國立臺灣大學電機資訊學院資訊工程學系

博士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Doctoral Dissertation

智慧家庭中的情境感知普及服務管理機制

Context-Aware Pervasive Service Management in Smart Home Environments

廖峻鋒

Chun-Feng Liao

指導教授：傅立成 博士

Advisor: Li-Chen Fu, Ph.D.

中華民國 100 年 6 月

June, 2011

國立臺灣大學博士學位論文
口試委員會審定書

智慧家庭中的情境感知普及服務管理機制

Context-Aware Pervasive Service Management in Smart Home Environments

本論文係 廖峻鋒 君（學號D93922006）在國立臺灣大學資訊工程學系完成之博士學位論文，於民國 100 年 6 月 16 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

傅立成

（指導教授）

曾煜棋　　陳良弼

逢愛君　　劉邦鋒

[illegible]　　郭斯彥

唐唯惠　　李蔡彥

系　主　任　呂育道

誌謝

博士學位對我來說不但曾經是一個遙不可及的夢想，攻讀的過程也充滿了艱辛。這樣一個艱鉅的任務，若沒有許多人的指導、支持與協助，是不可能完成的。首先，要特別感謝指導老師傅立成教授在這幾年的耐心指導，讓我在研究及專業知識上得以漸入佳境。其次是擔任博士論文口試委員，同時也是碩士班指導老師的李蔡彥教授，在攻讀學位過程許多重要的基本功是在李老師的指導之下才得以建立基礎，此外，從碩士班畢業後，老師也持續地給予許多鼓勵與關心。感謝林風教授、馮明惠所長、陳良弼教授、郭耀煌教授、曾煜棋教授、蘇豐文教授與逄愛君教授，在百忙之中抽空擔任我的博士論文口試指導委員，並給予許多建議與指導。也感謝許永真教授、朱浩華教授、林風教授與逄愛君教授擔任論文提案審查委員，並在撰寫過程中給予許多意見與協助。感謝台大智慧生活科技整合與創新研究中心劉佩玲主任、陳俊杉副主任與李劍峰博士，讓我有機會將論文中的技術實際應用到真實的智慧生活空間。此外，也要感謝政大資科系的陳恭教授，論文在發展初期老師對於方向給予了許多建議和協助。

同時也要感謝實驗室曾給予鼓勵的許多學長與學弟妹，特別是世勳學長，在口試前還特地抽空遠從高雄來台北幫我進行預演。此外也要感謝博士班學長與同學們，尤其是兆麟、敬互、益銘、士恒、勇成、宇傑，在與你們互動的過程獲益良多。亞文和觀音先後與我一同進行研究，為本論文的主要核心奠立紮實的基礎。此外，宗翰、羽君、慧文、雨喬、茂源及學鴻在論文完成的最後階段協助分擔、處理了許多繁重的專案與研究事務。

最後要感謝數年來始終在背後支持我的家人，尤其是爸爸、媽媽、佳虹、珮如、俊璟及佳虹的家人們。也要謝謝小女兒翊亘，總是帶給家人和我無限的希望與動力。

摘要

智慧家庭主要的願景為透過科技使人們的日常生活更加豐富。此一類型的智慧居住空間具備推測居住者意向並據以提供適切智慧居家服務的能力。大部份的智慧居家服務都由許多高異質性的元件所組成。本論文主要的研究目標即在於設計一組服務管理機制，使得智慧居家服務能夠具備高彈性、強健性、高效能及一致性的特色。

首先，系統的彈性與否大部份取決於其底層之架構型式(Architectural Style)，在比較過相關系統與文獻之後可發現訊息導向中介軟體架構(Message-Oriented Middleware, MOM)是最具備彈性且適合佈署於家庭網路之架構型式。在另一方面，雖然許多文獻都指出強健性為智慧家庭系統不可或缺之一環，但可發現在現存研究中對於加強訊息導向架構系統強健性著墨較少。因此，本論文提出一個兼具彈性與強健性的服務管理架構。本研究以嚴謹的正規程序代數(Process Algebra)的方式定義了一個可支援自主型組合、錯誤偵測及錯誤回復之訊息導向服務模型與其通訊協定，並對此一服務模型與協定進行強健性的正規驗證及實際的回復率與效能測試實驗。

其次，在智慧家庭中，無目錄式服務管理協定(如通用型隨插即用協定)被認為是較為適合管理智慧家庭服務的機制。此類協定大都以 IP 群播加以實現，但經常造成網路擁塞的問題。因此，基於前述之訊息導向服務模型，本研究提出一套可有效降低冗餘封包數量，進而提昇網路效能的機制來改善無目錄式服務管理協定所造成之網路擁塞的問題。經過分析與網路模擬實驗，可發現二者之間具有相當高的一致性，且均具備大幅提昇網路效能的效果。

近年來，有不少研究著重於普及服務的組合議題。在組合普及服務之前，使用者必須先行提出偏好(Preference)，但使用者之偏好不確定性高且可能彼此衝突。由於居家環境經當變動，也很可能造成所啟動之服務之間的相互衝突。為了

解決這些問題，本研究提出一組可同時表達列舉/可數及必要/可商議概念之偏好表示式(Preference Expression)。此一機制配合本研究所發展之一套可驗證的邏輯結合規則(Unification Rules)，可將不一致的使用者偏好表示式整合為一致的表示式。接下來並提出一套以模糊邏輯為基礎的方法，基於環境式情境資訊，評估已啟動服務間相互衝突之嚴重程度。經過實驗可發現，藉由整合上述機制，可同時維持相當高的服務組合之品質及成功率。

最後，本研究整合上述機制進行實作，並實際將多個智慧居家服務佈署於二個不同智慧實驗屋，以驗證所提出之各項機制之可行性。

關鍵字: 普及計算、通用型隨插即用協定、簡單服務管理協定、智慧家庭、服務模型、服務發現架構、IP 群播、服務系統、服務組合、使用者偏好。

ABSTRACT

The concept of Smart Home envisions a technology-enriched living space that is capable of anticipating intensions of occupants and providing appropriate services accordingly. Most of the services in such space are context-aware and are realized by an assemblage of heterogeneous components. The objective of this thesis is to design a suite of service management mechanisms that makes such context-aware services flexible, robust, efficient, and consistent.

The flexibility heavily depends on the underlying architecture style. After a thorough review on existing representative pervasive systems, it is concluded that the Message-Oriented Middleware (MOM) is one of the most flexible architecture styles for the Smart Home. Meanwhile, robustness is one of the key challenges for the Smart Home, but few researches have been done to improve the robustness of Message-Oriented Smart Home systems. Hence, this research work attempts to propose a flexible and robust service management framework by formally defining an MOM-based service application model and protocols that facilitate autonomous composition, failure detection and recovery of services. The proposed approach is evaluated by first proving the reliability property and then conducting experiments on recovery rate as well as performance.

Decentralized service management protocols such as UPnP are believed to be more suitable for Smart Homes. These protocols are usually realized by using IP multicast, which, if not carefully designed, often suffer from network flooding problems. This research proposes several efficiency boosting techniques that reduce the replications of unnecessary messages. The analytical predictions agree well with the simulated and experimental results, which show that the traffic can be greatly reduced by the

proposed approaches.

Pervasive service composition also attracts increasing interests. When composing services, the criteria for scoring and electing services are usually specified by users, which tend to be vague and subjective. Moreover, the deployment of services in smart homes is usually not as well-planned as that in traditional enterprise environments. Hence, the criteria can be contradictory and the activated components can interfere with one another. This thesis addresses these issues by first proposing the Preference Expression that is capable of specifying both enumerative/numeric as well as mandatory/negotiable preferences. Then, a set of unification rules for unifying conflicting preferences is presented. Finally, this thesis proposes a Fuzzy-based approach to estimate the degree of interference based on available context information. By incorporating the above-mentioned mechanisms, an integrated service composition framework is presented. Experiments that evaluate the effectiveness of the proposed framework are also conducted and reported.

Keywords: Pervasive Computing, UPnP, SSDP, Smart Home, Services Models, Services Discovery Architecture, IP Multicast, Service Systems, Service Composition, Feature Interaction, User Preferences.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

In recent years, the rapid emerging of Pervasive and Ubiquitous Computing [138], Context-Aware Computing [104], and Service Computing [147], and Machine Learning [30] have brought the concept of a "Smart Home" into reality. The concept of a "Smart Home" was first proposed officially in 1984 by the American Association of House Builders, which envisions a technology-enriched living environment that anticipates the needs and intensions of occupants and provides services accordingly to promote comfort, convenience, security, entertainment, and therefore an improved quality of life for them [21, 68]. Most of the services in the Smart Home have to be "context-aware" since "contexts" are essential information used to infer needs and intensions of inhabitants. A service is context-aware if it uses contexts, or it adapts to contexts [46], where a "context" is any information that can be used to characterize the situation of an entity which can be a person, place, or objet that is considered relevant to the provision of service [47].

Figure 1.1 depicts the relationship among occupants, the environment, contexts and the context-aware services in a Smart Home. In such environment, contexts are usually inferred from the environmental data gathered by sensors. According to the contexts, applications infer the user situations and then perform appropriate actions, such as to turn on a light or to play a media file. We can observe that the data flow discussed above forms a feedback loop between the environment and the context aware service (see Figure 1.1). In summary, Figure 1.1 reveals that providing a context-aware service in a Smart Home is a four-step procedure: 1) gathering context data from sensors, 2) inferring users' situations based on gathered context data, 3) anticipating users' needs or intensions based on their situations, and 4) determining and performing the most

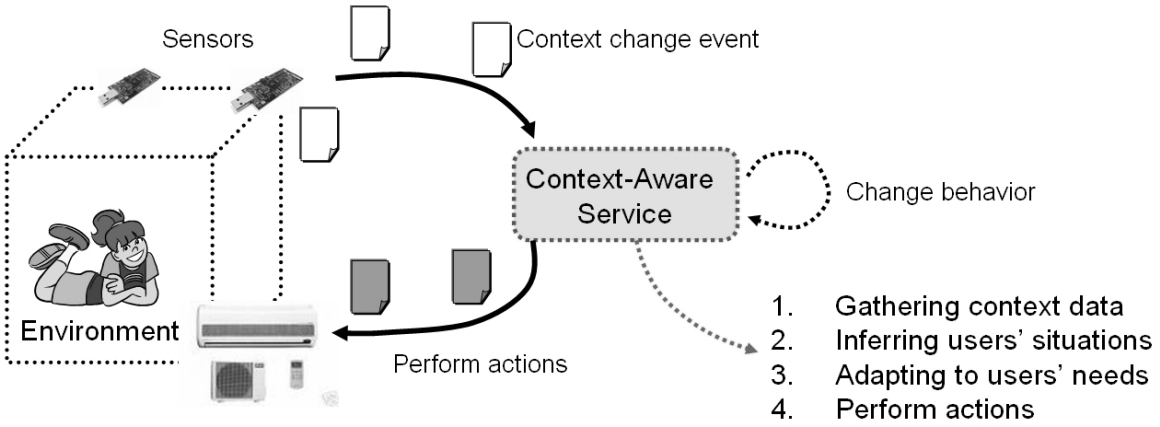


Figure 1.1: Providing a context-aware service in the Smart Home

appropriate actions such as manipulating appliances or displaying information to fulfill user's needs. It is important to observe that, a context-aware service is reactive (event-driven) by nature, since these services react to contexts or situations by performing desired actions.

In addition, heterogeneity is also a key feature of Smart Home services since a service in the Smart Home is realized by an assemblage of heterogeneous *service components* such as wireless sensors, networked appliances, and intelligent agent software that collaboratively offer context-aware habitual supports to residents. Moreover, these service components are usually interconnected by different wired or wireless protocols. Because of the context-awareness and heterogeneity of Smart Home services, a set of service management mechanisms are obviously required which composes services by discovering and selecting service components as well as makes the services work consistent and durable. It is worthy to point out that Bedrouni *et al.* [26] also reported the same observation that among all emerging issues associated with building services in ambient systems, none is more fundamental, challenging, and complex than the need to dynamically ensure adequate management of activities attributed to a large number of heterogeneous entities.

It can be concluded from the above discussions that service management is a core

issue in the Smart Home. However, managing services in such a complicated environment is not a trivial task. The objective of this thesis is therefore to investigate effective and efficient approaches for dealing with challenges of service management in the Smart Home. In Section 1.1, the challenges and users' expectations identified by related literatures are first discussed. Based on these challenges and expectations, this section also motivates the desired qualities, that is, flexibility, robustness, consistency, and efficiency. After that, Section 1.2 presents the contributions of this work and then in Section 1.3 the research scope is identified. Section 1.4 explains the research process of this work. Finally, Section 1.5 introduces the organization of this thesis.

1.1 Research Challenges and Objectives

This section examines challenges of service management in Smart Homes. Based on these challenges, several desired technical qualities of Smart Home service management that serve as the objectives of this research are then discussed.

Contrary to other smart environments such as Smart Offices or Smart Campuses where there are significant efforts in planning in advance, the deployments of services in Smart Homes are usually not well-planned and are upgraded incrementally. Hence, the design of a smart home is not benefited from holistic, ground-up approaches [53]. As a result, a context-aware service management platform is apparently required [17], that is, 1) flexible enough to be modified without affecting other interacting parts, and 2) is able to detect/resolve the service inconsistency arising from conflicting effects of appliances or conflicting user preferences. Moreover, due to the heterogeneous nature of Smart Homes, the platform must be interoperable so that heterogeneous hardware and software, incompatible wiring protocols are able to interoperate with one another [72, 62]. As a result, flexibility, consistency, and interoperability are essential qualities of the Smart Home.

In addition, Edwards *et al.* also observe that robustness is a paramount concern of Smart Home users [53, 62], since most of the domestic technologies are expected to work 24-7. Unlike in other types of smart environments, the Smart Home is in lack of professional system administrator [53]. What is more, the occupants of Smart Homes are usually non-technical users. The persons setting up and maintaining the home services are everyday consumers, with little or no knowledge of networking technologies. Since the consumers would be unable to pinpoint the source of failures [50], the service management platform must be highly reliable and be able to detect and to recover from failures autonomously. Note that the detection and recovery procedures have to be carried out in a minimal amount of time. Otherwise, the failures can lead to a very frustrating user experience and bad marketing perception for the vendors of home services. Consequently, a robust service management platform in the Smart Home has to be self-diagnosable, self-recoverable, and efficient.

It should be concluded, from what has been mentioned above, that a service management platform for a Smart Home with the following qualities is apparently required.

1. **Flexible:** The service management platform must be designed carefully by following an appropriate architectural style so that the platform is flexible enough for incremental deployment and is able to support impromptu interoperability.

2. **Robust:** The Smart Home services have to be available durably. The failures should be detected and recovered as soon as possible.

3. **Consistent:** The service management platform must be able to detect and to resolve conflicts arising from effects of appliances and user preferences.

4. **Efficient:** The proposed mechanisms that realize the qualities mentioned above have to be completed in a minimal amount of time.

1.2 Contributions

As discussed in the previous sub-section, the overall objective is to design a management platform for Smart Home services that are flexible, robust, and consistent. In addition, the proposed platform has to carry out service management mechanism efficiently. In order to meet the objective, this sub-section reports several technical contributions that have been achieved so far which serve as important milestones of this research. The contributions of this work are listed below.

1. **Message-oriented architecture for the Smart Home:** This work suggests that the message-orient architecture, or so-called publish-subscribe architecture, is one of the most appropriate architecture for the service management platform in Smart Homes among existing ones. The rationales behind this suggestion are reported and its superiority over comparable alternatives is also presented.

2. **Verifiable service application model for the Smart Home:** This work proposes a service application model, namely, Pervasive Service Application Model (PerSAM), that specifies the overall logical organization of the Smart Home from the point of view of its use or design. The proposed model is formally presented by using process algebra so that it is verifiable by mathematical proofs. This approach facilitates the analysis of communication complexity (see Item 4 below).

3. **Robust service management protocol for the Smart Home:** Based on PerSAM , this work proposes Pervasive Service Management Protocol (PSMP) which is a service management protocol that realizes autonomous failure diction and recovery by utilizing Universal Plug and Play (UPnP), a well-known home service network standard [15]. PSMP inherits the rigorous nature of PerSAM so that it can be mathematically validated to guarantee the service robustness.

4. **Boosting the efficiency of home service network:** Although the approach mentioned in Item 3 makes Smart Home services more robust, the UPnP-based service management protocol is usually realized by using IP multicast, which, if not carefully designed, often suffers from network flooding problems due to replications of too many unnecessary messages. Hence, boosting techniques that avoid replications of unnecessary messages are proposed here. The analytical predictions agree well with the simulated results, which show great improvements on efficiency.

5. **Service composition algorithms that ensure consistency:** Smart Home services usually have to allow user-in-the-loop service composition which is absent in most of the traditional enterprise service composition mechanisms. More specifically, the criteria for selecting and ranking services are usually specified by users, which tend to be vague and subjective. The criteria can be contradictory and the activated services can interfere with one another. This research addresses these issues by defining the Preference Expression (PE) that is capable of specifying both enumerative/numeric as well as mandatory/negotiable user preferences. A set of unification rules for possible conflicting preferences is also presented. Finally, it is suggested that the degree of interference be modeled and estimated by using a Fuzzy reasoner. A preference-guided and interference-aware service composition framework can therefore be obtained by incorporating the above-mentioned mechanisms. Experiments that evaluate the effectiveness of the proposed approaches are conducted and reported as well.

6. **Development support and rapid prototyping:** To enable the rapid and correct development of home services, a Java-based object-oriented application framework that provides design time supports is devised, which is composed of a set of reusable libraries, interfaces, and default implementations. One of the

salient features of this application framework is that it supports attribute-based programming [36], which implies that the resulting code becomes intuitive and more comprehensible. In addition, this framework provides template-based as well as drag-and-drop code generation services by a set of interactive wizards, which are realized as plug-in modules of the Eclipse IDE.

1.3 Research Scope

To date, Smart Home is still an emerging research field which requires technologies from several related fields such as communications, artificial intelligence, human-computer interfaces, service computing, and pervasive computing. The definition of a Smart Home, or more concretely, the *smartness* of a Smart Home is still a controversial concept. Inspired by Mann *et al.* [97] and Aldrich *et al.* [19], this thesis suggests a vertical architecture that tries to reflect the *smartness* of a Smart Home, as depicted in Figure 1.2.

One of the most essential characteristics of the Smart Home is the deployment of interconnected devices (see Figure 1.2, Level 1) so that they can be controlled and be mediated by computer programs (Level 2) to provide appropriate services to occupants. In order to facilitate more intelligent behaviors, it is necessary to design a management platform that can integrate, configure, and maintain devices as well as programs distributed over the home network (Level 3). Based on the platform, the Smart Home is able to be aware of the environment and occupants by analyzing contexts gathered by sensors (Level 4). Furthermore, by aggregating contexts and by employing machine learning mechanisms, the Smart Home can be aware of situations which are at higher level of abstractions than contexts [92] such as the behaviors of occupants (Level 5). The highest level should be the *Attentive home* which tries to infer deeper and unobservable situation of occupants, for example, emotion or intension of people, and thus

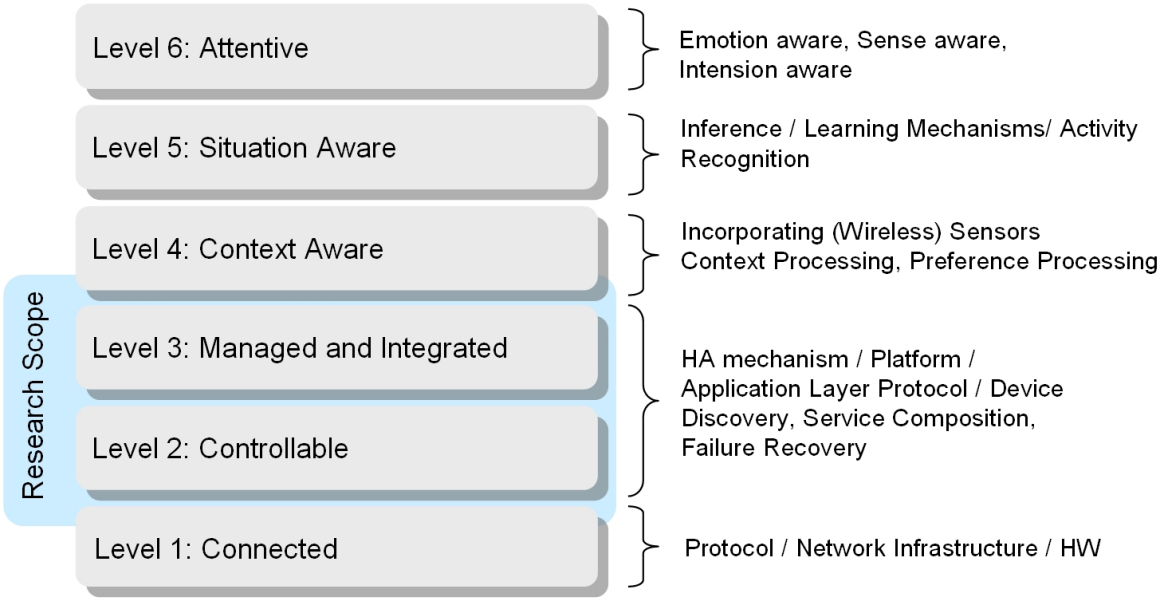


Figure 1.2: The vertical architecture of a smart home and the scope of proposed research

provides service in more attentive ways.

As revealed in Figure 1.2, this research focuses on service management issues. In other words, the issues and challenges posed by this thesis fall into level 2 and level 3. Note that in order to enhance the efficiency, this research also concerns some of the network issues (Level 1). For similar reasons, it is also necessary to take context information (Level 4) into account in order to resolve consistency problems in the Smart Home. To sum up, the primary interests of the proposed research lies in levels 2 and 3, while network and context-awareness issues lying in levels 1 and 4, respectively, are also within the scope of this research work.

