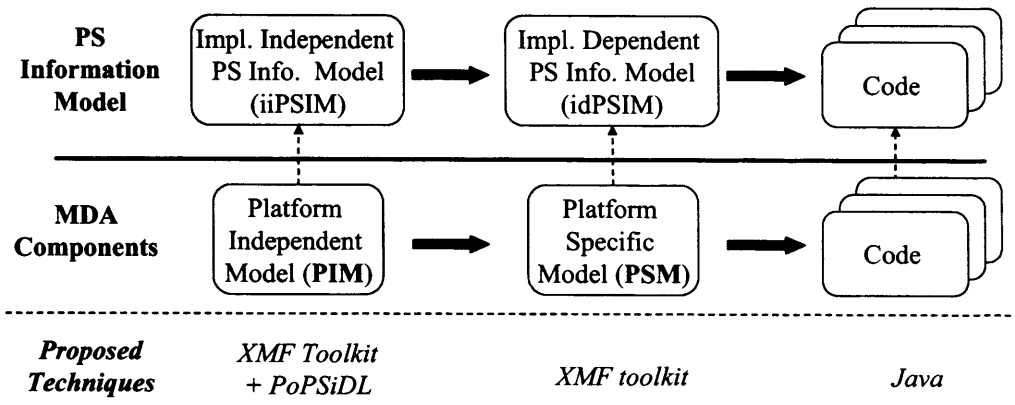


Figure 4: MDA Components for PSIM (Pervasive Service Information Model)

The OMG MDA defines the components for constructing model driven applications. However, it does not specify concrete techniques for each component. In order to make MDA useful in practice (e.g., using it for real pervasive service creation in POETRY), specific techniques need to be provided for each MDA component. The techniques employed by POETRY are shown italicised at the bottom of Figure 4. PoPSiDL, standing for Policy-based Pervasive Service Description Language, is a language used by pervasive service provider or developer to specify/update the possible behaviours (actions) of a pervasive service under different circumstances (conditions) by using policies. These policies are to guide the generation of PSIM.

Essence #3: Model-based Service Discovery for Wireless Mobile Networks

In POETRY, "service discovery" actually means the discovery of a service component that is needed to by a running pervasive service to deal with certain service requirement as a result of context changes. POETRY investigates the service discovery mechanisms that are not dependent on a centralized server but rather are carried out in an ad hoc manner, e.g., to discover a service component available in its nearby mobile devices. The POETRY service discovery algorithm, called MBSD for model-based service discovery, reuses the models that are generated during static service creation/development stage. This manner not only provides service description in a much easy way but also gives incentive for the discovered component services to be easily integrated with the existing service.

In summary, having polices defining the internal logic of a pervasive service (*essence #1*) and various models specifying the design and implementation views of the pervasive service (*essence #2*), which are further enhanced by runtime ad hoc service discovery (*essence #3*), a systematic and easy way of creating pervasive services and a

simple means of running and adapting services are created. And this is implemented by the POETRY framework.

2.4. The POETRY System Architecture

The POETRY system, whose architecture is shown in Figure 5, consists of two parts: static part for service creation and deployment – which is mainly designed for service providers and runs offline, and the dynamic part for service execution and adaptation (including service discovery) – which is designed for end users. The main function of the static part is to facilitate the creation of pervasive services by using policies and models, as mainly implemented by the *Service Creation Module*. The created services are deployed to a certain website for end users to subscribe where end users can also further customize the services by defining their preferences or security information. After a user's successful subscription to a pervasive service (which can be finalized by, e.g., the correct payment), the context variables involved in this service are automatically registered with the corresponding monitoring daemons. At the same time, the pervasive service is downloaded to the user device(s) as executable code. Then the service lifecycle gets into the dynamic status.

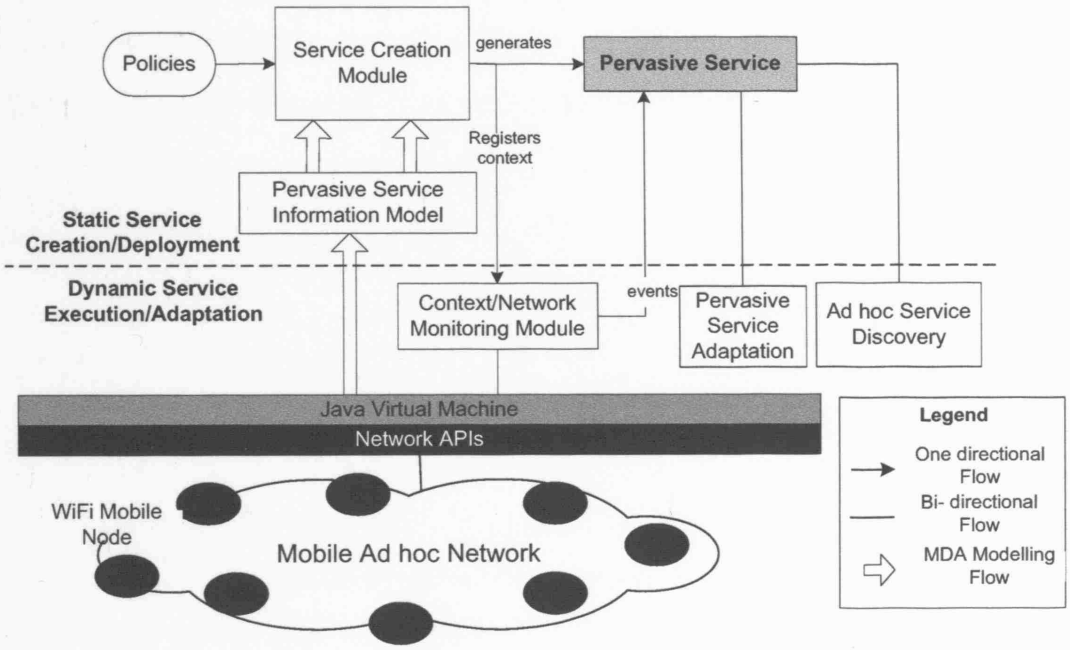


Figure 5: The POETRY System Architecture

During the service execution status, if there is any context change that goes beyond its threshold then an event is generated and sent to the pervasive service. The pervasive service, after picking up this event (which also carries the new value of the context),

will start decision making via reasoning over its policies, and in this manner service adaptation is carried out. If the context change leads to the request of the execution of a new component whose implementation is not currently available then the *Ad hoc Service Discovery* module is invoked to search for this particular component.

Here it is assumed that the underlying network functions are available via network APIs (Application Program Interfaces). In this Thesis we also assume that the network APIs are available in Java classes to match with the programming language used to develop pervasive services. Given the platform-neutral nature of the Java language and consequently the broad use for network applications, this assumption is reasonable. However the basic principle in POETRY is independent of the programming language used.

Chapter 3. Pervasive Service Creation

This Chapter focuses on the creation of pervasive services in POETRY by utilizing the design principles outlined in the POETRY essences in the previous chapter. First of all, the fundamental entities involved in a pervasive service are analysed in a scenario-driven manner in Section 3.1. These entities constitute the meta-models of a domain-specific language that is designated for pervasive services. Secondly, a pervasive service description language called PoPSiDL is designed in Section 3.2 to specify a pervasive service using rule-based policies. The main constructs, their syntax and semantics are presented, followed by the description of the PoPSiDL parser. While PoPSiDL provides a flexible way to describe a pervasive service, the fulfilment of the pervasive service is carried out by the pervasive service information model (PSIM), which is the content of Section 3.3. A hierarchical MDA models are utilized to represent the PSIM. After the core of pervasive services – the policy-based pervasive service decision making engine - is presented in Section 3.4 and before the chapter summary in Section 3.6, Section 3.5 details the whole procedure of POETRY service creation.

3.1. Pervasive Service Analysis and Modelling

Before embarking on the pervasive service description language and pervasive service meta-model study, an analysis of the composing entities common to the majority of pervasive services is needed. This is carried out via a typical pervasive service scenario study.

3.1.1. Scenario Description

Here a potential future pervasive service called *SNAMU* (transparent Service and Network Access for Mobile User) is described. The service tends to provide a means for a nomadic user to maintain a secure access to his/her enterprise/home network transparently after the user has subscribed to the pervasive service from the service provider. SNAMU also allows its user to dynamically search for new services and to use services in a context-aware manner.

Consider Julie, a medical professor in a famous university always with a busy schedule for medical consultation around Europe. Julie works from home several days a week, using her broadband home network that is connected to the consulting hospital's network to transfer materials including large-sized scan pictures. One day when she was working on an important medical case that had to be sent out by 4:30pm of the day, she found herself in a situation that she had to leave for the Heathrow Airport to catch a flight to Athens. So immediately she set off.

On her way to the Airport, she luckily finished the last part of the case analysis and needed to submit her analysis result together with loads of commented images to the consulting company. Immediately she used her laptop and an embedded GPRS modem to set up a network connection. Unfortunately, the bandwidth of the GPRS network was insufficient, and transferring these large files would take about several hours, far passing the deadline. However, luckily enough, she had subscribed to the new *SNAMU* service that detected bandwidth problem and tried to solve it. In the scenario the *SNAMU* service found out that there was an appropriate WLAN access point in the Airport, and as such it dynamically switched the network connection to the faster wireless LAN when Julie arrived at the Airport. At the same time, a secure tunnel was established automatically between the Airport and the consulting company. Julie was then connected to the local network so as to deliver the file more quickly. She noticed happily the significantly increased file transferring process, and her important medical analysis was submitted successfully before the deadline, which possibly could save a life.

While waiting at the lounge of the Airport, Julie received an instant message from her son saying that four five-minute video clips of his play show were just uploaded onto their home network server. Since it was still 20 minutes before boarding, Julie decided to download these video clips and to playback them on her PDA. However, the wireless LAN access points (APs) in the lounge started getting busier. Although Julie could get very stable connection, the bandwidth was not satisfactory to Julie's expectation. Furthermore, remaining battery power might not be sufficient to carry out the whole downloading, decoding and playback of even half of the video clips. Fortunately, Julie's PDA, having the *SNAMU* service installed, detected a surrogate which offered application offloading services. The *SNAMU* in her PDA partitioned the video-downloading-and-playing application, offloaded downloading and decoding components to the surrogate. By using the broadband network resource, the surrogate managed to download video clips in a very short time; and then it decoded the video

clips and sent the decoded video data to Julie's PDA for playback. Thanks to the powerful network and computational capacity, Julie got quicker application response and obtained higher video quality. Relying on the *SNAMU* service, Julie managed to view all the video clips before boarding. The saved battery power also enabled the *SNAMU* to dynamically discover doc-to-pdf converter software to get her camera-ready journal paper to the required format for final submission.

3.1.2. Analysis of Pervasive Service and Context

Describing it in an abstract way, this SNAMU scenario involves a *user* using her *user device* to make use of a *function* (e.g., file uploading, video streaming, service discovery) provided by certain *function provider* through her *access network* (GPRS/UMTS or WLAN) and the *core network*. A function can be of any type (and consequently different function providers getting involved in). For instance, video streaming needs a content provider to be involved in, language translation requests the existence of a third party service provider, a tourism information retrieval needs an information service provider, a secure IP tunnel setup needs supports from network infrastructure providers. And one individual function can be provided by multiple providers possibly at different price and quality.

There may be other ways of extracting these features. This Thesis adopts a more practical means that considers the final implementation of pervasive services. Namely, the entities that play different roles and locate at different physical and administrative domains are taken into consideration. The means reflects to some extent the business model of current ICT (Information and Communications Technology), aiming to making the provisioning and deployment of pervasive services more feasible.

The adaptability of a pervasive service largely relies on its context-awareness [Satyanarayanan01]. Therefore, before working on the pervasive service modelling itself, an investigation into what it is meant by *context* in the Thesis and how to classify it is carried out.

There is a heavily overloaded use of the term *context* with a wide variety of meanings [Yang03a], [Yang03b]. This Thesis is based on and further extends the context concept given by Dey in [Dey00a]. Dey's context definition constrains the context to these existing only between a user and an application – this also indicates the origination of the context awareness research: Human-Computer Interface or HCI. Given the impact of network features on the performance of application-level services, which is especially the case in wireless networks, this Thesis considers also network context information so as to give a full coverage of the entities involved in a pervasive

service. In summary, the context related to a pervasive service can be any information, obtained either explicitly or implicitly, that can be used to characterize one certain aspect of an entity that involves in a specific application or network service. An entity can be a physical object such as a person, a place, a router, a UMTS network gateway, a physical link, or a virtual object such as an IPsec tunnel, or a SNMP (Simple Network Management Protocol) agent. Actually, most of the policy variables in the policy pre-conditions of a pervasive service are regarded as contextual information for this particular pervasive service.

To guide the design and implementation of context classes in PSIM in a well-organized manner, the context needs to be classified. A proper classification of context types will help application designers decide the most likely pieces of context that will be useful in their applications. The abovementioned definition of context seeds this Thesis's development of context classification at a high level. Schilit *et al.* [Schilit94] list the important aspects of context as where you are, who you are with and what resources are nearby. This list does not consider *time* this critical factor. Ryan *et al.* suggest context types of location, environment, identity and time [Dey00a]. Dey thinks that there are certain types of context that are, in practice, more important than others, and these are location, identity, activity and time [Dey00b]. Sharing the same idea as proposed by Dey, this Thesis utilizes *location*, *identity*, *time*, and *activity* as basic context types for characterizing the situation of a particular context entity, as depicted in Figure 6.

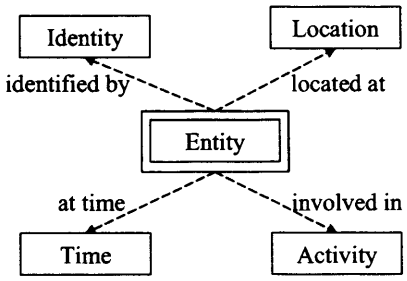


Figure 6: Classification of Context in terms of Entity

This context classification clearly answers the questions of who (*Identity*), where (*Location*), when (*Time*), and doing what (*Activity*) on a specific context entity. This contributes to one dimension of context information classification - the features common to most of the context entities. The other dimension of the context classification is about context entity itself, which can be categorized using the entities worked out above to describe the role of an entity in a pervasive service, i.e., user, user device, function, function provider, access network, and core network.

Note that some of the context types also act as indices into other sources of context information. For example, given a person's identity, many pieces of related information can be obtained such as phone numbers, addresses, email addresses, list of friends, and relationship to other people in the environment, or even the emotion of this person. This kind of information is related to the *basic* context discussed above and is called *derived* context (e.g., the email address) for that entity. Context information can be related to one another. For example, with an entity's location, it can be determined what other devices or people are near to the entity and what activity is occurring around the entity.

3.1.3. Meta-Modelling Pervasive Service as a Domain-specific Language

In summary, the following six generic entities are usually involved in a typical pervasive service: *user*, *user device*, *function*, *function provider*, *access network*, and *core network*, each having, apart from their own attributes, four extra attributes, namely, *identity*, *time*, *location* and *activity*. These artefacts constitute the meta-model of a specific domain: pervasive service, as depicted by XMF [XMF] in Figure 7. This meta-model is called the domain specific language or DSL in MDA terminology.

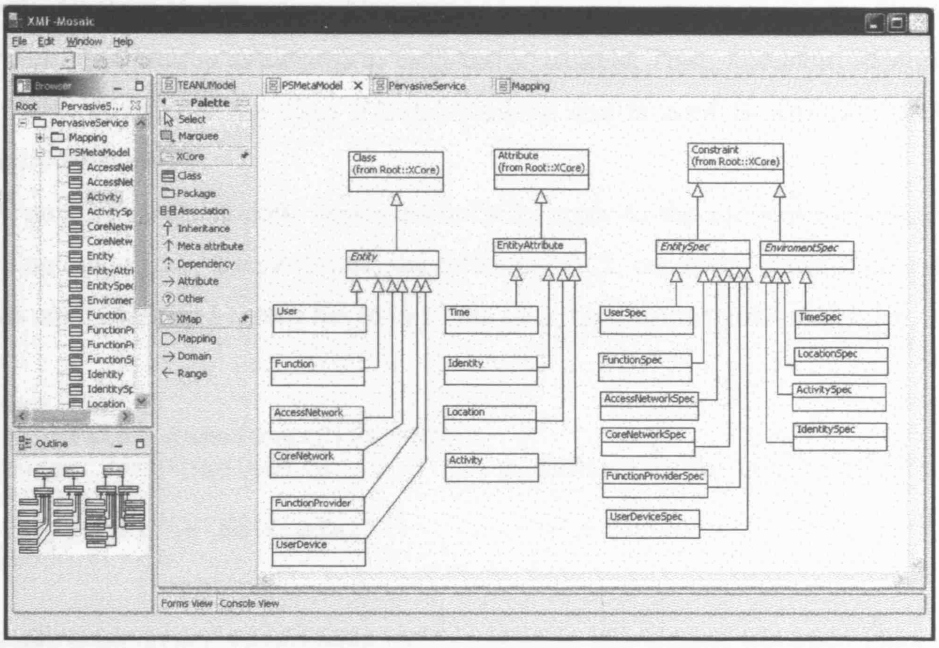


Figure 7: Domain Specific Language (or Meta-model) for Pervasive Services

Sitting at the highest level of the model hierarchy (refer to the MDA part of Chapter 2), this pervasive service modelling language consists of the following constructs, each specialising a corresponding class of the XMF *XCore* package that already has a well-defined executable semantics defined in the XMF toolkit:

- *Entity*, representing one of the abovementioned six types of context information of a pervasive service. It extends the class *Root::XCore::Class* so that it contains attributes, operations and constraints and it can be instantiated.
- *EntitySpec*, representing configuration of entities, mainly in terms of constraints, such as range of values or preconfigured settings on features of the *Entity*. Correspondingly, the API specification defines six subtypes of *EntitySpec*, each representing specification for one of the six entities respectively. *EntitySpec* specialises XCore:Constraint and is owned by an Entity and contains an evaluate-able OCL (Object Constraint Language) expressions.
- *EnvironmentSpec*, in the same way as *EntitySpec*, representing configuration of four special *EntityAttribute*, namely, *Time*, *Location*, *Identity* and *Activity*. Correspondingly, the API specification defines four subtypes of *EnvironmentSpec*, each representing specification for one of the four attributes respectively. *EnvironmentSpec* also specialises XCore:Constraint and contains an evaluate-able OCL expressions.
- *EntityAttribute*, representing relationships between *Entity* types and the four extra context information associated to each entity, namely, *Time*, *Location*, *Identity* and *Activity*. It inherits the class XCore::Attribute and is used to associate different Entity types.

A number of constraints, expressed in OCL, apply to the above pervasive service modelling language. As an example, the following OCL constraint states if an entity *a* extends another entity *b* then *a* has to be of the same type as the parent entity, i.e.,

```
context Entity
  @Constraint SameSuperClassType
    superClass->select(s     |     s.isKindOf(Entities::Entity))-
>forAll(s |
        s.of() = self.of())
  end
```

s.of() is an XOCL (Executable OCL) operation that returns the super-class of an entity.

So far these components commonly used by pervasive services are available for a service developer to develop his or her own pervasive services. Before illustrating how these generic, high-level components are to be used to create *SNAMU* pervasive service, a flexible pervasive service description language is designed and presented in the next

section. This language is used to describe the behaviour and features of a pervasive service, and is complementary to the pervasive service information model.

3.2. Policy-based Pervasive Service Description Language: PoPSiDL

3.2.1. Overview

A Policy-based Pervasive Service Description Language (PoPSiDL) is proposed here to specify/update the possible behaviours (actions) of a pervasive service under different circumstances (conditions) by using policies. As suggested by the IETF Policy Framework Working Group [IETF-policy], which has been investigating policies as a means for managing IP-based networks for many years, policies take the following rule-based format: `IF {condition(s)} THEN {action(s)}`. It means *action(s)* is/are taken if the corresponding *condition(s)* is/are true.

For instance, the following policy in a pervasive service forces the mobile to vibrate instead of beeping during meeting time:

```
IF (location == meetingRoom) and (time within meetingSchedule) THEN MobileVibratingOnly
```

A more network-related example of policy is as follows, which means that a secure IP tunnel needs to be set up between two ends of a communication if the destination host of the communication is within user's enterprise network and the user himself/herself is a manager of this enterprise. This piece of policy also implies that other traffic would not be protected by an IPsec tunnel. By this means, any policy related to a pervasive service can be defined. The execution of these policies eventually triggers the action of classes in policy-based pervasive service information model.

```
IF (destHost within EnterpriseNet) and (userRole is Manager) THEN useIPsecTunnel
```

By using rules as many policies as needed for certain pervasive service can be defined. All the policies are obligatory but subject to their conditions being satisfied. These rule-based policies can be further represented by XML due to XML's built-in syntax check and its portability across the heterogeneous platforms.

3.2.2. PoPSiDL Constructs

3.2.2.1. PolicyVariable (PV)

Policy Variables (PV) are the basic element in PoPSiDL. A PV usually appears at the condition part of a piece of policy where it is used to describe a piece of context information. A PV can also appear in the action part of a policy if the purpose of the

action is to change the value of this context information. This context information is obtained either from an object or from blackboard. Blackboard is a shared memory space used to temporarily store the reasoning information. An object can be a physical element such as mobile handset, IP router, firewall, wireless LAN (WLAN) access point, GPRS/3G gateway, or virtual element (software) such as SNMP agent, IPsec tunnel, etc. A PV, as a piece of context information, falls into one of the six categories for context entities as discussed in Section 3.1.

Definition 3.1: A PV is formally defined as a 9-tuple: PV::=<*Name, Type, Value, Unit, Description, Question, Multimedia, Procedure, ContextType*>, where *Name* is a unique string for identifying a PV and is mandatory; *Type* represents the data type of the PV, e.g., Integer, Real, Boolean or String; *Value* represents the value of the PV which follows the type defined by *Type*; *Unit* specifies the unit of the PV in the form of string, e.g. meter, kilogram, or null (meaning no unit); *Description* is a string describing the purpose of the PV and is null if no description provided; *Question* is a string containing the question to be raised to end users – it is designed for cases where a PV's value cannot be provided by the current system automatically and thus needs to be inputted by human users; *Multimedia* is a string that specifies a link to the multimedia format of the PV and it is optional; *Procedure* indicates the external procedure to get the value of the PV; and finally *ContextType* indicates to which type of context entity this PV belongs.

The *Question* string is to be displayed to human users during the real-time pervasive service execution indicating what kind of input is expected. This feature helps prevent a pervasive service from failing just because of a lack of some information that can be easily provided by users. A PV can also be in the form of *multimedia* such as voice, image or video in this PoPSiDL. For instance, different phone tones can be used for indicating different sources or types of incoming information. This attribute of PV, in the type of String, stores the path/link to a piece of multimedia information. The presence of this feature enriches pervasive services to make them possible to also include multimedia information.

Note that a *procedure* is usually assigned to a PV during service creation stage and this procedure can be the client side talking to a monitoring daemon or a standalone piece of code. The value of a PV, after it being registered with the corresponding monitoring daemon, is automatically updated in an event-based manner.

The introduction of *procedures* allows a service provider to provide its own code for specific context or actions. This avoids the frequent updating of policy information

model (PIM) once there is new service added. As a result, a stable and simpler PIM can be maintained and efficiently used by many services and management systems.

For example, in policy repository involving SNAMU pervasive service, policy variable (PV) "bandwidth" is specified as follows:

(bandwidth, Real, 1.2, Mbps, "bandwidth required by this customer", "how many bandwidth are you requiring?", "http://www.poetry.com/popsidl/bandwidth-help.html", NULL, "UserDevice")

3.2.2.2. Constants and Variables

Constants are values like 10, 1.2, "UCL", "128.40.40.48" which are not computed and recomputed but remain constant. In PoPSiDL, there are four types of constants: Integer, Real, Boolean and String. Variables in PoPSiDL usually are the names of PVs and there are four data types as in the case of constants, namely, Integer, Real, Boolean and String. Variables can appear anywhere constants appear.

Unlike some programming language such as C/C++/Java, variable in PoPSiDL can be used directly without prior declaration. This also means that there is no way to explicitly define the type of a PoPSiDL variable. This method has been used by many high-level shell or script languages due to its flexibility. This, on the other hand, is likely to cause type mismatch when operations such as comparison or addition are conducted on these variables. This potential type mismatch problem is to be solved by the corresponding operation (e.g., comparison operator).

The variables in PoPSiDL can be mapped to those provided by certain implementing language. For instance, all the above four PoPSiDL data types can be mapped to the corresponding classes in Java.

3.2.2.3. Mathematical and Logical Operations

Mathematical expressions in PoPSiDL use infix expression form. It supports basic arithmetic calculation like addition (+), subtraction (-), multiplication (*), division (/), brackets (). And it supports frequently used mathematical functions including `sin()`, `cos()`, `tag()`, `ctag()`, `log()`, etc. Other functions can be equally introduced. Variables can appear in PoPSiDL expressions. The calculation uses infix expression algorithms.

Logical expressions in PoPSiDL adopt CNF (Conjunctive Normal Form). PoPSiDL provides two logical operations: **OR** and **AND**. Logical negative ($\neg$) is expressed using comparison operation. E.g., the logical negative of (bandwidth <=20) is expressed in PoPSiDL as (bandwidth > 20), i.e.,

$$\neg(bandwidth \leq 20) \Leftrightarrow (bandwidth > 20)$$

The result of logical expression is either TRUE or FALSE.

3.2.2.4. ComparisonUnit

ComparisonUnit (CU) is a logical calculation unit used for the comparison of two PVs. The result of this comparison is either TRUE or FALSE. Six comparison operators are provided in PoPSiDL: <, <=, >, >=, ==, !=, similar to those provided in normal advanced programming languages. A CU in PoPSiDL can take one of the three following formats:

1. <*PVname comparisonOperator Constant Unit*>, e.g., <bandwidth == 2Mbps>
2. <*PVname comparisonOperator PVname*>, e.g., <useScreenSize < imageSize>
3. <*PVname*>, e.g., <bandwidthHigh>

The semantics of a CU can be summarized as "policy variable matches value" and its result is either FALSE or TRUE. The essence of this semantics lies in the meaning of *match*. The following method named *Compare* in *ComparisonUnit* class describes the semantics of *match*. This method carries out value comparison based on the results of type matching and unit conversion. The method takes three input parameters: the first policy variable *PV1*, comparison operator *comparisonOperator* and the second policy variable *PV2*. The return value of this method is Boolean.

Boolean ComparisonUnit.Compare (*PV1*, *comparisonOperator*, *PV2*)
Begin
1. **if** *PV1*.Type not match *PV2*.Type **then**
2. **return FALSE**;
3. **if** *PV1*.Unit not match *PV2*.Unit **then**
4. convertUnit();
5. **return** (*ComparisonOperator*(*PV1.Value,PV2.Value*));
End

3.2.2.5. Condition

In PoPSiDL condition is expressed as a series of **OR** operation of *ComparisonUnit*:

ComparisonUnit1 **OR** ComparisonUnit2 **OR** ... **OR** ComparisonUnit_n

For example,

(destination == "camden" **OR** destination == "skywarp")

Particularly, *ComparisonUnit* itself is also a *Condition*.

The result of a *Condition* is either FALSE or TRUE. The method *ConditionMatch ()* of *Condition* below describes the condition matching algorithm.

Boolean Condition.ConditionMatch ()
Begin
1. result = **FALSE**;
2. **for** *i*=1 to *n* **do**
3. **if** result == **TRUE then**

```
4.        return result;
5.     result = ComparisonUniti.Compare (PV1, comparisonOperator, PV2);
6.  return result;
   End
```

3.2.2.6. PolicyCondition

PolicyCondition is an ANDed set of *Conditions*. In PoPSiDL, *PolicyCondition* is in conjunctive normal form (CNF, i.e, ANDed set of OR conditions), as suggested by IETF Policy Framework [IETF-Policy].

Condition1 **AND** Condition2 **AND** ... **AND** Condition_n

For example,

(protocol == "ftp") **AND** (destination == "camden" OR destination == "skywarp")

Particularly, *Condition* itself is also a *PolicyCondition*.

The semantics of a *PolicyCondition*, which is "variable matches value" with result being either FALSE or TRUE, is described by the following *PolicyConditionMatch*() method of *PolicyCondition*.

```
Boolean PolicyCondition.PolicyConditionMatch ()
Begin
1. for i=1 to n do
2.    result = Conditioni.ConditionMatch ();
3.    if result == FALSE then
4.       return FALSE;
5. return TRUE;
   End
```

3.2.2.7. Actions

Action means executing certain operation(s). This, in PoPSiDL, can be blackboard operations (assert, delete), value assignment, information output, external program invocation and other operations provided by service providers.

Blackboard Operations: There are two types of blackboard operations: **Assert** and **Delete**.

Assert may take one of the following four formats (the semantics of each follows):

1. ***Assert (PV, String, Unit)***: Adds into the blackboard a policy variable *PV* of type *String* with unit as *Unit*. When a PV has already existed in the blackboard when the *Assert* operation is performed, *Assert* means update. The same applies to the other formats.

2. ***Assert*** *(PV, Expression, Unit)*: Adds into the blackboard a policy variable *PV* whose value is the result of *Expression* with unit as *Unit*. The type of *PV* is numeric.
3. ***Assert*** *(PV)*: means adding the Boolean *PV* into the blackboard.
4. ***Assert*** *(PV, Variable)*: adds *Variable* as *PV* into the blackboard.

For example,

```
Assert (ScreenSize, 5*6, centimetre)
```

Delete operation takes the following format:

Delete PV

Which means delete the *PV* from the blackboard.

File Operations: Four types of file operations are provided in PoPSiDL, e.g., to generate or to update a log file: (1) ***OpenFile*** *(filename, mode)*; (2) ***Read*** *(Var, n)*; (3) ***Write*** *(String)*; (4) ***CloseFile****(filename)*.

Display Operation: *Display* is used to show information to users during pervasive services' execution (or in another term showing the rule-based reasoning process). This is to, for instance, ask a PS user to input information or to show warning information or to show an information of user's interest (e.g., the room temperature or battery level of his mobile device), etc. The syntax of *Display* is:

Display (String, mode, type)

Where *String* is the information to be displayed; *mode* indicates the way (either in DIALOG frame or WINDOW format) information is to be displayed; *type* indicates the type of information: TIP/WARN/INPUT/OTHER.

External Method Invocation Operations: this type of operations invokes an external method of an object that is not known at the design stage, e.g., a dynamically discovered service component that provides services needed by the pervasive service. This is done in PoPSiDL by *Execute* operation using the following syntax:

Execute (MethodName, Parameters, ShowMode)

Whereas *MethodName* is the name of an external method including the object name (and the object's full path if needed); *Parameters* is String input/output parameters for the method; *ShowMode* indicates the display mode of this external program including: (1) SHOW for normal display, (2) MINSHOW for minimized display, (3) MAXSHOW for maximized display, and (4) HIDE for no display (i.e., running at the background without users' notice).

3.2.2.8. PolicyAction

PolicyAction is a series of *Actions* that are to be executed sequentially from left to right when it is triggered. *PolicyAction* takes the following format:

Action1 **AND** Action2 **AND** ... **AND** Action_n

3.2.2.9. Policy

In PoPSiDL, a policy is expressed in the following format:

PolicyId: ***IF*** *PolicyCondition* ***THEN*** *PolicyAction Priority Uncertainty_factor Comment*

PolicyId is for identifying a policy. *Comment* stores string comment for a piece of policy. The semantics of the policy is: actions in *PolicyAction* are taken when *PolicyCondition* is TRUE.

To consider the potential policy conflict and its resolution and to deal with uncertainty, the IETF policy format has been further extended to include *Priority* and *Uncertainty_factor* in the above *Policy* format. *Priority* denotes the level of priority of a policy in comparison with other policies and it is to be used for policy conflict resolution. *Uncertainty_factor* denotes the overall uncertainty of the policy, and it is an indicator as to how much this pervasive service trusts this piece of policy. Uncertainty factor can be equally applied to *PolicyCondition* and *PolicyAction* but will increase the complexity of the policy decision making procedure. The presence of uncertainty factor in a piece of policy provides a means to describe (and as such to tolerate) the vagueness or uncertainty of a piece of information (represented by policy conditions), or a tended action (represented by policy actions), or a whole strategy (policy itself), aiming to increase service's ability to cope with more complex circumstances. The default value ("1") to these two parameters means "same priority" and "100% certain" respectively.

3.2.2.10. Module

Module is a set of policies that aims to fulfil a *specific* network or service management task. Policies for a pervasive service can be grouped into multiple modules according to their functions or purposes. For example, in *SNAMU* pervasive service (refer to Appendix B), policies for *SNAMU* are grouped into the following modules: *UserInfoModule, PersonActivityModule, BatteryModule, ServiceModule, GPRSModule, CommonService*, etc.