1.4 Research Process

As Herbert A. Simon pointed out in the highly influential book *The Sciences of the Artificial* [122], the research field on information and communication technology (ICT) actually falls into the domain of Design Science. The research discipline of Design

Science is obviously different from Natural Science in view of ontology, epistemology, methodology, and axiology. More concretely, the research outcomes of Natural Science refer to a body of knowledge about objects or phenomena in the world that describes and explains how they behave and interact with one another. On the other hand, the goal of Design Science research is to obtain a body of knowledge about artificial (human-made) objects and phenomena designed to meet certain desired goals [135].

In this respect, the artificial object of interest in this work is the Smart Home service management platform which is designed to meet several desired qualities including flexibility, robustness, consistency, and efficiency. The rationales and motivations of setting up these objectives are explained in Section 1.1. The knowledge on how to design service mechanisms to achieve these objectives is therefore presented in the rest of this thesis.

Vaishnavi *et al.* [135] observed that Design Science research has a long history of knowledge building through making. The research process forms a general design cycle [130] which involves the construction of artifacts and the evaluation of artifact performance following the aforementioned construction. It is also important to note from the above discussion that every fragment of the research outcomes is valid only in certain situations [101] which is called the circumscription of research. The Design Science research is usually performed by iteratively proposing new methods that deals with the problems arising from the relaxation of circumscription of the previous research.

Consequently, this research follows the process mentioned above in which assumptions or restrictions are made in the earlier phase of research. After that, some circumscriptions are removed by enhancing the original research outcomes iteratively.

1.5 Organization

The rest of this thesis is organized as follows. Chapter 2 discusses backgrounds and related works, and Chapter 3 introduces the proposed mechanisms that facilitate flexible and robust service management in Smart Homes, namely, PerSAM and PSMP. After that, Chapter 4 proposes several efficiency boosting techniques to reduce the traffic of service management. Then, Chapter 5 concentrates on consistency issues among user preferences. Chapter 6 presents the implementations of the proposed mechanisms. Finally, in Chapter 7, conclusions and future works are provided.

Chapter 2

Background and Related Work

Mark Weiser [138] envisioned that due to the rapid advances of technologies, computing devices will become so small and so cheap that they can be embedded in everyday objects scattered over the living environment. In such an environment, computing devices become *pervasive* or *ubiquitous*. The technologies that facilitate this vision form a new emerging research domain, namely, Pervasive or Ubiquitous Computing. Therefore, the living environments equipped with Pervasive Computing devices is also known as *pervasive environment*, given that a collection of hardware and software components that cooperate to provide services in a pervasive environment is called a *pervasive system*. Pervasive systems are difficult to design and maintain because they involve heterogeneous hardware, software, wiring protocols, and programming paradigms. Moreover, services in pervasive environments are highly dynamic. Many pervasive systems have been proposed to deal with the above problems since the rise of Pervasive Computing. It is important to note that the Smart Home is a pervasive environment so that many existing works with previously developed architectures for pervasive systems are also applicable to a Smart Home. Hence, the purpose of this chapter is to provide backgrounds and the state of the art of the pervasive systems that are closely related to this work.

In this chapter, 11 representative Pervasive systems are investigated in detail (see Table 2.1). Primary issues concerned in these investigations are the contributions and methodologies of these works. For contributions, this work primary focuses on the architectural styles, service management functionalities, and the qualities obtained through the proposed systems, namely, flexibility, reliability, efficiency, and consistency. As for the methodologies, special attentions have been paid to the theoretical

Table 2.1: Sources for state of the art survey of the representative Pervasive systems

Name	Ph.D. Thesis	Journal	Conference	Source code
Context Toolkits	✓	2	3	✓
UPnP	(UPnP specifications)			✓(Reference Implementation)
Gaia OS	✓	2	3	-
Aura	✓	2	4	Partial
CoBra	✓	-	3	✓
SOCAM	✓	-	3	✓
One.World	✓	2	-	✓
Event Heap	✓	1	2	Archive file damaged
LIME	✓	1	2	✓
SOLAR	✓	1	3	✓
MIRES	-	-	1	-

underpinnings and the evaluation methodologies.

Before turning to further discussions, it is helpful to clarify the meanings of several terms used throughout this thesis precisely. A *service* in a pervasive environment is an assemblage of distributed components that collaboratively provides supports to a given user's intention. A service is also referred to as an *application* [15, 46, 114] or *task* [125] in some literatures. For example, a media-follow-me service is capable of choosing the most appropriate displays for playing media files according to user's current location. Under such a service, the user's intention is to listen to music or to watch a movie; the ingredients of a media-follow-me service include control programs, smart floors, all LCD displays and speakers that are capable of adapting to the user's location change cooperatively. Each of these ingredients is called a "service component"

or simply "component" in the sequel. In some pervasive systems [48, 115, 58], there is also a "service manager" for each service that is responsible for managing service components belonging to the service.

Although numerous pervasive systems have been proposed so far, however, most of them do not deal with the service management issues of a system. These works either rely on the existing general-purpose service management mechanisms, or leave these issues un-handled. For example, CoBra (Context Broker Architecture) [39, 40] and SOCAM (Service-Oriented Context-Aware Middleware) [64, 65] leave service management issues to their underlying platforms (i.e. JADE (Java Agent DEvelopment Framework) [27] and OSGi (Open Service Gateway Initiative) [11], respectively. Consequently, this research investigates the pervasive systems from both architecture and service management's points of view.

The following section concentrates on providing a bird's eye view for architectures and service management mechanisms of representative Pervasive systems. Deeper investigations of service management issues are then introduced consecutively in Section 2.2 and 2.3.

2.1 Pervasive Systems

Based on the architectural styles, pervasive systems can be classified into two categories: process-centric and data-centric [141]. In a process-centric system, the distributed components collaborate by invoking sequences of remote procedures. The flows of calls are controlled by software programs, which search services in a centralized service registry and invoke services in a synchronized way. The Context Toolkits [116, 46], UPnP [15], Gaia [115, 114], Aura [58, 125], CoBra (Context Broker Architecture) [39, 40] and SOCAM (Service-Oriented Context-Aware Middleware) [64, 65] fall into this category.

Due to synchronous and centralized nature of the process-centric systems, they suffer from many reliability issues. First, the distributed components of these platforms are usually tightly coupled since they are usually bound to a static network address. Hence, the components must start in a strict order. Moreover, if a service consists of a chained call sequences, all intermediates must be restarted when one of them fails. The failed services are hard to recover because all components must be shut down and then be restarted in a strict order. Finally, the distributed components communicate synchronously. Hence, both service provider and service user both must be ready to communicate at the same time. The caller gets stuck when the callee fails or when it is heavily-loaded.

Recently, more loosely-coupled and asynchronous data-centric architecture such as Tuple Space (TS) [59] and Message-Oriented Middleware (MOM) [24] are proposed. TS is essentially an associative virtual shared memory storing serialized objects. Distributed clients can read, write or take serialized objects from TS server. The Event Heap [76], One.world [61, 63], and LIME/TinyLIME [105, 42] fall into this category whereas SOLAR [87] and MIRES [127] is two of the few Pervasive systems based on MOM.

The succeeding sub-sections present the design and evaluation of the most frequently cited pervasive systems. Service management issues will also be briefly mentioned when these systems are presented, while the details will be discussed in Sections 2.2 and 2.3.

2.1.1 The Context Toolkits (CTK)

The Context Toolkits (CTK) is one of the earliest research works that attempt to propose a general architecture for Pervasive systems. In typical window-based desktop applications, a "widget" is a sub-component of a window such as a button, drop-down

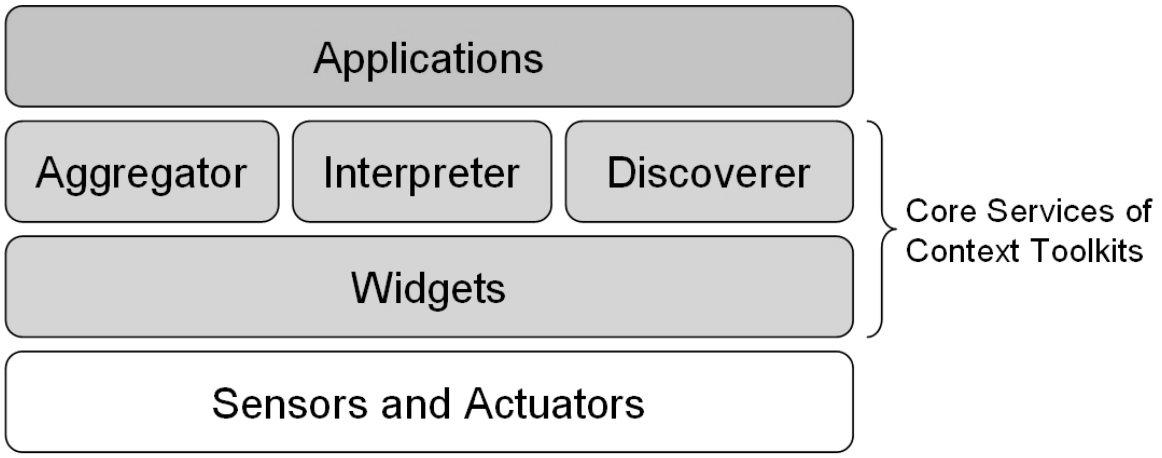


Figure 2.1: The layered architecture of Context Toolkits

list or a text area. Inspired by desktop applications, Dey *et al.* suggest a "widget" abstraction for sensors and actuators in pervasive environments [116, 46].

Figure 2.1 reveals the overall architecture of CTK. For each sensor, there is a component called a "widget" that is responsible for interacting with the sensors physically and then turning raw data into meaningful representations, referred to as "context". In CTK, a piece of context information is represented by a key/value pair that describes the situation of an entity. A service is called an "application" which operates by interacting with remote networked socket servers such as widgets, interpreters, aggregators, and the discoverer. Note that CTK provide three types of core services based on widgets. The aggregator is essentially a context query agent that gathers context data from several widgets according to certain criteria. For instance, a location context aggregator gathers all context information in a specific location. The context interpreter analyzes contexts and then transforms them into higher level contexts. For example, context information provided by location widgets can be used to infer human activities. Finally the discoverer is a directory service by which the applications find appropriate widgets, aggregators, or interpreters as context sources.

According to [46], abstracting sensors by using widgets has two benefits. First,

widgets hide the complexities of interacting with heterogeneous sensors such as floor sensor, pressure sensor, light sensor, and RFID from application developers and provide a uniform way for accessing them. Second, the widgets also become reusable building blocks. The most significant contribution of CTK is the sensor abstraction which separates the tasks of writing application logic from context gathering. This design not only relieves developers from the burdens of dealing with heterogeneous sensors, but also decouples context providers (e.g. sensors, interpreters, or aggregators) from context users (e.g. applications). Also, CTK provides development support with an object-oriented application framework written in Java. Developers of widgets can greatly reduce their efforts by inheriting template classes provided by the framework. Besides, the communication among CTK components rely on XML-based (eXtensible Markup Language) message, making CTK thus potentially interoperable, since CTK provides neither an XML schema nor DTD for defining the formal syntax of messages. In addition, a new component still needs to understand the semantics of the XML-based messages before interacting with CTK components.

The major limitation of CTK is that it lacks of sophisticated service management mechanisms, since it primary deals with context handling and leaves most of service management issues to the developers. More precisely, CTK only provides naïve service discovery mechanisms and leaves these burdens to developers. In real world cases where there are tens or hundreds of components, it is tedious and error prone to maintain the states and life-cycles of these components. More discussions on CTK service discovery can be found in Section 2.2.1.

2.1.2 Universal Plug and Play (UPnP)

The UPnP Device Architecture [15], revealed in Figure 2.2, is a well-known ISO/IEC standard for home network. The service component is called an UPnP Device in an

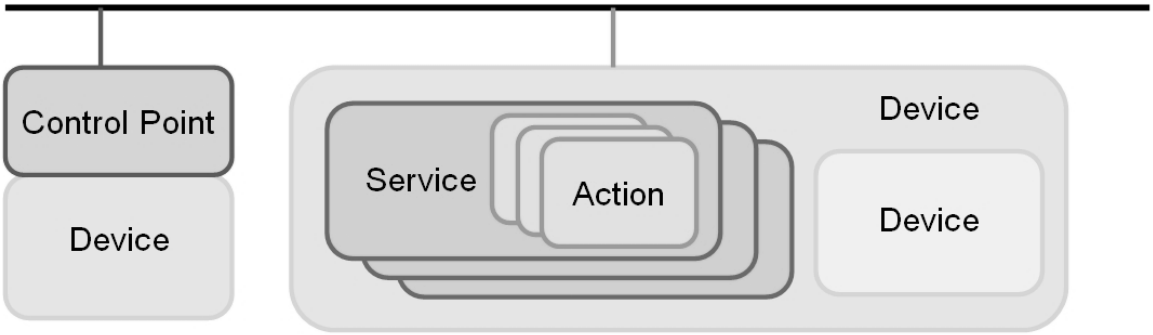


Figure 2.2: The layered architecture of Context Toolkits

UPnP network. Each UPnP Device is composed of a set of "UPnP Services", and each UPnP Service consists of a set of UPnP Actions. It is important to distinguish the term "service"defined above from the UPnP Services mentioned here. An UPnP Service always embedded in an UPnP Device, while a service refers to a collection of components that collaboratively support user's task.

From Remote Procedure Call (RPC)'s point of view, an UPnP Action is identical to a remote procedure which has a method name, parameters, a return value, and is located by an URL (Uniform Resource Locator). The application logic of UPnP is typically controlled by a Control Point which invokes UPnP Actions remotely. A component that plays the role of Control Point can also be an UPnP Device. It is also legal for an UPnP Device to contain another UPnP Device (see Figure 2.2, the bottom right block), which also contains a set of UPnP Services and UPnP Actions.

Despite the absence of sophisticated service management mechanisms, the service operation architecture of UPnP is very similar to that of CTK in the following aspects:

1. Device abstraction: Akin to CTK's widgets, UPnP abstracts sensors, appliances, or software programs by "UPnP Devices".

2. RPC-based: The Control Points of UPnP as well as CTK applications are both responsible for serially invoking remote procedures.

3. HTTP (Hyper Text Transfer Protocol) and XML-based wiring format: The wiring format of UPnP remote invocations is based on SOAP [12], a widely adopted XML-based for remote invocation standard, while CTK uses a proprietary XML-based wiring format. Besides, widgets and UPnP Devices are both implemented by embedding an HTTP server and an HTTP client.

UPnP provides more sophisticated support for service management than CTK. As mentioned earlier, the management of UPnP Devices is carried out by SSDP. Unlike CTK, SSDP is a decentralized protocol that does not require a dedicated discoverer. At first glance, it seems impossible to manage the presence of service components without a centralized broker. However, UPnP overcomes this issue by relying on the underlying network infrastructure. Specifically, UPnP eliminates the need for a centralized broker by using the IP multicast mechanism which is supported by the most low-end switches or home gateway. Hence, from the network layer's perspective, the multicast service provided by the switch becomes the centralized broker which is transparent to the service components in the application layer. Since multicasting is the realization of publish-subscribe style communication in the network layer, it is interesting to note that UPnP adopts the process-centric architecture for service operation and the data-centric architecture for service management, while CTK uses the process-centric architecture for both.

To sum up, the service management of UPnP is superior to CTK in both interoperability and reliability because that the encoding format of UPnP, that is, SOAP, is more widely recognized than the proprietary protocol proposed in CTK and that UPnP does not require a centralized broker.

2.1.3 The Gaia Meta-Operating System (Gaia OS)

The Gaia meta-operating system (Gaia OS) is proposed by Roman *et al.* [115, 114]. Compared with other pervasive systems, Gaia focuses more on large scale pervasive environments (or called Active Spaces in the literature) such as museums, office buildings or campus, where the deployment of a dedicated centralized high-end server is reasonable. Gaia OS takes a monolithic approach and aims to become a pervasive operating system, so that it addresses a wide range of issues such as distributed context file system, distributed event service, security policy, remote invocation, and databases. As a result, the design and implementation is based on CORBA (Common Object Request Broker Architecture) [1], a full scale industrial standard for enterprise distributed systems.

The service operation architecture of Gaia OS is inspired by both CTK and MVC (Model-View-Controller), a programming paradigm widely used in window-based desktop applications [115]. Roman *et al.* also propose a Model-Presentation-Adapter-Controller-Coordinator (MPACC) as standard service operation architecture for Gaia to fit the needs of Active Spaces which is describe in detail in [114]. The overall architecture of MPACC is depicted in Figure 2.3, a service is managed by a Coordinator. The Gaia OS's is responsible for composing a service, that is, to discover and to select most appropriate components for the service according to the predefined application description documents. An Application Generic Description (AGD) prescribes the default preferred types and attributes of a service, while an Application Customized Description (ACD) describes the user preferences. It is worthy to note that ACD is implemented as a script called LuaOrb [37] to facilitate rapid prototyping of services. Each device in the Active Space is controlled by a controller, where the function of a controller is akin to a widget in CTK or an UPnP Device in UPnP. After a controller reads from or writes to a device, the signals are then transformed by Adapters

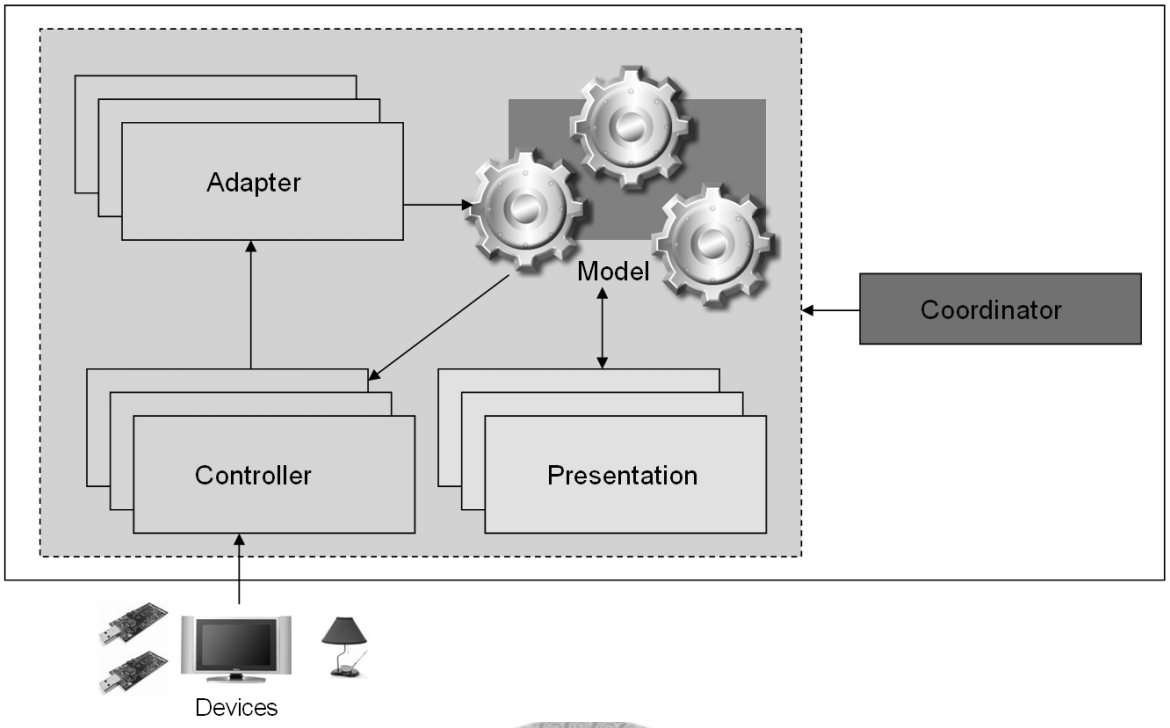


Figure 2.3: The MPACC Service operation architecture in Gaia

into standard format used in Gaia. The actual application logic is embedded in the Model component which determines an appropriate output and then delegates to the Presentation component.

Similar to UPnP Device Architecture, the architecture of Gaia is process-centric. Service management in Gaia relies on CORBA's event notification service. Each service component declares its presence by emitting heartbeats to CORBA's event notification service periodically [145]. However, Gaia considers neither service recovery nor service consistency issues.

The major criticism of Gaia comes form its dependency on CORBA. Although Gaia benefits from CORBA by reusing many standard services such as the directory service, shared repository, and event notification service, Gaia remains tightly coupled with CORBA. As pointed out by Chappel [38] and Henning [69], the design of CORBA standard is deficient and currently it has been regarded as a failed attempt to

standardized distributed systems: 1) CORBA has incomplete interoperability since it standardizes interface but not the wiring format; 2) The cost of implementing CORBA is high since the specification is too complex; 3) Most of CORBA services can not pass through firewalls. As a result, most of the existing CORBA applications have been replaced by XML-based Web Services in recent years. Besides, taking heavy weight and monolithic approach also makes it hard to be compatible with legacy applications, which makes Gaia infeasible in highly dynamic environments such as the Smart Home [76, 145].

2.1.4 The Aura Platform

Aura [58, 125] is a platform that aims to provide distraction-free services to occupants of pervasive environments by utilizing its service migration mechanisms among heterogeneous environments. The Aura platform is constructed on top of Linux kernel and is composed of four main building blocks. Specifically, Satyanarayanan *et al.* [118] proposed Coda and Odyssey, a file system for mobile user that supports ubiquitous file access with application-transparent adaptation; Spectra [56] is a self-tuned remote execution mechanism; Prism [109, 125] is a sophisticated service composition system that predicts and adapts to user's intent inspired by microeconomic model in Economics.

Figure 2.4 shows the overall architecture of Aura. A service is called a "task" in Aura platform, which is managed by a centralized Task Manager, a Context Observer that provides context information, an Environment Manager (EM), and many Suppliers that provide actual support to the task [125, 126]. Aura is process-centric, since the Task Manager is responsible for initiating, negotiating, and monitoring the progress of "tasks" in Aura according to pre-specified user preferences.

The developers of Aura recognized the importance of asynchronous communication between components in pervasive environments [125]. Hence, Aura components use

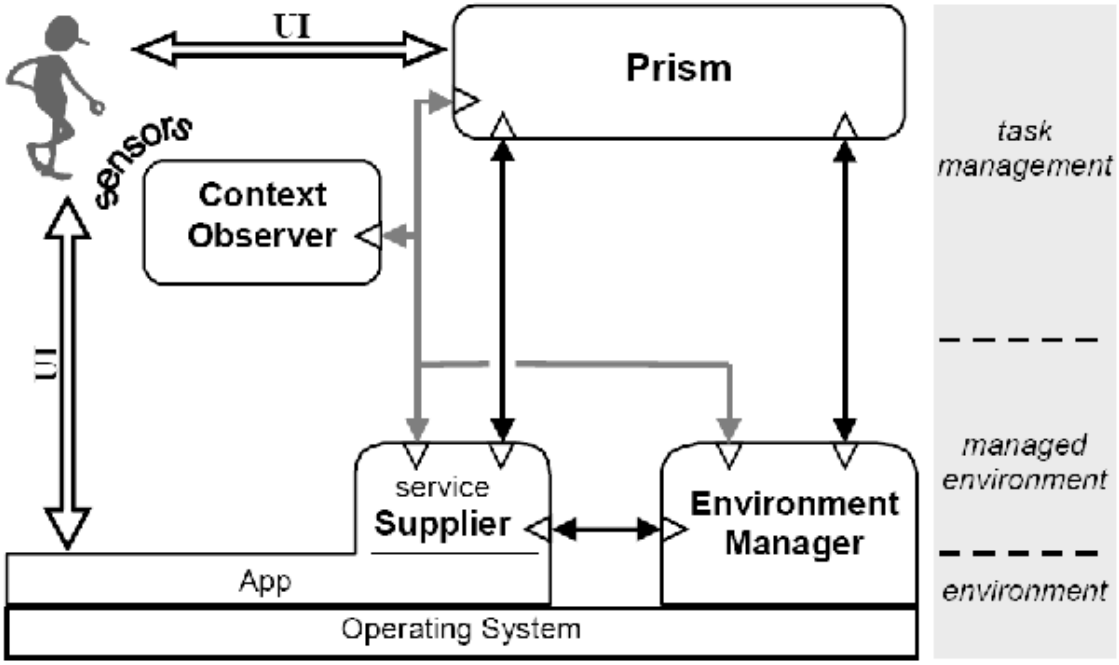


Figure 2.4: Aura's overall architecture (source: [125])

non-blocking sockets to communicate with one another. As pointed out by Eugster *et al.* [55], non-blocking sockets facilitate decoupling in time and synchronization. This feature makes Aura platform more robust than other process-centric systems mentioned in previous sections. However, since point-to-point communication still requires explicit address-binding, the locations of components are still tightly coupled. Similar to CTK, the communication among Aura components also rely on XML-based messages, hence Aura is also "potentially"interoperable. In addition, Aura adopts the asynchronous process-centric architecture for performing service management. The most notable service management mechanism is the utility-based and task-centric service composition [126] which will be elaborated in Section 2.3.

Despite the sophisticated service composition mechanism (Prism), Prism does not deal with inconsistency issues between services (tasks). As for the management about components' presence, Aura relies on Environment Manager as a centralized registry for detecting presences of Suppliers. In [125], the authors claim that current imple-

mentation of Aura can use existing tools such as INS [18] or Jini [20] as its default presence management mechanism. However, as INS only focuses on routing and Jini is tightly coupled by Java, it is not clear that how these mechanisms are applied or customized so that it can fit into the overall architecture. Another limitation of Aura is that although core components such as Environment Manager, Context Observer, and Task Manager are centralized, it does not deal with the single-point-of-failure issues.

2.1.5 CoBra (Context Broker Architecture)

CoBra [39, 40] emphasizes more on context reasoning. At the core of this architecture is a centralized server called Context Broker, which is the mediator of all components in the pervasive system. CoBra is tightly coupled with JADE (Java Agent DEvelopment Framework), a java-based multi agent platform that implements the FIPA (The Foundation for Intelligent Physical Agents) specifications. FIPA [10] is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. As a result, a service is composed of a set of collaborative agents, each of which resides in JADE containers.

CoBra delegates the presence management to the Directory Facilitator (DF) of JADE. Hence, the service management architecture of CoBra is process-centric. However, DF does not guarantee the validity of presence information [27], and the service composition mechanism is absent, as well. It can be concluded from the above discussion that the CoBra services are neither reliable nor user-centric. Like Aura, agent interaction is done by asynchronous peer-to-peer communication which uses ACL (Agent Communication Language) as the wiring format. Another limitation of CoBra is that the design of the Context Broker is purely centralized and lacks of recovery mechanism. Furthermore, all components have to be hosted by JADE (or at least conformed to FIPA specification) in order to access CoBra services, and CoBra does not fully uti-

lize the functionalities of JADE which aim to provide general support for multi agent systems. The library of JADE is complex and not easy to learn, and it is very likely that adopting the JADE platform is an overkill for pervasive environments.

2.1.6 SOCAM (Service-Oriented Context-Aware Middleware)

Similar to CoBra, SOCAM (Service-Oriented Context-Aware Middleware) [64, 65] primarily focuses on context reasoning. The service management issues are left to its underlying platform, namely, OSGi (Open Service Gateway Initiative) [11]. Note that OSGi is an emerging open standard for deploying services to smart home environments. Components deployed in the OSGi platform are called "bundles," and the bundles can be installed, updated, or removed on the fly without having to disrupt the operation of the device. Bundles are libraries or applications that can dynamically discover other services from the service directory or can be used by other bundles.

The OSGi framework is originally designed for a home gateway. However, the OSGi specification does not deal with the nature of distributed systems which is one of the important characteristics of a pervasive environment. As a result, SOCAM services have to be deployed in the same machine. All OSGi services are deployed locally so that the service management can be greatly simplified: the presence of components can be accurately detected by the OSGi ServiceRegistry service and the recovery of components can also be easily realized by utilizing OSGi ServiceTracker service.

Recent progress in the computing power of embedded systems has made it possible to embed the OSGi platform inside intelligent appliances such as the Interactive Television or home entertainment stations. Wu *et al.* [142] proposed a distributed architecture that enables interactions among distributed OSGi platforms, which is one of the baseline technologies of this research.

2.1.7 Tuple Spaces

As mentioned earlier, recently data-centric architecture has been proposed to deal with the flexibility and reliability issues of process-centric architecture. Contrary to the interaction style of process-centric architecture, the components in data-centric architecture usually interact with one another in publish-subscribe mechanism so that data-centric architecture is able to addresses the flexibility and reliability problems by enforcing decoupling in space, time, and synchronization [55].

The core idea of these decoupling techniques is to introduce a centralized mediator for all components in the system such as Tuple Space (TS) or Message-Oriented Middleware (MOM). The introduction of a centralized mediator can cause single-point-of failure problem, but it can be alleviated by deploying a cluster of mediators [59] or by delegating the mediating tasks to the underlying network infrastructure [15]. Among the data-centric pervasive systems, Event Heap [76, 77], One.world [61, 63], and LIME [105, 42] are implemented by using a centralized TS server. A TS sever is a remotely accessible associative virtual shared memory storing serialized objects. Therefore, components can read, write or take serialized objects from TS server.

Event Heap serves as the underlying infrastructure for a larger platform called iROS (Interactive Room Operating System). The service management mechanisms of iROS are carried out by another component called ICrafter [110]. Components announce their presence by a broadcasting mechanism called service bacon. In ICrafter, components describe themselves by SDL (Service Description Language) which is similar to UPnP service descriptions. SDL is capable of describing the type and supports operations of a component but it does not support attributes. ICrafter provides a naive service composition mechanism for iROS applications. Compared with Prism of Gaia, it lacks of advanced features such as attributed-based filtering and conflicts detection and resolution.

One.world proposes a programming model for TS, in which an application (i.e. a service) consists of a set of "scoped event handlers" (i.e. service components). The scope of these event handlers regulates their data access authority in the TS server. However, the application must be written according to specific guidelines and hard to support legacy applications [76]. One.world also proposes a robust presence management mechanism by an renewable centralized discoverer: upon failure of the discoverer, another new discoverer will be elected and initiated. However, it neither deals with service reliability nor service consistency issues. Finally, LIME emphasizes on customizing the TS server for applications in the pervasive environments and do not deal with service and service management issues.

The major problem of TS systems is their scalability and performance. As reported by Johanson [76], it is difficult to scale the TS system to large number of simultaneously communicating entities due to performance issues. Moreover, Grimm [61] reported that, LIME, Event Heap, and One.world all tightly coupled with Java, since TS server stores serialized java objects. Hence, the interoperability of these TS systems is poor.

2.1.8 Message-Oriented Middleware (MOM)

Message-Oriented Middleware (MOM) is an event-based mechanism that enables asynchronous communication and loosely-coupled integration. Hohpe and Woolf [71] point out that when compared with other paradigms, messaging is considered more immediate than file transfer, better encapsulated than shared database, and more flexible than RPC-based invocation. MOM creates a virtual "software bus" for integrating heterogeneous message publishers and subscribers, namely, the "nodes". The logical pathways between nodes are called "topics". Based on this architecture, the system provides services by chaining nodes and topics together. For instance, A, C, D, and F in Figure 3.2 collectively provide an "adaptive air conditioner" service. In this service,

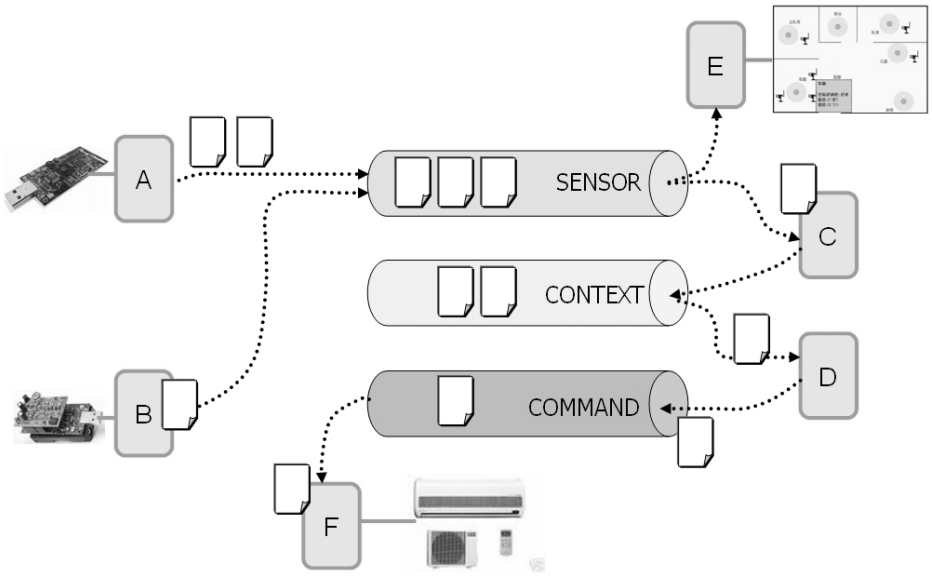


Figure 2.5: A typical message-oriented pervasive system

A is a software adapter of wireless temperature sensors, C is a context interpreter that transforms raw data into high-level context data, D decides the commands to be taken by performing logical reasoning based on the context data, and F is responsible for controlling fans or air-pumps based on messages coming from the COMMAND topic.

MOM has several advantages. First, it comes up with simple and intuitive abstractions of node behaviors. More specifically, all node behaviors can be reduced to three types (to receive messages, to process messages, and to send messages). Second, nodes are easier to "mock" and test. In Figure 3.2, node E can be tested separately without the presence of node A by using a "mock" node that feeds dummy sensor messages. In addition, MOM facilitates "separation of concerns", that is, since each node is isolated by the topics, developers are capable of concentrating only on the logic of the node to be built without worrying about the interferences with other nodes. Finally, due to the loosely-coupled nature of MOM, failures are isolated. In Figure 3.2, if D fails, the failure is isolated by the topics, but either C or F will be aware of the failure.

MOM and TS have similar advantages, that is, easy to integrate heterogeneous

hardware/software as well as failure isolation. However, they are two different architectures from the technology's point of view: TS is a way to access shared information across multiple concurrent clients, whereas MOM focuses on message delivery. More concretely, TS combines the concepts of centralized database and message delivery together. TS can simulate the event-driven feature of MOM; however, it tends to be less efficient as they are generally implemented using a remote accessible shared memory, which uses locks with read/writes to entries. Moreover, TS tends to store serialized objects, which is usually a penalty on performance and interoperability. Since that MOM does not enforce the wiring format of messaging content, the performance of MOM is better than TS. Because of the relief of performance and interoperability issues, MOM appears to be a good alternative that keeps the benefits of data-centric architecture and prevents performance and interoperability issues at the same time.

SOLAR [87] and MIRES [127] are two of the few Pervasive systems based on MOM. SOLAR is an infrastructure for processing context information, the primary application domain of SOLAR is large scale mobile and distributed systems so that P2P techniques such as Distributed Hashtable [23] are used, which is scalable but less efficient. The wiring format of SOLAR is proprietary text-based key-value pairs. It is noteworthy that SOLAR is capable of recover failed service components by restarting the failed ones. However, it does not support service-level recovery. Contrary to Aura, SOLAR deals with failures by reloading components into memory instead of finding an replacement. Meanwhile, MIRES is designed mainly for Wireless Sensor Networks (WSN). However, MIRES focuses on the gathering of contexts, and doesn't address reliability issues or how to compose services by grouping nodes of MOM.

2.2 Pervasive Service Discovery

Service discovery is the process by which an entity on a network (the service manager) is spontaneously notified of the presences of desirable resources (the service components) [52]. The process is usually initiated by issuing a query which contains a set of criteria the desired resources must comply with [136]. Although the idea of service discovery emerges from large scale enterprise systems, it has been extensively used to manage highly dynamic pervasive environments [82]. Typical objectives of service discovery include: 1) getting the locations (e.g. URLs or remote references) of components that meet certain criteria; 2) monitoring the presence or absence of affiliated service components, this is also known as presence management; 3) (optional) trying to recover failed services.

Many service discovery mechanisms have been proposed. They can be classified into three categories: directory-based, non-directory-based, and hybrid [44]. Directory-based systems [20, 3] usually have dedicated registries that maintain information and status of service components, while non-directory-based systems [15, 7] rely on broadcasting or multicasting mechanisms. Some systems support both model mentioned above and are capable to adapt themselves according to the environments [67].

Item 1 mentioned above implies that there is a matching process. To facilitate the matching process, each service component has to be associated with a "capability descriptor" that is to be matched by the "specification". Typically, a capability descriptor contains type and attributes of a service component. In a pervasive environment where heterogeneity is a concern, ontology standards such as OWL-S (Web Ontology Language for Services) [99] are employed to enhance interoperability. Ontology is a set of shared vocabularies used in a specific domain. Sycara et al. [129] suggested an extension of OWL-S for describing the capabilities of a service component. In OWL-S, the capabilities of a service component consist of three parts (see Figure 2.6): 1) Service

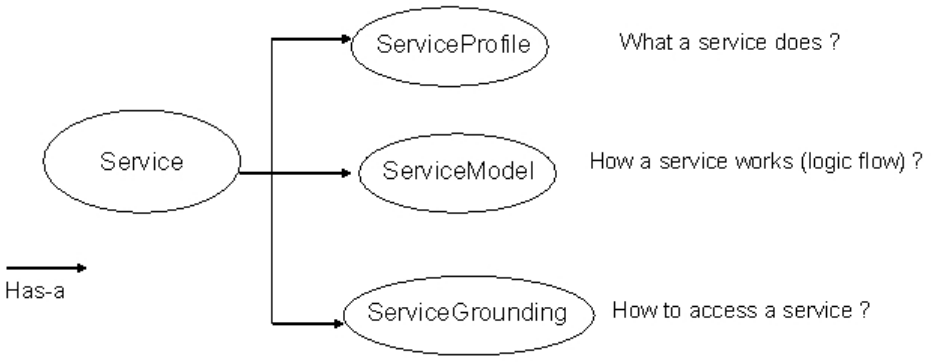


Figure 2.6: Overall structure of OWL-S

Profile that describes what a service component does; 2) Service Model that describe how a service component works (i.e. logic flow); 3) Service Grounding that provides the information of how to access a service. Paolucci et al. further refined the traditional capability descriptor by extending Service Profile of OWL-S, namely, Amigo-S [108]. Capabilities of a service component in Amigo-S are characterized by a type, IOPE (Input, Output, Precondition, and Effect), context parameter, and QoS parameters. In their paper, Paolucci et al. also defined a term "degree of similarity" that is used to estimate the quality of matching between the specification and a capability descriptor.

If there are more than one qualified resource, then more sophisticated mechanisms are required to rank these resources. Moreover, many systems take users' preferences into account so that the specified criteria are so complicated that they have to be processed by a dedicated interpreter. Finally, some other systems require that executing sequences of components to be constrained by a workflow. In this thesis, the mechanisms that resolve the above-mentioned design issues are called service composition which is a stage of service discovery. The taxonomy and the state of the art of service composition are discussed in the next section.

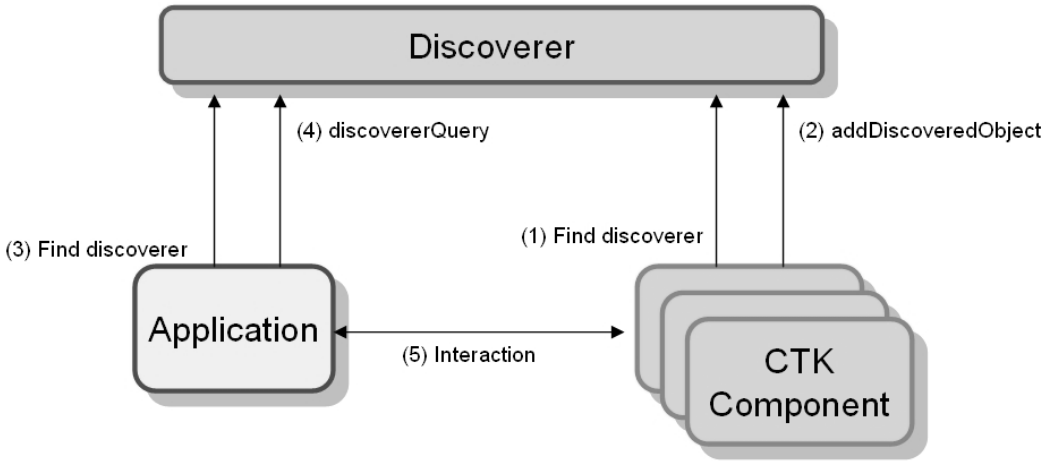


Figure 2.7: CTK service discovery architecture

2.2.1 Service Discovery in CTK

CTK service discovery is designed based on directory-based architecture. The centralized directory is called a Discoverer. After being initialized, a CTK component (e.g. a Widget, an Aggregator, or an Interpreter) searches for a Discoverer by using HTTP-MU (HTTP over UDP Multicast) (Figure 2.7, step 1) and then registers itself to one of the Discoverer (Figure 2.7, step 2). Although CTK allows multiple co-existing Discoverers, each component is only allowed to associate itself with one Discoverer. In CTK, a Discoverer is also a CTK component, so that a Discoverer can also register itself to another Discoverer. Consequently, the network of CTK components looks like a tree-like hierarchy structure (see Figure 2.8).

If there is only one Discoverer in the system, then components can discover one another other by simply querying the Discoverer (Figure 2.7, step 4). Currently, CTK Discoverers support query by ID, component type, and attribute. Finally, the application obtains remote references (IP and port) of the discovered components by which the application can then interact with these components (Figure 2.7, step 5). On the other hand, if there is more than one Discoverer, an application has to traverse the whole tree to obtain all qualified components. Taking Figure 2.8 as an example, the client

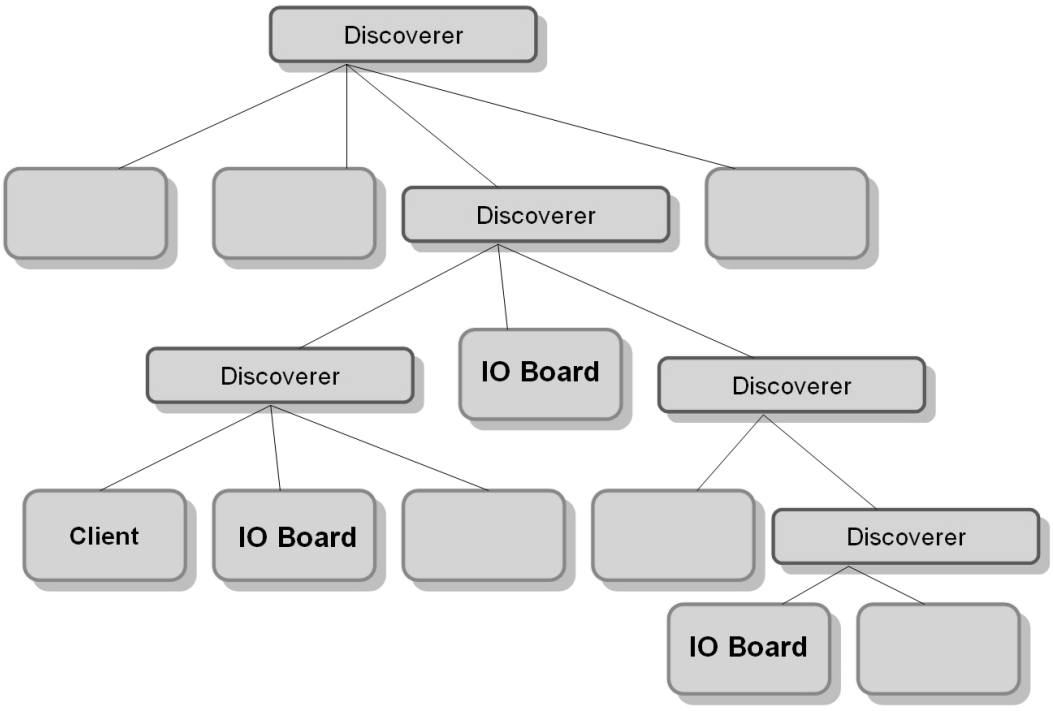


Figure 2.8: The hierarchical structure of CTK Discoverers

can find an IO Board by consulting the local Discoverer; however, exhaustive traversals are needed if the client wishes to find the IO Board that best fits its requirements.