Grouping policies into a number of modules makes the logical structure of a pervasive service description clearer to service developers, and as such reducing the searching space when performing policy reasoning (i.e. decision making). Parallel operation of independent modules is also possible which can improve the performance of pervasive services. Different modules may cooperate with one another. For instance,

some input of one module may come from the output of another module. As an example, the *securityLevel* variable of *CommonService* module in the *SNAMU* service comes from the *UserInfoModule*.

In PoPSiDL, *Module* takes the following form:

MODULE *Module-Name*
BEGIN_M
Policy1;
.....
Policy$_n$;
END_M

3.2.2.11. Pervasive Service - PS

Each complete PoPSiDL document describes a pervasive service via its **PS/ENDPS** pair. The syntax of **PS/ENDPS** is:

PS *PervasiveServiceName*
PolicyServer = *PolicyServerName;*
Module1;
.....
Module$_n$;
ENDPS

A detailed syntax description of PoPSiDL is presented in Appendix A using an extended BNF (Backus-Naur Form).

By using this format, as many as necessary policies can be defined to describe the behaviours of a pervasive service. For example, the *SNAMU* service can be described by PoPSiDL as shown in Appendix B. Each pervasive service corresponds to such a PoPSiDL description file (*.psd*). The policies specified in this file, maybe in collaboration with some network-specific policies imposed by network administrator for example, are reasoned over by a policy decision engine (or Policy Decision Point – PDP) to guide the operation of this pervasive service. Policy decision engine is to be discussed in Section 3.4.

In PoPSiDL, policies serve as the containers of context information. All the information used in policy conditions are regarded as context information. These changing frequently and dynamically, such as user location, user device type (as such its screen size), most of the network information like queue length, bandwidth available etc, are registered to the corresponding monitoring daemons so as to be monitored instantly. A sufficient change of such context (e.g., beyond certain threshold) will generate an event, which will then be captured by policy decision engine to make proper decision – refer to the POETRY system architecture in Chapter 2.

3.2.3. PoPSiDL Parser

The PoPSiDL parser is created in a semi-automated manner using JavaCC [JC]. JavaCC does not take standard BNF as input. So the BNF syntax of PoPSiDL (refer to Appendix A.1) has to be manually changed to a JavaCC-acceptable format (refer to Appendix A.2).

The parser generated by JavaCC outputs language tokens and parsing results to the standard output (typically a screen monitor). However many these tokens, e.g., *battery*, *location* etc in the SNAMU scenario, also called policy variables in PoPSiDL terminology, are not just variables as in normal programming languages but bearing much richer operational semantics. For instance, *battery* token is closely associated with a daemon that monitors the battery level of a mobile device. In other words, most of these tokens are context. Such context filtered out by the parser will need to be registered with the corresponding monitoring daemons so that changes on this context can be reported. In this light, the PoPSiDL parser needs to divert this context to the POETRY service creation module.

The PoPSiDL parser also needs to do other extra thing in comparison with the normal language parsers/interpreters. These includes: 1) to filter out all the context variables used by a pervasive service; 2) to filter out all the components to be used by the pervasive service, amongst others. More explanation on this can be found in the Pervasive Service Creation Section below.

3.3. Pervasive Service Information Model

After the modelling language and the description language of pervasive services have been defined in the previous last two sections respectively, a pervasive service's high-level behaviours can be specified by a PoPSiDL script as exemplified in SNAMU service. The next step is how to execute the script, i.e., to implement the pervasive service. This is how the information model for this pervasive service (i.e., PSIM) comes into place.

This section first presents a PSIM that straightforwardly reflects the rule nature of the service description. This PSIM is called EAOT-based PSIM because an extended AND-OR Tree (EAOT) is utilized. This special tree shows how the pervasive service and its policies are represented during execution. The PV nodes (i.e., the nodes representing context) need to be implemented in order to, e.g., retrieve/update the value of context. And this implementation is fulfilled by MDA models. Recall in the "POETRY Essences" section of Chapter 2, in particular in Figure 4, PSIM is

implemented via MDA models in two levels, namely, platform independent model (PIM) and platform specific model (PSM). So the two sub-sections following the EAOT subsection present how the PIM and PSM are created respectively.

3.3.1. A Generic EAOT-based Pervasive Service Information Model

A normal AND-OR Tree (AOT) contains only AND-nodes and OR-nodes, which might be enough to express the simple policies where only logical AND and OR exist, such as the one recommended by IETF Policy Working Group [IETF-policy]. However, in PoPSiDL some more complex structures are introduced such as ***modules***, ***PS***. In order to accommodate these new structures, an extension to the AOT is proposed in this Thesis: an Extended AND-OR Tree (EAOT). The EAOT tree is defined as follows.

Definition 3.2: *Comparison Unit Sub-Tree (CUST)* is a rooted binary tree whose root node is *"comparison operator" node* and whose left branch is composed of one leaf node – namely the PV on the left-hand side of the comparison operator, and whose right branch is composed of one leaf node – namely the PV on the right-hand side of the comparison operator or is empty if there is no right PV (refer to the format of *ComparisonUnit* in Section 3.2.2). comparison operator node can be one of the following: "<" node, "<=" node, ">" node, ">=" node, "==" node, "!=" node.

Definition 3.3: *Condition Sub-Tree (CST)*: as defined in Section 3.2.2, *Condition* is a series of **OR** operation of *ComparisonUnit*: *ComparisonUnit*1 **OR** *ComparisonUnit*2 **OR** ... **OR** $ComparisonUnit_n$. If n=1 then the CST is a CUST corresponding to *ComparisonUnit1*; if n>1 then the CST is a rooted tree whose root node is "**OR**" node and whose branches from left to right are the CUST's of $\{ComparisonUnit_i | i=1, 2, \ldots, n\}$ respectively.

Definition 3.4: *Conjunctive Condition Sub-Tree (CCST)*: as defined in Section 3.2.2, *PolicyCondition* is an ANDed set of *Conditions*: *Condition*1 **AND** *Condition*2 **AND** ... **AND** $Condition_n$ which is represented here by CCST. If n=1 then the CCST is a CST corresponding to Condition1; if n>1 then the CCST is a rooted tree whose root node is "**AND**" node and whose branches from left to right are the CST's of $\{Condition_i | i=1, 2, \ldots, n\}$ respectively.

Definition 3.5: *Policy Condition Sub-Tree (PCST)* is a rooted tree whose root node is "**IF**" node and whose only branch is the CCST of policy condition *PolicyCondition*.

Definition 3.6: *Policy Action Sub-Tree (PAST)*: as defined in Section 3.2.2, the action part of a policy, defined as *PolicyAction*, is a series of *Actions* that are to be executed sequentially from left to right: *Action*1, *Action2, ...,* $Action_n$. PAST is a rooted tree whose root node is "**THEN**" node and whose branches from left to right are these

leaf nodes $\{Action_i | i=1, 2, \ldots, n\}$ respectively. For example, an action node can be "ASSERT" or "DELETE".

Definition 3.7: *Policy Sub-Tree (PSB)*: is a rooted binary tree whose root node is a policy ID, whose left branch is the PCST corresponding to this policy's condition part and whose right branch is the PAST corresponding to this policy's action part.

Definition 3.8: *Module Sub-Tree (MSB)*: Suppose the composing policies of a module are *P*1, *P*2, ..., *Pn* from top down (refer to Section 3.2.2 for the definition of *Module*), then MSB is defined as a rooted tree whose root node (also called module node) is module name, whose branches from left to right are the PSB's of policies $\{P_i | i=1, 2, \ldots, n\}$ respectively.

Definition 3.9: *PoPSiDL Tree:* suppose PoPSiDL script includes *n* modules from top down: *M*1, *M*2, ..., *Mn* (refer to Section 3.2.2 for the definition of *PS*), then PoPSiDL Tree is defined as a rooted tree whose root node (also called "PS" node) is the name of a pervasive service and whose branches from left to right are the MSB's of modules $\{M_i | i=1, 2, \ldots, n\}$ respectively.

Definition 3.10: *Extended AND-OR Tree (EAOT)*: a PoPSiDL tree is also called an extended AND-OR tree or an EAOT tree.

It is clear that an EAOT not only contains the normal "AND" nodes and "OR" nodes but also contains the following extended nodes: (1) six types of comparison nodes, (2) policy condition node "IF" and policy action node "THEN", (3) policy name or id node, (4) different types of action nodes, (5) module node, (6) PS node. The addition of these nodes grants the EAOT tree more expressiveness. *The PSIM of a pervasive service is such an EAOT tree.*

The following algorithms show how to build up an EAOT tree from a pervasive service's service description in PoPSiDL. The following notations are used in the algorithms:

- NODE: a general node in an EAOT tree;
- **NEW** NODE: a method/procedure that applies for a node space and initializes it;
- insert(*child*, *parent*): a method/procedure that adds child node *child* to a parent node *parent*.

EAOTCreation (NODE **root*)
Input: PoPSiDL script file *PS*;
Output: an EAOT tree rooted from node *root*.

1. **for** each module $M \in PS$ **do**
2. *M*.insertNode(*root*);

```
M.insertNode(NODE *root)
1. newroot= NEW NODE;
2. insert (newroot, root);
3. for each policy P∈M do
4.    P.insertNode(root);

P.insertNode (NODE *root)
1. newroot = NEW NODE;
2. insert (newroot, root);

//dealing with conditions
3. newrootIF=NEW NODE;
4. insert (newrootIF, newroot);
5. if number of conditions in P >= 2 then
6.    newrootAND = NEW NODE;
7.    insert (newrootAND, newrootIF);
8.    for each condition∈P do
9.       condition.insertNode(newrootAND);
10. else
11.   condition.insertNode(newrootIF);

// dealing with actions
12. newrootTHEN=NEW NODE;
13. insert (newrootTHEN, newroot);
14. for each action∈P do
15.   insert(action, newrootTHEN);

Condition.insertNode(NODE *root)
1. if number of ComparisonUnit in Condition >=2 then
2.    newrootOR = NEW NODE;
3.    insert (newrootOR, root);
4.    for each comparisonUnit∈Condition do
5.       compaarisonUnit.insertNode(newrootOR);
6. else
7.    comparisonUnit.insertNode(root);

ComparisonUnit.insertNode(NODE *root)
1. newrootCU=NEW NODE;
2. insert(newrootCU, root);
3. insert(leftPV, newrootCU);
4. insert(rightPV, newrootCU);
```

For example, Figure 8 shows part of the EAOT-based PSIM for the SNAMU services. The particular part of SNAMU shown in Figure 8 is listed below – refer to Appendix B for the full description of the SNAMU service.

```
PS SNAMU
 MODULE BatteryModule
 BEGIN_M
       B001: If battery < 20% THEN reduce_resolution(50);
```

B002: IF battery <20% AND (urgencyFlag OR priority == HIGH) THEN execution=1;

END_M

ENDPS

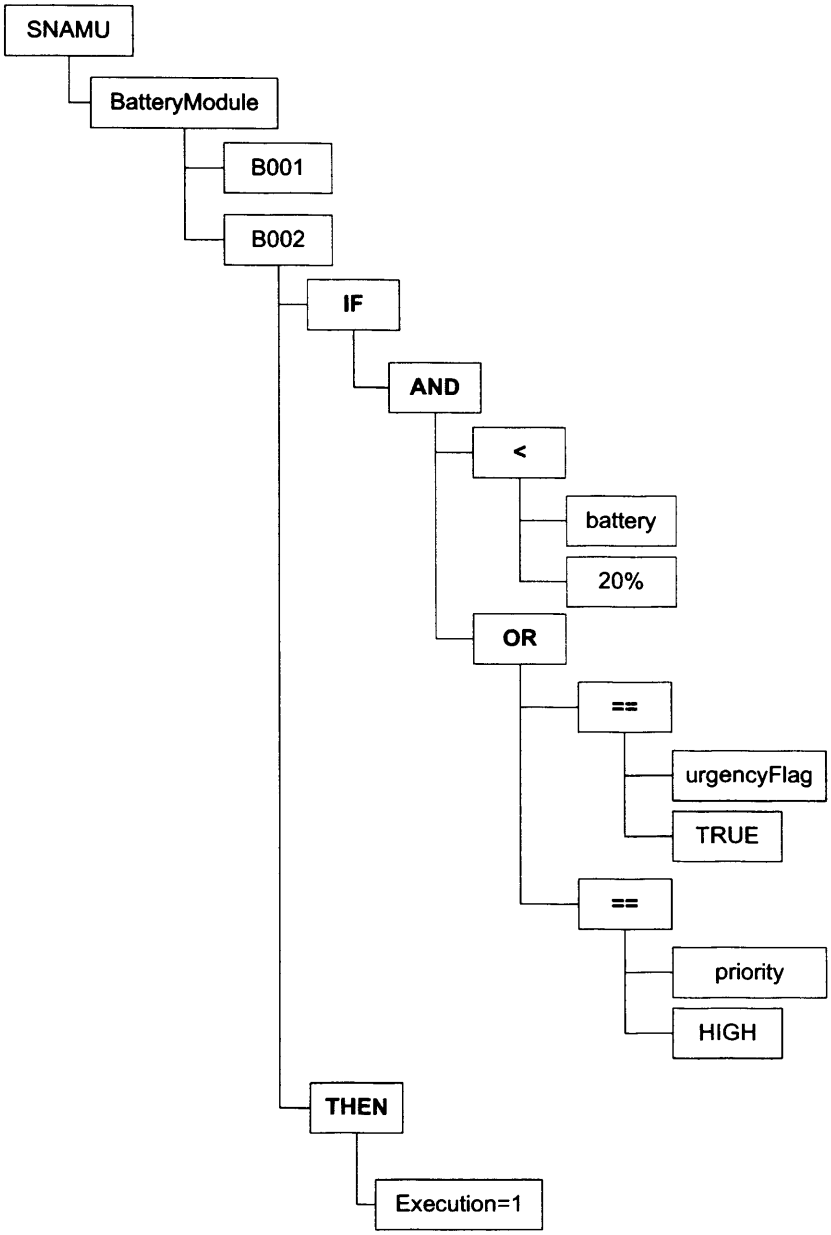


Figure 8: An Example of EAOT-based Pervasive Service Information Model

An EAOT tree gives a generic structure that represents a policy-based pervasive service. This structure is to be used by the service decision engine to select actions to be executed. As mentioned in the beginning of Section 3.3, the policy variables in the tree, such as *battery*, need to have the corresponding implementing classes. These classes are provided by MDA PIM and PSM.

3.3.2. Creation of the SNAMU PIM

Since PIM is application specific, this sub-section utilizes the SNAMU service as an example to demonstrate how the PIM part of a PSIM is created. Figure 9 illustrates the PIM of this *SNAMU* service.

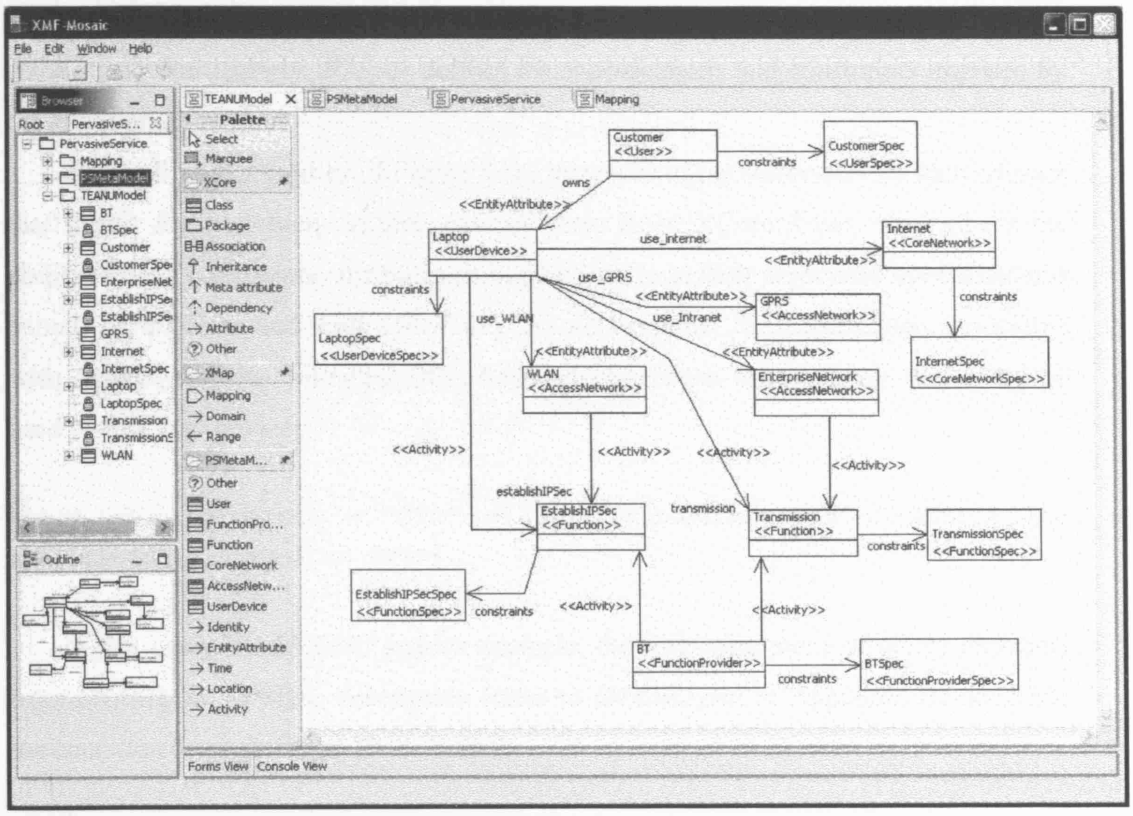


Figure 9: Part of the PIM of the Pervasive Service *SNAMU*

This SNAMU PIM indicates that all the building blocks in this diagram (including the relationships between two blocks) are sub-classes of the classes defined in the general PS modelling diagram in Figure 7. For instance, *Laptop*[*] is an instance of meta-class *UserDevice*, whereas *BT* is an instance of meta-class *FunctionProvider*. As such, this model is an instance of the pervasive service meta-model. For the sake of simplicity, this diagram only illustrates the highest-level entities and some of their specifications. In this SNAMU service, the *customer* makes use of four different types of networks: *GPRS*, *WLAN*, the *Internet* and her consulting company's enterprise network (*EnterpriseNetwork*), via her *Laptop* (or mobile phone) in order to do file *Transmission*. An IPsec tunnel may be set up by calling *EstablishIPSec* function between the *WLAN*

[*] The *Italic* font in this paragraph is specifically used to denote these blocks appearing in the SNAMU PIM diagram.

gateway in the WLAN access network and the ingress router of the *EnterpriseNetwork* depending on the customer's security level. In SNAMU, both *EstablishIPSec* and *Transmission* functions are provided by one function provider: *BT*. Individual features of the entities are defined in the accompanying *xxxSpec* constraints, e.g., *Laptop* by *LaptopSpec*, *Internet* by *InternetSpec*, *Transmission* by *TransmissionSpec*, and *BT* by *BTSpec*, etc. Particularly, *BTSpec* defines the specifications and constraints imposed by the BT internal system.

Since all model entities of Figure 9 are instances of pervasive service meta-classes that inherit *Entity*, which, in turn, extends class Root::XCore::Class, they inherit the ability to have constraints, attributes and operations (and their associated specialisations, namely, *EntitySpec* and *EntityAttribute*). As an example, the *LaptopSpec*, associated with *Laptop*, has the following OCL body which defines the protocols that could be used by this user device.

```
(self.Laptop.protocol = "FTP" or self.Laptop.protocol = "HTTP") and
self.Laptop.protocol <> "RTP"
```

Again using the *SNAMU* service example, the following piece of policy extracted from *SNAMU*'s PoPSiDL description (refer to *SNAMU.psd* in Appendix B) describes the condition to establish an IPSec tunnel:

```
CS001: IF UserDevice.protocol == "ftp" AND transmission.destIPaddress
== ConsultCompanyAddress AND securityLevel >= 5 THEN EstablishIPsec
<1>, <1>
```

This piece of policy can be translated into the following XOCL in *SNAMU* PIM:

```
context SNAMU
      @Operation WLANIPSecOp(SrcIP:NetworkSpec, DestIP:NetworkSpec,
User:String, Password:String)
        ......
    if self.User.securityLevel >= THRESHOLD_IPSec and
self.UserDevice.protocol = PROTOCOL.ftp and DestIP =
self.ConsultCompany.IPAddress
    then
      EstablishIPSec(SrcIP, DestIP, User, Password)
    end
```

Currently this procedure is carried out manually. With the research work going on, this procedure will be (semi-)automated in the future work. After a complete *SNAMU* PIM is ready, the next step is to transform this PIM to its PSM and Java source code.

3.3.3. Transformation of SNAMU PIM to PSM/Java

A MicroJava PSM and its corresponding transformation to MicroJava (a micro version of the Java programming language) have been implemented in XMF. Each element in the *SNAMU* PIM is a sub-class of XCore::Class. As every element in the XCore package has a mapping to a corresponding element in the MicroJava PSM, then the mapping from *SNAMU* PIM to MicroJava PSM is straightforward. One of the mechanisms provided by the XMF tool to transform PIM to PSM/MicroJava is XMap [XMF]. The diagram in Figure 10 illustrates how XMap was used to provide this transformation by means of mappings.

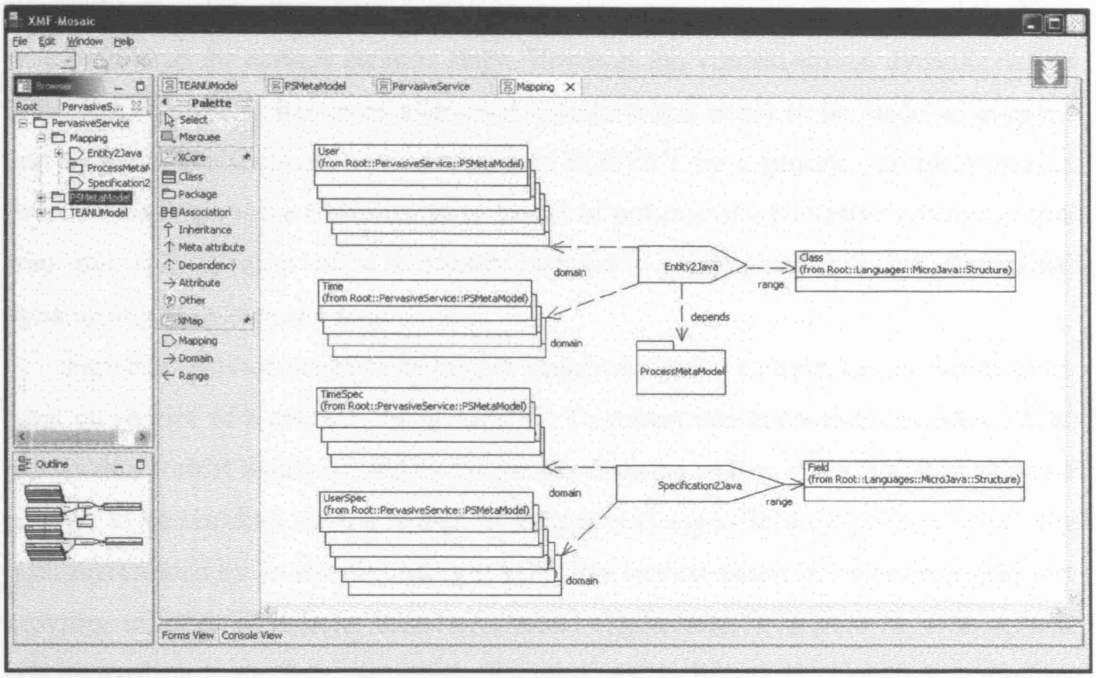


Figure 10: Transformation of SNAMU PIM to Java

A mapping consists of a sequence of *clauses* describing pattern matches. These clauses take an object with a specific structure and match it with another object of another structure. In the diagram, a mapping is represented by an arrow from source objects (the domain) to target objects (the range), and it contains pattern matches between their values. An example of simple pattern match is described by the following XMap code:

```
context Entity2Java
@Clause AccessNetwork_Clause0
  PervasiveService::AccessNetwork[name = A, attribute = B]
do
  Languages::MicroJava::Class [name = A, body = O]
  where
       O = B->collect(t | AccessNetwrok2JavaClass()(t))
End
```

So far the source code of classes that are locally available to *SNAMU* service has been generated. These classes constitute the building blocks for creating pervasive services. However, before discussing how a pervasive service is created, another important component of pervasive services, namely the service decision making engine is presented in the following section.

3.4. Policy-based Pervasive Service Decision Making

Apparently the PIM presented in the last section describes only a static set of relationships amongst SNAMU service components. In fact, due to the context-driven flexibility of a pervasive service, it is impossible and unnecessary to define all these relationships at the service creation stage. However this context-driven dynamic nature of pervasive services has to be addressed somehow and better to be made an inherent part of pervasive services. This is solved in POETRY by a generic pervasive service decision making engine reasoning over the set of policies of a pervasive service. In this way the collaboration of a pervasive service's components can be determined dynamically at service *run time.*

Pervasive service decision making is mainly driven by context, i.e., to decide to do what on receipt of a context change update. To reflect this context-driven idea, PSIM has another format which is used by the service decision engine in the first step to locate policies to be checked upon a receipt of a context change. Because policy content has been represented by its corresponding EAOT, this context-based information model just provides an index of policies based on context as illustrated in Figure 11. One context can be used by more than one policy. For example, in Figure 11, *Context*1 is used by three policies: *Policy i*, *Policy j*, and *Policy k*. And one policy can use multiple contexts such as *Policy i* which uses both *context1* and $context_n$.

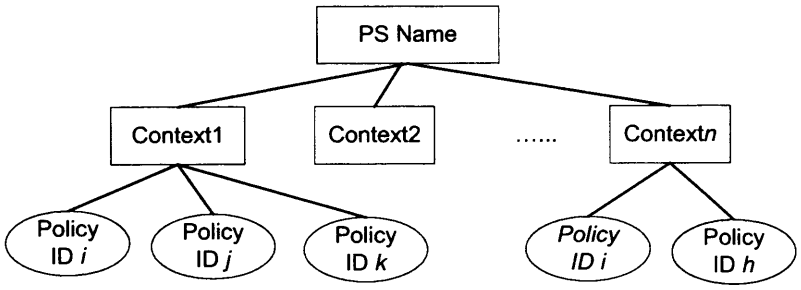


Figure 11: Context-indexed Pervasive Service Information Model

The reasoning process is described by the following *Reasoning* algorithm. It firstly finds all the policies that use the context on which a report of change has been received (Line 1). Then it further filters out these policies whose condition is TRUE, called

applicable policies (Line 2). Then the algorithm selects one policy from the set of applicable policies (Line 4) and executes its corresponding action (Line 6) until all the applicable policies are checked (Line 3).

Procedure *Reasoning*
Input: *C1*: the context information whose value is reported to have changed.
Output: none
1. $S \leftarrow$ all policies that use *C1*;
2. $S_a = \{p \mid p \in S \wedge Condition(p) = TRUE\}$;
3. **while** $S_a \neq \phi$ **do**
4. p=select_policy(S_a); //select a policy to execute
5. remove(p, S_a); //delete policy p from policy set S_a
6. p.Action.execute(); //execute the action defined by the policy p

Selecting applicable polices is straightforward – all needs to do is to check each policy's condition. If uncertainty is to be utilized by a pervasive service, e.g., in cases where a service user might not be sure how to deal with a situation, (i.e., the *uncertainty_factor* is defined to policies) then a corresponding policy triggering threshold is defined at the service level. The definition of this threshold, denoted as *PT* and $PT \in [0,1]$, can be made as a piece of *otherInfo* as part of service control information (refer to the PoPSiDL's syntax at Appendix A). Any policy whose *uncertainty_factor* is greater than the defined *PT* will be further checked on the validity of policy condition.

Selecting a policy from all the applicable policies for the current run is relatively more complex. In POETRY, it follows the following principles. Note that S_a denotes a set of applicable policies. High-priority first principle can be used to solve the notorious *policy conflict problem* in PBM [Dulay01].

1) **high-priority first**: select p_i, if

$$p_i, p_j \in S_a \wedge Action(p_i) \neq Action(p_j) \wedge priority(p_i) > priority(p_j);$$

2) **high-certainty first:** select p_i, if

$$p_i, p_j \in S_a \wedge Action(p_i) \neq Action(p_j) \wedge priority(p_i) = priority(p_j) \wedge uncertain\_factor(p_i) > uncertain\_factor(p_j);$$

3) randomly select one if both priority and the uncertainty factors of the remaining policies are same.

3.5. Pervasive Service Creation

Having designed the main components for creating a pervasive service, namely the pervasive service description language PoPSiDL in Section 3.2 and pervasive service information model in Section 3.3 and pervasive service decision engine in Section 3.4, the next thing is to integrate them together to create a pervasive service. This integration is carried out by the Service Creation Module (SCM) as illustrated in Figure 12. Figure 12 details the design of the POETRY system architecture in Chapter 2 as far as service creation is concerned.

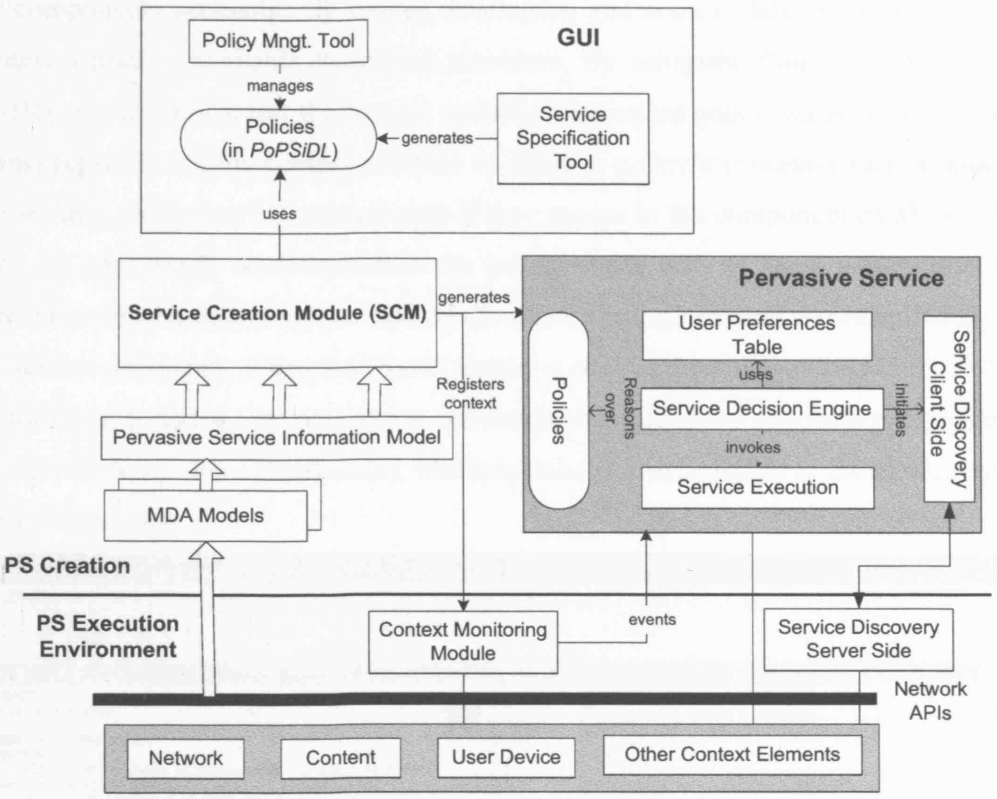


Figure 12: System Architecture for Pervasive Service Creation

The steps of creating a pervasive service are as follows – suppose the requirement analysis of the service has finished.

Step 1. Use MDA tools such as XMF to create these core and unique components that are needed by the pervasive service but are not currently available to the service provider. The components that the service developer believes can be found dynamically are left un-implemented. However these un-implemented components are each given an enabler to discover the components during service runtime. This enabler is the Service Discovery Client Side as shown in Figure 12.

Step 2. Use the POETRY service specification tool to define the service behaviours using policies.

Step 3. Then invoke the "Service Creation Module" to generate the pervasive service in Java language.

While Step 1 can be carried out by a MDA toolkit such as XMF, the POETRY system needs to implement the functions of Step 2 and 3 and this is shown in Figure 13. Firstly, policies are inputted using the *Policy Editor* Panel marked (1). The POETRY system maintains two databases: component database for keeping all the information of the components accessible by service developers, and context database storing all the context variables available to service providers. By using the Policy Editor, service developers can only select the context variable (also called policy variable in PoPSiDL terms) registered in the context database to define a policy's condition part. Similarly, actions in a policy can be selected only if they appear in the component database. This way, on one hand, semi-automates the policy input and as such makes sure the correctness of the policy syntax, and on the other hand, guarantees the executability of the service generated. When the "Save" button is clicked the policy edited in the Policy Editor Panel is added automatically to the current module of the service script shown in the Service Script Panel marked (2). Multiple policies can be added to the service script in such a manner.