The Discoverer is also responsible for the presence management over components. Components announce the presence and absence by registering and un-registering, respectively. After a component is registered, the Discoverer occasionally pings the component to validate its liveness. Once a component is not responding, the Discoverer notifies the application. But CTK does not deal with recovery tasks.

There are several issues with respect to the design and implementation of service discovery in CTK. First of all, although CTK allows multiple Discoverers, the single point of failure problem of the centralized discoverer is still not addressed since the hierarchical tree-like structure makes the system fragile: If one of the Discoverer crashes, then all registered components become undiscoverable. Second, due to the nature of multicast, when there are multiple co-existing Discoverers, the components do not have

a chance to choose among available Discoverer, in the latest released implementation (December 29, 2003). Hence, the registration holder of newly initiated components is chosen randomly causing the tree to become unbalanced. Third, the need to exhaustively search through the component tree makes the CTK service discovery a time consuming process. More sophisticated mechanisms are required to enhance efficiency of service discovery. Finally, to maintain the availability of an application means to make sure each participating component is available. However, CTK service discovery is supported by a centralized Discoverer which is assumed to reflect accurate presence or absence of components. If a component fails without notifying the discoverer, it is not clear that how long it will take for the Discoverer to recover from inconsistent state, since the pinging mechanisms are not elaborated in [46]. Even if the failures are detected accurately, the CTK-based applications are still unreliable since there is no recovery mechanism for applications.

To sum up, CTK service discovery is designed based on directory-based architecture without addressing single point of failure problem. In presence of multiple Discoverers, the efficiency and reliability of applications is questionable. It can be helpful if more efficient tree-search and tree-reformation algorithms can be proposed to deal with the efficiency and reliability issues.

2.2.2 Service Discovery in GaiaOS

The design of service discovery in Gaia is obviously different from that in CTK except that they both required a centralized directory. In Gaia, the centralized directory is called Service Repository which is realized based on the CORBA Trading Object Service [6]. As mentioned in Section 2.1.3, Gaia is tightly coupled with CORBA. In addition, its service management architecture is data-centric in the sense that its presence management heavily depends on a publish/subscribe communication service

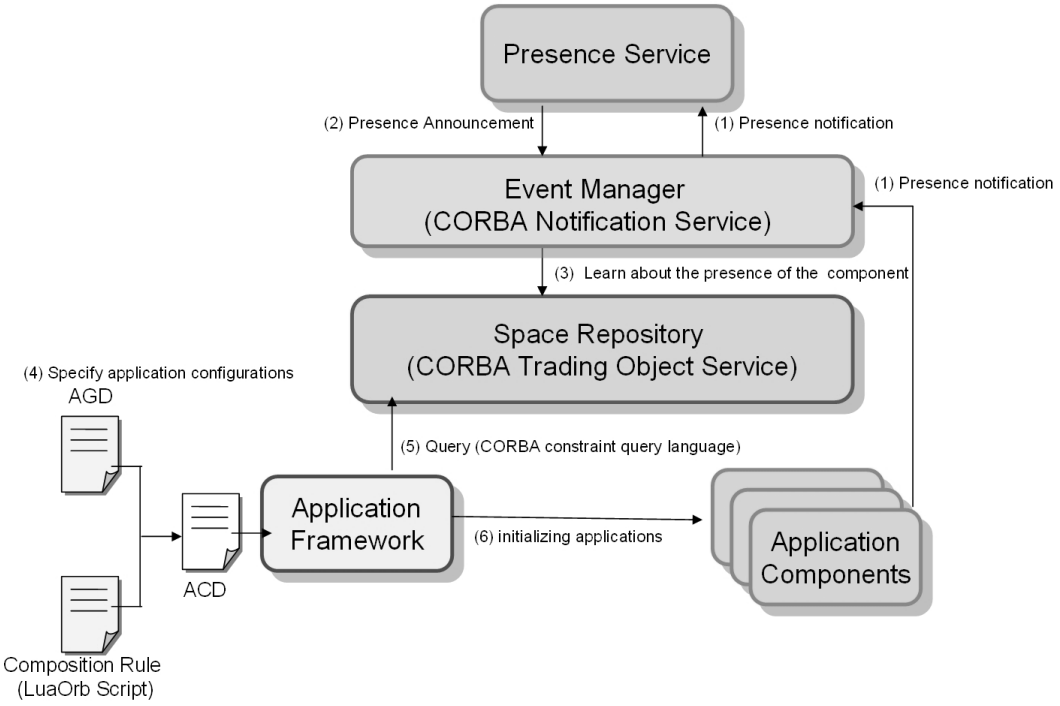


Figure 2.9: Discovering and invoking service components in Gaia

called Event Manager. The Gaia Event Manager is implemented by using CORBA Notification Service.

Figure 2.9 depicts the overall architecture of Gaia service discovery. It is carried out by the coordination among the following components: Event Manager, Service Repository, Presence Service, and Application Framework. Unlike in CTK, Gaia service components publish presence notification messages to Event Manager instead of registering themselves to the directory (i.e. Space Repository) directly. The Presence Service subscribes these notifications, and then these messages are forwarded to the Space Repository. After that, the Space Repository registers service components according to the received presence notifications. One advantage induced from this data-centric (publish-subscribe) approach is that the service components do not need to discover the location of Space Repository. Instead, the presence announcement messages are just sent to Event Manager, which are subscribed by the Presence Service.

The Application Framework is responsible for service composition. First of all, the

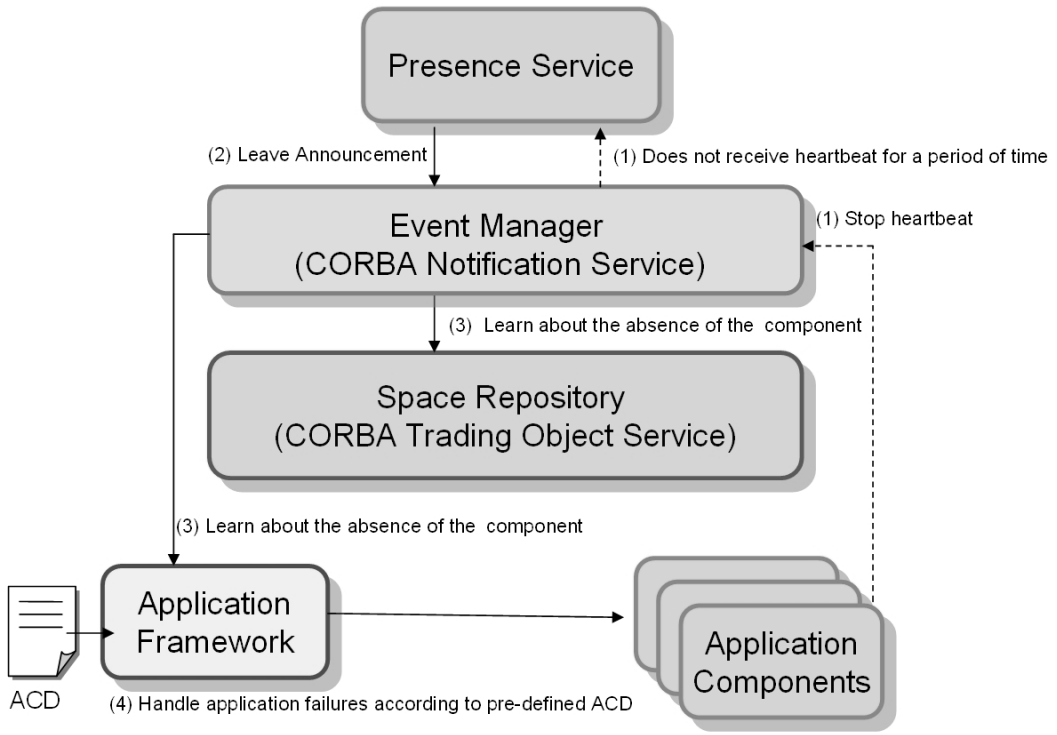


Figure 2.10: Presence management in Gaia

application developer specifies location independent configuration (AGD) as well as the composition rule (written by LuaOrb script). According to AGD and LuaOrb scripts, an ACD is automatically generated by the Application Framework. Then, the Application Framework search for appropriate component for composing the application based on ACD. In current implementation, Application Framework search for appropriate components by CORBA's constraint query language which is defined in [6]. If the Application Framework is capable of finding all qualified components specified in ACD, then it initializes the Coordinator of the application causing the application to be started.

In CTK, the Discoverer keeps track of presence information of components by polling each of them periodically. On the contrary, in Gaia, heartbeats are emitted from components periodically and then they are received by the Presence Service which is responsible for keeping track of the presence information of components. Similarly,

a failed component is identified if it has failed to emit heartbeat message for a certain period of time. After Presence Service is aware of the failed component, it notifies both Space Repository and Application Framework to remove the failed component from registry and to undertake failure handling. The failure handling mechanism is decided by ACD which is generated according to rules written in LuaOrb script (see Figure 2.10). By default, an application is stopped once failed components are identified. However, failure recovery mechanisms can be implemented by overwriting default rule in the LuaOrb script.

In short, Gaia service discovery is more sophisticated then that in CTK, since dedicated services such as Presence Service and Application Framework are developed for monitoring components and for handling failures. Nevertheless, the burdens of recovering failed application are still left to application developers. The architecture of Gaia is process-centric, where each component is tightly coupled, causing Gaia applications to become fragile. Hence, it is hard to design a generic failure recovery mechanism for Gaia application since the inter-dependencies among components can be very complex. In addition, these services are also possible point of failure, which also lack of recovery mechanisms. Finally, Gaia uses Event Manager Service (TCP-based) as its communication mechanism for service management instead of IP Multicast (UDP-based). However, as pointed out by Tran *et al.* [133], most packets used for service management in Pervasive systems are with short lengths, and the relevant sessions are not kept for a long time. In their experiments, even with reliable UDP, the system is still 4 times faster than with TCP.

2.2.3 CoBra/JADE Service Discovery

This sub-section elaborates service discovery mechanisms used in CoBra (see Section 2.1.5). Actually, CoBra itself does not deal with service management directly, and all

these tasks are delegated to its underlying platform, namely, JADE, an implementation of FIPA (The Foundation for Intelligent Physical Agents) specifications [10]. In FIPA, there are three specifications that concentrate on service discovery issues: Agent Management, Agent Discovery Service, and JXTA [60] Discovery Middleware. Among these specifications, only Agent Management is standardized [13], whereas the other two specifications are currently pending in preliminary state. Therefore, only Agent Management Service (AMS) are implemented in the current release of JADE platform.

As mentioned earlier, in CoBra each service component is implemented as a JADE Agent. Figure 2.11 illustrates the overall architecture of JADE. In each machine, there is a Container that manages local Agents. One of the Containers is chosen as the Main Container, in which the Directory Facilitator (the directory of JADE service discovery) and the AMS reside. The locations of Directory Facilitator and AMS are determined and can not be changed afterwards as soon as the system is initialized. Directory Facilitator provides yellow page service for Agents, while AMS is responsible for creating and suspending and deleting Agents. Note that according to FIPA specification, AMS is mandatory whereas Directory Facilitator is optional, so that AMS also plays the role of Agent directory when Directory Facilitator is absent. Context Broker, the core service of CoBra that serves as facade for acquiring context information, can be deployed in an arbitrary Container and then be found through Directory Facilitator.

To certain extent, JADE service discovery is less flexible, since the locations of Directory Facilitator and AMS are fixed and are known by all Agents. After an Agent is started, it registers its Agent Description and Service Description to the Directory Facilitator with a lease time. It is the Agent's responsibility to renew the lease time before it expires. After expiration of the lease time, the Agent is considered failed. Agents in JADE are addressed by AID (Agent Identifier). Before using a service

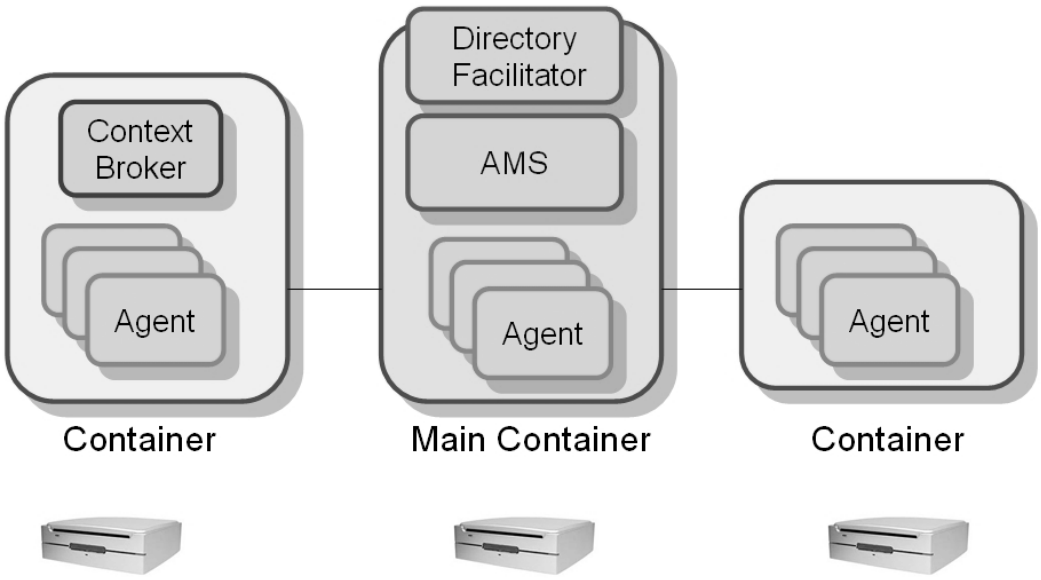


Figure 2.11: Overall architecture of CoBra/JADE

provided by an Agent, the client, which is also an Agent, first searches for the Agents that provide such service in the Directory Facilitator. Directory Facilitator returns the AIDs of matching Agents to the client, based on which it can then consume the service.

Apparently, the Main Container, in which Directory Facilitator and AMS resides, is the single point of failure. JADE reduces the possibility of failure by providing a replication service for Main Container [27]. As for the Context Broker, Chen *et al.* [40] suggests a persistent team based approach, which is inspired by Adaptive Agent Architecture [88], to prevent the single point of failure. However, since there is no evaluation on the proposed approach, it is not clear that this approach is cost-effective, namely, to what extent can the system be recovered, and to what extent does this approach affect the overall performance of the system.

In summary, CoBra/JADE service discovery is designed based on directory-based architecture with a replication service to alleviate the single point of failure problem. However, the synchronization of data among replicated Main Containers can cause flooding of additional network traffic. In addition, the locations of core services includ-

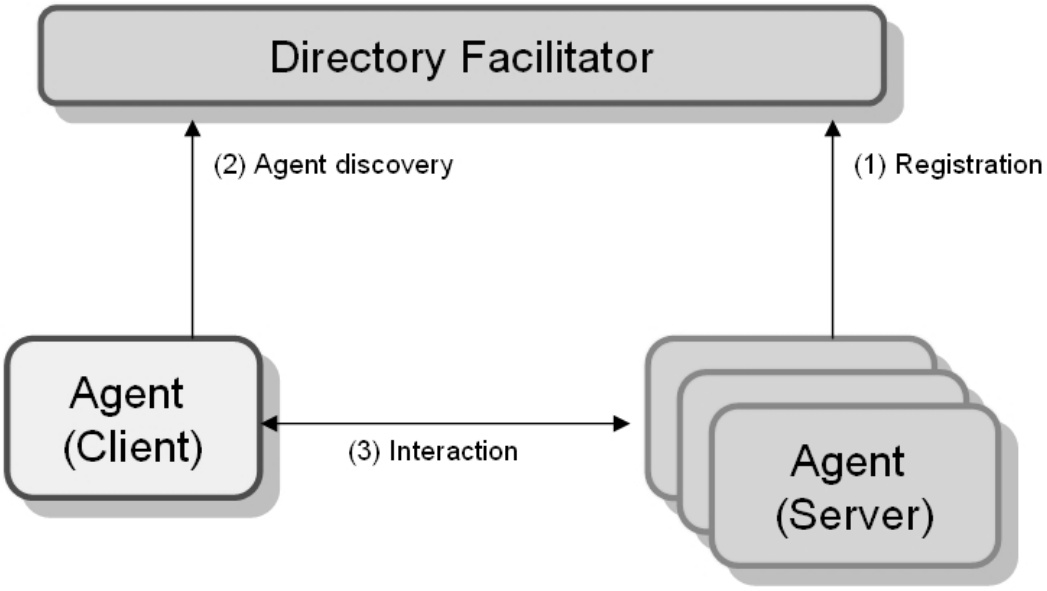


Figure 2.12: JADE service discovery architecture

ing AMS and Directory Facilitator are fixed. After theses services are re-located (i.e. the Main Container crashes and is replaced by a replicated one), the burdens of detecting new locations of these services as well as re-binding to these services are left to developers. CoBra focuses on context processing and provides only naive mechanisms for service discovery. Specifically, it lacks of supports for selecting and composing services according to users' needs. Finally, the whole JADE platform is Java-based and process-centric, and it is questionable that whether the legacy services can interact with CorBa/JADE Agents.

2.2.4 Aura/Jini

Aura does not propose dedicated service discovery mechanism. Instead, in [125], the author suggests that Jini [20] should be used as Aura's service discovery mechanism. Therefore, this sub-section examines Jini service discovery mechanism in detail.

Jini is designed based on directory-based architecture. The centralized directory is called a Lookup Service. After being initialized, a service component locates the

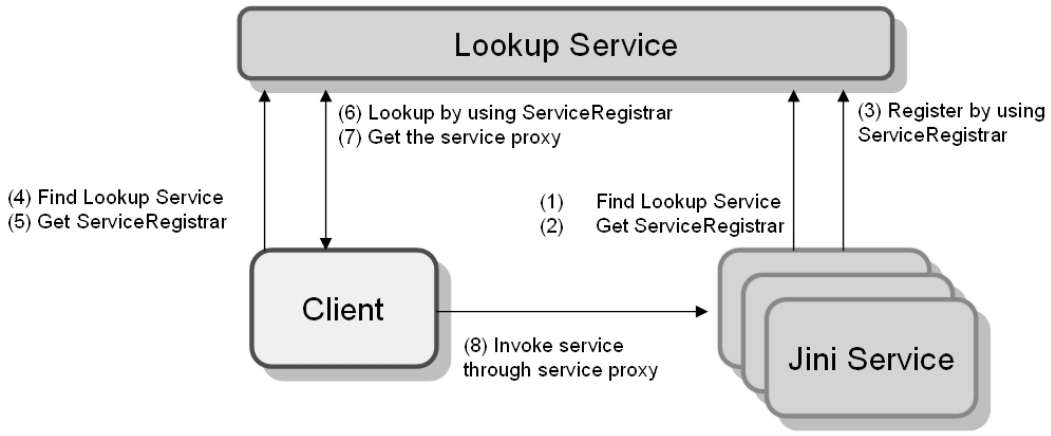


Figure 2.13: Jini service discovery architecture

Lookup Service either by emitting a UDP multicast request or by waiting for a multicast announcement from the existing Lookup Services. If the multicast service is not available (e.g. WAN), the location of the Lookup Service has to be known in advance so that Jini service components can contact the Lookup Service directly.

After a Lookup Service is located, the service component acquires a serialized Java object called a ServiceRegistrar from the Lookup Service. After that, the service registers itself via the ServiceRegistrar. Similarly, a client component discovers the Lookup Service by using the same way with service component mentioned above. The ServiceRegistrar is now used to invoke the lookup operation of Lookup Service. If the desired service is found, a proxy object associated with the service is downloaded. Finally, the client invokes the service through the downloaded proxy object.

From the architecture's point of view, Jini is identical to that of CTK and Gaia. However, the most significant difference is that Jini is tightly coupled with the Java RMI (Remote Method Invocation) technology. More concretely, instead of registering explicit remote references (e.g. IP and port), Jini service components upload a serialized Java proxy object to the directory, by which clients can invoke them. While this mechanism makes Jini independent of specific network protocol, however, all service

components in the system have to be implemented by using the Java technology.

Jini's presence management is similar to Gaia (Section 2.2.2) and CoBra (Section 2.2.3) which depend on the lease-based timeout techniques to detect the presence of components. A component is considered as failed if it fails to renew the lease within a certain period. Note that Jini does not support failure recovery. In Aura, components that constitute a service are monitored by the service composition engine, namely, the Prism service composer. A service is re-composed by Prism if a failed node belonging to the service is detected.

The major issues of Jini/Aura service discovery is two folds. First of all, Jini is tightly coupled with Java RMI, in which the interactions among components are synchronous. However, one of the important contributions of Aura is its asynchronous point-to-point communication mechanism. Hence, there is an obvious paradigm mismatch. The designers of Aura do not address these issues in the literature. As a result, it is not clear that how Jini is fitted into the overall architecture of Aura. Second, the adaptation of Jini as the underlying service discovery technology forces Aura tightly coupled with Java and process-centric service management, causing poor flexibility and interoperability.

2.2.5 Service Discovery in One.world

Similar to other directory-based service discovery mechanisms, the service discovery of One.world also relies on a centralized directory server. However, the design of One.world service discovery is very similar to CTK, Gaia, and Aura except that the directory server in One.world is electable. The election is performed based on the devices suitability to be the directory, that is, memory size and devices' uptime. Therefore, in addition to service components, and Directory Servers, there is also an Election Manager that is responsible for electing and initiating a new directory server aggressively.

More concretely, the Election Manager monitors presence announcement, sent by UDP multicast, emitted from Directory Servers, a new Directory Server will be elected when one of the existing Directory Server fails to perform presence announcement for successively two time periods or when one of the clients receives a malformed message. The core idea behind this design is to make sure there are always more than one Directory Servers in the system. The contents of a newly started directory server can be inconsistent with the system. Hence, the service components are assumed to cache most recent information and then forward to the newly started directory servers.

Although the design of electable directory servers increases the reliability of service management, One.world does not deal with the reliability of services or service components. Moreover, the aggressive approach can greatly increases the load of devices and network. This approach even causes the deployment of One.world runtime impossible for some resource constrained devices such as wireless sensor node. After all, it is unreasonable to assume that every device is capable of becoming a directory server.

2.2.6 Bluetooth SDP (Bluetooth's Service Discovery Protocol)

In the following two sub-sections, two non-directory-based service discovery mechanisms, that is, Bluetooth SDP and UPnP are introduced. In contrast to the above-mentioned directory-based approaches, Bluetooth SDP and UPnP focus more on residential applications and smaller networks.

Bluetooth is a short-range RF-based communication technology [52]. Different from other service discovery systems mentioned above, Bluetooth is realized based on non-IP network so that it can achieve robustness, low power consumption, and low cost. Bluetooth SDP (Bluetooth's Service Discovery Protocol) is an optional profile of Bluetooth core specification [7], which is tightly couple with L2CAP (Logical link

control and adaptation protocol). L2CAP is the base of many higher-level Bluetooth protocols, which hides the complexity of RF-based communication. Bluetooth devices form a small group called a "piconet" by coordinating nearby devices. Each piconet can only consist of up to 7 devices, and one of these devices is the master. The master can be a slave of upper level group called "scatternets". Hence, the entire Bluetooth forms a hierarchy network. Bluetooth is based on non-directory-based architecture.

The major limitation of Bluetooth SDP is it's tightly coupling with a specific protocol stack. Hence, it is not easy to interoperate with non-Bluetooth devices. Second, due to the low bandwidth, only 128-bits UUID-based search is allowed. The data structures used to represent attributes become more complex than competing service discovery protocols. The burden of processing these complex data structures is left to application developers. Third, the search scope of Bluetooth SDP is limited by physical distance which is usually less than ten meters. Finally, Bluetooth SDP is designed for Personal Area Network which typically contains less than ten devices. If it is used in the scale of a smart home, the monitoring of components can become inefficient and less accurate.

2.2.7 Simple Service Discovery Protocol (SSDP)

As mentioned in Section 2.1.2, UPnP is a well-known standard for home network, which composes of three HTTP-based sub-protocols: SDDP, GENA (General Event Notification Architecture), and SOAP. Among these protocols, SSDP takes charge of service discovery in an UPnP network.

By default, SSDP operates based on HTTPMU (HTTP over UDP multicast). Multicast is an IP-layer mechanism of forwarding IP datagrams to a group of interested receivers via a set of predefined addresses, which is supported by most network switching equipments. By default, SSDP uses the address 239.255.255.250:1900. Therefore,

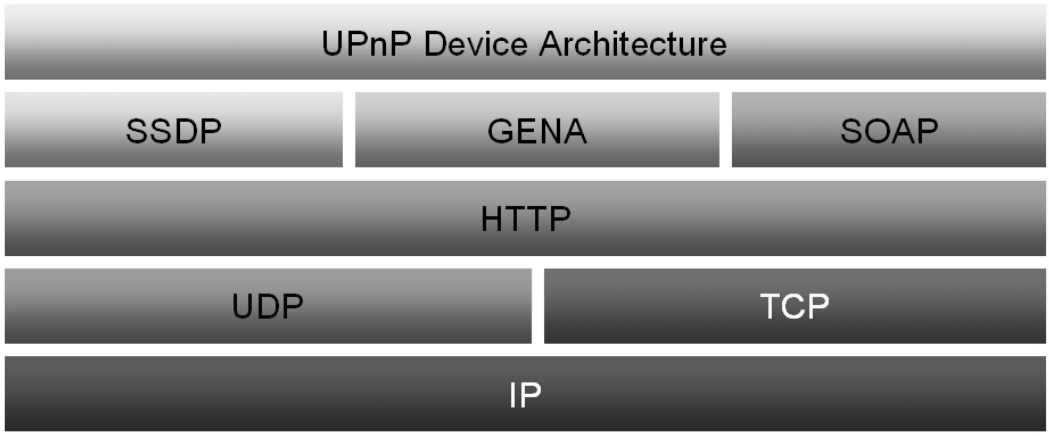


Figure 2.14: The protocol stack of UPnP

SSDP does not need a centralized directory. SSDP is a simple yet effective service discovery protocol. It extends HTTP with two message types: Notify and M-Search. Notify and M-SEARCH messages are used to encapsulate SSDP actions. There are three kinds of SSDP primitive actions: 1) ssdp:alive: announces the presence of a component by using a HTTP Notify message, 2) ssdp:byebye: announces that a component has left the network by using a HTTP Notify, and 3) ssdp:discover: finds a component that meets certain type specified in the ST (Search Target, see Listing 2.1) header in an M-SEARCH message. The matched component then replies by sending back an HTTP Response message (see Listing 2.1, lower part). It is noteworthy that it is possible that a device fails without sending a "ssdp:byebye" message. Therefore when issuing a "ssdp:alive" or a response message, the device attaches information of valid time period by using a HTTP "Cache-Control" header (see the bottom part of Listing 2.1). After this time period, the presence announcement becomes invalid.

SSDP/UPnP has the following benefits. First, it is an ISO standard. Second, it is one of the few dynamic service discovery protocols that do not need a dedicated and centralized service registry [148], which is more feasible in smart home environments. Furthermore, SSDP/UPnP is platform and language independent, as it is based on

Listing 2.1: An "ssdp:discover" request and its response

```
M-SEARCH * HTTP/1.1
ST: urn:schemas-upnp-org:device:sensor:1
MX: 3
MAN: "ssdp:discover"
HOST: 239.255.255.250:1900

HTTP/1.1 200 OK
Content-Type: text/html; charset="utf-8"
Server: Windows XP/5.1 UPnP/1.0 CyberLink/1.7
Content-Length: 0
Cache-Control: max-age=1800
EXT:
Date: Sun, 08 June 2008 13:35:12 GMT
ST: urn:schemas-upnp-org:device:sensor:1
USN: uuid:94b7fab5-52df-4222-b2f1-d5573e74859a::
     urn:schemas-upnp-org:device:TarokoSensorGateway:1
Location: http://192.168.4.102:4040/description.xml
MYNAME: Taroko Sensor Gateway
```

SOAP/HTTP protocol. Despite these advantages, there are still many issues in the original design of UPnP/SSDP. First of all, despite of high flexibility and interoperability, SSDP does not take care of both service composition and recovery. Second, service representation in SSDP is naïve. Specifically, service components can only be characterized by ID and type, while most of other service discovery mechanisms support at least ID, type, and attribute. Notice that in Smart Homes, services components with the same type do not mean they are substitutable. For example, the information obtained from sensors in living room is different from the information in the bed room. Therefore, supporting only type-based representation prevents UPnP network from applying sophisticated service composition mechanisms. Third, SSDP relies on UDP, which is unreliable. UDP is very likely to lose packets thus causing SSDP to become invalid in a busy network. Hence, the UPnP specification suggests broadcasting SSDP messages repeatedly for 2 to 3 times. Unfortunately, this approach tends to

make network traffic heavier. The situation is getting worse if we increase heartbeat rate to achieve higher availability. Finally, SSDP suffers from efficiency problems: the broadcasting nature of UDP tends to flood the network with unnecessary packets. We need more sophisticated mechanisms to save the bandwidth.

2.3 Pervasive Service Composition

Traditionally, service composition has been an integral part of service discovery. The major purpose of service composition is to select, to rank, and to assemble qualified service components according to a pre-specified service specification. Service composition is one of the most active research issues in Service Computing [147]. In the last few years, a considerable number of studies have been made on designing service composition mechanisms, but most of them focus on enterprise environments [103]. Many of these works [123, 132] aggregate service components according to workflow-based service template language such as Business Process Execution Language (BPEL) [79]. Researches using workflow-based approach focus on service units with predefined profiles. Other researchers propose to compose services by planning techniques [146, 83, 91], and they who suggest that the time of selection of components should be considered dynamically during runtime instead of in design time. Notably they emphasize on finding global optimal execution paths among numerous tasks and their subtasks. Generally speaking, planning techniques are more suitable when the structure of services can be divided into many sub-tasks hierarchically. In an enterprise environment, services are usually composed based on pre-defined business policies and are relatively well-planned. On the contrary, pervasive service compositions are often user-dependent and ad hoc.

Although numerous attempts have been made in studying service composition issues in enterprise environments; however, according to a recent study conducted by

Bronsted *et al.* [33], there are surprisingly few researches have been done on this issue for pervasive environments. After performing a thorough survey, Bronsted *et al.* noticed that few existing service composition solutions are feasible in pervasive environment. They also observed that there are some problems that made service composition in pervasive environments more challenging than that in traditional enterprise environments due to the following reasons: 1) composing services under changing contexts, 2) managing service contingencies, 3) device heterogeneity, and 4) taking user preferences into consideration. Among these issues, little research has been made on the last two problems, namely, the inconsistency problems among conflicting user preferences and service effects (the effects of services interfere with one another). They will be discussed further in the following sections.

2.3.1 Unifying Inconsistent User Preferences

The goal of service compositions in pervasive environment is to meet maximum satisfaction of users. Before composition starts, users specify their needs or preferences to the system via user interfaces. There are several methods proposed to model user preferences in intelligent information systems such as [90] and [96]. However, these approaches either do not provide a complete formal framework for users to represent their preferences or does not work when multiple conflictive preferences are present. Furthermore, complexity arises when multiple parties submit conflicting preferences simultaneously. For example, the preferences of energy saving policies and the user preferences are very likely to be conflicting.

Most of the existing pervasive service composition mechanisms only focus on capturing user preferences in either direct [110] or indirect ways [126, 111] to help to obtain optimal outcomes. For example, ICrafter [110] is a user-mediated service framework for Event Heap (see Section 2.1.7). In ICrafter, the term "service" is used in a narrow

sense, which refers to a user interface (UI) plus an appliance. An appliance can be a computer program or a real world device. At the core of ICrafter is a well-known service (e.g. every component knows its location) called the Interface Manager (IM) which generates UI for the users to operate appliances according to the contexts on the fly. Strictly speaking, ICrafter does not address most of service discovery issues such as presence management, service recovery and service locating.

Existing service composition mechanisms deal with conflicting policies problems by either explicitly defining the precedence among policies or by attempting to seek a common ground among conflicting policies using logic-based approach. Olympus [111] falls into the first category, which takes charge of service composition tasks for Gaia. In Olympus, the user preferences are specified by using LuaOrb-based policy files. According to AGD and the policy files, the Gaia service composition framework then generates ACD automatically. The conflicting policies are resolved by explicitly assigning a master policy for each criterion. The ranking of candidate service components are determined by heuristic utility functions predefined by users. On the other hand, Bettini [29] proposes a first-order-logic based profiling system for mobile systems. They discuss cases of conflicting profiles and offer their resolution strategies. Shankar et al. [119] proposes an Event-Condition-Action-Post-Condition (ECA-P) policy model. This work detects conflicts among policies by analyzing their semantic post-conditions and replaces conflicting ones with the one with preferred post-conditions. Due to limited expressiveness of ECA-P model, the results are decisive: users either come to a common agreement or the service is not provided at all. However, in real cases, users tend to change their minds and are usually negotiable. Obviously, more powerful representation techniques have to be developed in order to capture the negotiable user preferences.

2.3.2 Dealing with Inconsistent Service Effects

Services that work perfectly when they are isolated do not guarantee that they can still work perfectly when several services co-exist in the same environment. Usually, compatibility issues arise due to resource competing and interferences among different effects of services. This issue is traditionally referred to as the "Feature Interaction Problems (FIP)" [86], which is first observed in 1980s in telecommunications systems. FIP refers to some unexpected side-effects caused by interactions between or within services. It is important to note that from the users' point of view, not all interactions are unacceptable. Therefore, the pervasive service composition mechanisms should be able to distinguish acceptable interactions from undesired ones according to user preferences.

Earlier research that set their theme on telecommunications systems focuses on resolution after interferences occur rather than avoiding them in advance. The resolution processes usually involve rolling back the whole transaction of business calls [35, 81]. However, in pervasive environments such as smart homes, resolution approaches are usually infeasible, and hence, how to prevent service interferences becomes important in pervasive environments. Tsang et al. [134] propose a learning approach to capture sequences of behaviors and then detect service interferences. Low [93] propose a rule-based approach to detect interferences and improved the performance by using a cache. Kolberg et al. [86] divide service interference problems into several categories: 1) conflicting accesses to single device at the same time, 2) undesirable effects among activated devices, and 3) unexpected consequences caused by sequential connections among devices. Nevertheless, they only provide an architectural approach to handle the first two categories. Nakamura et al. [107] claim that there are two types of service interferences, namely, appliance interferences and environment interferences, depending on whether the interference takes place due to direct conflict among appliances or

via the environment. Existing prevention approaches take care of resource confliction, but in pervasive environments, the presences of interferences are usually dependent on users' perceptions, which are vague and subjective. Hence, it appears that fuzzy-based approaches which account for human's linguistic ambiguity can be promising.

It follows from the above discussions that there is still much space for further investigation on the consistency issues of pervasive service compositions. Currently, the scope of this thesis concentrates on dealing with two inconsistency issues of pervasive service composition, namely, inconsistent user preferences and inconsistent service effects.

2.4 Summary

Tables 2.2, 2.3 and 2.4 summarize the investigated systems with regard to the design issues this thesis has discussed so far. The expected contributions of this work are appended in the last row in the tables. As mentioned in previous sub-sections, in respect of flexibility (including extensibility and interoperability) of a pervasive system, data-centric architecture appears superior to process-centric architecture; standard-based wiring format is more interoperable than proprietary ones. Among the systems investigated so far, UPnP, Gaia, and CoBra are more interoperable. Strictly speaking, although CTK and Aura use XML, they are only "potentially interoperable" since the syntax and semantic of the XML-based wiring formats are still proprietary.

It is also important to point out that all systems being investigated except UPnP use TCP-based service management (see Table 2.3). Nevertheless, contrary to media streaming protocols, the packet size of service management is typically small (for example, most of the SSDP packets range from 200 bytes to 450 bytes), and can be transmitted by single UDP-based packet, since theoretical size of an UDP datagram is 65507 bytes, and empirical size is 576 bytes. Consequently, adopting UDP for service

management is more efficient.

Generally speaking, directory-based approach has better scalability and performance, but is poor in reliability because the directory can become single point of failure. On the contrary, non-directory-based approaches are more robust, but produce more network traffic. As reported by Meshkova et al., the design of service discovery systems depends on the scale of their deployment [102]. According to their classification, enterprise scale service discovery systems such as Jini, COBRA Trading Object Service, and JADE typically are designed based on directory-based approaches to reduce the traffic. On the other hand, small scale systems such as UPnP and Bluetooth are more suitable for non-directory-based approaches, since it avoids the single point of failure problem, and is easier to be implemented in embedded devices [148]. Nevertheless, in Table 2.3, all systems except for SSDP (UPnP) use directory-based approach for service discovery. This is because that these systems are designed for generality. The UPnP has been customized for home network. Therefore, its non-directory-based design is more suitable for the smart home.

CoBra and One.world deal with the single-point-of-failure problem by automatic recovery of directory servers and by re-electing directory servers, respectively. However, recovering the discovery server does not guarantee the reliability of services. Among the systems, only Aura is capable of recovering a failed service by re-composing the service. As for service consistency, only Gaia and Aura partially deal with the consistency problem which includes the consistency between service specification and the consistency between the effects caused by components.

Among four key qualities of the Smart Home systems (i.e. flexibility, reliability, consistency, efficiency), SSDP is superior to other mechanisms in flexibility but lacks of supports in reliability, consistency, and efficiency. Over the last few years, several mechanisms have been proposed to enhance UPnP/SSDP. Nakamura et al. [106] stud-

ied the efficiency problems of the interconnecting UPnP gateways. They proposed to store SSDP Presence Announcement messages in the caches of UPnP gateways in order to reduce the service discovery traffic. Knauth et al. [85] proposed to reduce traffic by introducing proxies among UPnP Devices that serve as cache in LAN. In [100, 73], the authors enhanced GENA (General Event Notification Architecture), a TCP-based sub-protocol of UPnP used for event notification by realizing GENA based on IP multicast. IP multicast mechanism is UDP-based, which is considered unreliable yet more efficient than TCP. However, these enhancements do not deal with service composition and recovery issues directly, which are critical for achieving high reliability, consistency, and efficiency.

To sum up, SSDP (of UPnP) tends to be very competitive in respect of flexibility, and none of existing systems fully address the reliability and consistency issues of service management. As a result, SSDP should be a good starting point based on which one can design more sophisticated mechanisms to address the reliability and consistency issues. This research proposes a self-organizing and self-healing service management protocol for smart homes by enhancing reliability of services (Chapter 3) and consistency of service composition (Chapter 5). Furthermore, efficient service management mechanisms in decentralized protocols are hard to design since it is apt to drain out bandwidths with a lot of heartbeat or polling messages. Hence, this research deals with efficiency issues by striking a balance between the robustness of system and the overhead of communication complexity by designing, analyzing and evaluating mechanisms to eliminate unnecessary network traffics. The details are elaborated in Chapter 4.

Table 2.2: Architectural styles and service management functionalities of Pervasive systems

Name	Architectural Style	Wiring Format	(Service Management Functionalities) Discovery	Recovery	Composition
CTK	Process-Centric	XML	✓	-	✓
UPnP	Process-Centric	XML (SOAP)	✓	-	-
Gaia OS	Process-Centric	IIOP	✓(COBRA)	-	✓
Aura	Process-Centric (Asynchronous)	XML	✓(Jini)	✓	✓
CoBra	Process-Centric (Asynchronous)	FIPA-ACL	✓(JADE)	✓	-
SOCAM	Process-Centric (Local)	-	✓(OSGi)	-	-
One.world	Data-Centric (Tuple Space)	Serialized Java Objects	✓	✓	✓
Event Heap	Data-Centric (Tuple Space)	Serialized Java Objects	✓	✓	✓
LIME	Data-Centric (Tuple Space)	Serialized Java Objects	✓	-	-
SOLAR	Data-Centric (MOM)	Text (Proprietary)	✓(INS)	✓	✓
MIRES	Data-Centric (MOM)	Active Message (Tiny OS)	-	-	-
This work	Data-Centric (MOM)	JSON	✓	✓	✓

Table 2.3: Detailed comparisons among Service Discovery mechanisms of Pervasive systems

Name	Category	Recovery	Network Scale	Transport Layer
CTK	Directory-based	-	LAN	TCP
UPnP	Non-directory-based	-	LAN	UDP
Gaia	Directory-based	Component	Internet	TCP
Aura	Directory-based	Service	LAN	TCP
CoBra	Directory-based	Backbone	Internet	TCP
SOCAM	Directory-based (Local)	-	-	-
One.world	Directory-based	Backbone	LAN	TCP
Event Heap	Directory-based	Backbone	LAN	TCP
LIME	Directory-based	-	Internet	TCP
Solar	Directory-based	Component	LAN	TCP
MIRES	-	-	-	-
This work	Non-directory-based	Service	LAN	UDP

Table 2.4: Detailed comparisons among Service Composition mechanisms of Pervasive systems

Name	Expression Support	Sophisticated Ranking	Expressiveness of Expression	Preferences Unification	Interferences Estimation
CTK	-	-	-	-	-
UPnP	-	-	-	-	-
Gaia	✓	✓	Enumerate; Mandatory	✓	-
Aura	✓	✓	Enumerate and Numeric; Mandatory	-	-
CoBra	-	-	-	-	-
SOCAM	✓	-	Enumerate; Mandatory	-	
One.world	-	-	-	-	-
Event Heap	-	-	-	-	-
LIME	-	-	-	-	-
Solar	✓	-	Enumerate and Numeric; Mandatory	-	-
MIRES	-	-	-	-	-
This work	✓	✓	Enumerate and Numeric; Mandatory and Negotiable	✓	✓

Chapter 3

Flexible and Robust Service Management in a Smart Home

As mentioned in Chapter 2, data-centric architectures such as MOM and Tuple Spaces are more flexible and robust than process-centric architectures. Furthermore, it has also been pointed out that MOM is superior to Tuple Spaces since it preserves the flexibility of data-centric architecture while prevents from performance and interoperability issues caused by Tuple Spaces. Despite these advantages, there are still several challenges when designing pervasive systems based on MOM.