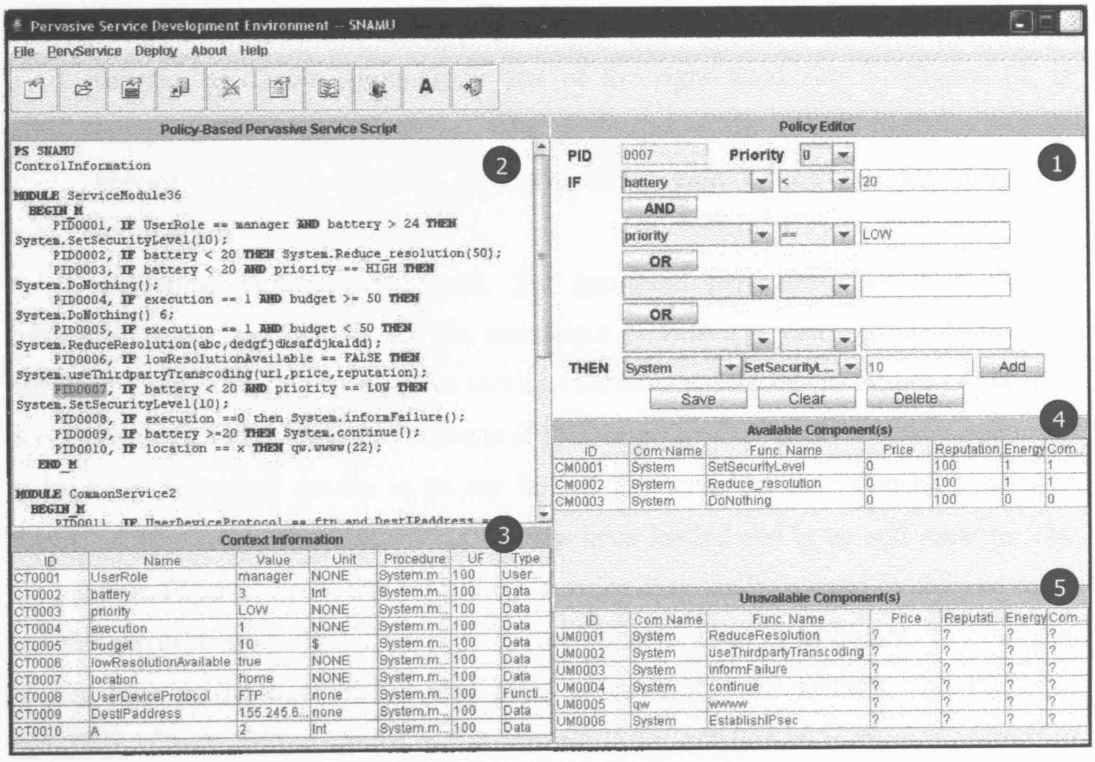


Figure 13: Graphical User Interface for Pervasive Service Creation

After the service is fully specified via policies, the PoPSiDL-based service script can be compiled using a sub-menu "PervService". The results of this compilation are threefold:

- Firstly, the PoPSiDL compiler filters out all the context information used by the service, as listed in the "Context Information" Panel marked (3) in Figure 13. Upon the subscription of the service from end users these context information will be registered with the corresponding monitoring daemon. Refer the definition of PV (policy variable) for the meaning of each field in panel (3). UF is short for Uncertainty Factor.
- Secondly, the PoPSiDL compiler also filters out all the components used by the service. If a component has been provided by the service developer (i.e., registered in the component database) then the component is put into the "Available Component" list marked (4) in Figure 13. Otherwise, it is put into the "Unavailable Components" list marked (5) in Figure 13. Implementation of the unavailable components will have to found during service runtime by using certain service discovery mechanisms – refer to the "Service Discovery Client Side" in Figure 12. At this stage, the service developer might decide to implement some of the unavailable components, register these components with the component database and then recompile the service.
- Finally, the PoPSiDL compiler generates the service. The generated pervasive service, as shown in the grey box of the right hand side of Figure 12, is composed of 5 parts: policies, user preference table, service decision engine, service execution module, and the client side of the service discovery mechanism.

Considering service deployment, the generated pervasive service is further compacted into a *.psx* file. A *.psx* file, meaning a pervasive service executable file, can run in a specially designed pervasive service execution environment. Figure 14 shows a pervasive service execution environment (PSEE) used by POETRY. It allows end users to load any pervasive service in its *psx* format. For example, the "Console" panel of Figure 14 shows that the *SNAMU* service has been loaded and is up and running. The top panel in Figure 14 lists the context information that can be altered by the end users and the corresponding right-click pop-up menu enabling any amendment. The purpose of this function is to allow the end users to customize the service. The PSEE of POETRY is very light-weighted and is suitable to be installed on resource constrained

mobile devices. And it has the feature of "install once and last forever". This PSEE can be set to run as a background process.

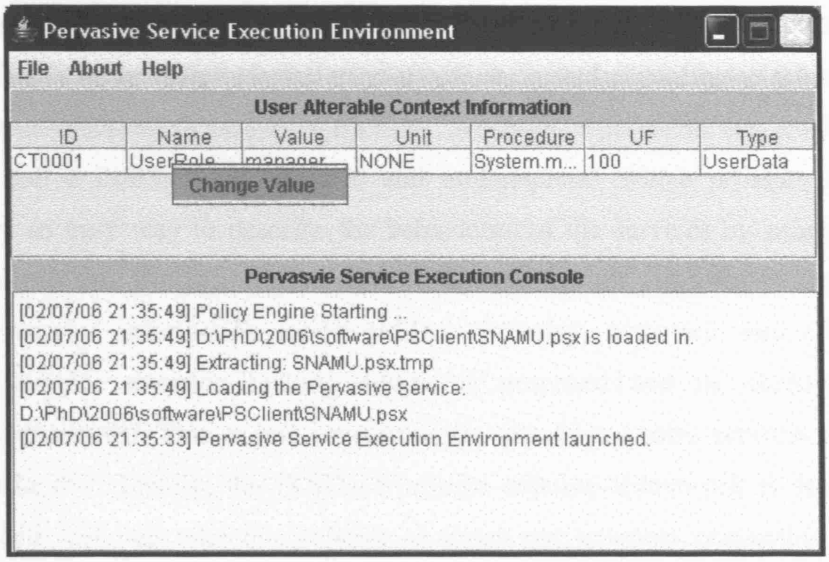


Figure 14: POETRY Service Execution Environment

The GUI in Figure 13 also provides service developers a facility to deploy the generated service. And this is implemented via the "Deploy" menu. In POETRY, the deployment of a pervasive service is to publicize the *psx* of a pervasive service to a designated web server. The web server acts as the service subscription portal of the service provider. Interested service customers can subscribe to any services via this portal. Every subscription to the *SNAMU* service creates an instance of *SNAMU* service with personalized information (e.g., different laptop or PDA physical address, different role in the enterprise network, different user preferences, etc). At the same time, the service's context variables are registered with the corresponding monitoring daemons in association with the subscriber or her/his devices.

SNAMU's execution is event-based and usually follows the following procedure: the change of the context information (and in turn the conditions of certain *SNAMU* policies) generates certain event which triggers the reasoning of the service decision engine over the *SNAMU* policies; then certain action(s), or no action if none of the conditions of the policies is satisfied, will be taken.

3.6. Summary

This Chapter presents a novel pervasive service creation solution called POETRY that takes advantages of two promising techniques: PBM and MDA. In particular, the POETRY service creation framework has investigated the thought of utilizing a policy-based management method to describe and control the internal logic of pervasive

services and the method of utilizing MDA models to describe the pervasive service information model. Apart from the above methodological novelty, the major technical contributions of the POETRY service creation framework can be summarized as follows. Firstly, a pervasive service meta-model is created according to the abstraction of the common entities of pervasive services. Secondly, a pervasive service description language called PoPSiDL is designed and implemented, which provides to service developers an easy way to describe the behaviours of the services by using linguistic rule-based policies. Thirdly, a hierarchical MDA models are proposed to represent the pervasive service information model (PSIM). Fourthly, a generic and event-driven pervasive service decision making engine is proposed and its decision making algorithms presented. This engine constitutes the core of pervasive services making the services adaptive. Finally, the POETRY service creation framework is designed and implemented that can take the policies as input and generate executable pervasive services.

Chapter 4.

Pervasive Service Discovery – Pull Method

The pervasive service created in the previous last Chapter contains a component conducting the client side function of service discovery to support dynamic service discovery. The discovered service components will be integrated into the existing pervasive service to fulfil user's requirements that arise at run-time. This chapter and the next chapter present two complementary service discovery algorithms for wireless mobile ad hoc networks. This Chapter focuses on a reactive, pull-based service discovery algorithm, i.e., model-based service discovery (MBSD). As briefly discussed in Chapter 2, the salient point of MBSD lies in the following two aspects: 1) the XML-based service description in MBSD can be automatically generated from existing MDA models; and 2) the model information carried in MBSD service description can be used to regenerate the original models which can be reused, either entirely or partially, by other mobile applications that locate at different places.

Based on the background and motivation discussion presented in Chapter 1 and Chapter 2, this Chapter starts from the rationalization of the model-based service discovery methodology. After justifying the method used, this Chapter details the design and creation of the service description language used for service discovery. Then detailed system architecture for service discovery algorithm MBSD is presented. The prototype implementation of MBSD is presented, which is followed by a performance evaluation.

4.1. A Methodology for Model-based Service Discovery

4.1.1. Service Discovery

In this sub-section, two main service discovery techniques, Jini and UPnP, initiated for wired networks are discussed first, followed by two others, DEAPspace and Konark, for wireless networks. Some features of MBSD are presented in general comparison with these service discovery mechanisms.

- Jini: Jini is a Java-based technology introduced by the Sun Microsystems. Three protocols works together to constitute the core of Jini: discovery, join and lookup [Helal02]. Jini lookup service is a directory service providing service brokerage. It

uses a multicast-based advertisement to announce its existence to the potential service clients. The service discovery is based on unicast after knowing the presence of service brokers. The service sharing strategy can be based on the business partnership known in priori. Jini technology uses the Java RMI (Remote Method Invocation) to provide service delivery (i.e., moving code around the network).

- UPnP (Universal Plug and Play): the UPnP Forum, headed by Microsoft, is in charge of the standard's developments. UPnP involves advertisement, discovery, and control of networked devices, services. Service description in UPnP is based on XML and the control messages are expressed as a collection of SOAP (Simple Object Access Protocol) objects and their URLs in the XML file. A Simple Service Discovery Protocol (SSDP) is used for service discovery via *announcement* and *search* messages. UPnP also provides automatic configuration of IP addresses via DHCP (Dynamic Host Configuration Protocol) server.
- DEAPspace: DEAPspace, developed by IBM [Hermann01], [Nidd01], evolves the service discovery from wired networks to wireless networks (i.e., *single-hop* mobile ad hoc networks). Each node equipped with DEAPspace maintains a view of all the services present in the network via periodical exchange of service information. Service discovery is carried out by a broadcast of service request message to a requester's neighbours. In DEAPspace, broadcast is scheduled sooner than usual in order to obtain prompt responsiveness and accuracy in service discovery.
- Konark: designed also for wireless networks, Konark [Lee03] furthers DEAPspace to *multi-hop* wireless ad hoc networks. Konark proposes a more efficient way for service information propagation. Each device in Konark community has a Konark SDP (Service Discovery Protocol) Manager that discovers required services on behalf of Konark applications and advertises the service information provided by the device. Konark also specifies a service description language using XML. Konark, like DEAPspace, utilizes caching of service information on each node to improve service discovery efficiency.
- MBSD proposed in this Thesis adopts a fully distributed architecture over multi-hop wireless mobile networks, i.e., each mobile node is a service provider and is its own service broker. XML is employed by MBSD to describe the service. However, rather than creating these description manually, MBSD makes re-use of

the service information already existing in the service models. Furthermore, the content structure of its service registry is well organized to reflect the domain-specific feature of mobile applications. Two service discovery mechanisms can be utilized: pull and push. In pull method, there is no need for service or directory service advertisement. Whereas in push method service advertisement is utilized for periodical update of service information (including the availability of services) across the concerned network.

4.1.2. Model-driven Approach

The Model-Driven Architecture (MDA) is an approach to IT system development fostered by the OMG. Some main terminologies and the basic principle of MDA have been covered in corresponding related work in Chapter 2.

In MDA technique family, MOF (Meta-object Facility) [MDA] is used for describing meta-data, which can be represented, for instance, by UML. On the other hand, service discovery typically utilizes meta information about services to locate a service. Rather than re-inventing metadata for services, the existing metadata that describe the features (typically via interface description) of services could be used directly for the benefit of service discovery. Since in MBSD service description is based on XML (eXtensible Mark-up Language), i.e., XML-based Service Description or XSD (more later on), a mechanism that transforms any MOF-based meta-model into an XML format is needed. This is where XMI (XML Metadata Interchange) comes into place.

XMI [MDA] is the official OMG specification for exchanging model information between modelling tools or model repositories. As a standard, XMI is part of OMG's MOF, and is thereby connected to the other MOF-related standards, such as UML and MDA. XMI uses XML as its carrier, implying that XMI can be friendly viewed by both human being and machine. Simply speaking, XMI is a way to save UML models in XML. Furthermore, XMI can be read by other modelling tools which then regenerate MDA models of the service based on the XMI.

4.1.3. MDA-based Service Discovery

In subsection 1 and 2 of this section, some representative service discovery techniques and MDA have been reviewed. The combination of these two techniques would be very powerful to provide a more precise service description and service discovery while taking advantage of the benefits of MDA for further service composition. Figure 15 illustrates how the above motivations (i.e., service description, service discovery and service composition) are neatly threaded together via MDA.

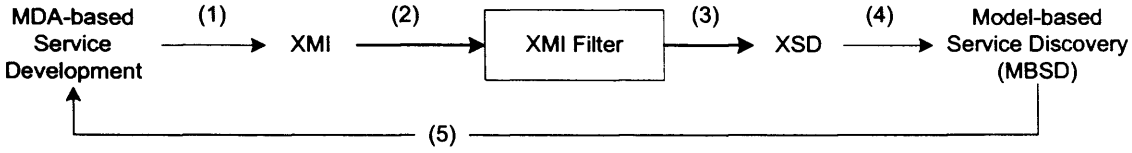


Figure 15: Model-based Service Discovery Methodology

MDA uses (and as such produces) models during its software development stage. Here MBSD utilizes platform independent models each representing a service that is composed of different implementing classes. Figure 16 shows a platform independent model for "MPEG4Decoder" service (note that for illustration purpose only some of the main classes are displayed). In this model, each square block represents a class of different functions or purposes. For instance, `VideoDecoder` *function* takes two input *parameters*: `VDecodeSetting` specifying the parameters of outputting video such as frames per second, etc and `MPEG4Video` providing the MPEG4 video that is to be decoded. And its output is `RawVideo` as a *parameter*. These models, after being mapped to platform specific models (PSM), can be transformed into source code (e.g., Java). Chapter 3 has presented how to use MDA for service development. Several other works have also addressed the application of MDA for service creation or composition [Orriens03], as this Thesis does in "Service Creation" Chapter. This Thesis goes one step further by proposing, for the first time to the best of our knowledge, to make reuse of the service information generated through this model-driven service development process for service discovery purpose.

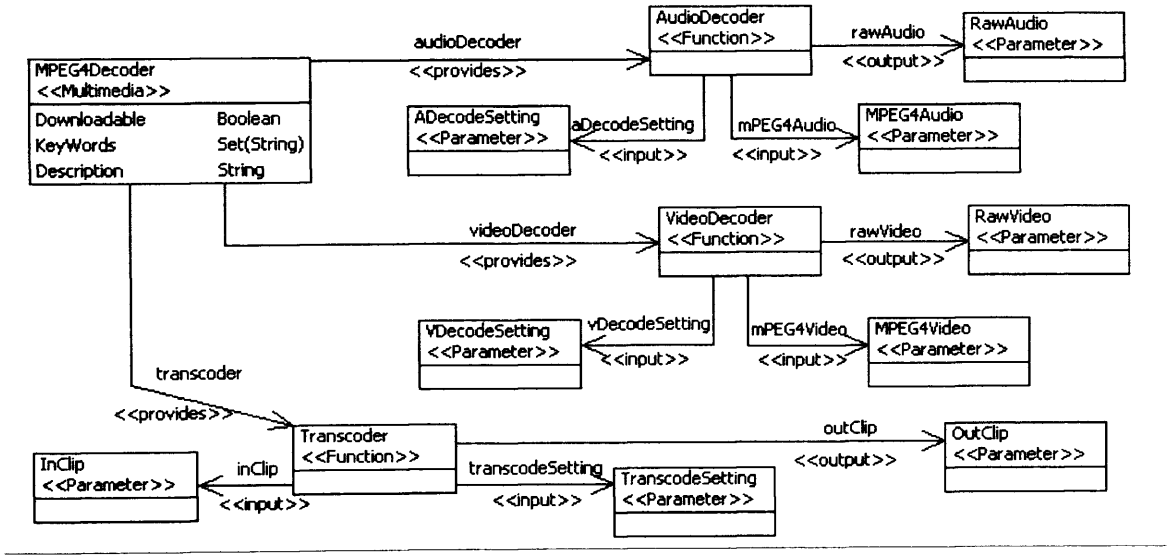


Figure 16: Model for "MPEG4Decoder" Service

This reuse of service information stored in models is fulfilled in the following manner, as illustrated in Figure 15: firstly transform the models into its XMI format (refer to Arrow (1) in Figure 15) - this transformation can be done automatically by most MDA tools such as XMF; secondly filter out from the XMI document the information that is useful for service discovery purpose (refer to Arrow (2) in Figure 15) and write these service information into XSD format (Arrow (3) in Figure 15). XSD, short for XML-based Service Description, is the language used to specify services in MBSD. Then MBSD can carry out service discovery based on the service description in XSD (refer to Arrow (4) in Figure 15). Finally the discovered service can be used as a building block to develop a new composite service or to extend an existing service (Arrow (5) in Figure 15). Functions for Arrow (1) and (5) are available in MDA tools. More explanation as to how to generate XSD from XMI is in next Section. In such a manner, service discovery mechanism MBSD matches with the service development methodology, i.e., OMG MDA and this makes significant positive impact on the future practical success of the next generation mobile systems.

The following two sections discuss how XSD document is generated automatically from an XMI document (Section 1.2) and how service discovery based on XSD is carried out in distributed wireless network environment (Section 1.3) respectively.

4.2. Service Description

Services first of all need to be described in terms of its main functions and interfaces before getting involved into service discovery procedure. Existing service discovery protocols have adopted various service description languages. For example, in Jini, services are represented by a Java interface, including not only methods that clients will invoke to use the services but also descriptive attributes [Helal02]. In SLP [Helal02], a service is defined by a service type, address, and a set of attribute-value pairs. IBM DEAPspace [Nidd01] also utilizes this attribute-value pair style for service description. Due to its cross-platform feature, XML has attracted increasing attention for service description as is the case in UPnP, WSDL, and Konark. XML provides richer mechanisms to express service description, therefore resulting in more powerful matching for service discovery. MBSD also adopts XML as the carrier of its service description, i.e., XSD (XML-based Service Description) as the name of the service description language in MBSD.

Since MBSD utilizes XMI documents to generate XSD service description, let's have a close look at an XMI document. One way of looking at XMI is as a method for

serializing UML diagrams – in most MDA tools such as XMF models are represented by UML models. For instance, Figure 17 illustrates the XMI format of the diagrams shown in Figure 16. Each diagram can be represented as XML text, stored and read back into a modelling tool. An XMI document usually consists of two sections. The first section is the header section containing information about the versions of the standards and the tool that created it. The second section contains the main content of an XMI document, mainly for representing the UML model. The XML content in the second section can be divided into two parts depending on their purposes: the first part is where the service information is stored and as such relevant to service description; while the second part is about how the model is displayed by a modelling tool and thus irrelevant to service description. For clarity purpose, Figure 17 shows only these main elements that are necessary for illustrating how an XMI document provides inputs to a service's XSD description. The elements with italic or underline fonts are of the main interest of the XMI filter. The elements with bold font provide direct input values to the XSD generation.

```
<XMI.content>
 <UML:Model name="MultiMediaServiceModel" xmi.id="Model_MMS_D0789324">

  <UML:Package name="MPEG4Decoder"                                    ← A
        xmi.id="MMS_MP4Dec_45B34F59"                                  ← B
        isRoot="false" isLeaf="false" isAbstract="false"  visibility="public">

   <UML:Class name="MPEG4Decoder"
        xmi.id="MP4_Dec_5CF99DA3"
        isRoot="false" isLeaf="false" isAbstract="false">
    <UML:ModelElement.stereotype>
     <UML:Stereotype name="Multimedia"/> //for service classification      ← C
    </UML:ModelElement.stereotype>
    <UML:Attribute KeyWords = "MPEG4|Decoder|Video|Audio|*"/>  ← D
    <UML:Attribute Description = "A decoder for MPEG4 Video."/>  ← E
    <UML:Attribute Downloadable = "True"/>                        ← F
    <UML:Attribute ServiceID = "S_MM_078D3F52A"/>
    <UML:Attribute videoDecoder = "VideoDecoder">                 ← G2-1
     <UML:ModelElement.stereotype>
      <UML:Stereotype name="provides"/>
     </UML:ModelElement.stereotype>
    </UML:Attribute>
    <UML:Attribute audioDecoder = "AudioDecoder"> ... </UML:Attribute>
    <UML:Attribute transcoder = "Transcoder"> ... </UML:Attribute>
   </UML:Class>

   <UML:Class name="VideoDecoder" .....                           ← G2-2
    <UML:ModelElement.stereotype>
     <UML:Stereotype name="Function"/>                            ← G1
    </UML:ModelElement.stereotype>
    <UML:Attribute visibility = "public"/>
    <UML:Attribute vDecodeSetting = "VDecodeSetting">             ← H2
     <UML:ModelElement.stereotype>
      <UML:Stereotype name="input"/>                              ← H1
     </UML:ModelElement.stereotype>
```

```
      </UML:Attribute>
    </UML:Class>
    ......
    <UML:Class name="VDecodeSetting" ......                    ← H2
      <UML:ModelElement.stereotype>
        <UML:Stereotype name="Parameter"/>                     ← H1
      </UML:ModelElement.stereotype>
      <UML:Attribute visibility = "public"/>
    </UML:Class>

  </UML:Package>
 </UML:Model>
</XMI.content>
</XMI>
```

Figure 17: XMI Document of the MPEG4Decoder Service

Based on the XMI document in Figure 17, the XSD document shown in Figure 18 is generated automatically by the XMI filter (refer to Figure 15). This XML document in Figure 18 also exemplifies the essence of the XSD language itself. The numbered arrows in the XSD document match the corresponding arrows in the XMI document, aiming to provide self-explanation as to how each value in XSD is obtained from its corresponding XMI document. These attributes that are not available in the XMI document such as *codeBaseURI* and *XMI_Link* are usually related to service deployment and as such their values are the locations where they are stored on the web-based deployment server. *XMI_Link* points to a service's XMI document which can be used by MDA tools to regenerate service models at the requester side. *XMI_Link* is empty if the service is not developed using MDA methodology. *codeBaseURI* stores the real implementation of the service objects that could be delivered to the client side by MBSD. *Downloadable* element is related to service delivery and it defines if the service provider allows the service objects to be downloaded to the requesting client. If not downloadable then the client has to use the service remotely (e.g., using Sun Java RMI). *Price* is given by the service provider as part of its business model or marketing plan. For normal users who are happy to share their applications freely on their mobile devices the *Price* tag is default value 0. *Functions* are these complex attributes that have the following features: 1) contain sub-tags (refer to G2-1 in the XMI document in Figure 17); 2) and have separate classes each defining further details of this function (refer to G2-2 in Figure 17); 3) and finally these classes each has a *Stereotype* of *Function* (refer to G1 in Figure 17). The XMI filter can automatically get these values and assign them to the corresponding attributes. The XMI filter is a Java program that retrieves information from an XMI document using the standard XML parsers and writes this information into another XML file following the XSD format.

```
<MobileService>
<-- basic service information -->
<ServiceName>MPEG4Decoder</ServiceName>  //match to package name in XMI        ← A
 <ServiceID>MMS_MP4Dec_45B34F59</ServiceID>                                     ← B
 <ServiceClassification>Multimedia</ServiceClassification>                      ← C
 <Keywords>  <-- for matching search requests --> // defined by service developers in MDA
tools                                                                          ← D
   <Word>MPEG4</Word>
   <Word>DECODER</Word>
   <Word>VIDEO</Word>
   ......
 </Keywords>
<Description>A decoder for MPEG4 Video</Description>                           ← E

 <Attributes>
   <Downloadable>TRUE</Downloadable>                                           ← F

<codeBaseURI>http://panda.ucl.ac.uk/multimedia/panda_MPEG4_Decoder.jar</codeBase
URI>
   <Price>0</Price>
   <XMI_Link>http://panda.ucl.ac.uk/ multimedia/panda_MPEG4_Decoder.xmi</XMI_Link>
 </Attributes>

 <Functions>
   <Function>                                                                  ← G1
     <fName>VideoDecoder</fName>                                               ← G2
     <fInputs>
        <fInput1> VDecodeSetting</fInput1>                                     ← H1&H2
        <fInput2> MPEG4Video</fInput2>
     </fInputs>
     <fOutputs>
       <fOutput1>RawVideo</fOutput1>
     </fOutputs>
   </Function>
.....// many other functions can be defined here, such as AudioDecoder, etc.
 </Functions>
</MobileService>
```

Figure 18: XSD Document of the MPEG4Decoder Service

The service information contained in the XSD document is stored in a designated directory at the service provider side waiting to be searched by the MBSD system.

There are two ways to get a service provider's service information known to service users: *proactive* and *reactive*, according to the time service information is publicised. In the proactive method, service advertisement messages are exchanged on a periodical basis and before the initiation of service discovery (i.e., in advance). This method is also called *push* because service information is pushed (via advertisement) to the client side. The reactive method asks for service information only at the time of service discovery. In this case the service information is pulled to the requesting client and thus is also called *pull* method or *on-demand* method for service discovery. This Thesis is to look into both methods. The remainder of this chapter first presents the overall system architecture of MBSD where only pull method is applied, then service

request broadcasting protocols and evaluations follow. Push method is to be discussed in the next chapter.

4.3. MBSD System Architecture

4.3.1. Overall System Architecture of MBSD

MBSD adopts a distributed architecture, namely, there is not a centralized service broker and it allows each mobile device to act as a server providing service and at the same time a client consuming services. So each mobile device has an identical MBSD system, whose protocol stack and architecture are illustrated in Figure 19 and Figure 20 respectively.

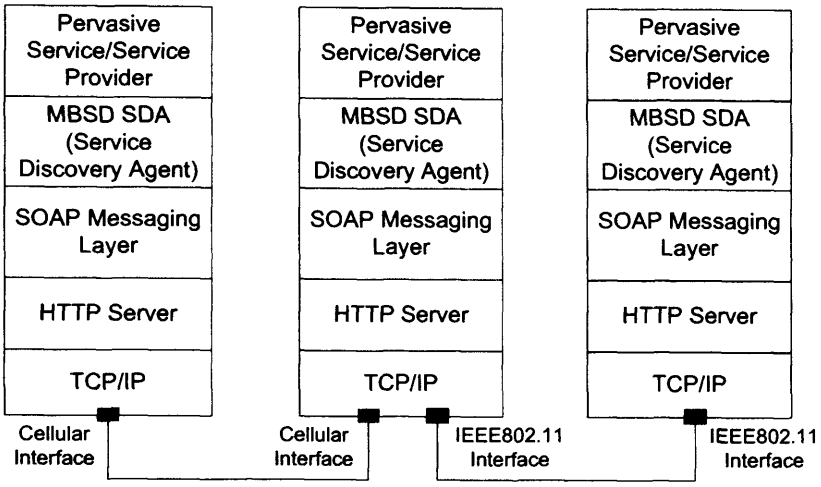


Figure 19: MBSD Service Discovery Protocol Stack

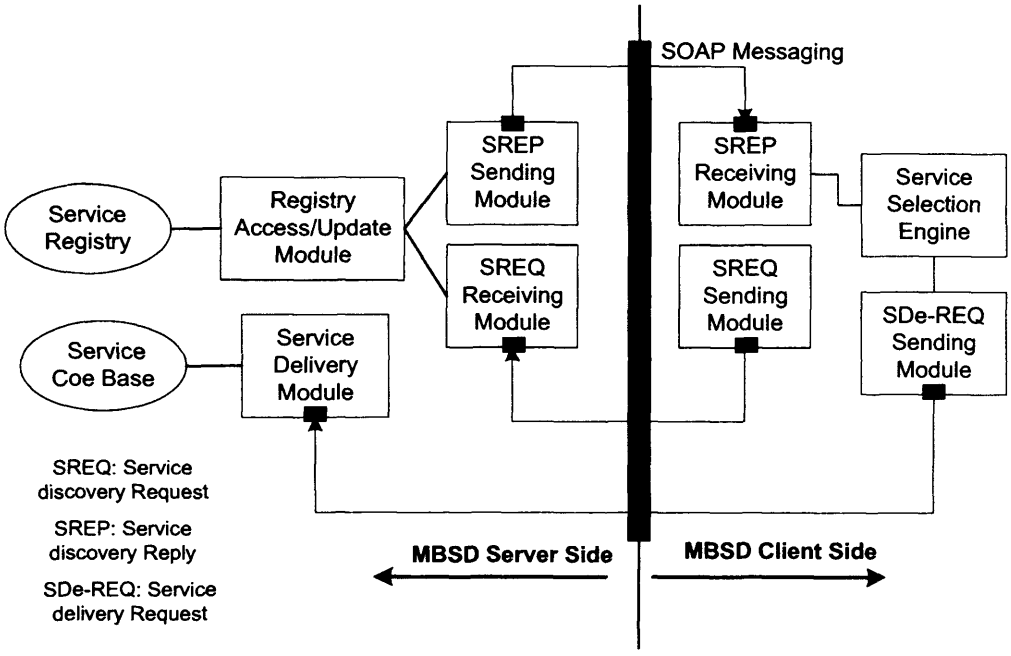


Figure 20: MBSD System Architecture

Each service provider in the MBSD community maintains a *service registry* to store information about services it makes available to other users. Service registry appears as an XML document maintaining multiple entries each describing a service in XSD language, such as *MPEG4Decoder* service. The location of the registry XML file is known to its local SDA (Service Discovery Agent) and the SDA can use normal XML parser to retrieve elements from the XML file and carries out matching.

SOAP (Simple Object Access Protocol) over HTTP is utilized for service discovery and delivery message passing across physical boundary. The generic HTTP can hide the heterogeneity of different hardware and software platforms utilized by different service providers as such achieving cross-platform service discovery and delivery. The XML nature of SOAP matches naturally with the format of service description in XSD – XML. As illustrated in Figure 19, service discovery messages can be physically transmitted via various air interfaces, such as cellular or ad hoc network interface (e.g., IEEE 802.11) depending on the hardware and software features of a mobile device.

4.3.2. Service Registry

The increase in the complexity of the next generation mobile systems implies a collectively large quantity of underpinning service components in terms of both number and varieties. As such service information in service registry should be stored in a manageable (e.g., well classified) and user-understandable manner. To this end, MBSD adopts a tree-based service registry where more generic service classifications reside at the higher levels of the tree and service classifications become more specific as the tree levels goes down. The registry on each service provider can start from different levels but as a branch of a generic mobile service tree. This methodology of grouping services into categories is also used by Konark [Lee03].

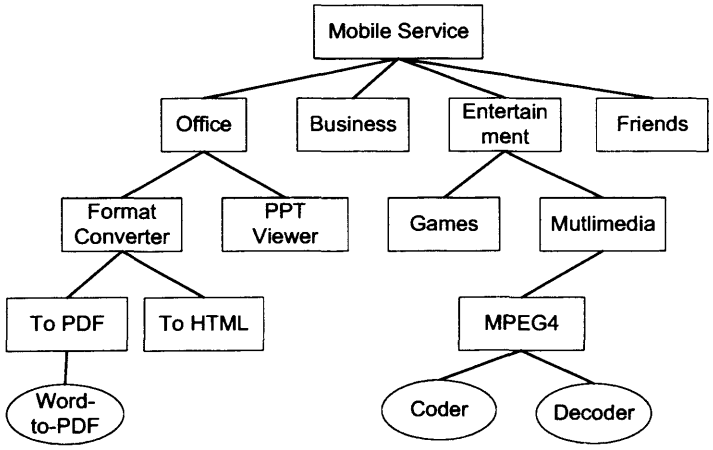


Figure 21: Example of a Pervasive Service Tree

Figure 21 illustrates such an example. This service tree potentially provides a template to the future MOF semantic structure for pervasive services. This hierarchical service organization is reflected in XSD via its *ServiceClassification* tag. For instance, in Figure 18, the *MPEG4Decoder* service is a kind of *Multimedia* services.