First of all, a pervasive system designed based on MOM is called a Message-Oriented Pervasive System (MOPS) which consists of a virtual "software bus" for interchanging messages among heterogeneous publishers and subscribers, namely, the "nodes". The logical pathways between nodes are called "topics". Contrary to traditional enterprise systems, pervasive systems are highly dynamic since the service components can join, leave or fail at any-time. However, MOM lacks of appropriate service management mechanisms to maintain and to keep track of the relationship between services and service components. (Recall that a service is composed of a group of collaborating service components.) Moreover, among the few MOPS such as SOLAR [87] and MIRES [127], none of them deal with service management issues. Second, regarding to robustness, MOM supports failure isolation, but it lacks of both failure detection and recovery mechanisms. In this thesis, the term "robustness" refers to the ability of a system to detect failed service components and then either to recover them from failure states or to find a replacement eventually. Third, in typical pervasive environments such as Smart Homes, the people setting up and maintaining the systems are consumers

with little technical knowledge. Hence, the proposed solutions to challenges mentioned above have to make the system as autonomous as possible. The robustness issue of a Pervasive system is a typical example: the system without autonomous failure detection and recovery capabilities may frustrate users from time to time, since they are hardly able to pinpoint the sources of all failed services and to recover them.

It follows from what has been discussed that the following features are of crucial importance: 1) an autonomous service discovery and composition framework that is capable of discovering, selecting, and activating nodes spontaneously; 2) a failure detection and recovery mechanism that is aware of service failures. Such mechanisms is capable of directing the system to identify failed components and then to recover the failed service by either replacing the failed components by alternative ones or restarting the failed components autonomously. In the following,the term "robust service management" will be used to refer to the two features mentioned above.

The objective of this chapter is therefore to design a robust service management framework for MOPS under the challenges listed above. In the following, a service model, namely, PerSAM (Pervasive Service Application Model), which defines key abstractions, data structures and a taxonomy of entities (see Figure 3.1) for MOPS is first presented. The reason for defining a service model is that MOM only comes up with the "node" and "bus" abstractions, which are insufficient to facilitate robust service management. After that, Section 3.2 describes an application layer robust service management protocol, namely, PSMP (Pervasive Service Management Protocol). It is important to point out that this section focuses on the application layer of the network stack, the protocol issues at lower layers such as the robustness of UDP and the efficiency of IP multicast are taken up in the next chapter. PerSAM and PSMP are presented by using Unified Modeling Language (UML) [31] and Communicating Sequential Processes (CSP) [70]. UML is useful in illustrating data structures (by using

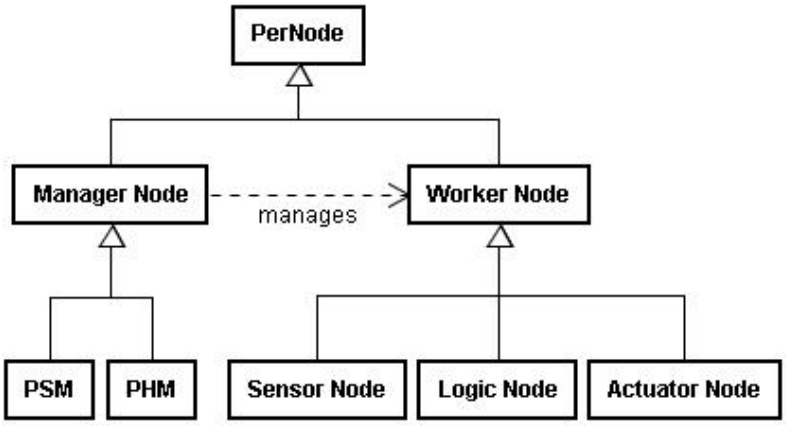


Figure 3.1: A taxonomy of PerNode

Class diagrams) and interacting flow (by using Sequence diagrams) visually. On the other hand, CSP is a member of the family of Process Algebra and is a widely used mathematical language for specifying distributed systems, where Process Algebra is a formal description technique for complex distributed systems, especially those with communicating or concurrently executing components [28]. As reported by Sharp [120], the Process Algebra not only enables us to describe protocol in a concise manner, but also makes it possible to analyze protocols. The benefits of using the CSP are: 1) the models and protocols can be specified accurately, 2) it enhances the reproducibility of PerSAM/PSMP since CSP is more precise than pseudo code and UML, and 3) it is easier to validate the desired system attributes formally because of the preciseness of process algebra. In this thesis, UML is used to convey high-level concepts, while recognizing that CSP is useful in increasing preciseness and reproducibility.

3.1 Pervasive Service Application Model (PerSAM)

This section focuses on presenting the proposed service model, namely, PerSAM. Several acronyms and notations are used throughout this thesis to keep the presentation concise, which are summarized in Table 3.1 and in Table 3.2.

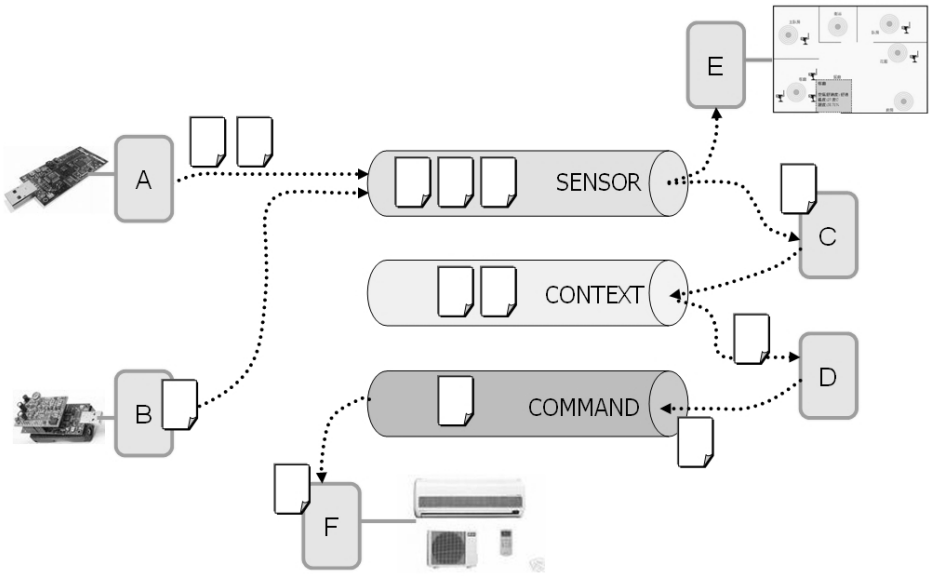


Figure 3.2: The message-oriented pervasive system

In PerSAM, the term "PerNode" refers to a service component in MOPS. Also note that the term "PerNode" or "node" are used interchangeably in the sequel. PerNodes fall into one of the two categories: the Manager Nodes and the Worker Nodes. Manager Node is designed for administrative purposes. Pervasive Service Manager (PSM) and Pervasive Host Manager (PHM) are both Manager Nodes, which are responsible for managing a Pervasive Service and a Pervasive Host, respectively. On the other hand, Worker Nodes are basic useful functional units. Worker Nodes can be further classified into three categories according to their behaviors: Sensor Nodes, Actuator Nodes, and Logic Nodes. Taking Figure 3.2 as an example, A and B are Sensor Nodes, which are connected to gateways of sensors and the sensed data are sent to the SENSOR topic. Similarly, C and D are Logic Nodes that encapsulate logics of message processing.

Figure 3.3 indicates the life-cycle of PerNode. The computing device on which a PerNode are deployed is called a Pervasive Host. The procedures of installing a PerNode on a Pervasive Host are: 1) placing binaries of the node in a directory, and 2) registering its metadata so that it is manageable. After being installed, a node en-

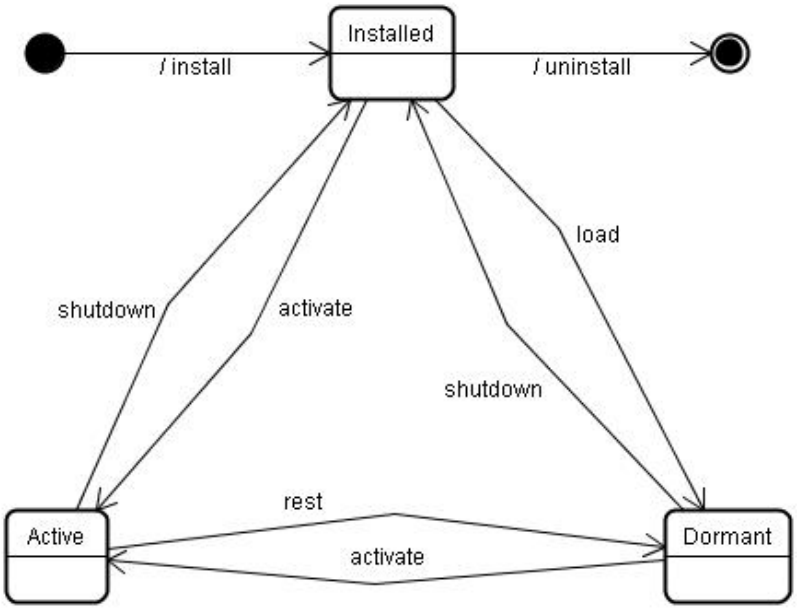


Figure 3.3: The states of a PerNode

ters INSTALLED state. Next the node is loaded into memory, starting in DORMANT state. Note that although a DORMANT node does not perform any message-processing task, it still issues heartbeat and is "discoverable" by discovery protocols. A node goes into in ACTIVE state when it is activated. An activated node can receive, process, and send messages. Similarly, nodes can be removed from memory by a "shutdown" operation, or fall back to DORMANT state by a "rest" operation. The formal definitions of PerNode and Worker Node are as follows, which are depicted in Figure 3.4.

Definition 1. (PerNode) *A PerNode $p \in P$ is an atomic stateful entity in MOPS, where P is the universe of PerNodes in the system, and state $\in$ {`INSTALLED,DORMANT,ACTIVE`} is an attribute of p.*

Definition 2. (Worker Node) *A Worker Node $w \in W$ is a PerNode that encapsulates a unit of application logic, where W is the universe of Worker Nodes in the system. In addition to the attributes inherited from PerNode, a Worker Node has three additional attributes: node type $nt \in NT$, where NT is the universe of node types in*

Table 3.1: Summary of acronyms

Abbreviation	Full Name
MOM	Message-Oriented Middleware
MOPS	Message-Oriented Pervasive system
PerSAM	Pervasive Service Application Model
PSMP	Pervasive Service Management Protocol
PH	Pervasive Host
PHM	Pervasive Host Manager
PS	Pervasive Service
PSM	Pervasive Service Manager
PA/LA	Presence Announcement or Leave Announcement

the system and the heartbeat period (hbp), which specify the functional category and the heartbeat rate of the node, respectively.

3.1.1 The Pervasive Communities

A Pervasive Community is a logical organization of nodes. There are two kinds of Pervasive Communities: 1) the Pervasive Service (PS) consists of one or more nodes

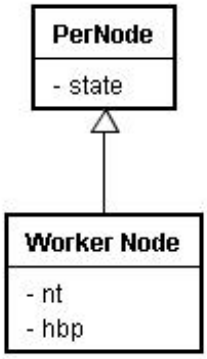


Figure 3.4: The structure of a PerNode and a Worker Node

Table 3.2: Summary of notations

Notation	Description
nt	An instance of node type
ps	An instance of Pervasive Service
ph	An instance of Pervasive Host
w	An instance of Worker Node
m^{ps}	An instance of PSM that manages ps
m^{ph}	An instance of PHM that manages ph
W^{ps}	The set of Worker Nodes belonging to ps
W^{ph}	The set of Worker Nodes belonging to ph
ST^{ps}	The Service template of ps
MT^{ps}	The set of missed or failed node types that prevent ps from being alive
T^{ps}	The set of timestamps that records the previous heartbeat time for each $w \in W^{ps}$
$\hat{m}$	A multicast channel
$\hat{t}$	An TCP-based unicast channel
$\hat{u}^n$	An UDP-based unicast channel to node n
$ssdp^x$	An instance of SSDP message, x indicates the message type
W^*_{nt}	A list of candidate Worker Nodes with type nt

that collectively provide a service to user, and 2) the Pervasive Host (PH) refers to a group of nodes that co-locate in the same computing device. Each community has a Manager Node that keeps track of its members. In other words, each PS has a PSM and each PH also has a PHM. A Worker Node can join several PSs at the same time.

Considering the example depicted in Figure 3.5, node A is a member of the "Adaptive Air Conditioner" PS ($ps1$) as well as the "Sensor Map" PS ($ps2$) at the same time.

Let us denote the PSM of a Pervasive Service ps as m^{ps} and the Worker Nodes that join ps as a set $W^{ps} \in 2^W$, where 2^W is the power set of W. Considering $ps1$ in Figure 3.5, we have $m^{ps1} = PSM1$ and $W^{ps1} = \{A, C, D, F\}$. The Service Template $ST^{ps} \in 2^{NT}$ is pre-defined by service designers. Each ST^{ps} specifies required node types that comprise ps. For example, in Figure 3.5, $ST^{ps1} = \{$`Temperature Sensor, Context Interpreter, Indoor Temperature Control Logic, Air Conditioner`$\}$. To compose a Pervasive Service ps, m^{ps} first finds the best $w \in W$ such that $w.nt = nt$ for each $nt \in ST^{ps}$. The definition of "the best" depends on a user-defined selecting function (see Table 3.3). The default selecting function is FCFS (First Come, First Select), but it can be substituted by a more sophisticated mechanism such as the one proposed in Chapter 5.

Before ps is successfully composed, some Worker Nodes of required types $nt \in ST^{ps}$ could be still missing. Let us denote the set of node types of missing Worker Nodes as MT^{ps}. Formally:

$$MT^{ps} = \{nt | nt \in ST^{ps}, \neg\exists w \in W^{ps} : w.nt = nt\} \tag{3.1}$$

In the previous example, assuming that $W^{ps1} = \{C, D, F\}$, then MT^{ps1} is $\{$`Temperature Sensor`$\}$, because the missing Worker Node A is of the type "Temperature Sensor". In the beginning of service composition, since there are no Worker Nodes found, thus $W^{ps1} = \phi$ and $ST^{ps1} =\{$`Temperature Sensor, Context Interpreter, Indoor Temperature Control Logic, Air Conditioner`$\}$. It is easy to observe from this example that $MT^{ps} \subseteq ST^{ps}$. It is worthy to point out that $MT^{ps} = \phi$ implies that ps is successfully composed and that ps is alive if and only if all $w \in W^{ps}$ are in ACTIVE states. Based on this observation, it is the time to define the liveness of a Pervasive Service.

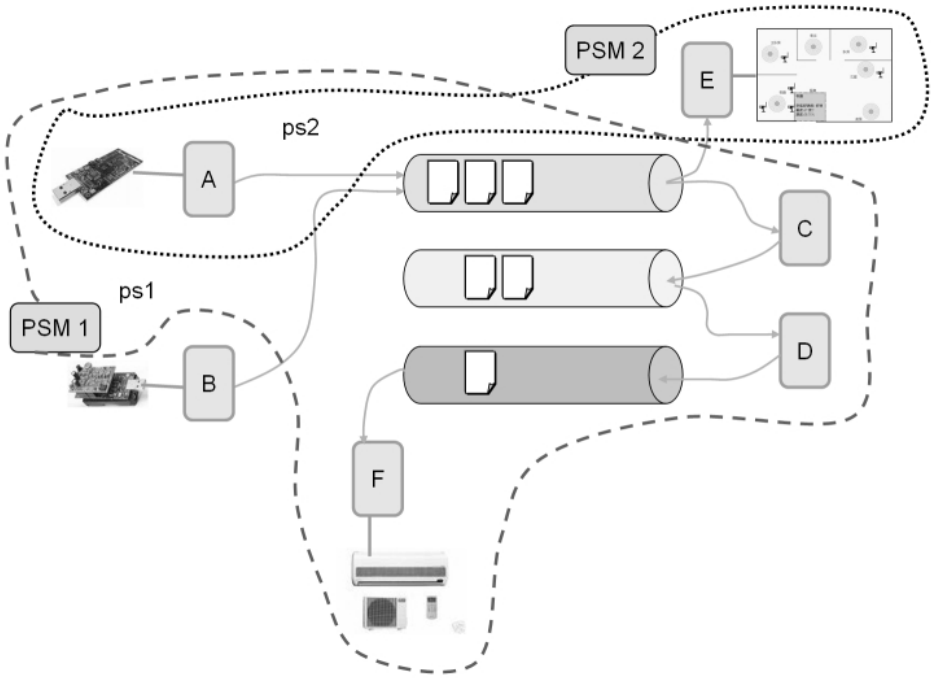


Figure 3.5: The Pervasive Service communities

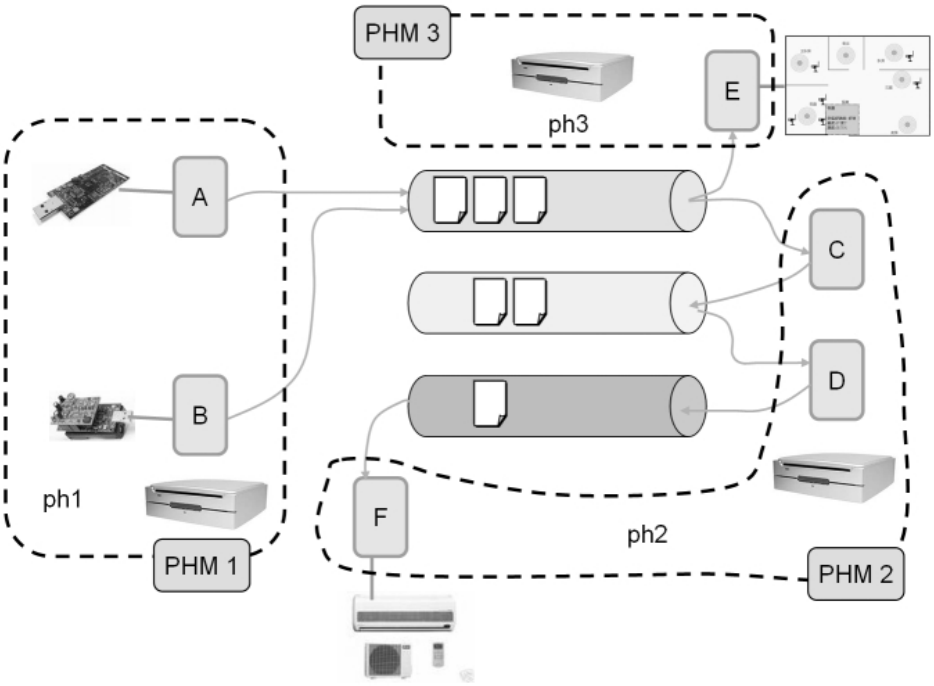


Figure 3.6: The Pervasive Host communities

Definition 3. (Liveness of a Pervasive Service) *A Pervasive Service ps is alive if and only if the following statement holds:*

$$MT^{ps} = \phi \wedge \forall w \in W^{ps}, w.state = \texttt{ACTIVE} \tag{3.2}$$

To detect possible failures of affiliating Worker Nodes, a PSM (m^{ps}) keeps track of timestamps of previous heartbeats t^w for each $w \in W^{ps}$ in a vector of timestamps T^{ps}, where $t^w \in T^{ps}$ and $|T^{ps}| = |W^{ps}|$. Based on the above discussions, a Pervasive Service can be formally defined as follows.

Definition 4. (Pervasive Service, PS) *A Pervasive Service ps is a tuple:*

$$ps = \langle m^{ps}, ST^{ps}, MT^{ps}, W^{ps}, T^{ps} \rangle , \tag{3.3}$$

where m^{ps} is PSM of ps, $ST^{ps} \in 2^{NT}$ is the Service Template, $MT^{ps} \subseteq ST^{ps}$ is set of missed types, $W^{ps} \in 2^{W}$ is the set of Worker Nodes join ps, and T^{ps} is a vector of timestamps that records the previous heartbeat time for each $w \in W^{ps}$.

Likewise, each Pervasive Host is composed of a set of Worker Nodes, whose life-cycles are managed by a PHM. The PHM and its affiliating Worker Nodes locate in the same device. Therefore, a PHM is able to detect the node states, to load and to shutdown Worker Nodes. In Figure 3.6, there are 3 Pervasive Hosts: A and B belong to $ph1$ since they are deployed in the same device; similarly, C, D and F belong to $ph2$; the $ph3$ has single member E. Note that $ph1$, $ph2$ and $ph3$ are managed by PHM 1, 2 and 3, respectively. The definition of a Pervasive Host is as follows:

Definition 5. (Pervasive Host, PH) *A Pervasive Host ph is a tuple:*

$$ph = \left\langle m^{ph}, W^{ph} \right\rangle , \tag{3.4}$$

where m^{ph} is the PHM of ph, and W^{ph} is the set of Worker Nodes that currently locate on ph.

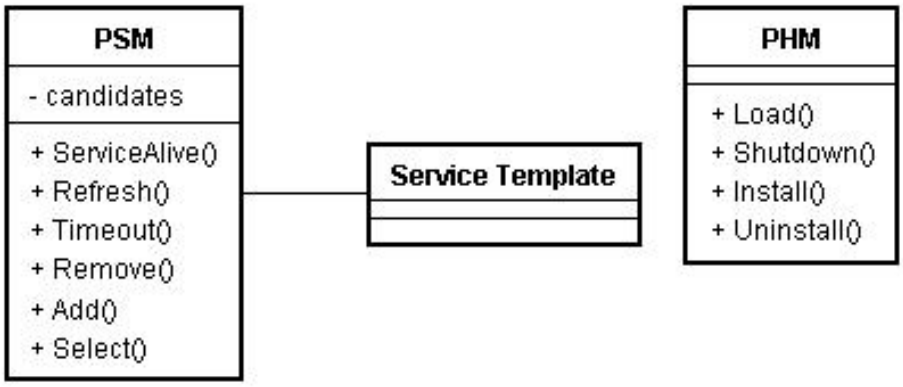


Figure 3.7: The structures of PSM and PHM

3.1.2 The Pervasive Managers

The responsibilities of a PSM are as follows: 1) to compose a PS according to Service Template ST^{ps}. As mentioned in Section 3.1, when there are many qualified candidates, PSM first stores them in a candidate list denoted W^{*}_{nt} and then selects the best one by invoking the pre-defined selecting function. 2) PSM monitors all $w \in W^{ps}$. Once PSM observes that a Worker Node does not heartbeat for longer than a pre-defined threshold, it emits a "suspect" message for that node. 3) PSM can add to or remove community members from PS. 4) PSM is responsible for keeping the PS alive. In case PS is not alive, PSM attempts to to re-compose PS. As depicted in Figure 3.7, there are six operations in PSM used to support the above-mentioned responsibilities. The input parameters, return values, definitions and explanations of these operations are revealed in Table 3.3. Fuller discussion of how these operations work will be presented in Section 3.2.