Figure 22 illustrates the logical structure of a service registry. Within a service XSD, there are links pointing to the service XMI document (*XMI_Link* in Figure 18) and service objects (*codeBaseURI* in Figure 18). Service XMI document is to be used by MDA tools to regenerate service models. The location of service objects is to be used by the SDA to deliver services.

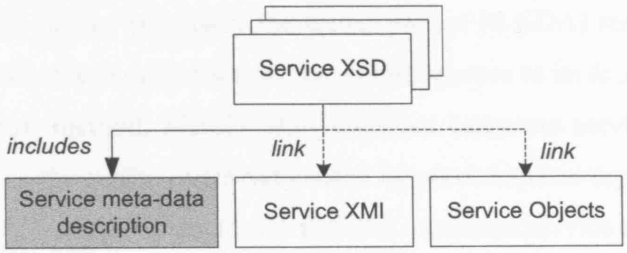


Figure 22: Logical Structure of a Service Registry

4.3.3. MBSD Operation

Figure 23 shows how MBSD operates when a pull mechanism for service discovery is employed.

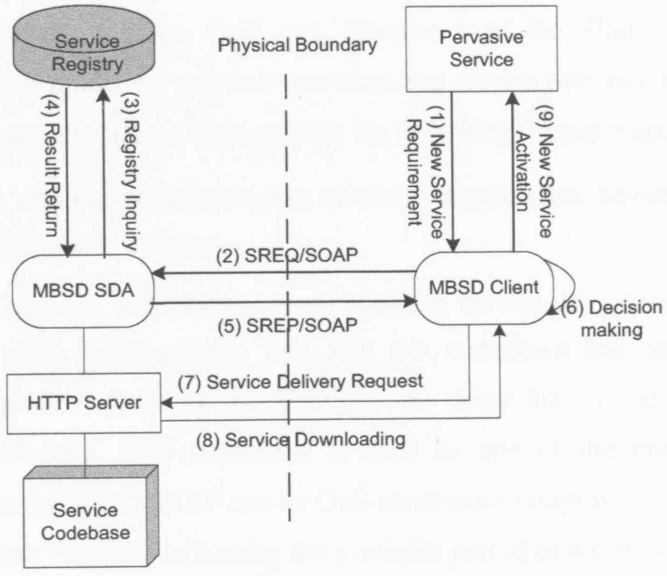


Figure 23: MBSD Operation Workflow

When a mobile application cannot find a component service in its local service registry, it interacts with the client side of SDA via local communication to make a new service request (refer to Step 1 in Figure 23), which triggers the service discovery

process. Upon receiving a new service request, the SDA client sends out a SREQ (Service REQuest) message (Step 2). Depending on its knowledge of potential service providers, the application (or the user of the mobile device in most cases) can adopt one of the following three approaches for efficient service discovery. 1) If a mobile device knows already where to get a cost-effective and quality implementation of this service then unicast is utilized, i.e., the SREQ is sent directly to this service provider. 2) If a mobile device knows nothing about the whereabouts of the requested service, SREQ is broadcast. 3) If a mobile device knows a list of service providers (e.g., via its business alliance and the user's friend list), then multicast is utilized.

When a mobile device (precisely the server part of its SDA) receives this SREQ, SDA tries to discover the required service by getting access to its *local* service registry (Step 3). In this pull method, MBSD SDA does not carry out service advertisement. Doing so can reduce the traffic overhead caused by service discovery mechanisms. If a SDA, upon receipt of a SREQ, could not find the requested service in its local service registry it simply does nothing. If a match is found (Step 4) then SDA packs the information of the discovered service in SREP (Service discovery REPly) message and sends it back to the service requester (Step 5). Multiple SREP messages might come back to the service requester as such bringing back more than one choice. In this case, a decision making is carried out at the client side (Step 6) to select the most proper one based on each service's price, QoS, etc. Chapter 6 of the Thesis is to present a comprehensive QoS model for pervasive services and service selection algorithms.

SREQ message and SREP message take the following format respectively:

- <SREQ_id, source, destination_list, service_classification, keywords, max_price, service_description, options>
- <SREP_id, source, destination, service_name, service_id, service_classification, keywords, price, downloadable_flag, XMl_link, codeBase_link, options>

Maximum price is included in SREQ if the client knows the targeted service provider is trustworthy. This parameter is used as one of the criteria for service matching. *Options* in SREQ/SREP can be QoS parameters (such as delay, media fidelity, etc.), or TTL (Time To Live) indicating the available period of a service.

After a proper service implementation is selected at Step 6, service delivery is invoked (Step 7&8). Service delivery is concerned about how a discovered service is used by a client. In MBSD, service delivery is carried out by a HTTP server locating at each service provider. If the service objects are downloadable (e.g., they do not use any native method or its execution is not dependant on any resource at the server side),

namely, the *Downloadable* tag is the XSD document (refer to Figure 18) is true, then the client prefers to download it to local to use the service locally. The idea is to avoid network communication as much as possible for security and performance efficiency purposes. An extension to this can be to allow the client side to negotiate with the service provider on service price and remote execution performance and then to make choice of downloading or not. However, whatever the way the service objects are executed (locally or remotely) the corresponding model information (as documented in XMI – refer to the *XMI_Link* tag in the XSD document) is downloaded to the client (Step 9) side for the next-step composite service creation or re-composition using MDA tools.

The service downloading procedure at a client side is described by the following pseudo code: `downloadService()`.

Procedure downloadService ( )
Input: *s* – the XSD file of the selected service
Output: none

1. **if** *s.Downloadable* == **TRUE then**
2. pay the downloadable service fare: *s.serviceFare*;
3. download service objects; // then use the service locally;
4. **else**
5. pay un-downloadable service fare: *s.serviceFare*;
6. download client stubs; // then use the service remotely;
7. **if** *s.XMILink* != **nil then**
8. download XMI document;

End

4.4. Service Discovery Broadcasting Protocol

4.4.1. Overview

The SOAP-wrapped SREQ message usually needs to be broadcast to the areas of networks concerned (e.g., within *k* hops from the service requester if not the whole wireless networks). SREQ broadcasting is a process where a SREQ sent by one node in a wireless network is received by all other nodes in the network concerned. In wireless networks where omni-antennas are normally utilized for signal transmission, radio signals from nearby nodes are likely to overlap with each other, thus making the broadcast via flooding result in serious redundancy, contention and thus collision, or *broadcast storm* problem as termed in [Tseng02]. Plenty of improved schemes have been proposed to reduce the number of redundant rebroadcasts so as to alleviate the seriousness of the broadcast storm problem while stilling ensuring (at least with high probability) that the broadcast message is delivered to all nodes in the network. The

survey and comparison works described in [Tseng02, Williams02, Stojmenovic04] collectively present a comprehensive overall view of the existing broadcasting algorithms in wireless ad hoc networks.

The fundamental problem to be solved by any broadcasting algorithm in wireless networks can be simply summarized as being for broadcast message receiver to decide whether to re-broadcast the received message or not. As to how to make this decision, there are basically two approaches according to the place the decision is made: sender-deciding and receiver-deciding. In the case of sender-deciding approach, before sender broadcasts a message it needs to generate a broadcast relay list (BRL) that contains all its next-hop neighbours that are going to re-broadcast the same message, and then piggybacks this list with the message and sends them out. Then what receiver needs to do is just to check if it is in this RBL list. If yes then it generates a new BRL and re-broadcast the message to its neighbours. Otherwise the receiver simply ignores the message. In the sender-deciding approach, the complexity of the decision making resides at the sender side. Whereas for receiver-deciding mechanism, sender simply sends out the packet and leaves the receiver to decide whether the packet needs to be broadcast or not. All the existing broadcasting algorithms, regardless of their complexity, can be classified to be either of the two. E.g., dominant pruning, multipoint relaying, AHBP (Ad-hoc Broadcast Protocol), CDS-based broadcasting algorithm as described in [Williams02] are all sender-deciding algorithms.

In terms of SREQ broadcasting, there is no detailed explanation in Konark. Konark utilizes the Gossip protocol for service discovery [Lee03]. There is a routing protocol in MANETs that shares the same name. However, it is not clear if the Konark Gossip is the same as that in MANETs. DEAPspace regards broadcasting as an important factor on the performance of service discovery. It discusses about the broadcast interval and its power-saving aspect. However, these discussions can hardly bring fundamental benefit to SREQ broadcast or service advertisement if there is no way to finer tune the broadcast mechanism itself. These facts have all motivated this Thesis to investigate a new SREQ broadcasting algorithm - MBSD-b.

MBSD-b investigates the receiver-deciding approach. MBSD-b makes use of location information to help reduce the number of nodes sending broadcast packets. It is believed that the advantages of using location information in MANETs outweigh its additional cost [Stojmenovic02b]. A location information service (LIS) is assumed to be available in the MBSD-b-supportive wireless mobile networks, such as the one presented in [Shen05]. The mechanism to be introduced in the next Chapter, which is

mainly for service advertisement, can also serve as a kind of efficient location update scheme because the node's location information can be piggybacked with the high-level service information for propagation. Furthermore, considering the fact that a pervasive service, which is typically context-aware, usually has more interactions with its nearby nodes than these nodes further away, this Thesis introduces a *grid* concept. In the grid concept, the part of the environment that is close to the sending node is partitioned into a number of grids. Then the broadcasts are carried out at the grid-level rather than the node-level. Since the number of grids is smaller than that of nodes, higher performance is expected to be achieved.

4.4.2. Data Structure and Terminologies Used

MBSD-b maintains a Message Duplication Table (MDT). The entries of the MDT on any node *a* record the information about SREQ messages that were recently received by node *a*. Node *a* uses MDT table to determine if a newly received message is a duplicate one or not. For each entry (i.e. each message) a list of nodes that are in the same grid as node *a* is maintained. MDT is formally defined as follow.

Definition 4.1*: the MDT on node a is defined as:* MDT_a*::=Entry* where Entry::=<MsgID, SrcID, NL>. * means there can be no entry, one entry or multiple entries; MsgID specifies the ID of the message this entry is about; SrcID defines the source the message is from; and NL specifies a node list:* $NL = \{n \mid grid(n)==grid(a)\}$, where *grid (n) returns the grid in which node n locates.*

If there exists one node in the *NL* that has broadcasted the message, node *a* will not broadcast this message. This way helps drop the number of broadcasting messages significantly.

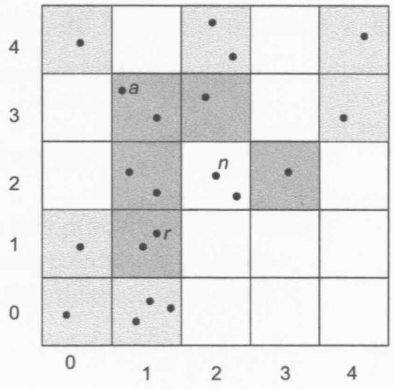


Figure 24: Broadcast Relay Gateways Selection

(one-hop active grids are in dark colour and two-hop active grids are in light colour)

Before embarking on the MBSD-b description itself, there are several terms surrounding the grid concept that need to be clarified. When a node sends or retransmits a broadcast message to other nodes in wireless ad hoc networks, it broadcasts the message to all its neighbours. In MBSD-b, when a neighbour node (e.g., *r*) receives a broadcast massage sent by node *n*, it first of all partitions the surrounding area of *message sending node n* into 24 *d*d* square grids, as shown in Figure 24. The 8 grids immediately surrounding node *n* are called *one-hop grids* of *n*. The set of the one-hop grids of *n* is denoted as *G*1(*n*). The 16 grids surrounding the 8 one-hop grids are called *two-hop grids* of *n*. The set of the two-hop grids of *n* is denoted as *G*2(*n*). It is further defined that a one-hop grid that contains at least one node as *one-hop active grid* (AG1) and denote the set of AG1 as *AG1*(*n*). Similarly, there are *two-hop active grids* (AG2) which mean these two-hop grids that contain at least one node each and *AG*2(*n*) denoting the set of AG2. For instance, in Figure 24, there are 4 AG1 as coloured in dark grey and 7 AG2 as coloured in light grey. We term these AG1 that are to be used to relay broadcast packet as *broadcast relay grids* (*BRGs*) and these nodes in AG1 that are to be employed to broadcast packet as *broadcast relay nodes* (*BRNs*)

4.4.3. Protocol Operations

In MBSD-b, when the neighbour node *r* receives a broadcasting message from node *n*, after partitioning the surrounding area of node *n* into square grids (refer to Figure 24), it calculates broadcast relay grids (BRGs) and decides if the node *r* itself is in one of these grids. If yes, it inserts the message into its sending queue and waits for t_w, and then rebroadcasts the message if no node in the same grid has ever sent the message during the waiting period. MBSD-b is presented in the procedure *MBSD-b (r, m, n)*, which describes the operation of MBSD-b on node *r* upon receipt of a broadcast message *m* (e.g., SREQ) from a neighbour node *n*.

Procedure MBSD-b *(r, m, n)*
Input:
r: the node on which this procedure is invoked;
m: the broadcast message received by node *r*;
n: the node that sent the message *m* to node *r*;
Output:
None
Purpose:
To decide if a received SREQ message needs to be broadcast by the receiving node itself or not

1. **if** *m* received the first time **then**
2. gridMapping(); // Map nodes into grids – described later

```
3.    BRG ← calc_BRG (n);      // Calculates BRGs of n – described later
4.    if r ∈ BRG then
5.       decide waiting time t_w;
6.       insert m into the broadcasting queue Q;
7.       insert each node i where grid(i)=grid(r) into NL_m in MDT_r;
8.  else // means the same message is received again
9.    if n ∈ NL then      // means r receives m from node j where grid(j)=grid(r)
10.      delete m from Q if m is in the queue;
11. while t_w not timeouts do
12.   wait();
13. if m ∈ Q then
14.   rebroadcast m;
    End
```

The nodes in the same grid are assigned different waiting time to avoid clash caused by simultaneous multiple transmissions. The criterion to decide the waiting time t_w of B in a relay grid is not unique. It can be scaled with the distance to the sending node, or can be scaled with the energy of the node r. In this Thesis's simulation, the distance to the previous BRG is used to decide t_w, i.e., the longer the distance is the shorter this waiting time t_w is. This implies that the nodes in broadcast relay grids closer to the edge of radio range send the message more quickly.

The essence of MBSD-b algorithm is the fact that in a given grid there is only one node that needs to re-send the broadcast message. In this way a significant number of redundant broadcast messages can be reduced, as to be demonstrated in the algorithm evaluation section. When node r receives the same broadcast message m again, it checks if the message was sent by a node within the same grid as itself. If yes, which means a node in the same grid has sent out the broadcast message, then node r stops sending the same message again.

4.4.4. Choosing the Grid Side Length

The choice of grid side length l is critical to the performance of the MBSD-b protocol. Side length of grids l should be chosen in such a way that it makes radio signal of n reach as much as possible all the surrounding 8 grids. Let R be the transmission radius of a radio signal. Three possibilities of l, as depicted in Figure 25, are discussed.

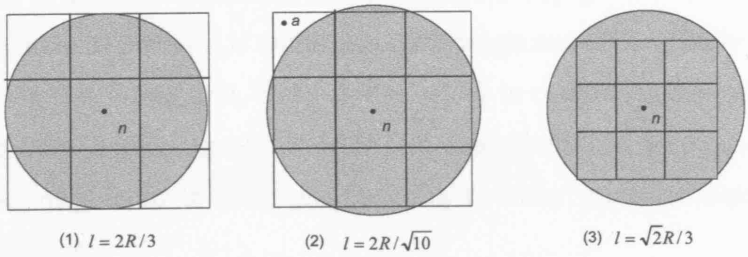


Figure 25: Different Grid Side Lengths

(1) $l = 2R/3$: This represents the situation where the nodes located at the centre of a grid cannot cover the whole area of any neighbouring grid. The nodes located outside the radio transmission range of the central node should not be selected as BRNs.

(2) $l = 2R/\sqrt{10}$: This represents the case where a node located at the centre of a grid is capable of talking to any gateway of its 4 side neighbouring grids, but can't cover the whole area of 4 grids in the corner. That is to say, when a relay grid is in the corner, there has to be mechanism in place to guarantee the selected BRN is within the radio transmission range of the node located at the centre of a grid.

(3) $l = \sqrt{2}R/3$: This represents a situation where any node in any one of the 8 one-hop grids is within the transmission range of the central node and as such is capable of talking to the central node *n*.

A larger value of l (case (1) and (2)) may lead to a situation where when a grid has been chosen as a broadcast relay grid (BRG), there is no node in this grid that can be chosen as a new BRN because all nodes in this BRG locate outside the radio transmission range of node *n* (e.g., node *a* in case (2) of Figure 25). On the other hand, a smaller value of l (case (3)) will lead to a situation where some nodes capable of communicating with the node *n* in two hops can not be considered as new BRNs because they are mapped outside the two-hop grids. This situation will reduce the number of BRGs and BRNs, and make some of broadcast message can not be received. At the same time, the number of broadcast hops and end to end delay will increase because new BRNs are closer to the initial BRN.

MBSD-b utilizes a side length of $l = 2R/\sqrt{10}$, which is a trade-off of case (1) and (3).

4.4.5. Dealing with Nodes in Four One-hop Corner Grids

In the case of $l = 2R/\sqrt{10}$, initial BRN *n* cannot cover fully the one-hop grids in four corners. For instance, node *a* in case (2) as shown in Figure 25 locates outside the transmission range of central node *n* – these nodes such as node *a* are called *orphan nodes*. In order to avoid the situation where such a one-hop grid at corner is selected as a BRG but no node of this grid is within the radio range and consequently no new BRN can be chosen in a one-hop grid, these orphan nodes in one-hop corner grid need to be treated at the very beginning of the MBSD-b algorithm. The method employed by MBSD-b is to map them to their neighbouring two-hop grids, as described by the following procedure.

Procedure gridMapping*(r, n)*
Input:
r: the node on which this procedure is invoked;
n: the node that sent a SREQ to the node *r*;
Output:
None
Purpose:
To map all the nodes within the transmission range of the sending node *n* into two-layer grids
Precondition:
r: the transmission range of nodes

1. **for** each node in one-hop grids **do**
2. **if** the node is in one of the four corner grids **then** {
3. $d \leftarrow$ the distance between the node and the previous BRN;
4. **if** $d > R$ **then**
5. Map the node into the corresponding on of three two-hop neighbour grids in the corner;
6. Delete the node from the one-hop grid;

End

For example, node *a* in Figure 24 is mapped to grids g_{03}, g_{04} or g_{14} (grids are numbered g_{xy} following the conventional *xy*-coordinates). The reason why an orphan node is mapped to three neighbouring corner two-hop grids rather than only one of them is to maximize the chance of this orphan node being covered by one-hop BRGs.

4.4.6. Relay Grids Calculation

The following procedure *calc_BRG(r, n)* describes how node *r*, upon receiving broadcasting SREQ *m* from a neighbour node *n*, to calculate the broadcast relay grids of node *n*. Let *AG1(n)* and *AG2(n)* be the sets of one-hop and two-hop active grids respectively, and *R(n)* the set of relay grids.

Procedure calc_BRG*(r, n)*
Input:
r: the node on which this procedure is invoked;
n: the node that sent a SREQ to the node *r*;
Output:
R(n): a set of relay grids
Purpose:
To calculate relay grids

1. $R(n) \longleftarrow \phi$;
2. **for** each $k \in AG2(n)$ **do**
3. **if** *k* has only one neighbour grid $j \in AG1(n)$ **then**
4. Put *j* into *R(n)*;
5. Delete *j* from *AG1(n)* ;
6. Delete two-hop neighbour grids of *j* from *AG2(n)*;
7. **elseif** *k* has no neighbour grid $j \in AG1(n)$ **then**

// k cannot be covered by any one-hop active grid
8. Delete k from *AG2(n);*

// deal with the remaining grids
9. **while** there still exists $h \in AG2(n)$ **do**
10. **for** each $g \in AG1(n)$ **do**
11. Compute the number of two-hop active grids of g in *AG2(n)*;
12. Add into *R(n)* the one-hop active grid g_m whose number is maximum; // always select first the grid that covers the maximum number of 2-hop neighbour grids
13. Delete g_m from *AG1(n)* ;
14. Delete all the two-hop neighbour grids of g_m from *AG2(n)*;

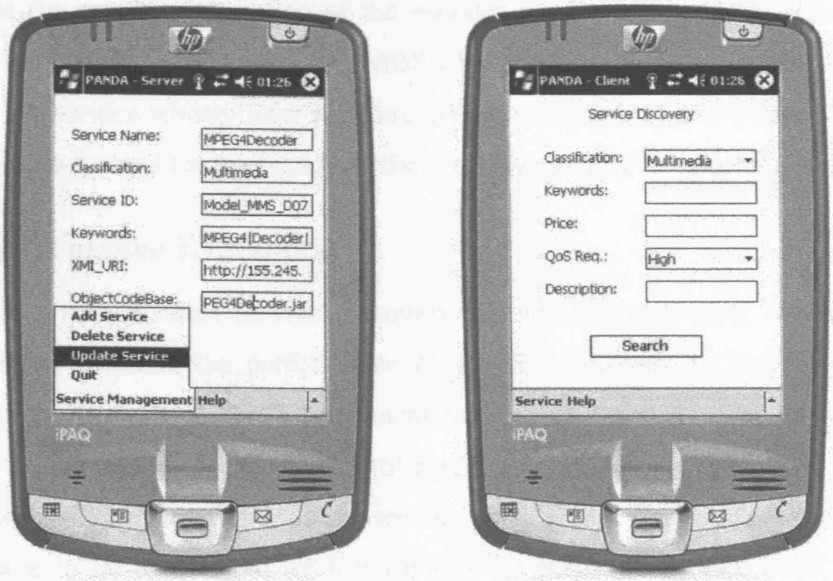
End

4.4.7. Discussion on the Algorithm Correctness

MBSD-b maps the neighbours of the sending node n into the one-hop and two-hop grids. Then node n regards every one-hop grid $g1 \in AG1(n)$ as a one-hop neighbour node $j \in N_1(n)$ and every two-hop grid $g2 \in AG2(n)$ as a two-hop node $k \in N_2(n)$. Then the part of the network (denoted as N_p) centred on node n is transformed to a network (denoted as N_g) whose nodes are grids and whose links represent the connected relationship of neighbouring grids. Procedure *calc_BRG(r, n)* uses a greedy algorithm to find BRGs (broadcast relay grids) in one-hop grids. These BRGs constitute a localized minimum connected dominating set [Stojmenovic02a] - a set is dominating if all the nodes in the system are either in the set or are neighbours of nodes in the set. Because BRGs are found from active grids which have at least one node, the BRNs (broadcast relay nodes) selected within these BRGs also constitute a connection dominating set thus proving the correctness of the MBSD-b algorithm.

4.5. MBSD Prototype Implementation and Experimental Validation

A prototype of MBSD service discovery mechanism has been implemented as part of POETRY. The experiments were carried out on two versions of Java platform, i.e., J2SE and J2ME by using two iPAQ HX2750 (OS: Microsoft Pocket PC 2003, JVM: IBM J9) and an IBM laptop X31 (PM-1.5GHz, 256M, OS: Redhat Linux 9, J2SE JVM 1.4.1), all equipped with an IEEE 802.11b wireless network adaptor. An ad hoc network is dynamically formed. Simulation on large-scale wireless mobile ad hoc networks using network simulator ns-2 is to be discussed in the next section.



(a) Server Side (b) Client Side

Figure 26: MBSD Prototype Implementation

Figure 26 shows two graphical user interfaces (GUI) at server side (a) and client side (b) respectively. Server GUI is used for managing the services made available to other mobile devices. For example, one can update a service by changing its name, classification (i.e., the branch on mobile service tree), keywords, etc. A unique service ID is generated automatically based on the service name, classification and the location of the mobile service. As illustrated in Figure 26 (a), one can also provide a link to the corresponding XMI document of the service and the code-base of the objects implementing the mobile services. Client GUI (Figure 26 (b)) illustrates which kind of information is needed for service discovery. A set of APIs has been implemented to automate the service discovery process.

Suppose a user John is implementing a pervasive service that will need to play a movie clip on a mobile device that requests a MPEG4 decoder. When he finds out that he does not have the decoder himself, he decides to use MBSD to discover the decoder because he would need to integrate the decoder into his pervasive service programmatically rather than just plug-and-play and MBSD allows him to do so. After a quick discovery by MBSD he successfully gets this service called *MPEG4Decoder* as detailed in Section 4.1.3 and Section 4.2. Associated with this service include its XMI document and the implementing object classes. He imports the XMI document into his MDA tool (e.g., XMF) and regenerates the models of this decoder service which are the same as that in Figure 16. Then via his MDA tool John can use partial or entire classes from *MPEG4Decoder* to construct his pervasive service. Though John might not be

aware that the service description of the decoder service was actually also generated based on these models, he knows that MBSD has helped him find precisely the service he needs – a service whose inner structure can be seen by him and whose components can be separately used for the benefit of the pervasive service he is composing.

4.6. Performance Evaluation

Konark, as the latest service discovery for wireless networks, is selected as a benchmark to evaluate the performance of MBSD-b. Konark is implemented using Microsoft C# and no detailed implementation source code is available. However, Konark, as presented in [Lee03], utilizes multicast routing protocol for SREQ propagation when evaluation was carried out in [Lee03]. This implies that multicast groups have to be maintained by Konark and the pervasive service itself or its user needs to know which group(s) the SREQ shall be sent to. This is an unrealistic to assume that users have this knowledge. As such this Thesis replaces the multicast routing algorithm in Konark with an existing ad hoc broadcast protocol called AHBP [Peng02]. The reason why AHBP is selected is because AHBP was evaluated in [Williams02] as the algorithm of best performance amongst many other algorithms. The updated Konark is named as Konark-b in this Thesis. Simple flooding is also selected as a comparative reference point.

This section firstly proves numerically that the core of MBSD – MBSD-b is less complex than the above Konark-b computation-wise. Then simulation method is utilized to illustrate the overall performance of MBSD under different network congestion and node mobility situations. Finally the other benefits of MBSD are discussed.

4.6.1. Numerical Analysis

Suppose the number of nodes in the network is N, network area is S, the node transmission radius is R. Given each node's location information, to obtain the link information of the two-hop nodes of the receiving node r, Konark-b has to decide which nodes are located within $2R$ of node r. This will need to use distance formula ($d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$) $N-1$ times. The expectation of the number of one-hop neighbours of node r is: $\frac{\pi R^2}{S} \cdot N - 1$. The expectation of the number of nodes between R and $2R$ is: $\frac{\pi(2R)^2 - \pi R^2}{S} \cdot N - 1 = \frac{3\pi R^2}{S} \cdot N - 1$. As such, the above distance formula

needs to be used $(\frac{\pi R^2}{S} \cdot N-1) \times (\frac{3\pi R^2}{S} \cdot N-1)$ times so as to determine the links between one-hop nodes and two-hop nodes of *r*. After this, Konark-b needs to use the greedy algorithm to find from $(\frac{\pi R^2}{S} \cdot N-1)$ one-hop nodes the relaying nodes in two-hop nodes of *r* to cover all $\frac{3\pi R^2}{S} \cdot N-1$ two-hop nodes.

In the case of MBSD-b, in order to map $N-1$ nodes into grids as shown in Figure 24, $N-1$ times $|x_1 - x_2|$ and $|y_1 - y_2|$ calculation are needed. The detection of the covered nodes (such as node *a* in Figure 25) in the one-hop concern grid (refer to 4.4.5) will use the above distance formula up to $\frac{\pi R^2}{S} \cdot N-1$ times – much less the amount of times used by Konark-b above. Finally MBSD-b needs to find from the maximum 8 one-hop active grids *AG1(n)* the BRGs and BRNs that fully cover up to 16 *AG2(n)*. Because on average each grid contains more one node it holds that

$$|AG1(n)|<|N1(n)| = \frac{\pi R^2}{S} \cdot N-1, \text{ and } |AG2(n)|<|N2(n)| = \frac{3\pi R^2}{S} \cdot N-1.$$

$|x|$ means the size of *x*. As such the number of times the greedy algorithm is invoked by MBSD-b is also less than that by Konark-b. In conclusion, MBSD-b has less computational complexity than Konark-b.

4.6.2. Simulation Environment and Experiment Design

Using the network simulator ns-2 [ns] that is extended by the wireless protocol stack model developed at CMU [CMU], the simulation environment detailed in Table 1 was established. These parameters are needed for any comprehensive simulation and as such are commonly utilized by most research works in MANETs [Williams02].

The performance of each algorithm was assessed in a network of 50 mobile nodes, each having a transmission range of 100m. In Table 1, the derived parameters, which are calculated from the basic parameters, are also listed, intending to demonstrate an overall flavour of the wireless mobile ad hoc network used in the simulation. Node coverage area is the area of the circle whose radius is the node's transmission distance. The maximum path length is the distance from the lower left corner to the upper right corner in the simulation area. The network connectivity is expressed using the number of one-hop neighbours a node is connected to (i.e., the total degrees a node has on average), and it is roughly calculated by dividing the node coverage area by the node density.

In terms of mobility model, *random waypoint* [ns] is used in a rectangular area because of its reputation in effectively simulating the movement of mobile nodes in wireless networks. To simulate the mobility of the network, a constant pause time is utilized as listed in Table 1. Movement scenarios were generated by using movement-connection program *./setdest* provided by ns-2. For each mobility speed, 10 simulation trials were generated and the average was used as the final result.

Table 1: Simulation Environment

Simulator	
Name	ns-2 (ns2.27 all-in-one)
MAC Protocol	IEEE 802.11
Link Bandwidth	2Mbps
Basic Parameters	
Simulation Duration	1300 seconds, starting SREQ generation from 1000th second onwards (lasting for 100 seconds)
Simulation Area (width×height)	300×600 m
Number of Nodes	50
Node Transmission Range ($Range_{trans}$)	100 m
Derived Parameters	
Node Density	1 node per 3,600 m^2 (= number of nodes divided by the total simulation area)
Node Coverage Area	31,416 m^2 ($=\pi \cdot (Range_{trans})^2$)
Maximum Path Length	671 m ($=\sqrt{width^2 + height^2}$)
Max Hops	6.71 hops (=Maximum Path Length divided by $Range_{trans}$)
Network Connectivity	8.73 (=node coverage area divided by node density)
Mobility Parameters	
Mobility Model	Random waypoint
Mobility Speed	1,4,8,12,16
Pause Time	10s
SREQ Parameters	
Message Size	64 bytes payload
Message Origination Rate	5,10,20,30,40,50 messages/sec

In the simulation, the SREQ generation was started from 1000th second onwards rather than from the start of the simulation process. This is to avoid the big vibration in average number of neighbours that occurs usually during the first 600 seconds of the simulation process as pointed out by [CMU]. When the algorithms were evaluated under different network congestion, the average speed of nodes is fixed to 8m/s, whereas SREQ origination rate is set to 5, 10, 20, 30, 40, 50 per seconds. When the algorithms were evaluated under node mobility, the SREQ origination rate is fixed to 30 per second whereas varying the average speed at 1, 4, 8, 12 and 16 m/s.

4.6.3. Simulation Results and Analysis

The simulation was carried out from two aspects: the performance under different network congestion conditions and the performance under different node speeds.

4.6.3.1. Performance under Different Network Congestion

Figure 27 shows the delivery ratio as network congestion increases. It is observed that increasing congestion of network introduces more competition and conflicts for each protocol. The MBSD-b algorithm has better delivery rate than flooding from the SREQ origination rate of 10 messages per minutes onwards. MBSD-b always shows better delivery ratio than that of Konark-b. In Konark-b, due to the fact that some of relay nodes may fail to receive the message because of the congestion, some part of the network can not be covered by broadcasting and as a result the delivery ratio is negatively affected. In MBSD-b where the relay *grids* are utilized instead of relay *nodes* (be noted that a grid covers bigger area that a node does), message could still be retransmitted even if there is only one node receiving the message in the relay grid.

Figure 28 illustrates the end-to-end delay as network congestion increases. Because the nodes ready to send SREQ message will spend a waiting time in MBSD-b, the end-to-end delay of MBSD-b is slightly longer than other protocols in low congestion conditions. However it performs better when the network congestion gets worse.

Obviously, increase in network congestion deteriorates the performance of broadcast protocols in all cases due to the channel competition and data confliction.

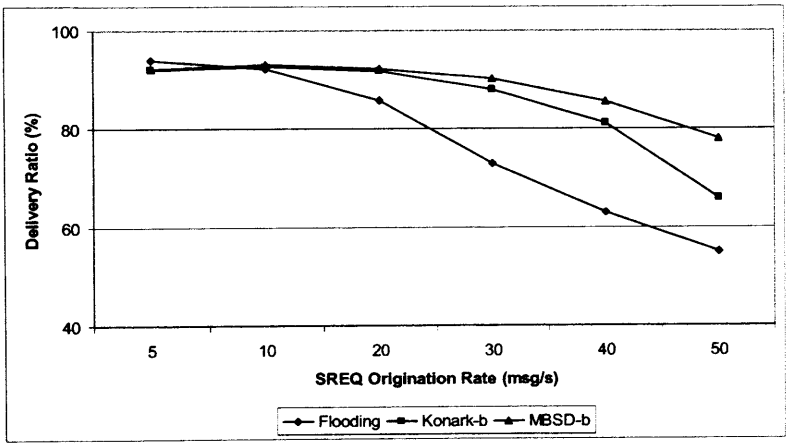


Figure 27: Delivery Ratio versus SREQ Origination Rate

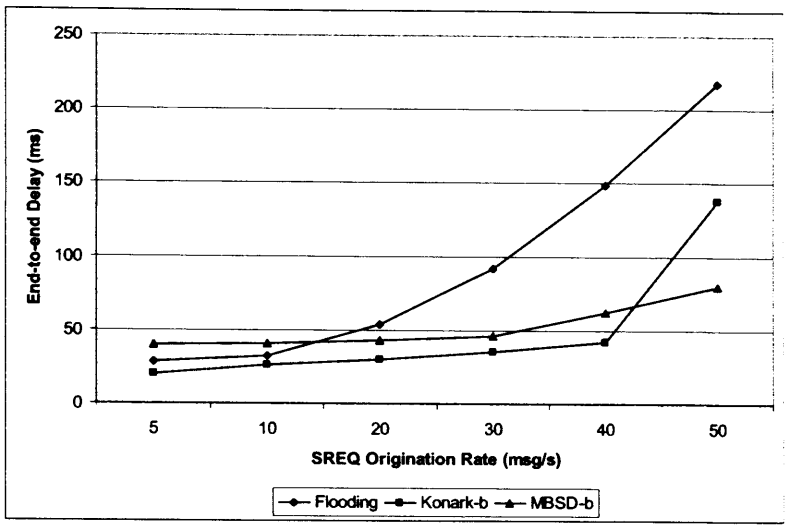


Figure 28: End to End Delay versus SREQ Origination Rate

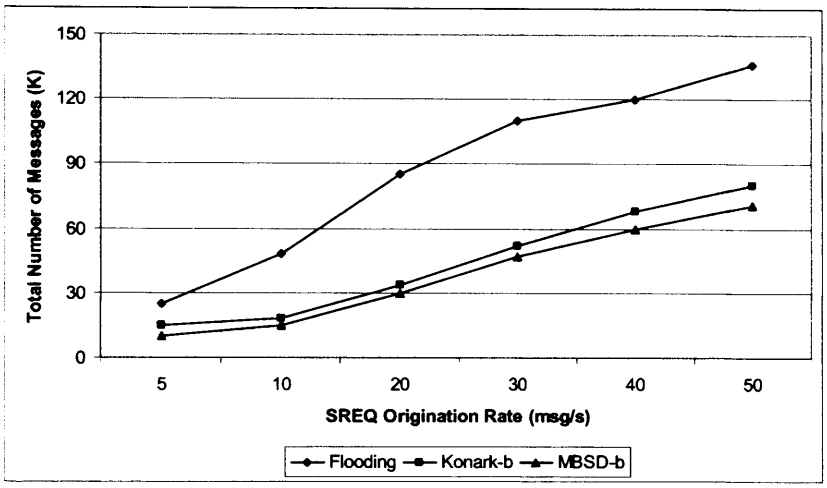


Figure 29: Total Message Number versus SREQ Origination Rate (average speed 8m/s)

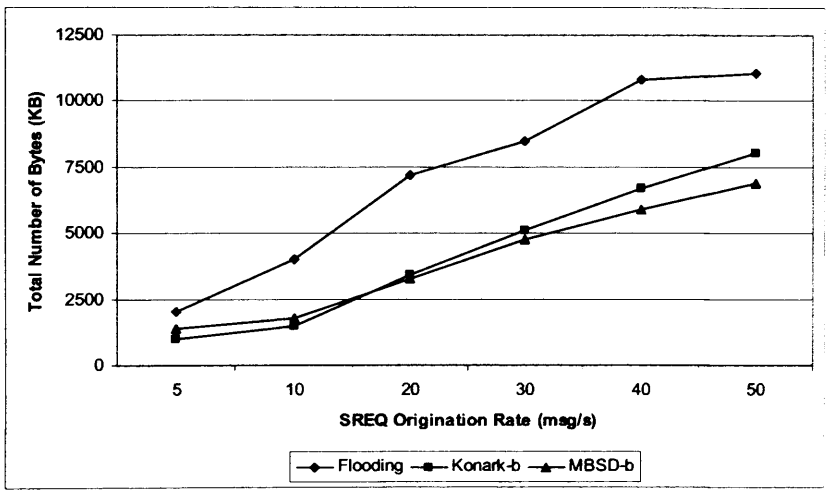


Figure 30: Total Bytes Number versus SREQ Origination Rate (average speed 8m/s)

In terms of overhead, the total message number and byte number were both calculated. Figure 29 shows the total message number as network congestion increases. MBSD-b algorithm sends the least number of SREQs in all situations, which implies a less power consumption than the other algorithms.

Figure 30 illustrates the total byte number as network congestion increases. The byte number calculated here is the number of bytes sent in routing layer. The total byte number of MBSD-b is more than that of Konark-b when the SREQ origination rate is less than 15 messages per minute. This is due to the extra transmission of location updating packets imposed by the location information service employed by MBSD-b. And these location updating packets are much longer than normal hello packets employed in Konark-b. Information of rebroadcast list of neighbours is piggybacked as part of message in Konark-b, causing Konark-b having more byte overhead when network congestion is higher. It is worth mentioning that the location information maintained by the same location updating procedure can also be used for other MANET operations, such as routing. Overall speaking, MBSD-b consumes less network bandwidth.

4.6.3.2. Performance under Different Node Mobility

Figure 31 shows the delivery ratios of different protocols as mobility of network increases under the condition of SREQ origination rate being 30 messages per second. Under this condition, the delivery ratio of MBSD-b is the highest. Figure 32 illustrates the end-to-end delay under the same network conditions, which demonstrates its insensitivity to the average node speed in the network for all tested protocols. The delay of MBSD-b is longer than that of Konark-b.

Figure 33 depicts the total SREQ number as mobility of network increases. The total SREQ number of MBSD-b is the fewest in the three algorithms and is insensitive to the average speed of nodes. Therefore, MBSD-b consumes the least energy and network bandwidth in this situation. Figure 34, showing the total byte number as mobility of network increases, demonstrates a similar performance result as that of message number overhead.

This evaluation also demonstrates that the increase in average speed of nodes in the network does not affect the performance of all service discovery protocols too much.

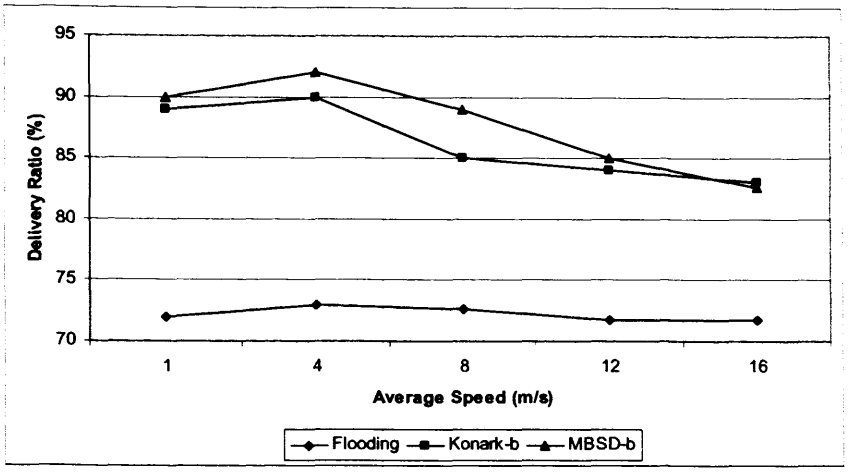


Figure 31: Delivery Ratio versus Average Speed of Nodes (30 SREQ's per second)

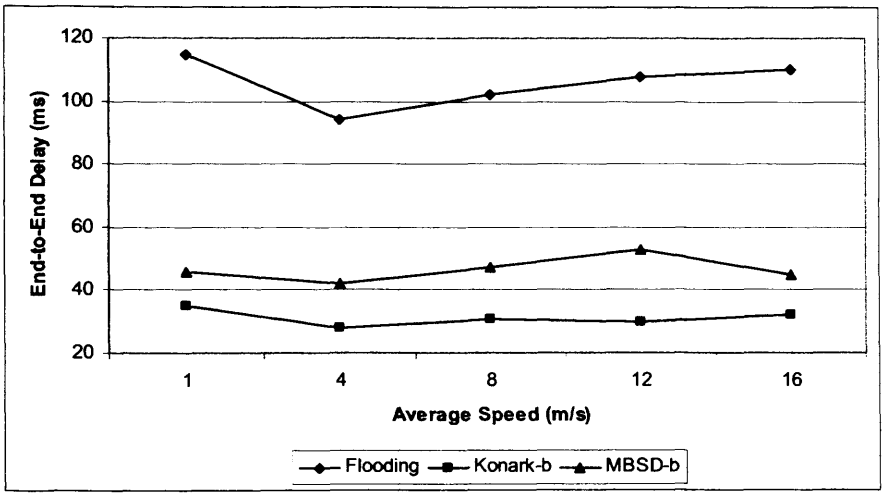


Figure 32: End to End Delay versus Average Speed of Nodes (30 SREQ's/second)

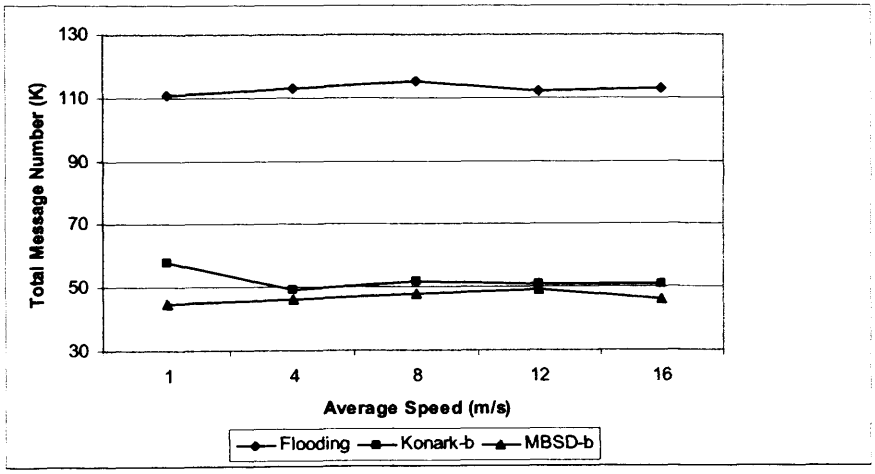


Figure 33: Total Packets Number versus Average Speed of Nodes (30 SREQ's/second)

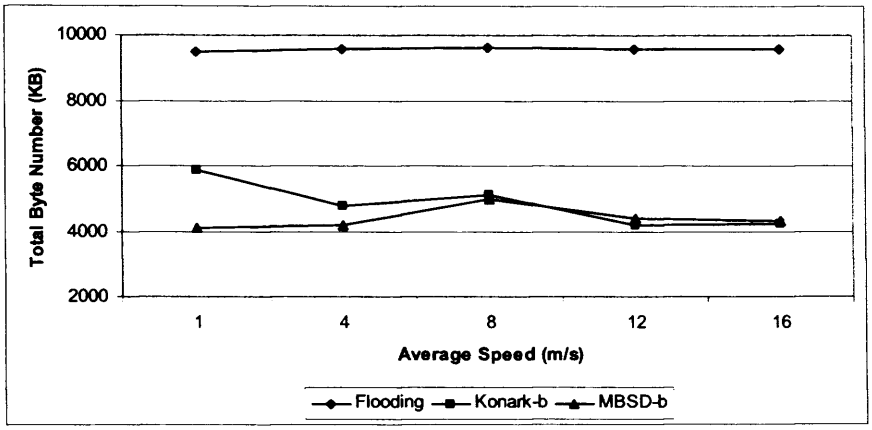


Figure 34: Total Byte Number versus Average Speed of Nodes (30 SREQ's/second)

4.6.3.3. Experimental Conclusions

The evaluation results have shown the efficiency and stability of the proposed service discovery broadcasting protocol MBSD-b. Compared to Konark-b, MBSD-b provides higher delivery ratio in most network congestion conditions. This broadcasting reliability is achieved by significant reduction of the number of re-broadcasting messages, resulting in the reduced network contention and collision. The experiment has also demonstrated that the performance of MBSD-b stays stable in different network conditions. It has shorter end-to-end delay in most cases and consumes less network resources.

Overall speaking, MBSD-b provides improved overall performance (especially in terms of the message delivery rate) over other typical protocols evaluated. The efficiency of MBSD-b appears to be directly related to the following two features of MBSD-b: location-awareness and dynamic grid concept. MBSD-b takes advantage of location information service. The location information service is not only used for broadcasting purpose but also for unicast routing and other applications. The grid concept exploited in this paper is similar to the widely-used cluster concept [Yu05] in that it also provides a means to hierarchically organize nodes. However, it differs in that there isn't a special head and the structuring of grids is dynamic. Most importantly, message re-broadcast is carried out at grid level rather than single node level. The MBSD-b algorithm guarantees that as long as there is one node in a grid that is able to do the re-broadcast the broadcast message will be further propagated, not like other cluster-based broadcasting protocols where only cluster-head is responsible for the message re-transmission [Yu05].

4.6.4. Other Performance Issues

In comparison with other service discovery protocols, including both these for wired networks (such as Jini and UPnP) or these for wireless networks (such as Konark, DEAPspace), MBSD has the following advantages:

- *Expressiveness*: MBSD utilizes XML as its syntax for service description, which, though might be more expressive than other protocols using attribute-value pair to describe services, is no more advantage than other XML-based service discovery protocols such as UPnP and Konark. However, MBSD's service description is not manually written by service providers in a separate phase from service development. It is generated automatically from the model information of a service that is developed based on MDA approach. So when a service has been MDA-developed, the corresponding service description is created at the same time. This integrated approach maximizes the expressiveness of the MBSD's service description language, XSD, at no extra cost. And most importantly this approach guarantees the semantic transition of the service to its description, whereas it is impossible in any other service description approaches in the current service discovery protocols.
- *Impact on software development*: within XSD, there is also links pointing directly to the service's XMI document. As demonstrated in Section 4.2, this XMI document can be imported to an MDA tool to regenerate its visual models and these models can then be used for service re-composition. The composed new services can again be described by XSD and be made available for use by other software developers. Therefore, MBSD service discovery takes into consideration the whole software development procedure, both before service discovery (i.e., the description of services) and after service discovery (i.e., service models regeneration), and makes positive impact on the development of software systems.
- *Mobile service classification:* MBSD also proposes a natural way of classifying and organizing services that speeds up the service discovery response time.

4.7. Summary

This Chapter presents a service discovery algorithm that utilizes a reactive SREQ broadcasting method. The major contributions of MBSD are summarized as follows. Firstly, for the first time MBSD utilizes the model information created during the service component development stage for the benefits of service discovery (including

for both pre-service-discovery service description and post-service-discovery service integration and composition). Secondly, MBSD is empowered by a location-aided grid-based service request broadcasting algorithm that is more efficient than other existing broadcasting algorithms. Thirdly, the feasibility of MBSD is demonstrated using a prototype implementation and its efficiency is presented using larger-scale simulations.

Chapter 5.

Pervasive Service Discovery – Push Method

The MBSD discussed in previous chapter utilizes client-initiated queries to obtain service information. While this pull-based method does not require extra service advertisement it suffers from long service response time and robustness problems and as such not very suitable for mission-critical or real time tasks that are part of pervasive services. Most service discovery protocols for wireless networks such as Konark and DEAPspace adopt service advertisement, i.e., pushing the service information to or close to clients so that when a client needs a service it can use shorter time to locate it. The Thesis has not found any published work of Konark concerning how the service advertisement is carried out. However one of the main contributions DEAPspace brings to the service discovery community is its power-saving algorithm that is highly related to service advertisement [Nidd01, Hermann01]. As a piece of seminal work in service discovery for wireless networks, DEAPspace benefits from a broadcast technique that replaces n broadcasts of one SA (service advertisement) each with one broadcast of n SAs. Considering the benefits service advertisements bring to the performance of pervasive services, the Thesis introduces in this Chapter a push method (i.e., advertising the service information to the service clients). The essence of this push method is service advertisement. Complemented by broadcast-based service discovery on demand, service advertisement in MBSD can be triggered in a relatively longer interval. Chakraborty et al. [Chakraborty06] proposes a distributed service discovery algorithm for pervasive computing environment where service advertisement is also utilized. However the focus of this particular paper is on peer-to-peer caching rather than service advertisement.

5.1. Introduction

After services (or called component services if they are used to compose more complex services) have been created, the next step is how to make them known by other users. This is how service advertisement is used for.

Service advertisement in mobile wireless networks has its specialities that differ it from that in stationary wired networks. Since in mobile wireless networks, a currently

available service on host *i* might become unavailable to a user on host *j* if host *i* moves outside of the radio transmission reach of host *j*. Therefore, service availability and quality in mobile wireless networks also depends on the location of the service concerned. A comprehensive discussion on the QoS of pervasive services is in next chapter. A term *LoS*, short for Location of the Service, is utilized here to specify the location feature of a service, alongside other service semantic features such as the name, functions, interfaces, QoS issues, etc. The existing service advertising mechanisms can spread only service semantic features and leave the location information update to a separate mechanism. However, considering both mechanisms involve broadcasting to certain extent and broadcasting is very expensive in wireless networks, this Thesis proposes to combine the spreading of both types of information together. Namely the service advertisement mechanism in this Thesis advertises not only service semantics as other service advertisement protocols such as UPnP, JINI, Konark, DEAPSpace do but also location information of services. This location information can be used for many other purposes besides service availability, e.g., efficient SREQ broadcasting as discussed in the previous Chapter.

The ultimate goal of any service advertisement mechanism is to reduce the number of *service advertisement message* (*SAM*) or bytes (as overhead) transmitted across the mobile wireless networks while at the same time to keep as high as possible the accuracy level of service information. The simplest SA (Service Advertisement) mechanism is service information flooding where each node broadcasts its own service information on a periodic basis and unconditionally forwards whatever received from neighbouring nodes to its next-hop nodes. A service table, containing service information received by the node, is maintained in every node in the pervasive computing environment. Flooding causes massive transmission of redundant messages and consequently collisions, and usually is not an ideal solution to information updating. The author has not found any dedicated work discussing about service advertisement in mobile wireless networks in the current literature. However, the related researches discussed in mobile ad hoc networks have inspired this Thesis to consider a much better alternative service advertisement mechanism than flooding. Since the availability of a service in wireless mobile networks largely depends on its location to the service user, the literature survey effort to be presented below has been made on the location information update algorithms available in wireless mobile ad hoc networks.

T. Camp *et al.* discussed in [Camp01] a Simple Location Service (SLS). SLS transmits location update packets to its neighbours at a give rate. The difference

between SLS and other location update schemes such as DLS [Basagni98] lies in the type of information exchanged and the distance the information is propagated. Specifically, SLS transmits *table* containing multiple nodes' locations to neighbours and then neighbours will carry out some processing before the next hop transmission; whereas DLS transmits only the sending *node*'s location to its neighbours and then immediately to other nodes via neighbours. DEAPspace's one broadcast of n SAs idea coincides with the table transmission idea in SLS. The service advertisement algorithm proposed in this Thesis, named as MBSD-sa, also utilizes the table transmission idea as that in DEAPspace or SLS but is different in terms of the content of the table entry, updating strategy, etc. Another salient point of MBSD-sa is that MBSD-sa explores *prediction* to predict a node's current location based on its previous location and movement model aiming to significantly reduce the number of SAMs. Note that the computation (used for prediction) consumes far less energy than transmission (used for advertisement). Obviously, prediction cannot replace advertisement completely because location errors as a result of prediction can propagate and accumulate rendering the location information predicted in the second time less accurate than that done in the first time. The proper cooperation and trade-off between prediction and advertisement constitute one of the investigations of this Chapter.

The service discovery system architecture developed in the previous Chapter stays the same for the push method of service discovery. Here in this Chapter the focus is given to the distinguishing aspect of push-based service discovery mechanisms, namely service advertisement algorithm MBSD-sa.

5.2. Preliminaries

5.2.1. MBSD-sa Location Prediction Scheme

The movement of each node in the mobile wireless ad hoc network is assumed to be a uniform velocity linear movement between two service advertisements (SA). Then based on the most recent location (x_1, y_1) of a node at the past time point t_1, the current location (x_2, y_2) of the node at the time point t_2 can be predicted by using the following formulas:

$$x_2 = x_1 + v \cdot (t_2 - t_1) \cdot \cos\theta \tag{5-1}$$

$$y_2 = y_1 + v \cdot (t_2 - t_1) \cdot \sin\theta \tag{5-2}$$

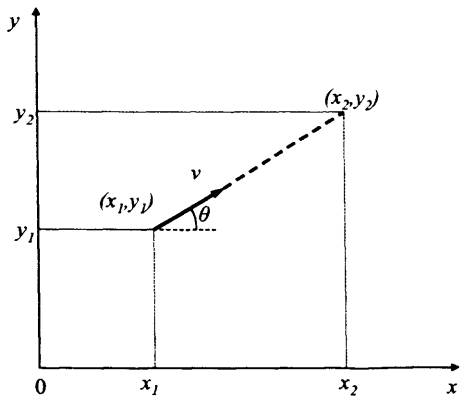


Figure 35: Node Positions and Movement

As depicted in Figure 35, it is assumed that the node moves at a velocity of v with movement direction being angle θ to x axis. Velocity v and angle θ can be calculated via the following formulas:

$$v = \frac{\sqrt{(x-x')^2 + (y-y')^2}}{t-t'} \tag{5-3}$$

$$\theta = \begin{cases} \arccos \dfrac{x-x'}{\sqrt{(x-x')^2 + (y-y')^2}} & y-y' \geq 0; \\ 2\pi - \arccos \dfrac{x-x'}{\sqrt{(x-x')^2 + (y-y')^2}} & y-y' < 0. \end{cases} \tag{5-4}$$

where (x, y) and (x', y') are the locations of the node at the time point t and t' respectively. These two location information is the most recent two pieces of information known to a node. They are obtained either via prediction using formulas (5-1) and (5-2) or received via service advertisement.

5.2.2. Data Structure

The MBSD-sa algorithm is a kind of service information update scheme where each node (if it has some service to be made available) in the network concerned sends out SAM (service advertisement message) to its neighbours (i.e., nodes within its transmission radius) on a periodical basis. SAM is formally defined as follows:

Definition 5.1*: a service advertisement message (SAM) is defined as: SAM::= <SN, SMD, NMD, TS> | <SAM>*, where SN is the name of the service, SMD defines the service meta-data, NMD defines the meta-data of the node the service is on, and TS represents the time stamp indicating when these information is collected.*

SAM is defined in a nested manner. This means that a SAM can contain multiple items of the above single service. *SMD* and *NMD* are further defined as follows.

***Definition 5.2**: service meta-data is defined as: SMD::=<FUNC, P^{in}, P^{out}, Q^S>, where FUNC presents the provided service function, P^{in} and P^{out} specifies the input parameters and output parameters of the service respectively, Q^S defines the quality of the service.*

***Definition 5.3**: node meta-data is defined as: NMD::=<NID, x-coordinate, y-coordinate, SPD, DIR, RESR>, where NID represents the node id, x-coordinate and y-coordinate define the location of the node at TS (refer to SAM definition), SPD and DIR define the moving speed and direction of the node respectively, RESR specifies the resource information of the node.*

Here *RESR* is reserved for future use in cases where resource information is needed. For instance, this field could contain the current power level of the node identified by *NID*, which is useful in power-aware algorithms. This field is set to "nil" in the current MBSD-sa algorithm. It could be easily relaxed to contain any resource-related information of variable length in the future. The QoS issue of a service is mainly employed for service selection and constraint-satisfying service composition which are to be discussed in detail in the next chapter.

SAM is employed to wrap node (as identified by *NID* field)'s service information at the time point indicated by *TS*. In a SAM, the location information includes not only the geometric coordinates (*x-coordinate* and *y-coordinate*) but also the *speed* and the *direction* of the movement. Be note that the speed and the moving direction of any individual node may change as time goes along, so the speed and direction values in SAM reflect only the status at the given time (i.e. *TS*). Both speed and direction are calculated by the sender itself, who is capable of knowing its coordinates at any time point, by using the formula (5-3) and (5-4) respectively. The presence of *speed* and *direction* fields in SAMs enable the receivers of a SAM to predict the current location of this SAM sending node using formula (5-1) and (5-2). It helps the MBSD-sa to decide if the difference of location information is too much.

In order to implement the MBSD-sa algorithm, the following two tables are maintained at each node: *Service Update Table (SUT)* and *Transmitted Service Information Table (TSIT)*. SUT contains all the fields packed in SAMs (refer to Definition 5.1), plus an extra status field called *New* indicating the readiness of this piece of location information to be checked for transmission to other nodes (1 means ready to be checked for possible sending and 0 means the opposite). Upon the receipt of a SAM, the receiving node invokes an procedure called *ServiceTableUpdate*() to update its local SUT.

Procedure *ServiceTableUpdate* (SAM *sam*)
Input:
sam: the newly arrived Service Advertisement Message
Output:
None
Purpose:
Update local service registry upon receipt of a service advertisement message

```
1.  SN-List ← the names of all the services currently available in SUT;
2.  for each item sam(i) = (SN, SMD, NMD, TS) ∈ sam do
3.    if sam(i).SN ∉ SN-List then          // a new service
4.      newService ← add new field to sam(i);
5.      newService.New ← 1;
6.      append newService into SUT;
7.    else                                 // not a new service
8.      existingService ← SUT.getRow(sam(i).SN);
9.      if existingService.TS older than sam(i).TS then
10.       replace existingService in SUT with sam(i);
11.       set the new field of existingService in SUT to 1;
                                    // newer info needs to be propagated
End
```

Every node in the network also has the information of services provided by itself written in its SUT. This information is updated periodically by the node itself. If the difference (e.g., the location information due to the node movement) between the new service information and the previous one is different sufficiently enough, the *New* field of this local node is set as well.

TSIT (Transmitted Service Information Table) contains exactly the same fields as SAMs. TSIT in a node *a* is updated every time a new SAM is broadcasted by node *a*. TSIT on node *a* contains all the service information that has been broadcasted to other nodes by node *a*. In other words, the neighbouring nodes should at least know the information stored in the TSIT. TSIT represents the view of other nodes about the service information of these nodes stored in the TSIT table as this service information is the information received and kept by the other nodes.

As such, two views, as represented by two tables respectively, are maintained in each node *a*: the local node *a*'s view about the other nodes' service information as represented by SUT (it is called *myView* here) and the other nodes' view about the location of the same set of nodes as represented by TSIT (here it is called *othersView*). These two views are used by the service information update algorithm MBSD-sa to check if there is a service information difference between these two views. If the difference is significantly enough, a SAM is generated by MBSD-sa.

5.3. MBSD-sa Algorithm Operation

The service advertisement algorithm MBSD-sa involves the following two complementary types of service advertisements or updates.

Type-1 is to carry out normal service advertisement and update. The Type-1 advertisements are generated by MBSD-sa on a periodical basis with a variable interval $Interval_1$. Obviously, it is hoped that 1) the bigger the node's transmission range ($Range_{trans}$) is the longer the $Interval_1$ is; and 2) the faster the node is moving the shorter this interval is. As such $Interval_1$ can be calculated using the following formula:

$$Interval_1 = \frac{Range_{trans}}{\alpha \times v_{avg}} \tag{5-5}$$

where v_{avg} is the average speed of the node and α is a scaling factor. In Type-1, all the entries in a node's SUT whose *New* field is set to 1 are packed into SAMs and broadcasted but at a relatively longer interval $Interval_1$. This longer interval, while reducing the number of SAM packets, may lead to a situation where large changes in service information might not be able to be propagated quickly enough. To avoid this situation, MBSD-sa employs another type of service advertisement – *Type*-2.

Type-2 service advertisement is triggered when there is a considerable change in the service information (e.g., node coordinates or price adjustment) but the time for triggering Type-1 advertisement has not arrived yet. Type-2 service advertisement in node *a* periodically predicts, by using the prediction formulas (5-1) and (5-2) given above, the current information of each service in the SUT table by calculating separately its entries in both the SUT table and the TSIT table. If the difference of the two calculations is greater than a given threshold, which means a quite different understanding of a certain service's information between node *a* and the other nodes, then node *a* will broadcast this service information (as stored in node *a*'s SUT table) to its neighbours immediately.

Type-2 service is triggered periodically at an interval ($Interval_2$) which is shorter than $Interval_1$ so as to propagate significant service information changes in a quicker manner than the normal service advertisements (i.e. Type-1 advertisements). Note that if there is no significant change in any service then no SAM is sent out during Type-2 procedure. In MBSD-sa, $Interval_2 = Interval_1/3$. The introduction of Type-2 updates contributes to the high performance of MBSD-sa while keeping a fairly low average service error, as to be shown in the simulation results later in this Chapter.

In MBSD-sa, all service advertisements are transmitted using SAMs of variable length. The more the number of pieces of node information to be broadcasted is the longer the SAM length is. The MBSD-sa service advertisement algorithm runs on each node. The following pseudo code illustrates the operation of the MBSD-sa algorithm.

MBSD-sa algorithm
Input :
sut: service update table SUT
tsit: transmitted service information table TSIT
Output:
None
Purpose:
Carry out service advertisement using both Type-1 and Type-2 advertising

1. specify the value of $Interval_1$, $Interval_2$; //optional: can be hard-coded
2. **if** $Interval_1$ timeouts **then** //Type-1
3. create an instance of SAM called *sam1;*
4. **for** (each entry *entry_sut* in *sut* whose *new* field is 1) **do**
5. append the content of *entry_sut* into *sam1*;
6. update the peer entry in *tsit* using the content of *entry_sut*;
7. reset the *new* field of *entry_sut*;
8. **if** *sam1* ≠ **nil then**
9. broadcast(*sam1*);

10. **if** $Interval_2$ timeouts **then** // Type-2
11. create an instance of SAM called *sam2;*
12. **for** each entry *entry_sut* in *sut* **do**
13. *entry_tsit* ← the entry of the same service as *entry_sut* in *tsit*;
14. $x\_predicted\_sut \leftarrow entry\_sut.x + v \cdot (t_2 - t_1) \cdot \cos\theta$; // formula 5-1
15. $y\_predicted\_sut \leftarrow entry\_sut.y + v \cdot (t_2 - t_1) \cdot \sin\theta$; // formula 5-2
16. $x\_predicted\_tsit \leftarrow entry\_tsit.x + v \cdot (t_2 - t_1) \cdot \cos\theta$;
17. $y\_predicted\_tsit \leftarrow entry\_tsit.y + v \cdot (t_2 - t_1) \cdot \sin\theta$;
18. **Boolean** *others* ← **FALSE**;
19. **if** other service information changes **then**
20. *others* ← **TRUE**;
21. **if** ($|x\_predicted\_sut - x\_predicted\_tsit| >$ X_CHANGE_THRESHOLD) **or** ($|y\_predicted\_sut - y\_predicted\_tsit|$ >Y_CHANGE_THRESHOLD **or** *others*) **then**
22. append the content of *entry_sut* into *sam2*;
23. update *entry_tsit* using *entry_sut*;
24. reset the *new* field of *entry_sut*; } }
25. **if** *sam2* ≠ **nil then**
26. broadcast(*sam2*);

// $|x|$ means the non-negative value of x.

In terms of service information change, the above MBSD-sa algorithm mainly illustrates how the node location changes are calculated using the prediction method

presented in 5.2.1. This is because node location change is the most frequent one amongst all the service information due to the mobile nature of the devices on which pervasive services run. Furthermore the change of location may easily render a service unreachable. The guidelines for other service information changes are as follows: 1) changes in items *SPD* and *DIR* of *NMD::=<NID, x-coordinate, y-coordinate, SPD, DIR, RESR>* (refer to Definition 5.3) can follow the same methodology as *coordinate*'s, namely setting up a corresponding threshold; 2) *RESR* (node resource) of *NMD* is currently reserved for future use but the threshold method can be equally applied here; 3) any change in *SMD* (service meta-data – refer to Definition 5.1) will immediately set the Boolean variable *others* in the above algorithm true because this means a semantic change of the service (e.g., the change of service input) that has to be acted on immediately.

5.4. Numerical Analysis of MBSD-sa Performance

The performance of the algorithms is numerically analyzed in terms of overhead and delay. Overhead is mainly concerned about the number of SAM messages. Delay is measured via the average SAM transmission delay, namely the average time used to propagate a SAM to all the other nodes in the network. In this sub-section delay is numerically calculated whereas overhead is evaluated using simulation in the next sub-section.