On the other hand, PHM is an agent that administrates nodes located in the same computing device. The tasks of PHM include monitoring and maintaining local Worker Nodes, loading local Worker Nodes from the file system into memory, and killing the failed Worker Nodes that do not emit heartbeat messages. It is important to point

Table 3.3: The Operations of a Pervasive Service Manager

Name	Input	Output	Definition	Comments
ServiceAlive	$\varnothing$	*Boolean*	see Def. 3	Returns the liveness of a Pervasive Service
Refresh	$w \in W^{ps}$	$\varnothing$	$T^{ps}[w] := t^{now}$	Update heartbeat timestamps of w with current time
Timeout	$w \in W^{ps}$	*Boolean*	$t^{now} - T^{ps}[w] \geq k$	Returns if w does not heartbeat more than a threshold k
Remove	$w \in W^{ps}$	$\varnothing$	$W^{ps} := W^{ps} - w$	Removes w from ps
Add	$w \in W^{ps}$	$\varnothing$	$W^{ps} := W^{ps} \cup w$	Add w to ps
Select	W^*_{nt}	$w \in W^*_{nt}$	User defined	Returns a best node w among candidate list W^*_{nt}

out that the task of discovering community members is faster and more robust for PHM than for PSM, since no network-based communications are involved. The life-cycles of Worker Nodes deployed in the same PH can be altered by the PHM locally. As indicated in Figure 3.7, PHM contains two operations. The *Shutdown* operation removes a local Worker Node from memory. Similarly, the *Load* operation loads a local Worker Node into memory. The input parameters, return values, definitions and explanations of these operations are shown in Table 3.4. Notice that the *Install* and *Uninstall* operations are invoked when a Worker Node installed to or uninstalled from a computing device.

Table 3.4: The Operations of a Pervasive Host Manager

Name	Input	Output	Definition	Comments
Load	$w \in W^{ph}$	$\varnothing$	$w.state = DORMANT$	Returns the liveness of a Pervasive Service
Shutdown	$w \in W^{ph}$	$\varnothing$	$w.state = INSTALLED$	Update heartbeat timestamps of with current time
Install	$w \in W^{ph}$	$\varnothing$	$W^{ph} := W^{ph} \cup w$	Add a Worker Node into a Pervasive Host
Uninstall	$w \in W^{ph}$	$\varnothing$	$W^{ph} := W^{ph} - w$	Remove a Worker Node from a Pervasive Host

3.2 Pervasive Service Management Protocol (PSMP)

PerSAM and PSMP are realize by extending UPnP, a home networking protocol standard (ISO/IEC 29341) [15]. The reason for choosing UPnP is three fold: 1) it is one of the few dynamic service discovery protocols that do not need a dedicated and centralized service directory [43], which is more feasible for robust service management. 2) UPnP is independent of platform and programming languages. 3) UPnP is a widely used and well-known standard. SSDP takes charge of service discovery in an UPnP network. By default, SSDP operates based on HTTPMU (HTTP over UDP and Multicast). HTTPMU uses IP multicast, which is supported by most network switching equipments. IP multicast forwards packets to a group of interested receivers via a set of pre-defined virtual IP addresses. Therefore, SSDP does not need a centralized server since the multicast service is carried out by the underlying infrastructure. SSDP extends HTTP by two HTTP methods: NOTIFY and M-SEARCH.

UPnP specifies a Device Architecture. An UPnP Device consists of a set of UPnP Services, and each UPnP Service comprises several UPnP Actions. A node that is

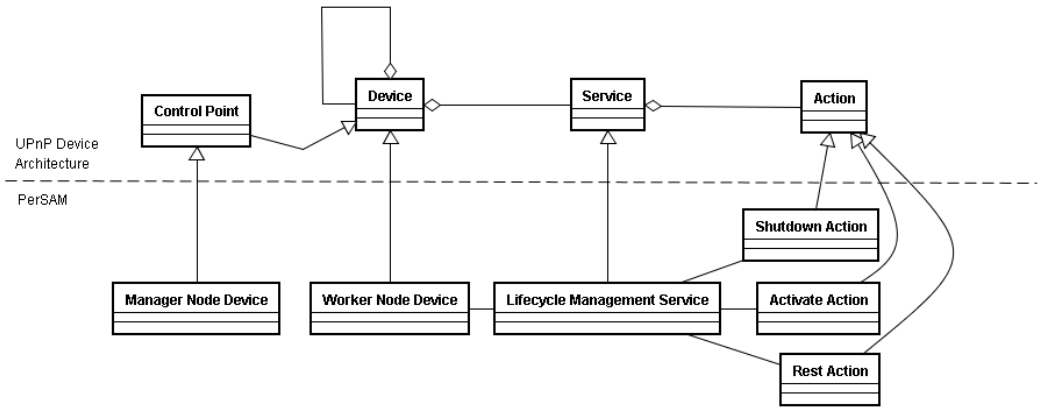


Figure 3.8: The projection of PerSAM to UPnP Device Architecture

capable of invoking UPnP Actions is called a Control Point, which can also be embedded in an UPnP Device. In this research, PerNode is implemented based on UPnP Device Architecture (see Figure 3.8). More precisely, each PerNode Device consists of an UPnP Service, the PerNode Life-cycle Management Service, which manages PerNode life-cycle by using three UPnP Actions (Activate, Rest, and Shutdown). Manager Nodes (PSM and PHM) are special types of PerNode Devices because they contain a Control Point. The reason for this design is that a Control Point is capable of invoking UPnP Actions of remote PerNodes to manage their life-cycles. It is important to point out that despite the similarity in their names, the UPnP Services are different from Pervasive Services: An UPnP Service always embedded in an UPnP Device, while a Pervasive Service is a virtual community that consists of a group of nodes.

Before turning to a closer examination of PSMP, let us first take a look at some basic CSP [70] syntax. CSP uses the form $P \triangleq e \rightarrow R$ to describe the behaviors of a Process P, which first takes part in an event e, and then behaves like process R. Parameters can be passed to a process by enclosing with square brackets. For example, in $P[x] \triangleq f(x) \rightarrow R$, the x enclosed by square brackets is passed to the function $f(x)$ in the right hand side. In CSP, $c!m$ denotes an output event, in which a message m is emitted through network channel c. In similar way, $c?m$ denotes an input event, in

which a message m is received through channel c. A special process $SKIP$ denotes a process that terminates without error. Table 3.5 summarized the CSP notations used throughout this paper.

3.2.1 Presence Announcement, Leave Announcement, and Life-cycle Management

In PSMP, Presence Announcement (PA) and Leave Announcement (LA) in PSMP can be formally described as follows:

$$PA[p] \triangleq \hat{m}!ssdp^{alive}[p] \rightarrow SKIP \tag{3.5}$$

$$LA[p] \triangleq \hat{m}!ssdp^{byebye}[p] \rightarrow SKIP \tag{3.6}$$

In (3.5), PA sends an "ssdp:alive" to the multicast channel $\hat{m}$ to announce the presence of the node p, and then terminates. The same syntax applies to the definition of LA except that the message is "ssdp:byebye". Based on (3.5) and (3.6), we can now define Life-cycle Management (see Protocol 1). The Life-cycle Management protocol (LM) enables Manager Nodes to change states of nodes remotely.

Protocol 1. (Life-cycle Management, LM) *A Life-cycle Management (LM) protocol changes state of PerNode according to incoming calls to UPnP Actions. A function NewState is used to decides new state based on the current state and the action being invoked. If a node is changed to INSTALLED state, it performs a leave announcement (LA).*

$$\begin{aligned} LM[p] \triangleq \hat{t}?call \rightarrow & if(call.action = shutdown) \\ & then\ LA[p]; NS[p]\ else\ NS[p] \end{aligned} \tag{3.7}$$

$$\begin{aligned} NS[p] \triangleq & p.state := NewState(p.state, call.action) \\ & \rightarrow LM[p] \end{aligned} \tag{3.8}$$

Table 3.5: Summary of CSP notations used in PerSAM/PSMP

Notation	Description
$P \triangleq e \rightarrow R$	A process P takes part in an event e and then behaves like another process R
$c?m$	Listening for an incoming message m from channel c
$c!m$	Emitting a message m to channel c
$P;Q$	P and Q run sequentially
$P \| Q$	P and Q run concurrently
$\coprod_{x \in X}[e]$	For each $x \in X$ do e
$SKIP$	A process terminates successfully

In (3.7), $\hat{t}$ is a call channel to a UPnP Action from remote Manager Nodes, the notation *call* denotes an incoming call to UPnP Actions, and ";" is used to concatenate two sequential processes. The ":=" symbol assigns values to variables.

3.2.2 Service Composition and Activation

The purpose of service composition and Activation in PSMP is first to find appropriate Worker Nodes for a PS, and then to ensure that the chosen nodes are in `ACTIVE` states persistently. If there are multiple matching nodes, PSM selects one from them according to a pre-defined strategy. Currently, FCFS is the default strategy. PSMP provides extension points for more sophisticated service selection strategies such as the one proposed in Chapter 5. Figure 3.9 illustrates the interactions between these nodes when performing service composition. Whenever the service is not alive (see Definition 3), the PSM issues a "psmp:discover" to find PS members (Figure 3.9, step 1) to initiate a service composition.

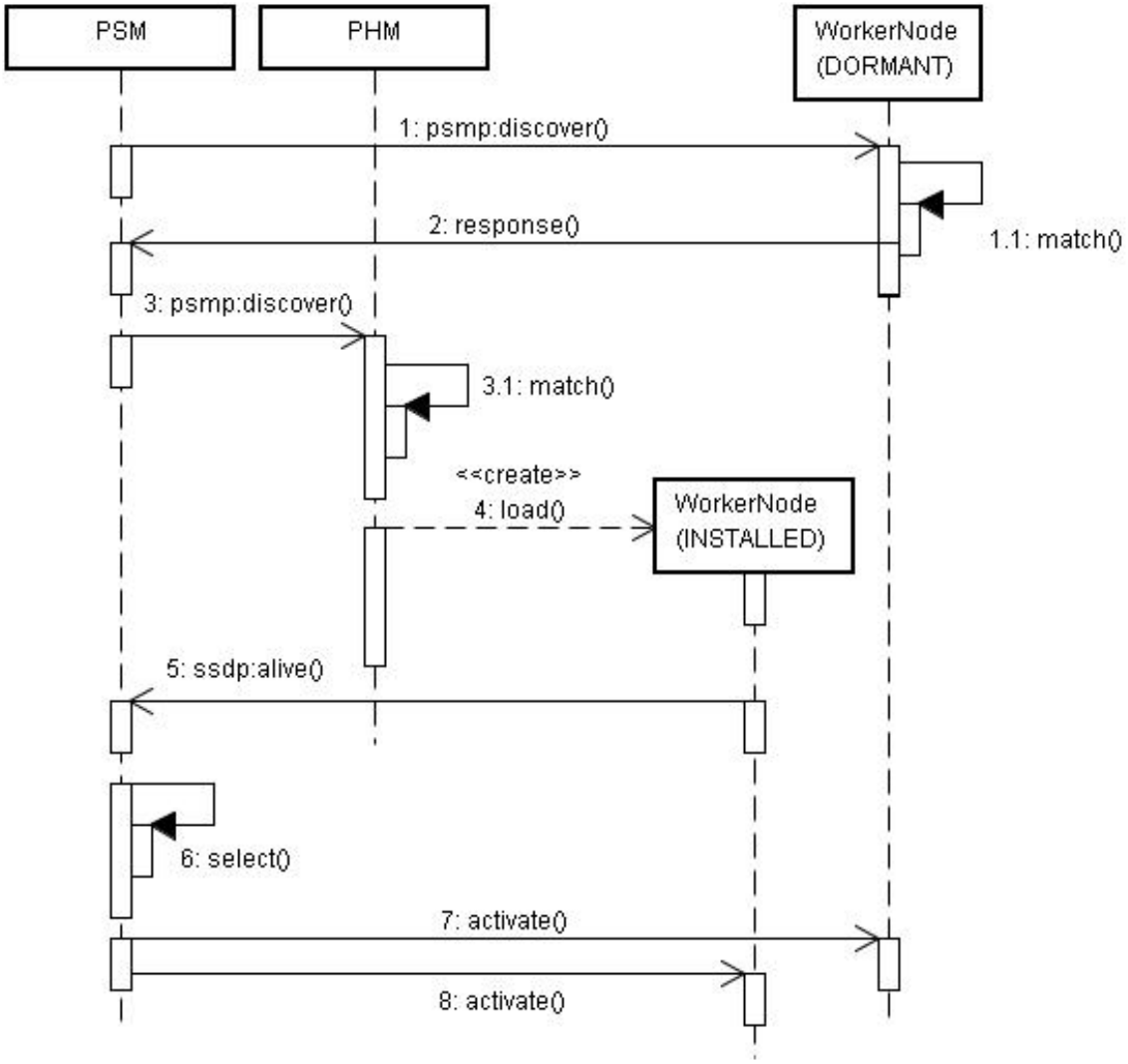


Figure 3.9: PSMP service composition

Note that finding qualified Worker Nodes can not be achieved by simply issuing a "ssdp:discover" action. The reasons is twofold. First, "ssdp:discover" only discovers nodes that are already loaded into memory (i.e. in `DORMANT` state or in `ACTIVE` state). To put it another way, the nodes that are in `INSTALLED` state do not respond to PSM. This problem causes low degree of support and low composition sustainability [80]. A typical case is that for a newly booted system, nearly all nodes are in `INSTALLED` states. Hence, few services can be successfully composed in this circumstance.

Second, "ssdp:discover" does not support property-based lookup. SSDP messages contain only type information (ST), so that in the matching phase of a `DORMANT` node (Figure 3.9, step 1.1), only node types are compared. In pervasive environments such as Smart Homes, components with the same type does not imply that they are interchangeable. For instance, the contexts obtained from sensors in the living room is different from the contexts observed in the bed room. As a result, PSMP propose the following extensions to deal with the above-mentioned problems: 1) PSMP defines a new "psmp:discover" action. This action is issued by PSM to perform "eager loading" of nodes. In simple terms, when PHM receives "psmp:discover", it loads all qualified local nodes that are in `INSTALLED` state. Once the qualified nodes become `DORMANT`, they will send presence announcements so that PSM will be able to discover them (Figure 3.9, Step 3-5). The "psmp:discover" can discover nodes that are not loaded, therefore the first problem mentioned above is solved. 2) A new header, CRITERIA, is added to support the property-based lookup. In CRITERIA header, the key-value pairs are separated by a comma. The comma represents an "and" relation, that is, a Worker Node is matched if and only if it fulfills all constraints specified by the key-value pairs. The key-value pairs are helpful to specify additional contexts of the nodes such as location and time. Hence, the search results are more accurate.

Listing 3.1: M-SEARCH message content with a "psmp:discover" action and a CRI-

TERIA header

```
M-SEARCH * HTTP/1.1
ST: urn:schemas-upnp-org:device:sensor:1
MX: 3
MAN: "psmp:discover"
CRITERIA: id=21, place=livingroom, type=thermo
HOST: 239.255.255.250:1900
```

Listing 3.1 shows the contents of a "psmp:discover" message that discovers a sensor service type. In this example, the criteria for this search are 1) the sensor id is 21, 2) the sensor type is thermometer, and 3) it locates in living room. The behavior of PSM in service composition is formally defined below.

Protocol 2. (PSM Service Composition) *A PSM Service Composition (SC_{PSM}) is initiated whenever the Pervasive Service is not alive. For each node type in MT^{ps}, PSM issues a discovery message (m-search) to the multicast channel ($\hat{m}$) and then performs PSM Service Selection (SS_{PSM} , see Protocol 5), formally:*

$$\begin{aligned} SC_{PSM}[ps] \triangleq\ & if(\neg ServiceAlive()) \\ & then \coprod_{nt \in MT^{ps}} \hat{m}!ssdp^{msearch}[nt] \rightarrow SS_{PSM}[ps] \\ & else\ SC_{PSM}[ps] \end{aligned} \tag{3.9}$$

In (3.9), the $\coprod$ operator is a shorthand for iteration. For instance, $\coprod_{nt \in MT^{ps}} P$ means that a process P executes one time for each $nt \in MT^{ps}$. Upon receiving a discovery message, a qualified Worker Node responses with a message in which describes its accessing information (Figure 3.9, step 1.1 and step 3). Protocol 3 describes the behavior for Worker Node in service composition. In (3.10), $\hat{u}^{psm}$ represents a UDP unicast channel corresponding to the searching PSM.

Protocol 3. (Worker Node Service Composition) *A Worker Node Service Composition (SC_W) examines incoming discovery messages. If there is a match in its node type, then it sends an response message ($ssdp^{resp}$) indicating its accessing information*

via the UDP unicast channel ($\hat{u}^{psm}$) corresponding to the source of the discovery message.

$$SC_W[ps] \triangleq \hat{m}?ssdp^{msearch}[psm, nt] \rightarrow$$
$$if(w.nt = nt)\ then\ \hat{u}^{psm}!ssdp^{resp} \rightarrow SC_W[w] \tag{3.10}$$
$$else\ SC_W[w]$$

Meanwhile, PHM is responsible for discovering `INSTALLED` nodes. Upon receiving a "psmp:discover", PHM compares the node type against local `INSTALLED` nodes (Figure 3.9, step 3). If there is a match, then PHM loads the node, causing it enters the `DORMANT` state (Figure 3.9, step 4).

Protocol 4. (PHM Service Composition) *A PHM Service Composition (SC_{PHM}) examines incoming discovery messages. According to the specified node type, a PHM iteratively match against local nodes and then load the matched node into memory (i.e. DORMANT state). It also emits a response message for the matched node.*

$$SC_{PHM}[ph] \triangleq \hat{m}?ssdp^{msearch}[psm, nt] \tag{3.11}$$
$$\rightarrow \coprod_{w \in ph} LS[w, psm, nt, ph]$$
$$if(w.nt = nt)\ then\ \hat{u}^{psm}!ssdp^{resp} \rightarrow SC_W[w]$$
$$else\ SC_W[w]$$

$$LS[w,psm, nt, ph] \triangleq \tag{3.12}$$
$$if(w.nt = nt \wedge w.state = \texttt{INSTALLED})$$
$$then\ Load(w) \rightarrow \hat{u}^{psm}!ssdp^{resp} \rightarrow SC_{PHM}[ph]$$
$$else\ SC_{PHM[ph]}$$

In (3.12), *Load* is an operation of PHM (see Table 3.4). PSM selects and activates Worker Nodes for a PS. The protocol for PSM Service Selection and Activation is shown below:

Protocol 5. (PSM Service Selection and Activation) *A PSM Service Selection protocol (SS_{PSM}) examines responses from Worker Nodes. For each response, PSM add the node into a list of candidates for a specific node type (W^*_{nt}). After MT^{ps} being empty, PSM selects the best ones for each node type according to a user-defined selecting function. After that, it executes PSM Service Activation (SA_{PSM}).*

$$
\begin{aligned}
SS_{PSM}[ps] \triangleq\ & \hat{u}?ssdp^{msearch}[w] \rightarrow \\
& if(MT^{ps} = \phi) \\
& then \coprod_{nt \in ST^{ps}} \begin{bmatrix} Add(Select(W^*_{nt})) \rightarrow \\ SA_{PSM}[Select(W^*_{nt}), ps] \end{bmatrix} \\
& else \begin{bmatrix} if(w.nt \in MT^{ps}) \\ then\ MT := MT - w.nt \rightarrow \\ \qquad W^*_{nt} := W^*_{nt} + w \rightarrow SS_{PSM}[ps] \\ else\ SS_{PSM}[ps] \end{bmatrix}
\end{aligned}
\tag{3.13}
$$

$$
\begin{aligned}
SA_{PSM}[w, ps] \triangleq\ & \hat{t}?call[activate] \rightarrow W^*_{nt} := \phi \\
& \rightarrow SC_{PSM}[ps]
\end{aligned}
\tag{3.14}
$$

In SA_{PSM}, PSM invokes an UPnP Action (activate) of the selected Worker Node, and then reset W^*_{nt}. After all Worker Nodes in a PS are activated, the PS becomes alive.