Recap how MBSD-sa works as follows:

1) At time 0, T, 2T, 3T, ..., MBSD-sa on node *a*, advertises all the items in its SUT table whose *New* field is 1, i.e., carrying out Type-1 advertisement.
2) And at time $\frac{T}{3}$ and $\frac{2T}{3}$ and for each time duration *T*, MBSD-sa checks if there is any Type-2 advertisement. If yes then a Type-2 SAM is sent out (refer to the MBSD-sa algorithm).

The performance of Type-1 advertisement is firstly analyzed, based on which the whole MBSD-sa, including both Type-1 and Type-2 advertisements, is analyzed.

5.4.1. Type-1 Service Advertisement

Suppose the interval for each node to send SAM is *T* and the starting time for each node to send SAM is randomly distributed in $[0,T]$. The number of nodes in the network is *N*. Since the starting time for each node to send SAMs is random, the chance for each node to advertise its service information is equal. As such the expectation of the number of nodes that receive SAMs before they send their own SAMs is *N*/2. It is true to

MBSD-sa that all the newer service information (i.e., the information whose *New* field is set to 1) can be advertised in just one SAM. When a SAM arrives at node *i*, the expectation of the SAM delay before being further propagated is:

$$\int_0^T \frac{1}{T} t dt = \frac{T}{2} \tag{5-6}$$

If K hops are needed for a piece of service information to be propagated to the whole network, then the delay of this piece of service information when it just goes through k-th hop is: $\frac{T}{2} \cdot k$. Suppose the radio transmission range of each node is the same (R) and the nodes are evenly distributed in the network area S, then the number of nodes encountered by a SAM in each hop k, denoted as n_k, can be calculated as follows:

$$n_1 = N \cdot \frac{\pi R^2}{S}$$

$$n_2 = N \cdot \frac{\pi (2R)^2 - \pi R^2}{S}$$

$$n_3 = N \cdot \frac{\pi (3R)^2 - \pi (2R)^2}{S}$$

.....

$$n_k = N \cdot \frac{\pi (kR)^2 - \pi ((k-1)R)^2}{S} \tag{5-7}$$

Since during $[0,T]$ each node sends one SAM to its neighbouring nodes, so if the number of nodes in the network is N then the number of SAMs is also N.

Then the average SAM delay across the network is:

$$\frac{1}{N} \sum_{k=1}^{K} \frac{T}{2} \cdot k \cdot n_k \tag{5-8}$$

5.4.2. Type-2 Service Advertisement

Suppose the speed threshold for Type-2 is v_{td} and the speed change is Δv, the direction (or angle) threshold is θ_{td} and the direction change is $\Delta\theta$, then the following probabilities hold:

$$P(\Delta v \geq v_{td}) = \frac{(V - v_{td})^2}{2V^2} \tag{5-9}$$

$$P(\Delta\theta \geq \theta_{td}) = \frac{(2\pi - \theta_{td})^2}{2(2\pi)^2} \tag{5-10}$$

Denote the event "Type-2 advertisement occurs" as A then the probability of A, denoted as $P(A)$, is:

$$P(A) = P(\Delta v \geq v_{td}) \cup P(\Delta\theta \geq \theta_{td}) = P(\Delta v \geq v_{td}) + P(\Delta\theta \geq \theta_{td}) - P(\Delta v \geq v_{td}) \cdot P(\Delta\theta \geq \theta_{td}) \tag{5-11}$$

There are three cases when a Type-2 checking is carried out at time $\frac{T}{3}$ and $\frac{2T}{3}$ respectively:

1) Case 1: A does not occur, namely there is no Type-2 advertisement. This case is equivalent to a Type-1 advertisement only during [0, T] and as such the SAM transmission delay for this case is the same as that of Type-1, namely:

$$\frac{1}{N}\sum_{k=1}^{K}\frac{T}{2}\cdot k \cdot n_k \tag{5-12}$$

2) Case 2: A occurs at both $\frac{T}{3}$ and $\frac{2T}{3}$. Following the way of case 1 the SAM transmission delay for this case can be calculated as follow:

$$\frac{1}{N}\sum_{k=1}^{K}\frac{T}{6}\cdot k \cdot n_k \tag{5-13}$$

3) Case 3: A occurs at either $\frac{T}{3}$ or $\frac{2T}{3}$. Then the SAM transmission delay for this case is the mean of Case-1 delay and Case-2 delay, namely:

$$\frac{1}{N}\sum_{k=1}^{K}\frac{T}{3}\cdot k \cdot n_k \tag{5-14}$$

Then the average SAM transmission delay is:

$$\begin{aligned}&(1-P(A))\cdot(1-P(A))\cdot\left(\frac{1}{N}\sum_{k=1}^{K}\frac{T}{2}\cdot k\cdot n_k\right)+P(A)\cdot P(A)\cdot\left(\frac{1}{N}\sum_{k=1}^{K}\frac{T}{6}\cdot k\cdot n_k\right)\\&+2\cdot(1-P(A))\cdot P(A)\cdot\left(\frac{1}{N}\sum_{k=1}^{K}\frac{T}{3}\cdot k\cdot n_k\right)\\&=(3-2P(A))\cdot\left(\frac{1}{N}\sum_{k=1}^{K}\frac{T}{6}\cdot k\cdot n_k\right)\end{aligned} \tag{5-15}$$

5.5. Performance Evaluation

5.5.1. Experimental Design

As far as service discovery protocols for wireless mobile ad hoc networks are concerned, DEAPspace is a representative one whose service advertisement mechanism has been publicized [Nidd01]. DEAPspace transmits service advertisement packets to its neighbours at a given rate. The salient point of DEAPspace lies in the novelty that DEAPspace transmits n SAs in one broadcast rather than 1 SA in n broadcasts. However DEAPspace does not concern the service availability affected by the node

mobility. For comparison purpose, this Thesis implemented an updated DEAPspace called DEAPspace+ that obtains the location information of the service from positioning devices such as GPS and transmits the location information together with service information. Service location is collected at the time of advertising service information. DEAPspace+ does not utilize prediction to predict a node's current location as MBSD-sa does. Flooding is also selected as a benchmark (mainly serving as a worst-case reference point) to be compared against MBSD-sa. The specific value for each parameter used in each algorithm is discussed below.

In simple flooding, every node generates a SAM which is flooded into the network after a constant interval. In this Thesis's simulation, this interval was set to 13 seconds. And the SAM used by flooding includes the node's coordinates, the corresponding sampling time, service id, node id and some service syntax and semantic information of fixed length.

In DEAPspace+, the service advertisement interval was calculated using the following formula:

$$Interval - DEAPspace = \begin{cases} \dfrac{Range_{trans}}{\alpha \times v_{avg}}; & Interval \le 13\,\text{sec} \\ 13\,\text{sec}; & Interval > 13\,\text{sec} \end{cases}$$

Formula (5-5) was used here where α was set to 4 and $Range_{trans}$ was set to 100m. Nodes broadcasted SAMs in *Interval-DEAPspace* interval. SAMs in DEAPspace+ includes the service location information (precisely the x and y coordinates), their corresponding sampling time and service information.

As to MBSD-sa, $Interval_1$ was calculated using formula (5-5) where $Range_{trans}$ stayed the same (i.e., 100m) as DEAPspace+ but α was set to 8/3. That is to say, the Type-1 update interval is longer than the interval employed by DEAPspace+. As mentioned in MBSD-sa algorithm description, $Interval_2 = Interval_1/3$. The other threshold values used in MBSD-sa were as follows: `X_CHANGE_THRESHOLD` = `Y_CHANGE_THRESHOLD` = 6m. For simplicity the evaluation focus regarding service information accuracy is on the location information only. However other service information such as price or functions can be equally evaluated in the same manner and with a reduced complexity. The changes of other service information are only determined by the service providers themselves in a static manner and are usually not relevant to the dynamic node mobility. In order to test the average location error, the real locations of nodes (as generated by ns-2) and the location information predicted

were recorded every 2 seconds. The differences between these two sets of values indicate the degrees of location accuracy.

The simulation environment is similar to that used for evaluating MBSD-b in the previous Chapter. The performance of each algorithm was tested using network simulator ns-2 (extended by the wireless protocol stack model developed at CMU [CMU]) in a network of 50 mobile nodes each emulating a mobile user. Each node has a transmission range of 100m and uses 2Mbps IEEE802.11 as the MAC (Medium Access Control) layer. The simulation area is 300m*600m. Random waypoint is utilized to model node mobility, where a constant pause time of 10s is employed. Movement scenarios were generated by using movement-connection program *./setdest* provided by ns-2. For each mobility speed, 10 simulation trials were generated and the average was used as the final result. In the simulation test, SAM generation was started from 1000th second onwards to avoid the big vibration in average number of neighbours occurring at beginning of the simulation process [CMU]. To avoid neighbour nodes sending SAMs at the same time (and thus causing data collision), a random jitter was added before every transmission.

5.5.2. Simulation Results and Analysis

The above 3 service advertisement algorithms were evaluated from two aspects: 1) overhead in terms of both the number of SAM *messages* broadcasted and the number of *byte* broadcasted for service advertisement purpose during the simulation period; 2) the average service information errors (using location errors as an example) caused by the node mobility during the simulation period.

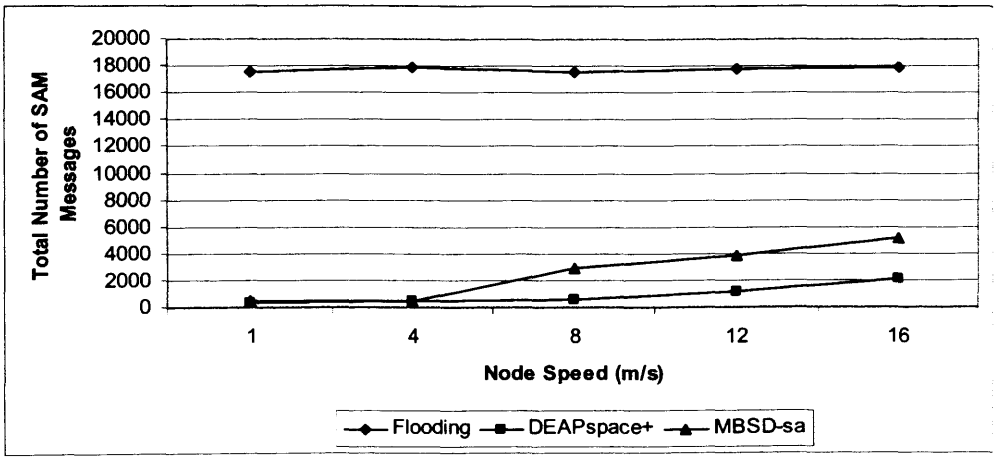


Figure 36: SAM Message Overhead vs. Node Speed

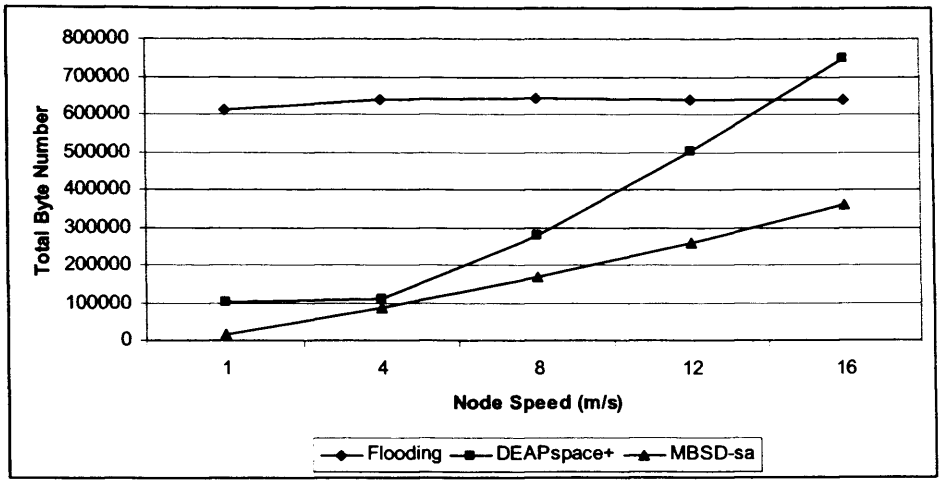


Figure 37: SAM Byte Overhead vs. Node Speed

Figure 36 illustrated the SREQ message overhead versus the node speed for each algorithm. Theoretically, in flooding the number of SAMs broadcasted during every update interval equals to N^2 (N is the number of nodes in the network). As shown in Figure 36, flooding uses the biggest amount of packets. DEAPspace+ and MBSD-sa consume fewer packets due to their one-hop broadcast feature. Given the same location information update interval, MBSD-sa uses more SAM packets than DEAPspace+ because it introduces an extra type of update.

Figure 37 describes the byte overhead versus node speed. The byte overhead represents the network bandwidth requirement. The simulation result shown here counted only these bytes transmitted by the routing layer. Actually, an increase in the transmitted packet number at the network layer means a corresponding increase in number of bytes transmitted at the link layer. In flooding, a SAM contains only service information and timestamp of *one* single node (i.e., the sending node) so the length of SAM is constant. As a result, the byte number keeps scale with the transmitted packet number. MBSD-sa consumes less byte than DEAPspace+. The byte number overhead for DEAPspace+ and MBSD-sa increases when mobile node moves more quickly; this is because these protocols transmit SAMs more frequently when the average node speed becomes faster. MBSD-sa consumes fewer bytes because it transmits far less Type-1 messages and has shorter SAM messages.

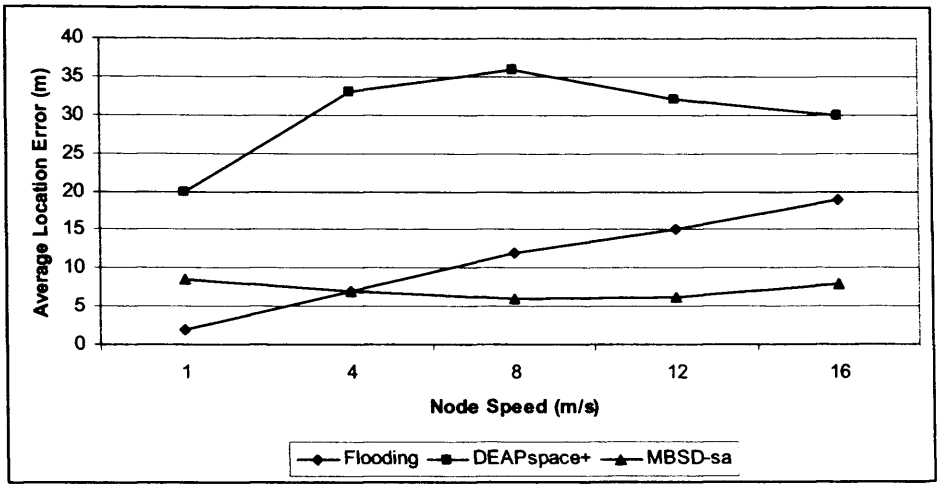


Figure 38: Average Location Error vs. Node Speed

Figure 38 shows the average location error (in meters) of the protocols versus the node's speed. It is observed that DEAPspace+ introduces the largest location errors whereas in most cases (apart from the very beginning) MBSD-sa has the smallest average location errors. The average location error increases linearly in the flooding algorithm because the nodes generate updates at a constant interval. The accuracy of node's location information in DEAPspace+ is worse than that of the flooding; this is because most of the SAMs in DEAPspace+ only transmit to their neighbours and not being able to propagate to the whole network. As the speed increases, the update interval becomes shorter in both DEAPspace+ and MBSD-sa, which means more SAMs being transmitted and consequently increasing location accuracy. Thanks to the prediction mechanism, MBSD-sa shows better location accuracy in comparison with the other protocols. When the nodes move in low speed (lower than 5 m/s) the advertisement interval in MBSD-sa is larger (refer to Formula (5-5)). As a result the less service updates lead to a bigger average location error –even bigger than that of flooding. Figure 38 demonstrated that MBSD-sa can provide constantly good location accuracy when the nodes are moving at a higher speed (regardless how relatively high the speed is).

5.6. Summary

This paper presents a service advertisement algorithm MBSD-sa that complements the service discovery solution in the last Chapter. The major contributions of MBSD-sa are summarized as follows. Firstly, MBSD-sa advocates the importance of service location to the service availability and integrates the service location information together with the service semantic information into service information for

advertisement. Secondly, MBSD-sa utilizes prediction to estimate the service location so as to reduce the number of service advertisement messages. Thirdly, two complementary types of service advertisement mechanisms (Type-1 and Type-2) are employed by MBSD-sa to strike the balance between the SAM overhead and the accuracy of service information. Fourthly, a detailed numerical analysis of MBSD-sa performance is presented. Finally, the efficiency of MBSD-sa is demonstrated using large-scale simulations.

Chapter 6. QoS-aware Service Selection for Pervasive Service Composition

During the execution of a created pervasive service, upon the change of context information, this pervasive service might sometimes invoke a task whose implementation is not provided at the service creation stage. In this case the component services implementing this task will have to be found, which is conducted by service discovery discussed in Chapter 4 and 5. If there is more than one service providing the similar or identical function to the requested task then service selection is needed. The selected service is to be composed into the existing pervasive service. Here, the purpose of service selection is for service composition.

Service composition mechanisms enable *individual services* to be composed together to generate a more powerful service (*composite service*). Here is this Thesis, pervasive services are composite services. Service composition greatly improves service reuse [Papazoglou03]. Automatic service composition has been researched for web services in terms of semantic service discovery, validation of composite service logic and service execution monitoring, etc [Dustdar05]. In particular, *QoS-aware service composition* is receiving increasing attention. In general, it refers to a problem as to how to select a set of appropriate services to instantiate composition logic while satisfying user's QoS requirements. Some researches have been conducted in the web services field. However, not too much effort has been given to the service selection and composition for pervasive services that run over wireless mobile networks. This Chapter of the Thesis aims to help facilitate research work in this aspect.

This Chapter firstly investigates a comprehensive model for describing the quality of pervasive services. This model integrates both network conditions and user perception. Secondly a QoS-aware service selection problem is described and its algorithm (including its variations) is presented. Then system architecture for dynamic service composition is described. Finally some evaluation results are presented and discussed.

6.1. Modelling Quality of Pervasive Services

The purpose of quality of pervasive service (QoPS) is two-fold. On one hand QoS can be used to differentiate multiple services with overlapping or identical functionalities; on the other hand and more importantly, QoS is needed for service composition process to compose a service that satisfies the QoS requirements from users or for certain mission-critical applications. According to O'Sullivan et. al [O'Sullivan02] and Zeng et. al [Zeng04], service quality is a broad concept that encompasses a number of non-functional properties such as availability, reliability, reputation and price. These properties apply to both individual services and composite services. The latter are composed of small services. These small services are also called *component services*. In order to compose a QoS-aware pervasive service, a QoPS model that quantifies the important features of pervasive services needs to be investigated.

O'Sullivan et. al [O'Sullivan02] suggest some non-functional properties for electronic services. These include the methods of charging and payment, temporal and spatial availability, service quality, security, trust and the rights attached to a service. Largely inspired by O'Sullivan et. al's work, Zeng et. al [Zeng04] provide a more precise QoS model that is specific to web services (a kind of electronic services). Zeng's model is based on a set of quality criteria (also non-functional properties) that are applicable to all web services. Five generic quality criteria are considered in this model: execution price, execution duration, reputation, successful execution rate and availability. For each criterion, both definition and rules to compute its value for a given service are presented. However, these researches do not consider the network aspects of services and assume services run over the Internet. Gu et al. [Gu06, Gu04] take into account the impact of network topology on the service selection and composition, but the network environment considered by Gu's work is peer-to-peer wired networks serving as an overlay network on top of the Internet.

The QoS model for pervasive services proposed in this Thesis differs from the existing related researches in the following aspects. Firstly, the considered network environment is wireless mobile networks whose unique features (such as radio transmission, mobility etc) pose new challenges to service selection. Secondly, the *network* features that affect the quality of services (such as availability, reliability) are also considered and modelled, together with these perceived directly by end users. Finally, the relationship between network-oriented QoS criteria and user-perceived QoS criteria are discussed and formulated where applicable. The reason why network parameters need to be considered when describing QoPS is because the locations,

mobility and error-prone transmission channel of wireless nodes all make significant impact on the availability and reliability of a service running in wireless mobile networks. Whereas in the wireline networks such as the Internet and stationary peer-to-peer networks, far more reliable data transmission can be guaranteed by copper or fibre, plus nodes are stationary in most cases. In wireless mobile networks, the properties of the node on which the service locates are too critical to be neglected when selecting a component service.

Since the QoS of composite services is determined by the QoS of its underlying component services, this section firstly investigates the QoS criteria for elementary services and then these of composite services. For each criterion, both definition and rules/formula to calculate its value are presented.

6.1.1. Network Quality Criteria for Elementary Services

The quality criteria utilized by POETRY QoPS are divided into two groups: one from network's view and the other from user's view, denoted as q^n and q^u respectively. In terms of network quality criteria the following are considered to be making big impact on the upper layer services: node availability, delay and network reliability.

6.1.1.1. Node Availability

In wireless mobile networks such as ad hoc networks, the availability of a node to its neighbour nodes is highly related to the node's mobility – here it is assumed that batter power is not a concern and as such a node will not out of reach because of battery exhaustion. If node i moves outside the transmission range of its neighbouring node j then node i is unreachable by node j and as a result the services on node i become unavailable to node j either. Node availability, to some extent, also expresses network availability because the connection between any two neighbouring nodes in a route from a service provider to a service requester becomes unavailable then the whole route also becomes unavailable. Here node mobility is utilized to calculate the node availability.

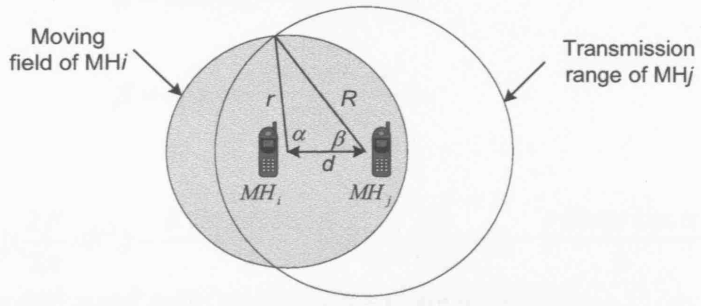


Figure 39: Network Availability Calculation

Consider, as illustrated in Figure 39, two mobile hosts MH*i* and MH*j* of the same transmission range *R*. Each node moves randomly and it is assumed that the moving field is a circle with a radius of *r*. *d* represents the distance between MH*i* and MH*j*. These three parameters are to be used for calculating the node availability. The transmission range of a node *R* is known (e.g., pre-defined or changing according to certain algorithm). Suppose the location (i.e., coordinates) of each mobile host is known (e.g., via GPS - global positioning system), then distance *d* can be calculated using the Euclidean distance formula, i.e., $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ where (x_i, y_i) and (x_j, y_j) are the coordinates of MH*i* and MH*j* respectively. Finally let's discuss how to calculate *r*.

The moving radius of a mobile host (*r*) is its speed (*s*) multiplied by the average service time (*t*). Here *t* can be statistically calculated as the average value of last *n* servings of this component service, namely, $t = \sum_{i=1}^{n} t_i / n$. The speed of a mobile host *s* can be calculated based on its moving distance during a period from t_1 to t_2 [Ko98], namely: $s = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{t_2 - t_1}$. Then $r = s \times t$.

Knowing the values of these three parameters *R*, *r*, and *d*, the probability of MH*i* staying inside the transmission range of MH*j* (denoted as P_i^{IN}) can be calculated by:

$$P_i^{IN} = \frac{A_i^{IN}}{A_i^T} \tag{6-1}$$

Namely, P_i^{IN} equals to the area of the MH*i*'s moving field inside the transmission range of MH*j* (denoted as A_i^{IN}) divided by the overall area of the MH*i*'s moving field (A_i^T).

As shown in Figure 39,

$$\alpha = \arccos(\frac{r^2 + d^2 - R^2}{2r \times d}) \tag{6-2}$$

$$\beta = \arccos(\frac{R^2 + d^2 - r^2}{2R \times d}) \tag{6-3}$$

Then,

$$\begin{aligned} A_i^{IN} &= [(\frac{2\beta}{2\pi}\pi R^2) - \frac{R\sin\beta R\cos\beta}{2}2] + [(\frac{2\alpha}{2\pi}\pi r^2) - \frac{r\sin\alpha r\cos\alpha}{2}2] \\ &= \beta R^2 + \alpha r^2 - (R^2\sin\beta\cos\beta + r^2\sin\alpha\cos\alpha) \end{aligned} \tag{6-4}$$

There is also,

$$A_i^T = \pi r^2 = \pi \times (s \times t)^2 \tag{6-5}$$

Therefore, there is:

$$P_i^{IN} = \frac{A_i^{IN}}{\pi s^2 t^2} \tag{6-6}$$

Therefore, the probability of MH*i* staying inside the transmission range of MH*j* (P_i^{IN}) can be calculated. Suppose service *s* running on MH*i* is a candidate service for a task running on MH*j*, and the node availability with regard to service *s* is denoted as $q_{av}^n(s)$, then finally there is:

$$q_{av}^n(s) = P_i^{IN} = \frac{A_i^{IN}}{\pi s^2 t^2} = \frac{\beta R^2 + \alpha r^2 - (R^2 \sin\beta\cos\beta + r^2 \sin\alpha\cos\alpha)}{\pi s^2 t^2} \tag{6-7}$$

6.1.1.2. Network Delay

The delay of a packet in a network is the time it takes the packet to reach the destination after it leaves the source. Similar to that in the fixed network, the delay in wireless mobile networks is the sum of time spent at each relay on the route. However, in wireless mobile networks, delay depends on the velocity *v* of each relay. [Gamal04] gives the average packet delay for a network with *n* nodes as

$$D(n) = O(\sqrt{n} / v(n)) \tag{6-8}$$

where *v(n)* denotes the way node velocity scales with *n*. This Thesis directly uses this result – for details with regard to this formula please refer to [Gamal04].

6.1.1.3. Network Reliability

In wireless networks, network reliability is largely affected by the quality of wireless channels. In this light, the network reliability $q_{re}^n(s)$ of service *s* is measured in this Thesis by the packet loss rate of the wireless link via which the service host is connected to the outside world. In each receiver node there is a queue accepting arriving packets. Due to the finite-length queuing, arriving packets will be dropped when the queue is full. Let P_d denote the packet dropping probability on the queue (caused by either overflow or blocking) and P_e denote the packet error probability of wireless channel. A packet from a service requester is correctly received by a service provider (or vice versa) only if the following two conditions hold:

C1: the packet is correctly received through the wireless channel (with probability $1 - P_e$), and

C2: the packet is not dropped from the queue (with probability $1-P_d$).

Thus, the probability of a packet received correctly is $(1-P_e)(1-P_d)$, and the packet loss rate η can be expressed as:

$$\eta = 1-(1-P_e)(1-P_d) \tag{6-9}$$

Then as far as the service provider node (i.e., the node on which service s is located) is concerned, the network reliability $q_{re}^{n}(s)$ of service s is:

$$q_{re}^{n}(s) = \eta = 1-(1-P_e)(1-P_d) \tag{6-10}$$

Let η^i denote the packet loss rate of node i. If considering also the intermediate wireless nodes on the multi-hop route from service requester r to service provider p then $q_{re}^{n}(s)$ is computed by:

$$q_{re}^{n}(s) = 1-\prod_{i=r}^{p}(1-\eta^i) = 1-\prod_{i=r}^{p}(1-P_e^i)(1-P_d^i) \tag{6-11}$$

where P_e^i and P_d^i denote the packet error probability of the outgoing channel of node i and the packet dropping probability of the queue on node i respectively.

Note that the method for computing P_e and P_d is not unique. It varies depending on channel characteristics, medium access control and also researcher's preference. For example, Liu et al [Liu05] present a Markov-Chain-based means of calculating P_e and P_d for time-slotted wireless channel. Detailed discussion of P_e and P_d is out of the scope of this Thesis. The concept of network reliability proposed here is independent of these computation methods.

6.1.2. User-Perceived Quality Criteria for Elementary Services

From the user's perspective, the following four generic quality criteria are considered: *availability*, *execution delay*, *price*, and *reliability*.

6.1.2.1. Availability

The availability $q_{av}^{u}(s)$ of a service s is the probability that service s is usable. The factors that influence the accessibility of service s come from two aspects. Firstly, the service s itself is still functioning well on its host *h(s)* but the host itself is inaccessible either because the host moves out of the range of the service requester or because the interference is too high to maintain a decent data transmission. Secondly, the service itself is broken or inaccessible e.g., due to too many accesses (as such not enough processing power or bandwidth capacity) or sudden hard disk failure. The probabilities

of the above two events are denoted as q_{av}^{n} and q_{av}^{s} respectively. Then the integrated service availability observed by users can be expressed as:

$$q_{av}^{u} = 1 - (1 - q_{av}^{n})(1 - q_{av}^{s}) \quad (6\text{-}12)$$

q_{av}^{n} is calculated via Equation (6-7) above. Similar to Zeng's work [Zeng04], q_{av}^{s} is computed using the following expression: $q_{av}^{s}(s) = T_a(s)/\delta$ where $T_a(s)$ is the total amount of time (in seconds) in which service s is available during the last δ seconds. The calculation of q_{av}^{s} is carried out by the service host itself and is then sent to the service requester upon receipt of any service request. δ is a constant set by the service provider of service s.

6.1.2.2. Price

The price $q_{pr}^{u}(s)$ of a service s is the fee that a service requester has to pay for using the service. The value of this QoS parameter is given by the service provider. Service providers either provide means for potential requesters to inquire about it or simply write it into the service description. Price advertisement to the whole wireless network is not a feasible in wireless networks due to the significant overhead caused by broadcasting though it is utilized in wired networks such as for web services [Zeng04]. In our system, service discovery will bring back the price value for a certain service – refer to the "price" tag of the service description language in Chapter 4.

6.1.2.3. Reliability

The reliability $q_{re}^{u}(s)$ of a service s is the probability that a service request is correctly responded, namely, the requester has received the expected results, within the maximum expected time frame indicated in the service description. Reliability is a measure related both to the software and hardware configuration $q_{re}^{c}(s)$ of service s and to the quality of the network connection $q_{re}^{n}(s)$ between service requesters and service providers, namely,

$$q_{re}^{u}(s) = q_{re}^{c}(s) + q_{re}^{n}(s) \quad (6\text{-}13)$$

The calculation of $q_{re}^{c}(s)$ uses history-based method (similar to the method used by Zeng et al [Zeng04] to calculate the successful execution rate of a service), namely, using data of the past service invocations. The following formula is employed:

$$q_{re}^{c}(s) = N_f(s)/T(s) \quad (6\text{-}14)$$

where $N_f(s)$ is the number of times that service s has been successfully finished within the maximum expected time frame, and $T(s)$ is the total number of invocations on service s.

Merge Equation (6-14) and Equation (6-11), there is:

$$q_{re}^{u}(s) = q_{re}^{c}(s) + q_{re}^{n}(s) = N_f(s)/T(s) + 1 - \prod_{i=p}^{r}(1-P_e^i)(1-P_d^i) \qquad (6\text{-}15)$$

6.1.2.4. Delay

The delay $q_{de}^{u}(s)$ of a service s is a measure of duration between the time point when a service request is sent out and the time point when the results are received by the requester. The following expression shows the factors contributing to the delay of a service:

$$q_{de}^{u}(s) = T_{local}(s) + T_{net}(s) \qquad (6\text{-}16)$$

This shows that the delay is caused by both local processing time $T_{local}(s)$ and the network transmission time $T_{net}(s)$. Given the increasing computational power of mobile devices the local processing time is usually ignorable (i.e., by default $T_{local}(s)$ =0). To some computation-intensive services (such as video coding/decoding) $T_{local}(s)$ can be given by service providers (e.g. in the service description or via an inquiry mechanism). $T_{net}(s)$ is given by Formula (6-8).

Given the above service quality criteria, the overall quality of an elementary service s can be described by a quality vector defined below.

Definition 6-1: quality vector of a pervasive service s is 4-element tuple defined as: $QV(s) = (q_{av}^{u}(s), q_{pr}^{u}(s), q_{re}^{u}(s), q_{de}^{u}(s))$, *where the value of each element is calculated by the corresponding formula given above.*

Considering the quality of a pervasive service is measurement for end users, the above QV can be simply represented as:

$$QV(s) = (q_{av}(s), q_{pr}(s), q_{re}(s), q_{de}(s)) \qquad (6\text{-}17)$$

6.1.3. Quality Criteria for Composite Services

The same quality criteria utilized by elementary services are equally applied to composite services. Let's discuss how the value of each quality criterion for a composite service s^C is calculated. Suppose $s^C = \{s_i \mid i = 1,...n\}$ where s_i is an elementary service.

Availability: the availability $q_{av}(s^C)$ of a composite service s^C is given by the product of the availability values of all its component services, namely, $q_{av}(s^C) = \prod_{i=1}^{n} q_{av}(s_i)$.

Execution Price: the price $q_{pr}(s^C)$ of a composite service s^C is the sum of the prices of all its component services, namely, $q_{pr}(s^C) = \sum_{i=1}^{n} q_{pr}(s_i)$.

Reliability: the reliability $q_{re}(s^C)$ of a composite service s^C is the product of the reliability values of all its component services, namely, $q_{re}(s^C) = \prod_{i=1}^{n} q_{re}(s_i)$.

Execution Delay: the delay $q_{de}(s^C)$ of a composite service s^C is the sum of the delay introduced by each of its component service, namely, $q_{de}(s^C) = \sum_{i=1}^{n} q_{de}(s_i)$.

Given the above service quality criteria, the quality vector of a composite service s^C can be defined in the same way as Definition 6-1 as:

$$QV(s^C) = (q_{av}(s^C), q_{pr}(s^C), q_{re}(s^C), q_{de}(s^C)) \quad (6\text{-}18)$$

6.2. QoS-driven Service Selection Algorithm

6.2.1. Service Selection Literature and Goals

Some work has been carried out for service selection, mainly in the field of web services. Service selection constrained by multiple QoS requirements either from end users or from amongst the internal syntax and/or semantics of component services is usually NP-hard [Gu04, Zeng04]. Many heuristic approaches have been proposed to strike the trade-off between the level of optimization and the computational complexity. In Zeng's work [Zeng04], three service selection algorithms are proposed and compared. Amongst these three algorithms, the globally optimized integer programming computation model demonstrates a relatively better performance. However it suffers from high computational complexity and it is not easy for distributed operation in mobile ad hoc networks. Canfora et al [Canfora05] propose a genetic algorithm-based approach to optimize the selection process and compare the results with that of integer programming approach. Yu et al [Yu04] model the service selection problem as a multi-choice Knapsack Problem and present a Pisinger's algorithm to solve the problem. The same work also points out that the network issues should be taken into account in service selection. The network considered by [Yu04] is wired fix network which is assumed of having constant RTT (round trip time). This assumption hardly holds in wireless mobile networks.

Gu et al [Gu04, Gu06] tackle the QoS-aware service selection and composition problem for another kind of more dynamic service rather than web services - peer-to-peer (P2P) multimedia services. This work models explicitly link delay and link availability but the networks considered are still wired networks. The optimized Dijkstra algorithm employed by Gu's work requires the global view of the whole network. Maintaining a global topology view in wireless networks has been proved hardly feasible [Stojmenovic02b].

Analyzing the requirements for the dynamic composition of pervasive services, one can find that it is less dynamic than web service composition. Web service composition (e.g., [Zeng04]) starts from scratch, i.e., each and every task within a composite service needs to find its web service implementation. However, in the case of pervasive services, a big percentage of tasks within a pervasive service have been implemented during service creation stage by pervasive service providers or developers, as such leaving only a limited number of tasks to be discovered at runtime. Figure 40 illustrates this difference.

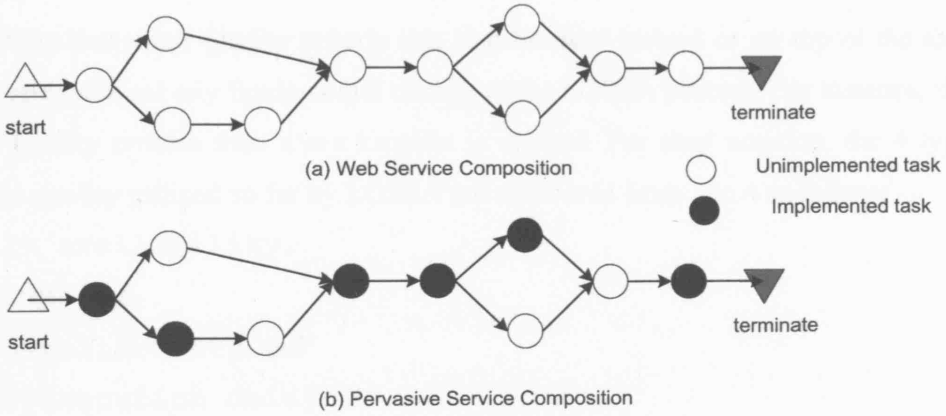


Figure 40: Examples of Initial Task State for Service Composition

This unique feature of pervasive service composition can relax the service composition problem by considering only the individual missing tasks without worrying about the other tasks in a composite service. However, output/input parameter matching of consecutive services needs to be respected. When a task needs to be executed (for instance, Julie needs a German-to-English converter if she arrives at Berlin rather than a Spanish-to-English translator at Barcelona), service discovery mechanism will collect QoS information of each candidate service discovered that can execute this task. After collecting this QoS information, a quality vector is computed for each candidate service. And based on these quality vectors the service selection algorithm (SSA) selects one

candidate service. The service selection algorithm in this Thesis is named as LOSSA, short for Local Optimization based Service Selection Algorithm.

6.2.2. Service Selection Problem Formulation

A pervasive service is composed of a set of tasks $\{t_i \mid i = 1,...,n\}$. Suppose task t_i is needed by a user but its implementation is currently missing in this pervasive service and POETRY has discovered a set of m candidate services $S(t_i) = \{s_{i,1}, s_{i,2},..., s_{i,m}\}$ that can fulfil task t_i. Each service $s_{i,j}, j = 1,..., m$ has 4 quality criteria as discussed earlier. By expanding each $s_{i,j}$ vertically by its 4 quality criteria, a $m \times 4$ matrix is created as below (note that for notation simplicity the task subscript i is omitted in the following notations because here the focus is only on one task: task t):

$$\mathbf{M}=\begin{pmatrix} q_{av}(s_1), q_{av}(s_2),..., q_{av}(s_m) \\ q_{pr}(s_1), q_{pr}(s_2),..., q_{pr}(s_m) \\ q_{re}(s_1), q_{re}(s_2),..., q_{re}(s_m) \\ q_{de}(s_1), q_{de}(s_2),..., q_{de}(s_m) \end{pmatrix} \tag{6-19}$$

Note that other quality criteria can also be used instead or on top of the existing four types without any fundamental change to the LOSSA process. For instance, if there are k quality criteria then a $m \times k$ matrix is created. For easy notation, the 4 types of service quality utilized so far by LOSSA are numbered from 1 to 4 as follows:

1= availability,

2=price,

3=reliability, and

4=execution delay.

Then the matrix in (6-19) can be expressed as

$$\mathbf{M}=(q_{i,j}(s_i); 1 \le i \le m, 1 \le j \le 4) \tag{6-20}$$

In matrix **M**, each column q_i corresponds to a component service (s_i) whereas each row q_j corresponds to a quality dimension.

The constraints on task t are represented as a 4-element vector $C =< c_1, c_2, c_3, c_4 >$ where each element from left to right denotes the constraint on availability, price, reliability and delay, respectively.

Given a task t and its constraint vector C, LOSSA is to choose one component service that implements t and satisfies all the user constraints C for t and at the same time having the optimal score. The selection is carried out at two levels: the individual criterion level (or *I-level*) and the aggregated criteria (or score) level (or *A-level*). The

purpose of I-level is to filter out these candidates that fail the requirement on an individual criterion. For these that have passed each individual criterion test, they are further assessed based on an aggregated score. To calculate the aggregated score of a candidate component service the following two steps are needed: *normalizing* and *scoring*.

Normalizing: Amongst these 4 criteria for a given component service, some are positive, i.e., the higher the value the higher the service quality, such as reliability and availability; whereas some are negative, i.e., the higher the value the lower the service quality, such as price and execution delay. To express these criteria in a uniform manner, they are *normalized* using the formula below.

$$a_{i,j} = \begin{cases} (q_{i,j} - q_j^{\min}) \Big/ (q_j^{\max} - q_j^{\min}) & if\ q_j^{\max} - q_j^{\min} \neq 0 \\ 1 & if\ q_j^{\max} - q_j^{\min} = 0 \end{cases} \quad (6\text{-}21)$$

$$a_{i,j} = \begin{cases} (q_j^{\max} - q_{i,j}) \Big/ (q_j^{\max} - q_j^{\min}) & if\ q_j^{\max} - q_j^{\min} \neq 0 \\ 1 & if\ q_j^{\max} - q_j^{\min} = 0 \end{cases} \quad (6\text{-}22)$$

For positive criteria, the values are scaled according to (6-21), and for negative criteria the values are scaled according to (6-22). In (6-21) and (6-22), $q_j^{\max} = \max(q_{i,j}), 1 \le i \le m$ is the maximal value of quality criterion j; $q_j^{\min} = \min(q_{i,j}), 1 \le i \le m$ is the minimum value of quality criterion j. Applying these two equations on **M**, a new matrix **N** is created:

$$\mathbf{N} = (a_{i,j}; 1 \le i \le m, 1 \le j \le 4). \quad (6\text{-}23)$$

Matrix **N** is a normalized matrix of **M**. For example, assume there are m=6 component services that can fulfil a given task t. The price values of these component services are given by vector $q_{pr} = q_2 = (10,15,60,45,20,35)$. In this case, $q_2^{\max} = 60$ and $q_2^{\min} = 10$. Since price is a negative criterion, Formula (6-22) is used for scaling. Then there is:

$$N_2 = (1,0.9,0,0.3,0.8,0.5).$$

Scoring: the aggregated score of a service reflects its overall quality and is calculated using a weighted mean as follows:

$$score(s_i) = \sum_{j=1}^{4} (a_{i,j} * w_j) \quad (6\text{-}24)$$

Where w_j represents the weight of criterion j and it satisfies: $w_j \in [0,1] \wedge \sum_{j=1}^{4} w_j = 1$.

The value of each w_j is given by users based on their preferences regarding service quality. By default $w_{j=1,...,4}$=0.25.

6.2.3. Proposed Service Selection Algorithm

Based on this aggregated score A-level filtering can be carried out. The candidate that scores the highest is the final choice of task *t*. If after A-level filtering there are still more than one candidate then a finer selection mechanism called *distance-based filtering* is utilized. Note in Zeng et al's work [Zeng04], a random selection is applied if there are multiple candidates after A-level selection process. The following pseudo code describes the LOSSA algorithm.

LOSSA Algorithm
Input:
a set of component services $S(t) = \{s_1, s_2, ..., s_m\}$ that each fulfils a given task *t;*
a set of constraints on task t – represented as a 4-element vector $C = < c_1, c_2, c_3, c_4 >$;
Output: an optimal component service $s_i \in S(t)$ that fulfils task *t*.

Step 0: initialization. Create $m \times 4$ matrix **M** using formula (6-20);
Step 1: I-level filtering
for *i*=1 to *m* do
for *j*=1 to 4 do
if $(q_{i,j}(s_i) \lhd c_j)$ then delete column M_i (i.e., service s_i) from **M**;
Step 2: A-level filtering
Set *l* as the number of column in updated **M**;
Generate normalized quality vector **N** from **M** using formula (6-21) and (6-22);
Use formula 6-24 to generate an *l*-element vector (i.e., weighted services) *V*1;
Sort *V*1 according to its value in a descending order into *V*2;
Step 3: Distance-based filtering and result return
if *element*(*V*2, 1) == *element*(*V*2, 2) then
Set *k* as the number of elements in *V*2 that all have the highest score;
Set *x*=L (the maximum diagonal length of the network concerned);
for *i*=1 to *k* do
calculate the distance *y* between the requesting node and the node where service represented by *element*(*V*2, i) locates;
if *y*<*x* then *x*=*y*;
return the first s_i where its distance to the requesting node is *x*;
else
return s_i where $score(s_i) == element(V2,1)$;

$(x \lhd y)$ means the quality of x is worse than the quality of y. For any positive QoS criterion there is: if $(x < y)$ then $(x \lhd y)$. For any negative QoS criterion there is: if $(x > y)$ then $(x \lhd y)$. *element*($V2$, i) returns the value of the i-th element in $V2$. Note that the distance-based filtering algorithm selects the service whose hosting node stays the closest to the requesting node. This implies the following potential benefits: 1) fewer hops from the service requester to the selected service provider and as such less likely the route is to break; 2) less distance means shorter distance of data transmission and as such less energy consumption; 3) less likely that these two nodes move out of each other's reach.

The above LOSSA algorithm is for one task only. In this case its computational complexity is $O(m*4)$ or $O(m*n)$ where m is the number candidate component services for the task and n is the number of criteria. If there are k tasks to be discovered in a pervasive service then the above LOSSA needs to repeat k times making the computational complexity of the service composition $O(m*n*k)$. As illustrated in Figure 40, k is a small number for pervasive services in comparison with that of web services. Furthermore, here in this Thesis n=4. Considering the significant side effect of broadcasting in wireless networks, the above LOSSA algorithm can be limited in terms of the number of hops from the service requester within which services can be discovered. This limited-broadcast based LOSSA is named as LOSSA-k where k defines the number of hops. LOSSA-k also reduces the number of candidate components to be found, i.e., m is reduced.

6.3. Implementation and Evaluation

6.3.1. Implementation

In order to evaluate the proposed service selection approach in wireless mobile networks, network simulator ns-2 is utilized, as in the previous two chapters. LOSSA is implemented at the application layer of ns-2 stack following the service composition architecture in Figure 41. Each mobile host is identically equipped with these modules and works with other mobile hosts in an ad hoc manner. Namely, each mobile host is a service provider and at the same time a service requester. This is a fully distributed architecture.

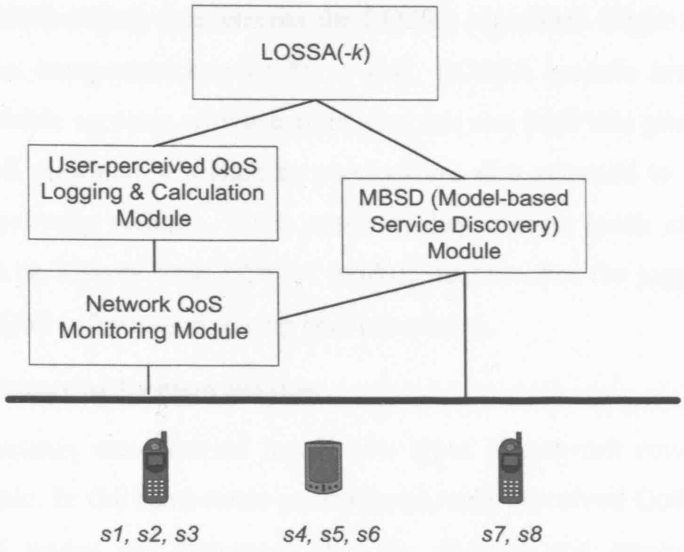


Figure 41: Service Composition System Architecture

The *User-perceived QoS Logging and Calculation Module* carries out the following two things. Firstly, it locally logs the necessary data (e.g., service component availability) into a local repository (a text file in this case) after each service invocation. Secondly, it implements the formulas discussed in Section 6.1. The variety of QoS criteria values used by these formulas are obtained by interacting with the underlying Network QoS Monitoring Module and the log files.

The *Network QoS Monitoring Module* provides values for the network QoS criteria using the following means respectively:

- Mobility (used by the network availability criterion): different mobility scenarios can be generated by using movement-connection program *./setdest* provided by ns-2.
- Delay: is defined as the packet receiving time at destination node minus the packet sending time at source node and can be calculated by ns-2.
- The channel error rate P_e and the queue dropping rate P_d (used by the network reliability criterion): they are related to the MAC (Medium Access Control) protocol used. In the evaluation experiments below, IEEE 802.11 is used – it has the widest application in practice and it is the most well-implemented wireless MAC protocol in ns-2. In our current implementation, these two parameters are not obtained individually. However, they can be calculated by changing the source code of 802.11 implementation (refer to ~ns/mac-802_11.{cc,h} [ns]).

MBSD Module has been implemented in ns-2 in the previous chapters. Here in the evaluation, the pull method in utilized.

LOSSA Module mainly implements the LOSSA algorithm. Upon receipt of a new request to find a component service for a task, LOSSA module invokes MBSD to discover all available services within certain area that can fulfil this given task. Some of the network QoS parameters related to services are also returned to LOSSA module after service discovery process. Since some quality criteria (such as reliability) are calculated based on history data, LOSSA module also invokes the logging functions to log appropriate QoS information during task executions.

6.3.2. Experimental Design Issues

The experiments were carried out in two types of *network environments*: semi-static and dynamic. In the semi-static environment, user-perceived QoS parameters stay unchanged, and nodes are stationary, but the channel and queue status changes according to the IEEE802.11 protocol (so precisely it is not fully static – however Zeng et al [Zeng04] regard this as "static" because they do not consider network status). It is also assumed that in the semi-static environment each component service is able to execute the tasks successfully and in conformance with its QoS expectation. In the dynamic environment, the QoS values, including the user-perceived QoS parameters and the node mobility, might undergo changes during the execution of a composite service. As a result, an existing service might become unavailable. For simplicity and presentation clarity, during the evaluation under the dynamic environment, two out of the four QoS criteria, namely, availability and price, were changing their values whereas the two remaining QoS criteria (delay and reliability) stayed unchanged. However, the same experimental procedure is applicable to whatever QoS criteria employed.

Another dimension of experimental design is related the *metrics*. In this Thesis, the service selection algorithm, together with service discovery algorithm, is evaluated from two aspects: communication overhead and the quality of the composite service. The network environment is the same as that in Chapter 4 (refer to Table 1 of Chapter 4). For each test case, the average of five measurements was utilized as the final result.

Other variables that affect the performance of our service selection algorithm include: the number of tasks whose implementations are missing and as such need to be discovered; and the number of candidate component services for a given task.

6.3.3. Communication Overhead

The communication overhead is measured in a semi-static network environment in terms of the number of bytes sent out during service selection. The number of tasks in a pervasive service is varied to observe the communication overheads of both LOSSA

and LOSSA-*k*. Here each task is discovered separately. Figure 42 shows the results of this experiment.

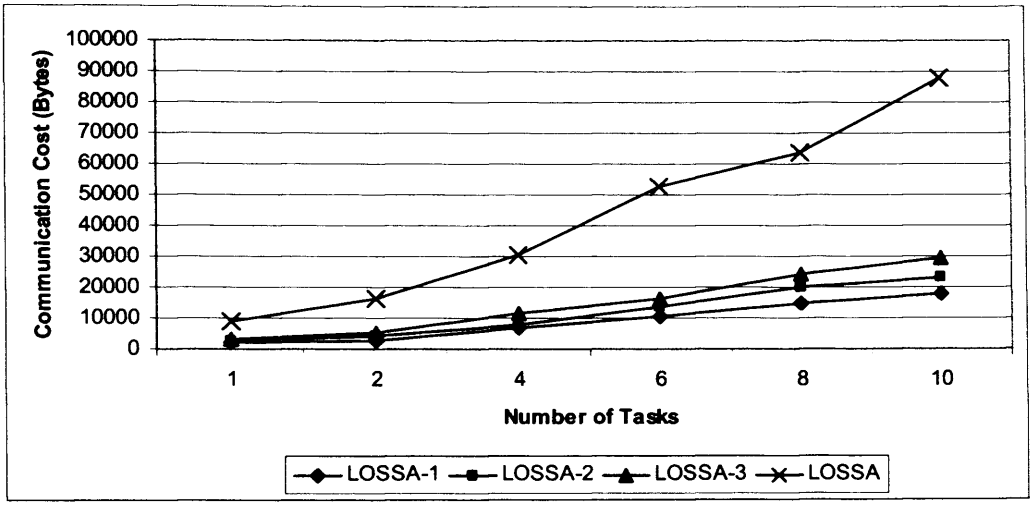


Figure 42: Communication Cost vs. Number of Tasks

It is observed that for each algorithm the communication cost increases with the increase of the number of tasks missing in the current pervasive service. Furthermore, the wider the search cope is the more communication overhead incurs. For instance, when there are 8 missing tasks in a pervasive service, using LOSSA will incur about 320% the communication of LOSSA-2. This polynomial increase is mainly due to the fact that each task is discovered separately. If the composite service knows it is missing more than one task (actually their implementation) then it can discover these tasks in one round of MBSD to reduce the communication overhead. The results indicate that the global-search-based LOSSA should be avoided if possible, especially when the number of missing tasks becomes bigger.

6.3.4. QoS of Pervasive Services

The QoS changes in the dynamic environments are carried out from two aspects. For the user-perceived QoS parameters such as price and service availability (q_{av}), they are randomly changed in the following ranges: $5 \leq price < 100$, $50\% \leq q_{av} \leq 100\%$. For network QoS parameters such as mobility the existing mechanism in ns-2 is utilized. Every time a selected component service becomes unavailable, the LOSSA or LOSSA-*k* is triggered.

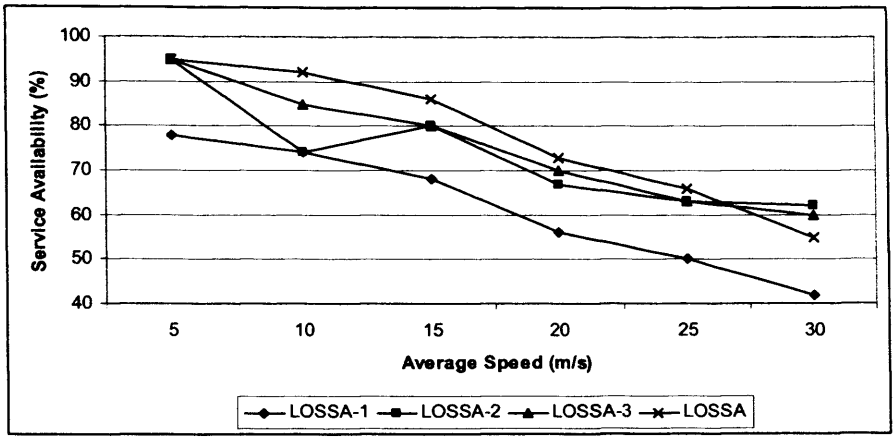


Figure 43: Service Availability vs. Node Mobility (Number of tasks is 6)

Figure 43 shows the changes of service availability against the average moving speed of nodes. In general, for each algorithm, the service availability deteriorates as the average node speed increases. The global-search-based LOSSA gives the best service availability. This is because the underlying routing protocol will conduct automatic route repair if the route is broken due to certain nodes on the route moving away. Here a widely-used routing protocol for mobile wireless ad hoc networks called Dynamic Source Routing (or DSR) [Johnson01], [ns2] is utilized for routing purpose. The multi-hop nature of LOSSA gives DSR more chance to find an alternative route to the destination. However, when the average node speed is more than 28m/s both LOSSA-2 and LOSSA-3 (both using limited broadcast) slightly outperform LOSSA. This is because high node speed makes it difficult to maintain a stable route and this is more of a case if there are more hops (which is the case for LOSSA). Comparing this against the results in Figure 42, it can be observed that this QoS gain is at the cost of significant communication. For instance, at the point where the average node speed is 15m/s, the 6% QoS gain of LOSSA over LOSSA-3 is at the cost of 323% the communication overhead of LOSSA-3. Therefore, when the wireless network concerned is congested it is recommended not to use global-search based LOSSA.

Figure 44 shows the average service price. For each algorithm, there is not a clear and stable trend to follow. This is reasonable because the tasks and their price nature are largely decided by their providers. Amongst these algorithms, LOSSA offers better performance price-wise. Actually, the wider the search scope is (i.e., the bigger k is) the better the chance to find a lower price is. Namely, LOSSA-i usually outperforms LOSSA-(i-1) price-wise.

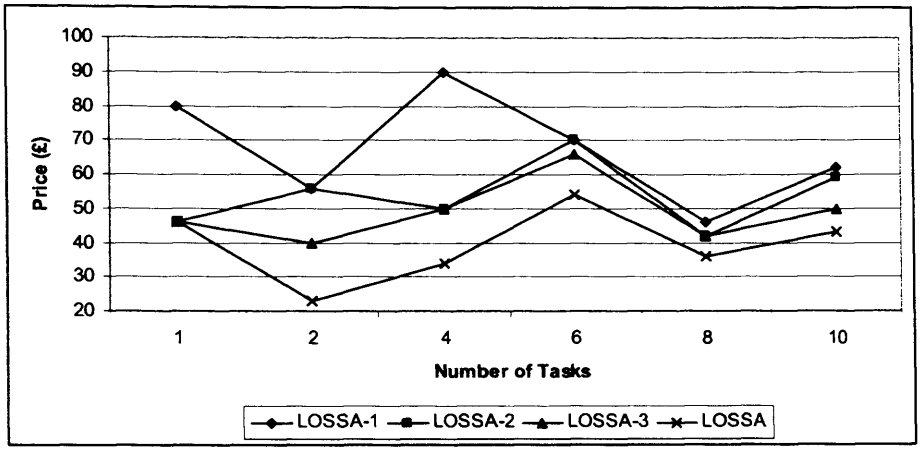


Figure 44: Average Service Price vs. Number of Tasks

(The average node speed is 15m/s)

Based on the experimental results above, it can be concluded that the LOSSA offers better QoS (e.g., both price and service availability) but at a big cost of communication overhead. The final choice of LOSSA and LOSSA-*k* (and *k* itself) depends on the constraints imposed by users such as price, end-to-end delay. LOSSA-3 seems a better trade-off based on the above results.

6.4. Summary

This Chapter has presented a QoS-aware service selection algorithm for pervasive service composition. The major contributions of this chapter are summarized as follows. Firstly, for the first time a QoS model specific for pervasive services is proposed and this model considers not only user-perceived factors but also mobile wireless network characteristics. The formulas to calculate each QoS criterion are also given. Secondly, this Chapter has identified the differences between web service composition and pervasive service composition, based on which a local optimal service selection algorithm (LOSSA) is introduced. Thirdly, this Chapter has formulated the QoS-aware service selection problem for pervasive service composition and proposed some solutions to the problem, i.e., global-search-based LOSSA and limited broadcast based LOSSA-*k*. The corresponding evaluation of the algorithms are also presented and discussed.

Chapter 7. Conclusions

This Thesis addresses the problem of QoS-aware pervasive service engineering in wireless mobile networks by mainly focusing on service creation, service discovery and service selection/composition. A software framework fulfilling the above tasks, called POETRY, has been presented. In comparison to the conventional service engineering approach which emphasizes on the importance of building a comprehensive service execution environment, the POETRY provides service adaptability by focusing on the service internal logic itself. In this light, policies have been adopted to describe the features and behaviours of a pervasive service, and a pervasive service itself is modelled as a policy-based management system whose execution relies little on the supporting environment. The POETRY framework also takes advantage of the emerging software engineering technique MDA (Model-driven Architecture) and uses it to model pervasive service information model. A prototype POETRY system has been developed which enables both off-line service creation and online service discovery and QoS-aware service composition. POETRY not only presents a methodology and framework to benefit service providers but also provides a flexible way for mobile service end users to use their pervasive services in a more adaptive manner.

This Chapter firstly summarizes the major contributions of the Thesis, then briefly discusses some future research directions.

7.1. Major Contributions

In the course of designing and implementing an integrated framework (POETRY) for QoS-aware pervasive service engineering in wireless mobile networks, the Thesis has made the following major contributions to the state-of-the-art of service engineering:

- **A new methodology for pervasive service engineering.** POETRY proposes a novel service engineering methodology that advocates a shift of pervasive service engineering focus from the *external service execution and supporting environment* to the *internal service logic* itself. To this end it proposes to utilize policies to describe pervasive service logics and for the first time creates a pervasive service as a standalone and self-contained PBM system. A pervasive service developed in such a manner bears inherent adaptability to context changes.

- **MDA Models for Pervasive Service Information Model (PSIM).** In order for a pervasive service to run as a PBM system, the corresponding pervasive service information model has been developed. Furthermore, this PSIM is developed using a new method, namely, a MDA approach instead of a rigid class hierarchy. The technical details of this approach have been presented. This new way of implementing the classes needed by pervasive services brings the following benefits to the POETRY framework: 1) giving flexibility to class implementation, e.g., the classes and their relationships can be designed without having to consider the implementing platforms or languages and this higher-level design is mapped to specific platform or programming language only when services need to be deployed to certain specific environments; 2) making the service components discovered more easily to be integrated into the current service, especially when both are developed using MDA techniques because then the integration can be carried out at platform independent model level. Since pervasive services in POETRY are developed using policies, the relationship between polices and models is also discussed.
- **Ad hoc service discovery algorithms using MDA models.** The ad hoc service discovery mechanisms proposed in POETRY reuse the models that were generated during static service creation/development stage in the following two ways: firstly, service description is generated automatically; secondly, the model information carried in service description is used to regenerate original models which can be reused to compose new services or to update existing services using the MDA approach. In terms of the service discovery procedure, two complementary service discovery algorithms based on the pull method and the push method respectively have been presented and evaluated. Different from the most existing service discovery algorithms for wireless networks, the two algorithms proposed in this Thesis considers also the wireless, mobile and ad hoc natures of the underlying networks, making them more suitable to be used in pervasive services that also run over wireless mobile networks.
- **A comprehensive quality model for pervasive services.** A comprehensive model for describing the quality of pervasive services (QoPS) has been proposed. This model accommodates not only these parameters that are directly perceived by end users but also network parameters. Both network QoS parameters and user-perceived QoS parameters have been quantified and integrated together for both elementary services that are the building blocks of pervasive services and

composite pervasive services. Then making use of this QoPS model, a QoS-aware service selection algorithm for service composition has been designed, implemented and evaluated.

- **A functioning pervasive service development toolkit**. Using this toolkit, service developers can easily create the source code of an executable pervasive service skeleton. This skeleton bears a generic but intelligent decision making engine that endorses the adaptability of the pervasive service via an event-based context sensing mechanism. The corresponding service discovery algorithms are also embedded inside this skeleton and are triggered automatically in response to certain context-event change.

The feasibility of efficiency of POETRY is validated using both prototype implementation and simulations. To demonstrate the effectiveness of POETRY, a pervasive service scenario *SNAMU* has been prototyped using POETRY, including both the code generation and the model-driven aspect of the service discovery. To evaluate the network aspects of POETRY, such as service request broadcasting, service advertisement propagation, and the communication overheads for service selection, POETRY (or more precisely the corresponding algorithms in POETRY) is compared against other existing high-performance algorithms. The experiments and simulation results show that the POETRY approach to pervasive service engineering is feasible, effective and efficient.

7.2. Open Research Issues

Open research issues still exist in realizing fully automatic pervasive service creation and a practically autonomic service execution system. Several interesting issues are discussed as follows.

- **Energy-aware pervasive service engineering.** Given the battery-supplied nature of mobile devices on which pervasive services run, energy efficiency becomes an important issue for pervasive service execution and adaptation in wireless mobile networks. Energy consumption of a mobile device depends on the amount of data transmitted, transmission distance, channel quality, and other factors. The service discovery algorithms in POETRY try to reduce the control message traffic, e.g., trying to reduce the number of service request (SREQ) and service advertisement retransmission. This contributes to the energy saving. However, energy-efficiency has not been a design and evaluation metric in POETRY and there is no systematic

investigation of the energy-efficiency of the POETRY system. Thus, one interesting future research issue is to look into how the performance of POETRY affects the energy consumption of mobile devices and how to revise the POETRY design (especially the dynamic part such as service discovery and service selection) to provide energy-efficient pervasive service engineering solutions.

- **Security, privacy and piracy issues in pervasive services.** POETRY currently does not provide security support at its current version. Secure pervasive service execution and composition is important in open, mobile and resource-sharing computing platforms such as wireless mobile ad hoc networks. Commercialized pervasive services will have to meet certain security criteria before being largely deployed. [Hao06] and [Verissimo06] together provided a comprehensive discussion on security issues arising in wireless computing environment. Privacy [Beresford03], [Ren06] is equally important in multi-hop wireless networks such as the network environment concerned in this Thesis because end users would not like their information transmitted in a multi-hop manner to be interpreted by any intermediate nodes. Nor would the intermediate nodes like their information to be interpreted by the passing code. Thus, how to protect privacy when conducting service selection and discovery is an interesting future topic. For example, a trust management system can be integrated into the POETRY framework. Proper authentication and authorization mechanisms can not only provide privacy and security protection but also help control software piracy epidemic [Ren06]. Jacobs *et al.* proposed a framework for comparing perspectives on privacy and pervasive technologies where several key cases relating to privacy and technology were discussed [Jacobs03]. Because security mechanisms bring overhead a good balance between levels of security and the corresponding overhead introduced needs to be worked out – typically on a per scenario basis. [Zhu06] provides a good example of integrating security and privacy with user-centric information exposure during service discovery.

- **Cross-layer pervasive service engineering.** While wireless mobile ad hoc networks provide ample opportunities to provide various services, its effective commercialization is still challenging due to not only limited

battery supply but also various other constraints such as time-varying fading effect, limited bandwidth, different protocols and application requirements. *Cross-layer design (CLD)* is a new paradigm that addresses this challenge by optimizing communication network architectures across traditional layer boundaries defined by ISO (International Standard Organization) OSI (Open Systems Interoperability) protocol stack [Khan06]. Design of pervasive services over these networks using cross-layer design method hold great promise for addressing these challenges and providing more flexible and robust algorithms for services operating in such networks. The work carried out so far in the Thesis has accommodated to some extent the CLD flavour, which can be reflected for example in quality model of pervasive services and service discovery algorithms. A more interesting work is to investigate how to design a cross-layer optimization strategy that are driven mainly by the application-layer services but also jointly optimizing the lower layers such as data link layer, etc. An application-oriented objective function can be integrated into the POETRY system in order to adaptively maximize user satisfaction.

- **A pervasive service DSL or domain specific language underpinned by rich and universal web semantic.** The entities used by the pervasive service DSL in this Thesis are proprietary and do not bear any fixed meaning that is universally understandable by a computer. Though due to the fact there is not such standardized ontology for service engineering in general in the current community, this largely impedes the re-use of services developed out of such a DSL. How and to what extent to import the existing research outcomes in semantic web areas, or even to spark a new research area in semantic web areas, and to refine the pervasive service DSL (or CIM in MDA terminology), is also an interesting future research topic.

References

[Banavar00] G. Banavar, J. Beck, E. Gluzberg, J. Munson, etc. "Challenges: An Application Model for Pervasive Computing", *Proc. of the sixth annual IEEE/ACM Int. Conf. on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[Basagni98] S. Basagni, I. Chlamtac, V. R. Syrotiuk etal. "A distance routing effect algorithm for mobility (DREAM)." *Proc. of the ACM/IEEE Int. Conf. on Mobile Computing and Networking (MOBICOM)*, Dallas, 1998

[Beresford03] A. Beresford, F. Stajano. "Location privacy in pervasive computing", *IEEE Pervasive Computing,* Vol. 2, Issue 1, Jan-Mar 2003 Page(s):46 – 55.

[Camp01] T. Camp, J. Boleng, and L. Wilcox. "Location information services in mobile ad hoc networks". *Proc. of IEEE. Int. Conf. of Communications (ICC)*, 2001.

[Canfora05] G. Canfora, M. Di Penta, R. Esposito, M. L. Villani. "An approach for QoS-aware service composition based on genetic algorithms", *Proc. of the 2005 conference on Genetic and Evolutionary Computation (GECCO)*, Washington DC, USA. Pages: 1069-1075.

[Chakrabarti01] S. Chakrabarti, A. Mishra. "QoS issues in ad hoc wireless networks", *IEEE Communications Magazine*, Volume 39, Issue 2, Feb. 2001 Page(s):142 – 148

[Chakraborty06] D. Chakraborty, A. Joshi, Y. Yesha, T. Finin. "Toward Distributed service discovery in pervasive computing environments", *IEEE Trans. On Mobile Computing,* Vol. 5, Issue 2, Feb. 2006 Page(s):97 – 112.

[Chen04] G. Chen, B. Kang, M. Kandemir, et. al, "Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, No. 9, Sept. 2004

[Chetan05] S. Chetan, A. Ranganathan, R. Campbel. "Towards fault tolerance pervasive computing", *IEEE Technology and Society Magazine,* Vol. 24, Issue 1, Spring 2005 Page(s):38 – 44

[CMU]ns-2 network simulator at CMU: http://www.monarch.cs.cmu.edu/cmu-ns.html

[Damianou01] N. Damianou, N. Dulay, E. Lupu, M Sloman. "The Ponder Specification Language". *Proc. of Workshop on Policies for Distributed Systems and Networks (Policy2001)*, HP Labs Bristol, UK, Jan. 2001.

[Decker00] S. Decker, S. Melnik, F. van Harmelen, etc. The Semantic Web: the roles of XML and RDF", *IEEE Internet Computing*, Vol. 15, Issue 3, October 2000, page(s): 63-74.

[Dertouzos99] M. Dertouzos. The Oxygen Project. *Scientific American*, 281(2):52-63, August 1999.

[Dey00a] A. K. Dey and G. D. Abowd. "Towards a better understanding of context and contextawareness". *Proc. of Workshop on the What, Who, Where, When and How of Context-Awareness*, April, 2000.

[Dey00b] A. K. Dey. "Providing Architectural Support for Building Context-Aware Applications". PhD thesis, Georgia Institute of Technology, November 2000.

[Dulay01] N. Dulay, E. Lupu, M Sloman, N. Damianou. "A Policy Deployment Model for the Ponder Language". *Proc. IEEE/IFIP International Symposium on Integrated Network Management (IM'2001)*, IEEE Press, Seattle, May 2001.

[Dustdar05] S. Dustdar, W. Schreiner; "A survey on web services composition", *International Journal of Web and Grid Services*, Vol. 1, Issue 1, 2005.

[EMF] "Eclipse Modelling Framework - EMF", Eclipse Foundation Inc. http://www.eclipse.org/emf/

[Erl05] T. Erl. "Service-Oriented Architecture (SOA): Concepts, Technology, and Design", Prentice Hall PTR, August, 2005. ISBN: 0131858580

[Esler99] M. Esler, J. Hightower, T. Anderson, and G. Borriello. "Next Century Challenges: Data-Centric Networking for Invisible Computing - The Portolano Project at the University of Washington". *Proc. the Fifth ACM/IEEE International Conference on Mobile Networking and Computing (MobiCom)*, August 1999. Pages 256-262.

[Flinn02] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing Performance, Energy, and Quality in Pervasive Computing," *Proc. of 22nd IEEE Int'l Conf. on Distributed Computing Systems*, Austria, July 2002.

[Gamal04] A. Gamal, J. Mammen, B. Prabhakar, D. Shah. "Throughput-delay trade-off in wireless networks", *Proc. of the IEEE Infocom*, March 2004.

[Genco06] A. Genco, S. Sorce, G. Reina, G. Santoro. "An agent-based service network for personal mobile devices", *IEEE Pervasive Computing,* Vol. 5, Issue 2, April-June 2006 Page(s):54 – 61.

[Georgalas04] N. Georgalas, M. Azmoodeh, T. Clark, A. Evans, P. Sammut, J. Willans. "MDA-Driven Development of standard-compliant OSS components: the OSS/J Inventory Case-Study". *Proc. of the Second European Workshop on Model Driven Architecture with emphasis on Methodologies and Transformations (EWMDA 2004)*, Canterbury, UK, 7-8 September 2004.

[GMF] "Graphical Modelling Framework - GMF", Eclipse Foundation Inc. http://www.eclipse.org/gmf/

[Grimm04] R. Grimm. "One.world: experiences with a pervasive computing architecture", *IEEE Pervasive Computing,* Volume 3, Issue 3, July-Sept. 2004 Page(s):22 – 30.

[Gu04]X. Gu. "SpiderNet: A Quality-Aware Service Composition Middleware", Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, October, 2004.

[Gu06]X. Gu, K. Nahrstedt. "Distributed multimedia service composition with statistical QoS assurances", *IEEE Transactions on Multimedia*, Volume 8, Issue 1, Feb. 2006, Page(s):141 – 151.

[GuT04] T. Gu, H. K. Pung, D. Zhang. "Toward an OSGi-based infrastructure for context-aware applications", *IEEE Pervasive Computing*, Vol. 3, Issue 4, Oct-Dec 2004 Page(s):66 – 74.

[Hao06] Y. Hao, F. Ricciato, S. Lu, L. Zhang. "Securing a wireless world", *Proceedings of the IEEE*, Vol. 94, Issue 2, Feb. 2006 Page(s):442 - 454

[Harroud03] H. Harroud, M. Ahmed, A. Karmouch. "Policy-driven Personalized Multimedia Services for Mobile Users", *IEEE Trans. On Mobile Computing*, Vol. 2, No. 1, Jan.-March 2003. Page(s): 16-24.

[Helal02] S. Helal. "Standards for service discovery and delivery", *IEEE Pervasive Computing*, Volume 1, Issue 3, July-Sept. 2002. pp.:95 – 100.

[Hermann01] R. Hermann, D. Husemann, M. Moser, M. Nidd, C. Rohner, and A. Schade. "DEAPspace-transient ad hoc networking of pervasive devices," *Computer Networks*, vol. 35, no. 4, pp. 411-428, March 2001.

[IC-Policy-web] Policy Research Group, Department of Computing, Imperial College London website: http://www-dse.doc.ic.ac.uk/Research/policies/index.shtml.

[IETF-OPES] The IETF Open Pluggable Edge Services (opes) website: http://www.ietf.org/html.charters/opes-charter.html.

[IETF-policy] IETF Policy Framework Working Group web page: http://www.ietf.org/html.charters/policy-charter.html.

[Jacobs03] A. R. Jacobs, G. Abowd. "A framework for comparing perspectives on privacy and pervasive technologies", *IEEE Pervasive Computing*, Vol. 2, Issue 4, Oct.-Dec. 2003 Page(s):78 – 84.

[JC] JavaCC website: https://javacc.dev.java.net/

[Johnson01] D. Johnson, D. Maltz, and J. Broch; "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks". in *Ad Hoc Networking*, edited by Charles E. Perkins, Chapter 5, pp. 139-172, Addison-Wesley, 2001.

[Kagal03] L. Kagal, T. Finin, A. Joshi. "A Policy Language for a Pervasive Computing Environment". Proc. Of IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, June, 2003

[Khan06] S. Khan, Y. Peng, E. Steinbach, M. Sgroi, W. Kellerer. "Application-driven cross-layer optimization for video streaming over wireless networks", *IEEE Communications Magazine*, Volume 44, Issue 1, Jan. 2006 Page(s):122 - 130

[Ko98]Y. Ko and N. H. Vaidya. "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks", *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (Mobilcom)*, 1998. Page(s): 66-75

[Koch96] T. Koch, C. Krell, et al. "Policy Definition Language for Automated Management of Distributed Systems". *Proc. of the IEEE Second International Workshop on Systems Management*, Toronto, Canada, June, 1996.

[Kumar03] M. Kumar, B. Shirazi, S. K. Das, B. Sung, D. Levine. "PICO: A middleware framework for pervasive computing", *IEEE Pervasive Computing*, July-Spet. 2003. Page(s): 72-79.

[Lee03] C. Lee, D. Nordstedt, S. Helal. "Enabling smart spaces with OSGi", *IEEE Pervasive Computing,* Vol. 2, Issue 3, July-Sept. 2003 Page(s):89 – 94.

[Lee03] C. Lee; A. Helal, N. Desai, et. al. "Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services", *IEEE Transactions on Systems, Man and Cybernetics, Part A*, Volume 33, Issue 6, Nov. 2003 Page(s): 682 – 696.

[Liu05] Q. Liu, S. Zhou, G.B. Giannakis.; "Cross-layer scheduling with prescribed QoS guarantees in adaptive wireless networks", *IEEE Journal on Selected Areas in Communications (JSAC)*, Volume 23, Issue 5, May 2005 Page(s):1056 – 1066.

[MDA]Model Driven Architecture website: http://www.omg.org/mda

[Medjahed05] B. Medjahed, A. Bouguettaya; "A multilevel composability model for semantic Web services", *IEEE Transactions on Knowledge and Data Engineering*, Volume 17, Issue 7, July 2005 Page(s):954 – 968.

[Meyer94] B. Meyer, and C. Popien. "Defining Policies for Performance Management in Open Distributed Systems". *Proc. of the IEEE/IFIP Workshop on Distributed Systems Operations and Management (DSOM94)*, Toulouse, France, October 1994.

[Miller03] J. Miller, J. Mukerji (editor). "MDA Guide Version 1.0.1". online on OMG document webpage: http://www.omg.org/docs/omg/03-06-01.pdf.

[Nidd01] M. Nidd. "Service discovery in DEAPspace," *IEEE Personal Communications*, vol. 8, Aug. 2001, Page(s): 39-45.

[Niebert04] N. Niebert, A. Schieder, H. Abramowicz, G. Malmgren, J. Sachs, U. Horn, C. Prehofer, H. Karl. "Ambient Networks – An Architecture for Communication Networks Beyond 3G", *IEEE Wireless Communications* (Special Issue on 4G Mobile Communications – Towards Open Wireless Architecture), April 2004. Page(s): 14-22.

[ns2] The Network Simulator - ns-2 website: http://www.isi.edu/nsnam/ns/

[O'Sullivan02] J. O'Sullivan, D. Edmond and A. Hofstede; "What's in a Service?" *Journal of Distributed and Parallel Databases*, vol. 12, nos. 2-3, Sept. 2002, Page(s): 117-133.

[Orriens03] B. Orriens, J. Yang, and M. Papazoglou. "Model Driven Service Composition". *Proc. of ICSOC'03*, Dec. 2003. Page(s): 75-90.

[Papazoglou03] M. P. Papazoglou; "Service-oriented computing: concepts, characteristics and directions", *Proc. of the 4th International Conference on Web Information Systems Engineering*, 2003. Page(s): 3-12.

[ParcTab] http://www.ubiq.com/hypertext/weiser/UbiHome.html

[Pashtan04] A. Pashtan, A. Heusser, P. Scheuermann. "Personal service areas for mobile Web applications", *IEEE Internet Computing,* Vol. 8, Issue 6, Nov.-Dec. 2004 Page(s):34 – 39.

[Peng02] W. Peng and X. Lu. "AHBP: An efficient broadcast protocol for mobile ad hoc networks," *Journal of Science and Technology* - Beijing, China, 2002.

[QoSForum] QoS Forum. "White Paper: QoS Protocols and Architectures", www.qosforum.com, 1999.

[Ren06] K. Ren, W. Lou, K. Kim, R. Deng. "A novel privacy preserving authentication and access control scheme for pervasive computing environments", *IEEE Transactions on Vehicular Technology,* Vol. 55, Issue 4, July 2006 Page(s):1373 – 1384

[Satyanarayanan01] M. Satyanarayanan. "Pervasive computing: vision and challenges". *IEEE Journal of Personal Communications,* Vol. 8 Issue: 4, Aug. 2001, Page(s): 10 -17.

[Schilit94] B. Schilit, M. Theimer. "Disseminating Active Map Information to Mobile Hosts". *IEEE Network*, 8(5): 22-32, 1994.

[Shen05] J. Shen, K. Yang, S. Zhong. "A Prediction-based Location Update Algorithm in Wireless Mobile Ad-hoc Networks". *Proc. of the 2005 International Conference on Computer Networks and Mobile Computing (ICCNMC'05)*, Aug. 2005, Zhangjiajie, China. Springer LNCS-3619: 111 – 120.

[Sloman94] M. Sloman. "Policy Driven Management For Distributed Systems," *Journal of Network and System Management*, vol. 2, no. 4, Dec. 1994, Page(s): 333–360.

[Stojmenovic02a] I. Stojmenovic, M. Seddigh, J. Zunic. "Dominating Set and Neighbor Elimination-Based Broadcasting Algorithms in Wireless Networks," *IEEE transactions on parallel and distributed systems*, Vol 13,NO.1, Jan.2002.

[Stojmenovic02b] I. Stojmenovic. "Position based routing in ad hoc networks", *IEEE Communications Magazine*, Vol. 40, No. 7, July 2002, 128-134.

[Stojmenovic04] I. Stojmenovic and J. Wu. "Broadcasting and Activity - Scheduling in Ad Hoc Networks," in *Ad Hoc Networking*, S. Basagni et al., eds., IEEE Press, 2004.

[Tseng02] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu. "The broadcast storm problem in a mobile ad hoc network," *Wireless Networks*, vol. 8, no. 2/3, pp. 153–167, Mar.-May 2002.

[Wang04] X. Wang, J. S. Dong, C. Y. Chin, etc. "Semantic Space: an infrastructure for smart spaces", *IEEE Pervasive Computing*, Volume 3, Issue 3, July-Sept. 2004 Page(s):32 – 39.

[Weis94] R. Weis. "Policies in Network and Systems Management - Formal Definition and Architecture". *Journal of Network and Systems Management*, 2(1): 63-83, Plenum Publishing Corp., March 1994.

[Weis95] R. Weis. "Using a Classification of Management Policies for Policy Specification and Policy Transformation". *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management*, California, USA, May 1995.

[Verissimo06] P. Verissimo, N. Neves, C. Cachin, etc. "Intrusion-tolerant middleware: the road to automatic security", *IEEE Security & Privacy Magazine,* Vol. 4, Issue 4, July-Aug. 2006 Page(s):54 - 62

[Williams02] B. Williams and T. Camp. "Comparison of broadcasting techniques for mobile ad hoc networks," *Proc. of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2002, pp. 194–205.

[WS] Web services homepage at W3C: www.w3.org/2002/ws

[WSDL] Web Service Description Language homepage at W3C: http://www.w3.org/TR/wsdl

[Wu01] H. Wu, C. Qiao, S. De, and O. TonguZ. "Integrated Cellular and Ad hoc Relaying Systems: iCAR", *IEEE Journal on Selected Areas in Communications*, vol. 19, NO. 10, pp2105-2113, October 2001

[XMF] XMF toolkit webpage: http://www.xactium.com

[Yang02] K. Yang, A. Galis, T. Mota and S. Gouveris. "Automated Management of IP Networks through Policy and Mobile agents". Proc. of Fourth International Workshop on Mobile Agents for Telecommunication Applications (MATA2002): 249-258. LNCS-2521, Springer, Barcelona, Spain, October 2002.

[Yang03a] K. Yang, A. Galis. "Policy-driven Mobile Agents for Context-aware Service in Next Generation Networks". *Proc. of the 5th International Workshop on Mobile Agent for Telecommunication Applications* (MATA03). Marrakech, Morocco, October 8-10, 2003. Springer LNCS-2881: 111 - 120, ISSN: 0302-9743.

[Yang03b] K. Yang, A. Galis, J. Serrat, K. Jean, N. Vardalachos, X. Guo. "Network-centric Context-aware Service over Integrated WLAN and GPRS Networks". *Proc. of the 14th IEEE International Symposium on Personal, Indoor and Mobile Radio*

Communications (PIMRC 2003). Beijing, China. 7-10 Sept., 2003. IEEE Press. Pp. 854-858.

[Yang05a] K. Yang, S. Ou, M. Azmoodeh , N. Georgalas. "Policy-based Model-driven Engineering of Pervasive Services and the Associated OSS". *British Telecom (BT) Technical Journal (BTTJ)*, Springer Netherlands, Vol 23, No 3, July 2005. pp. 162-174

[Yang05b] K. Yang, Shumao Ou, Antonio Liotta, Ian Henning. "Composition of Context-aware Services Using Policies and Models". *Proc. of IEEE Globecom 05*, IEEE Press, 28 Nov. - 2 Dec. 2005, St. Louis, USA.

[Yew05] A. Yew, A. Liotta, K. Yang. "Adaptive Content for the Mobile User: A Policy-Based Approach". *Proc. of the 2nd International Workshop on Mobility Aware Technologies and Applications - Service Delivery Platforms for Next Generation Networks: MATA05*. October 17-19, 2005, Montréal, Canada. Springer LNCS-3751: 111 – 120.

[Yu04]T. Yu, K. Lin. "Service selection algorithms for Web services with end-to-end QoS constraints", *Proc. of IEEE Int. Conf. on e-Commerce Technology*, July 2004, page(s): 129- 136.

[Yu05]J. Y. Yu, P. Chong. "A survey of clustering schemes for mobile ad hoc networks", *IEEE Communications Surveys and Tutorials*, Volume 7, Issue 1, 2005 Page(s):32 – 48.

[Zeng04] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, H. Chan; "QoS-aware middleware for Web services composition", *IEEE Transactions on Software Engineering*, Volume 30, Issue 5, May 2004, Page(s):311 – 327.

[Zhu06] F. Zhu, M. Mutka, L. Ni. "A private, secure, and user-centric information exposure model for service discovery protocols", *IEEE Trans. On Mobile Computing,* Vol. 5, Issue 4, April 2006 Page(s):418 – 429.

Appendix A: PoPSiDL Syntax Specification

A.1 PoPSiDL Syntax in Extended BNF

<**PoPSiDL**> ::= **PS** <ServiceName> <ServiceBody> **ENDPS**
<ServiceName> ::= <name>
<ServiceBody> ::= <ControlInfo> < ModuleSet>
<ControlInfo> ::=<PolicyServerInfo> | <otherInfo>*
<PolicyServerInfo>::=<String>
<otherInfo>::=<String>
<ModuleSet> ::=<Module>{<Module>}
<Module> ::=**MODULE** <ModuleName> <ModuleBody>
<ModuleName>::=<name>
<ModuleBody> ::=**BEGIN_M** <PolicyGroup> **END_M**
<PolicyGroup> ::=<Policy>{<Policy>}
<Policy> ::=<PolicyId><colon><PolicyBody><semicolon>
<PolicyId> ::=<name>
<PolicyBody> ::=**IF**<PolicyCondition>**THEN**<PolicyAction><Priority>
<UncertaintyFactor><Comment>
<Priority>::=< RealNum>
<UncertaintyFactor>::=< RealNum>
<PolicyCondition> ::=<Condition>{**AND** <Condition>} | **NULL**
<PolicyAction> ::= <Action> {**AND** <Action>}
<Condition> ::=<ComparisonUnit>{ **OR** < ComparisonUnit>} | (<Condition>)
<ComparisonUnit> ::=<PolicyVariable><ComparisonOperator><Constant><Unit>
|< PolicyVariable>< ComparisonOperator>< PolicyVariable>
|< PolicyVariable> | (<ComparisonUnit>)
<PolicyVariable> ::= <name>
<Unit> ::= <name> | **NULL**
<Constant> ::= <Number> | <String>
<CompareOperator> ::=<EQ>|<NQ>|<LT>|<GT>|<LE>|<GE>
<Action> ::= <BlackboardOperation> | <FileOperation> | <OtherOperation>
<BlackboardOperation> ::=<AssertAction> | <DeleteAction>
<FileOperation> ::= <OpenAction> | <WriteAction> | <ReadAction> | <CloseAction>
<OtherOperation> ::=<DisplayAction> | <ExecuteAction>
<AssertAction> ::=**Assert** (<PolicyVariable> <Comma> <String> <Unit>)
| **Assert** (<PolicyVariable> <Comma> <Expression> <Unit>)
| **Assert** (<PolicyVariable>)
| **Assert** (<PolicyVariable> <Comma> <Var>)
<Var> ::=<name>
<Expression> ::=<Number> | (<Expression>)
| <Expression><Operator><Expression>
| <Funcion>(<Expression>)
<Operator> ::= + | - | * | / | **exp**
<Function> ::= **abs** | **int** | **log**| **sin** | **cos** | **tag** | **ctag**
<DeleteAction> ::= **Delete** PolicyVariable
<OpenAction> ::=**Open** (<FileName><Comma><Mode>)
<FileName> ::=<name>
<Mode> ::= **"r"** | **"w"** | **"a"**
<WirteActin> ::=**Write** (<String>)
<ReadAction> ::= **Read** (<Var><Comma><INTNUM>)
<CloseAction> ::= **CloseFile** (<String>)

<DisplayAction> ::=**Display (** <String><Comma><DisplayMode>
<Comma><DisplayType>**)**
<DisplayMode> ::=**DIALOG | WINDOW**
<DisplayType> ::=**TIP | ALERT | INPUT | OTHER**
<ExecuteAction> ::=**Execute (**<FileName><Comma><Parameter>
<Comma><ShowMode> **)**
<Parameter> ::=<String>
<ShowMode> ::= **SHOW | MINSHOW | MAXSHOW | HIDE**
<Comment> ::=<CommentSymbol><String>
<CommentSymbol> ::= //
<EQ> ::= == <NQ> ::= != <LT> ::= <
<GT> ::= > <LE> ::= <= <GE> ::= >=
<Comma> ::= , <Colon> ::= : <Semicolon> ::= ;
<Number> ::=<IntNum> | <RealNum> | + <Number> | -<Number>
<IntNum> ::=<Digit>{<Digit>}
<RealNum> ::=< IntNum>. < IntNum>
<Digit> ::= **0 | 1| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<name> ::= string beginning with letter
<String> ::="String"

A.2 PoPSiDL Syntax in JavaCC format

```
options {
 STATIC = false;
}

PARSER_BEGIN(PoPSiDLParser)

public class PoPSiDLParser {

 public static void main(String args[]) {
  PoPSiDLParser parser;

System.out.println("\n==========================================");
  System.out.println("Pervasive Service Description Language (PoPSiDL) Parser
Version 0.01");

System.out.println("==========================================");
/*   if (args.length == 0) {
     System.out.println("Reading from standard input . . .");
     parser = new PoPSiDLParser(System.in);
   } else if (args.length == 1) {
     System.out.println("Reading from file " + args[0] + " . . .");
     try {
       parser = new PoPSiDLParser(new java.io.FileInputStream(args[0]));
     } catch (java.io.FileNotFoundException e) {
       System.out.println("File " + args[0] + " not found.");
       return;
     }
   } else {
     System.out.println("Usage is one of:");
     System.out.println("         java PoPSiDLParser < inputfile");
     System.out.println("OR");
     System.out.println("         java PoPSiDLParser inputfile");
     return;
   }
*/
//    while(true) {
        {   try {
            parser = new PoPSiDLParser(System.in);
//          parser.ExpressionList();
            parser.PolicyBlock();
            /* System.out.println("Syntax OK!"); */
           } catch (ParseException e) {
            System.out.println(e.getMessage());
            System.out.println("Encountered errors during parse.");
           }
           try {
                Thread.sleep(100);
           } catch (Exception ee) {}
    }
```

```
  }
}

PARSER_END(PoPSiDLParser)

/* WHITE SPACE */

SKIP :
{
  " "
| "\t"
| "\n"
| "\r"
| "\f"
}

/* RESERVED WORDS AND LITERALS */

TOKEN :
{
  < PS: "PS"|"ps" >
| < ENDPS: "ENDPS"|"endps" >
| < MODULE: "MODULE"|"module" >
| < BEGIN: "BEGIN_M"|"begin_m" >
| < END: "END_M"|"end_m" >
| < IF: "IF"|"if" >
| < THEN: "THEN"|"then" >
| < AND: "AND"|"and" >
| < OR: "OR"|"or" >
}

TOKEN :
{
  < GT: ">" >
| < LT: "<" >
| < EQ: "==" >
| < LE: "<=" >
| < GE: ">=" >
| < NE: "!=" >
}

TOKEN :
{
  < ID: ["a"-"z","A"-"Z","_"] ( ["a"-"z","A"-"Z","_","0"-"9"] )* >
|
  < INT: ( ["0"-"9"] )+ >
|
  < COMPONENT_FUNCTION: ["a"-"z","A"-"Z","_"] ( ["a"-"z","A"-"Z","_","0"-
"9"] )* "." ["a"-"z","A"-"Z","_"] ( ["a"-"z","A"-"Z","_",",","0"-"9"] )* "(" (["a"-
"z","A"-"Z",",","_","0"-"9"])* ")" >
|
  < FLOATING_POINT_LITERAL: (["0"-"9"])+ "." (["0"-"9"])* >
```

```
}


/*************************************************************
 * THE PERVASIVE SERVICE DESCRIPTION LANGUAGE GRAMMAR STARTS
HERE *
 *************************************************************/
/*
void ExpressionList() : {}
{
  { System.out.println("\nPlease type in a PoPSiDL expression followed by a \";\" or
Ctrl-C to quit:");
    System.out.print(">>");}
  ( Expression()
        {
          { System.out.println("\nPlease type in a PoPSiDL expression followed by a
\";\" or Ctrl-C to quit:");
            System.out.print(">>");}
        }
  )*<EOF>
}
*/
void PolicyBlock() : {}
{
  { System.out.println("\nPolicy Block Parsing:"); }
  ( <PS> ProgramName() ProgramBody() <ENDPS>
  )*<EOF>

}

void ProgramName() : {Token t;} {
        (t=<ID>
        { System.out.println("ProgramName:\t"+t.image); })
}

void ProgramBody() : {} {
        ControlInfo() ReasoningInfo()
}

void ControlInfo() : {Token t;}  {
        (t=<ID> | t=<INT> )
                { System.out.println("Control Info:\t" + t.image); }
}

void ReasoningInfo() : {}  {
        ModuleSet()
}

void ModuleSet() : {} {
        Module() [ Module()]
}
```

```
void Module() : {} {
        <MODULE> ModuleName() ModuleBody()
}

void ModuleName() : {Token t;} {
        (t=<ID>
        { System.out.println("ModuleName:\t"+t.image); })
}

void ModuleBody() : {} {
       {System.out.println("\n--------------------Begin a policy Group--------------------");}
        <BEGIN> PolicyGroup() <END>
       {System.out.println  ("--------------------End of a policy Group--------------------");}
}

void PolicyGroup() : {} {
        Policy() [ Policy()]
}

void Policy() : {} {
        PolicyID() "," PolicyBody() ";"
}

void PolicyID() : {Token t;} {
        ( t=<ID> )
     { System.out.println("PolicyID:\t"+t.image); }
}

void PolicyBody() : {} {
        Expression()
}

void Expression() : {}  {
        IFStatement()
}

void IFStatement() : {}  {
        <IF> PolicyCondition() <THEN> PolicyAction() [ Priority()] ["," Uncertainty()]
}

void PolicyCondition() : {} {
        Condition() [<AND> Condition()]
}

void Condition() : {} {
        ComparisonUnit() [<OR> ComparisonUnit()]
}

void ComparisonUnit() : {} {
        Literal()
        ( <GT> {System.out.println("CompareOp:\t>"); }
```

```
        | <LT> {System.out.println("CompareOp:\t<"); }
        | <EQ> {System.out.println("CompareOp:\t=="); }
        | <LE> {System.out.println("CompareOp:\t<="); }
        | <GE> {System.out.println("CompareOp:\t>="); }
        | <NE> {System.out.println("CompareOp:\t!="); }
      )
      Literal()
}

void Literal() : {Token t;} {
      (t=<INT>
          { System.out.println("Constant:\t"+t.image); }
      )|(t=<ID>
      { System.out.println("Variable:\t"+t.image); })
}

void PolicyAction() : {Token t;} {
      ( t=<ID> | t=<INT> | t=<COMPONENT_FUNCTION> )
    { System.out.println("Policy Action:\t"+t.image); }
}

void Priority() : {Token t;} {
      (t=<INT>
          { System.out.println("Priority:\t"+t.image); }
      )
}

void Uncertainty() : {Token t;} {
      (t=<INT> | t=<FLOATING_POINT_LITERAL> )
          { System.out.println("Uncertainty:\t"+t.image); }
}
```

Appendix B: SNAMU Service in PoPSiDL Format

```
PS SNAMU
# Control Information

MODULE UserInfoModule
 BEGIN_M
  UI001: IF User.role == manager THEN securityLevel=10 <1> <1>;
  UI002: IF User.role == consultant THEN securityLevel=5 <1> <1>;
  UI003: IF User.role == employee THEN securityLevel=3 <1> <1>;
  UI004: IF User.role == non-employee THEN securityLevel=0 <1> <1>;
  UI009: IF User.contactAllowedAttribute == TRUE THEN
                    UserDevice.setAvailability(TRUE) <1> <1>;
  ......
 END_M

MODULE PersonActivityModule ...

MODULE BatteryModule
BEGIN_M
 B001: If battery < 20% THEN reduce_resolution (50) <0.9><1>;
 B002: IF battery < 20% AND priority == HIGH THEN execution = 1 <0.95><1>;
 B003: IF execution ==1 AND availableBudget>=50 THEN buySurrogate()
                                                    <0.8><1>;
 B004: IF execution ==1 and availableBudget<50 THEN reduceResolution() <0.9><1>;
 B005: IF lowResolutionAvailable() == FALSE THEN useThirdpartyTranscoding(url,
                                        price, reputation) <0.8><1>;

 B006: IF battery<20% and priority == LOW THEN execution =0 <1><1>;
 B007: IF execution ==0 then informFailure()<1><1>;

 B008: IF battery >=20% THEN continue()<1><1>;
  ......
 END_M

MODULE ServiceModule
BEGIN_M
 S001: IF location ==x THEN searchContact=x <0.8><1>;
  ......
 END_M

MODULE GPRSModule
 BEGIN_M
  GPRS001: IF gPRSConnected == FALSE AND networkAccess == TRUE THEN
                    UserDevice.establishGPRSConnection() <0.8><1>;
  GPRS002: IF tearDownGPRSConnection == TRUE THEN
                    UserDevice.closeGPRSConnection() <1><1>;
  ... ...
 END_M

MODULE WLANModule ...
```

other modules

MODULE CommonService
BEGIN_M
CS001: IF UserDevice.protocol == "ftp" and transmission.destIPaddress == ConsultCompanyAddress and securityLevel >=5 THEN EstablishIPsec <1> <1>;
CS002: IF wLANdeviceAvailable == TRUE AND UserDevice.protocol == "ftp" THEN UserDevice.handOverToWLAN() <0.7> <1>;
CS003: IF IPsecOn == TRUE THEN billingTariff.increaseBy(20) <1> <1>;
...
END_M
ENDPS