3.2.3 Failure Detection and Recovery

To keep a PS alive, all affiliating Worker Nodes must be in `ACTIVE` state lastingly. If one of them fails, then the PS becomes unavailable. Therefore, PSM has to be aware of the failures of Worker Nodes first (Failure Detection) and then resumes or substitutes the failed ones (Service Recovery). Recall that the term "robust service management" is used to refer to the mechanism that enables a system to detect failures and to recover

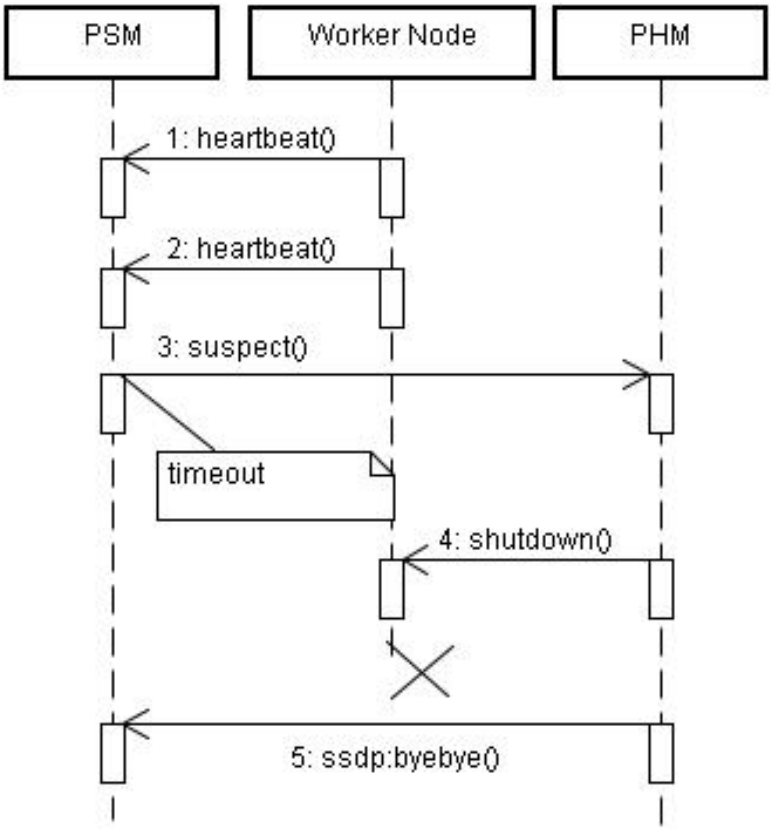


Figure 3.10: PSMP failure detection

from failures autonomously. Having clarified the semantics of robustness, a "Robust Pervasive Service" is now formally defined as follows.

Definition 6. (Robust Pervasive Service) *A Pervasive Service ps is robust if and only if the following statement holds:*

$$\Diamond Fail(w^f) \Rightarrow \Diamond\neg ServiceAlive() \wedge \Diamond ServiceAlive(), \tag{3.15}$$

where $w^f \in W^{ps}$ is a failed Worker Node, and $Fail(w^f)$ represents the fact that w^f fails.

The use the symbol $\Diamond$ to denote "eventually" and the symbol $\Box$ to denote "always" in the logic statements. These symbols are borrowed from Temporal Logic [89]. Note that $\Diamond\neg ServiceAlive()$ happens before $\Diamond ServiceAlive()$; therefore, their conjunction is not necessarily false. This sub-section presents protocols that ensure the robustness of PS. These protocols are designed based on the following assumptions.

1. **Eventually correct local failure detector (A1):** A Worker Node stops performing heartbeat eventually after it fails. This assumption requires that a failed Worker Node stops heartbeat eventually. It is weaker than the Fail-Stop model in the sense that a failed node does not need to stop executing or to stop sending packet, and the heartbeat does not need to stop immediately. It is also assumed that an eventually correct local failure detector exists. This assumption is reasonable since detecting failed processes or threads locally (i.e. within the same host) is much easier than detecting failed processes distributed over an asynchronous network. In real cases, eventually correct local failure detectors are usually supported by the underlying OS. For example, in Embedded Linux platforms, local failure detectors can be implemented by means of either built-in system monitoring hardware or software such as "watchdogs", which is included in the standard kernel package [143]. In brief, this assumption is made in theory for gaining rigor

of the result to be obtained. In most real cases, this assumption can be fulfilled by implementation techniques which are platform dependent.

2. **Perfect-Link assumption (A2):** Since we concentrate on application layer in this research work, a Perfect-Link model is assumed, in which all messages are guaranteed to be successfully delivered. In addition, a message does not appear in the network unless a node sends one. In network layer, this assumption can be ensured by using reliable multicast protocols such as RMP [140] or SRM [57].

3. **Persistent Manager Nodes assumption (A3):** Manager nodes will not experience failure, of which such assumption is made since we don't consider the robustness issues of Manager Node for the time being. In real cases, a simple yet effective solution to the reliability issues of Manager Nodes is to handle them by means of techniques in the implementation level. For example, one can develop a program (or use the watchdog services provided by the underlying OS) to detect the failures of Manager Nodes. In real world, many popular mission critical enterprise systems adopt this approach. For example, Oracle WebLogic Cluster uses similar design, in which a "Domain" is a logical division of application, which contains a cluster of "Managed Servers". Each domain is administrated by an "Admin Server", which is responsible for detecting the failures of Managed Servers in the same Domain. Actual services are provided by Managed Servers, but Managed Servers belonging to the same Domain are not necessarily located in the same host. In other words, a host contains Managed Servers may belong to different Domains. In each host, there is a "Node Manager", which is responsible for monitoring and recovering all Managed Servers. Apparently, the design mentioned above does not take care of reliability issues of Admin Servers and Node Managers, which are in fact guaranteed by the watchdog services provided by the underlying OS. The Nanny Servers of IBM WebSphere Servers use

similar approaches. To sum up, the rationale behind using hierarchical architecture (Manager-Worker) is that the possibility of manager node failure is much less than that of Worker Nodes in practice, since 1) actual heavy-loaded user tasks are handled by Worker Nodes; 2) the quantity of Manager Nodes is less than that of Worker Nodes; 3) The failures of Manager Nodes can be detected and be recovered by using mechanisms provided by their underlying OS/Platform. Consequently, we make this assumption in theoretical level, which however can be replaced by employing either consensus protocols or implementation level techniques. We are recently designing consensus-based protocols that make the failures of Manager Nodes detectable and recoverable without centralized coordinators [78]. When failure detection protocols for Manager Nodes are absent, one simple yet effective solution is to use the watchdog services provided by the underlying platform to detect and to recover the failed Manager Nodes.

4. **Composable service assumption (A4):** All services are composable. In other words, for each PS, for all types specified in the Service Template of the PS, there is at least one node of such type exists in the system. If this assumption is not hold, then it is impossible to recover the PS. The PSMP failure detection is shown in Figure 3.10. The behaviors of PSM, PHM, and Worker Node are formally defined as follows.

Protocol 6. (Worker Node Heartbeat) *A Worker Node performs heartbeat by emitting PA periodically. The Worker Node attribute hbp is a pre-defined interval between each heartbeat.*

$$HB_W[w] \triangleq sleep(w.hbp) \rightarrow PA[w]; HB_W[w] \tag{3.16}$$

Protocol 7 reveals how PSM emits suspecting message. There are two processes running in parallel, one for refreshing T^{ps} and the other for timeout eviction (EV_{PSM}).

In (3.17), || is used to combine two parallel processes. In Protocol 7, $Refresh$, $Timeout$, and *Remove* are operations of PSM (see Table 3.2).

Protocol 7. (PSM Node Suspecting) *PSM Node Suspecting protocol checks if there is an affiliated node stops performing heartbeat. If PSM does not receive any heartbeat for more than a pre-defined interval, it sends a suspecting message indicating a possible node failure.*

$$NSU_{PSM}[ps] \triangleq \begin{bmatrix} \hat{m}?ssdp^{alive}[w] \rightarrow Refresh(w) \\ \rightarrow NSU_{PSM}[ps] \end{bmatrix} \quad (3.17)$$
$$|| \; EV_{PSM}[ps]$$

$$EV_{PSM}[ps] \triangleq$$
$$\coprod_{w \in W^{ps}} \begin{bmatrix} if \; Timeout(w) \\ then \; Remove(w) \rightarrow \hat{m}!psmp^{suspect}[w] \end{bmatrix} \quad (3.18)$$
$$\rightarrow EV_{PSM}[ps]$$

After a node is suspected, PHM stops the node and then sends a leave announcement on behalf of it (Figure 3.10, step 5). These operations are described as follows.

Protocol 8. (PHM Shutdown Suspects) *PHM Shutdown Suspect protocol stops the suspected nodes according to the incoming suspect messages. The PHM also emits LA on behalf of the suspected nodes.*

$$SSU_{PHM}[ph] \triangleq \hat{m}?psmp^{suspect}[w]$$
$$\rightarrow \text{if } (w \in W^{ph})$$
$$\text{then} \begin{bmatrix} Shutdown(w) \\ \rightarrow LA[w]; SSU_{PHM}[ph] \end{bmatrix} \quad (3.19)$$
$$\text{else } SSU_{PHM}[ph]$$

After a failure is detected, PSM is aware that the service is not alive, since *ServiceAlive* returns false. Thus, according to Protocol 2, a new service composition procedure is then triggered to recover the PS. Finally, we can define PSMP by composing the above protocols together. The robustness of PSMP will be validated in Section 3.3.1.

Protocol 9. (Pervasive Service Management Protocol, PSMP) *PSMP is a composite protocol that describes interactions between PSM, PHM, and Worker Nodes to realize reliable Pervasive Services.*

$$PSM[ps] \triangleq PA[ps]; (SC_{PSM}[ps]||NSU_{PSM}[ps]) \tag{3.20}$$

$$PHM[ph] \triangleq PA[ph]; (SC_{PHM}[ph]||SSU_{PHM}[ph]) \tag{3.21}$$

$$W[w] \triangleq PA[w]; (SC_{W}[w]||HB[w]||LM[w]) \tag{3.22}$$

3.2.4 Security

This sub-section presents the mechanisms used to ensure several security issues in PerSAM/PSMP. The costs of employing security mechanisms are: 1) the efficiency of services is degraded, and 2) setting up security policies, authentication, and authorization is labor intensive and may cause inconveniences to users. These mechanisms are independent of the original design and therefore they are optional.

Confidentiality

Since PSMP is designed based on HTTP, it is able to ensure data confidentiality based on SSL/TLS [49] and WS-Security [9]. In fact, the UPnP security profile [54] adopts this approach. However, the devices (nodes) in Smart Homes typically have limited computing resources such as network bandwidth, CPU, and memory. As a result, Symmetric-Key Encryption mechanisms such as DES/Triple DES [124, 25] or AES [45] are considered more feasible. For example, ZigBee [16] uses AES encryption with

128-bit key length. The major challenge of using a Symmetric-Key Encryption is how to transmit the secret key over a unsecured network. In the residential mode, ZigBee chooses to ignore the potential vulnerability.

One possible solution is to distribute the secret key using Asymmetric-Key Encryptions. As a result, the following key exchanging procedure is proposed for ensuring data confidentiality in PSMP:

1. A new Manager Node called Security Manager which is responsible for keeping track of public keys as well as the security policies of PerNodes has to be developed and deployed.

2. PerNodes have to be configured so that each of them has a embedded private key as well as a corresponding public key. The key pairs is set up in a Security Console [54] (identical to the Security Manager in this thesis) and can be re-configured by users. Also, the user has to set up a secret key for symmetric encryption through the Security Console.

3. When performing PA, a node sends its public key without encryption to the multicast address.

4. When the Security Manager receives a public key embedded in a PA message, it encrypts the secret key by using the received public key and then sends the encrypted secreted key back to the newly joined node.

5. After the node receives the encrypted secret key, it decrypts the key by using its private key. Now, the node is able to send and receive encrypted data based on Symmetric-Key Encryption mechanisms such as AES by using the secreted key.

Figure 3.11 depicts the overall process of registering the public key and acquiring the secret key in PSMP.

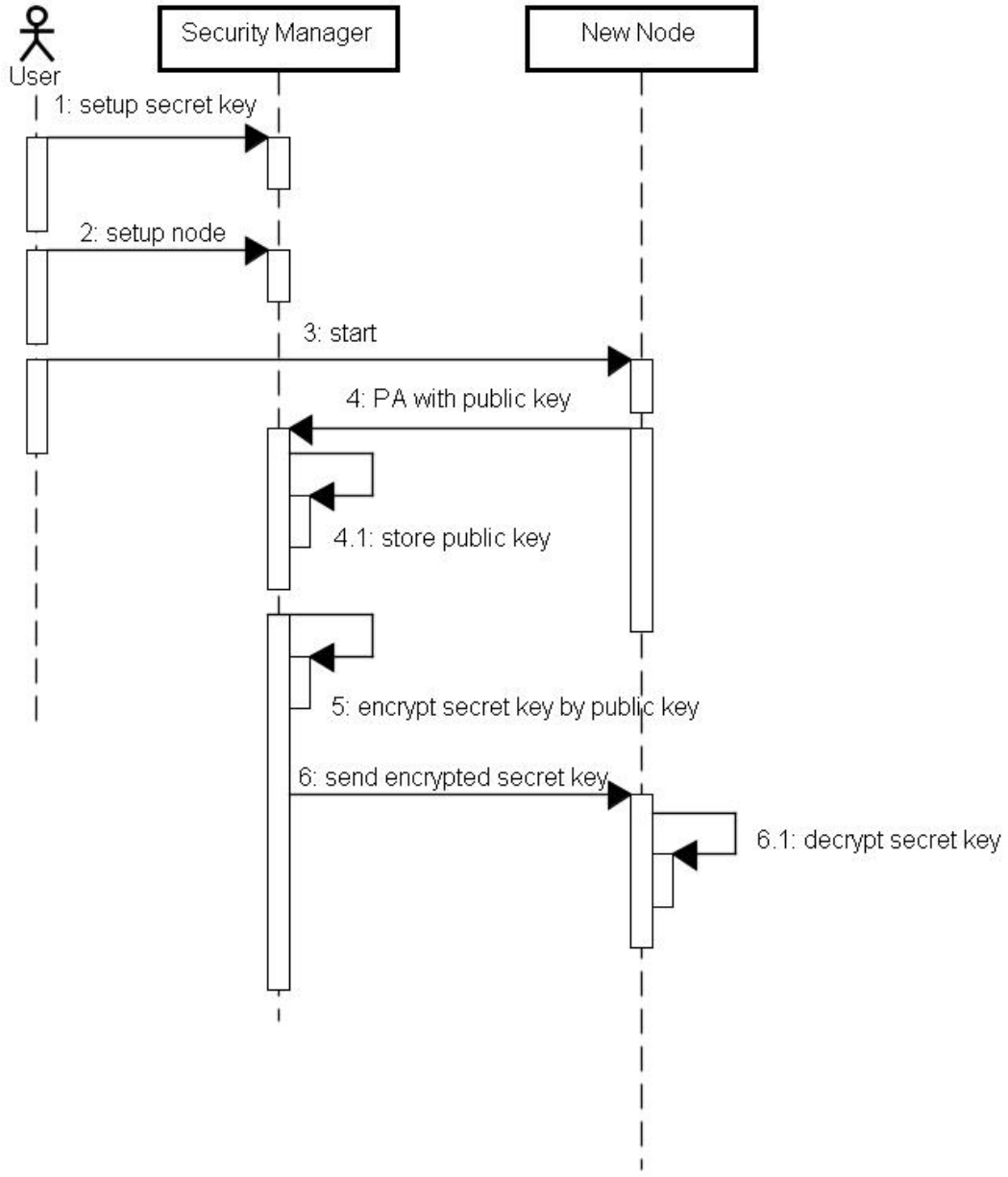


Figure 3.11: Registering the public key and acquiring the secret key in PSMP

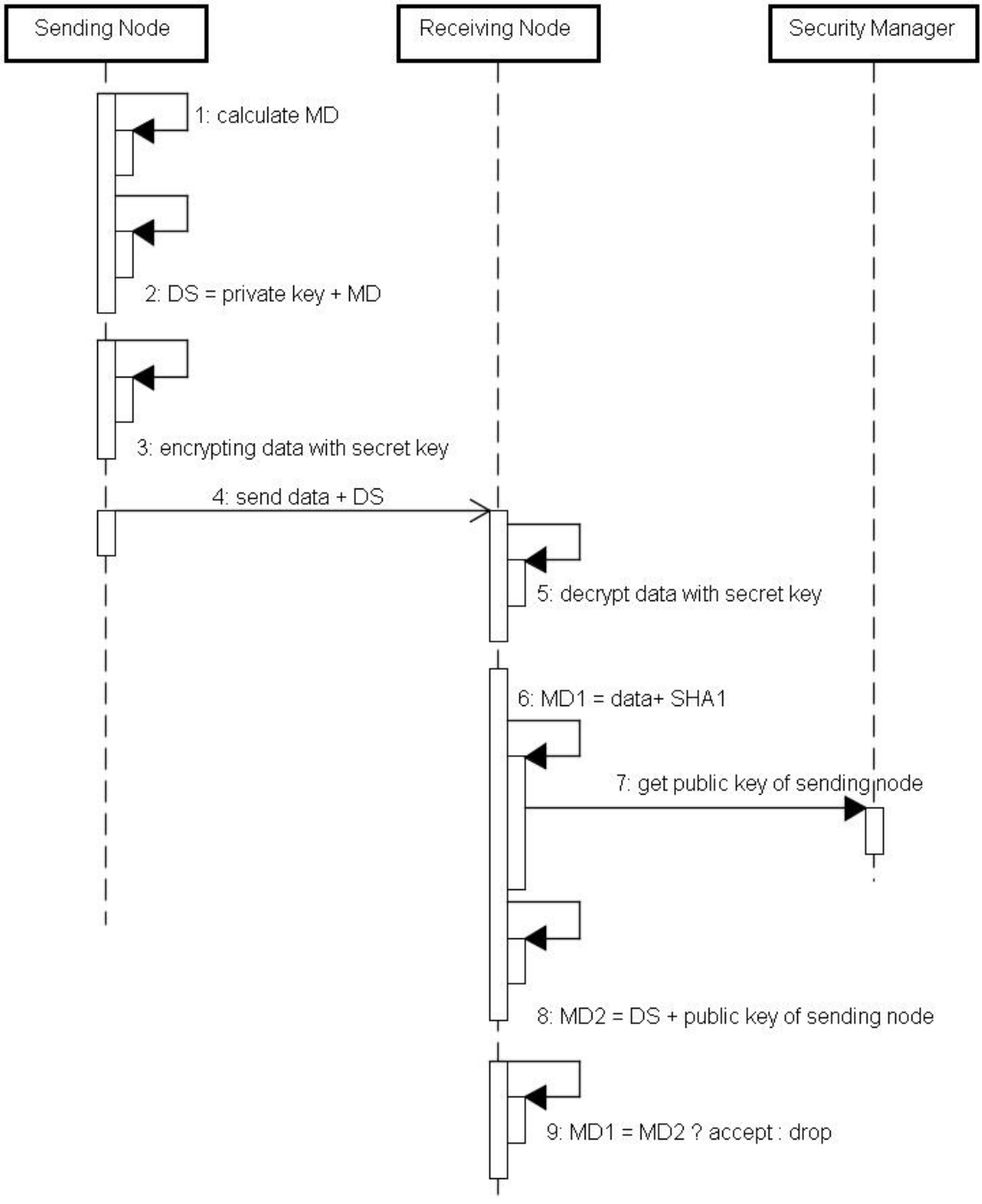


Figure 3.12: Sending and receiving data in PSMP

Integrity and Non-repudiation

Data integrity refers to the mechanisms that prevent the transmitted data from being corrupted or modified, whereas non-repudiation refers to sender of a message is actually the one claimed in that message. Integrity and non-repudiation are realized by using the message digest and digital signature mechanisms. To ensure data integrity and non-repudiation in PSMP, the sending node first obtains a message digest by using hash algorithms such as SHA (Secure Hash Algorithm) [51]. The digital signature can be generated by encrypting the message digest using the private key of the sending node, which is then placed in the header of the message before it is sent. After the receiving node receives the message, it first obtains a message digest from the decrypted message and then compares it with the one obtained by decrypting the digital signature. Finally, the receiving node can then ensure that the message is sent from a specific sender if the message digests are identical. Figure 3.12 depicts how PSMP ensures integrity and non-repudiation when sending and receiving encrypted data.

Authentication and Authorization

In order to support authentication and authorization, each PerNode has to be enhanced according to the UPnP Security Ceremonies [54]. Specifically, every node has an additional DeviceSecurity Service which supports authentication and authorization functionalities. The security policies are also pre-configured by users in the Security Console.

3.3 Evaluation

This section reports the results of evaluating PerSAM and PSMP. The following subsections explain the evaluations with respect to robustness, recovery capability, performance, cost and limitation.

3.3.1 Robustness

The purpose of this sub-section is to show that services in PerSAM/PSMP are robust, that is, to validate that (3.15) holds, which is stated as follows:

Theorem 1. *PerSAM/PSMP-based Pervasive Services are robust.*

Before presenting the proof, it is helpful to define an auxiliary function $\eta : E \rightarrow Boolean$ that maps an CSP event to a logical assertion, where E is a set of CSP events. For example, $\Diamond\eta(e)$ represents the fact that an event e eventually happens.

Lemma 1. *A failed Worker Node does not send any PA after the failure occurs eventually.*

Proof. From (A2), all messages are guaranteed to be delivered to their sinks; and a message does not appear in the network unless a node sends one, hence the following statements hold:

$$\eta(c!x) \Rightarrow \Diamond\eta(c?x), \text{ and} \tag{3.23}$$

$$\neg\eta(c!x) \Rightarrow \Diamond\neg\eta(c?x). \tag{3.24}$$

Assume there is a Worker Node w^f fails, according to (A1), (3.22) and (3.16) must stop working. From (3.5), no PA will be sent from w^f after this failure. As a result, we have:

$$\eta(Fail(w^f)) \Rightarrow \Diamond\neg\eta(\hat{m}!ssdp^{alive}[w^f]). \quad \square \tag{3.25}$$

Lemma 2. *After w^f fails, it will eventually be removed from its affiliating Pervasive Service ps and a suspecting message with respect to w^f will be sent.*

Proof. From (3.24) and from (3.25), the following statement holds.

$$\Diamond\neg\eta(\hat{m}?ssdp^{alive}[w^f]) \tag{3.26}$$

Thus, from (3.17) and from (3.26), $Refresh(w^f)$ never executes after the failure, which causes $Timeout(w^f)$ returns true. Finally, according to (3.18), $Remove(w^f)$ and $\hat{m}!psmp^{suspect}[w^f]$ will occur. To sum up, the following statements can be deduced from (3.26):

$$\Diamond\neg\eta(Refresh(w^f)) \Rightarrow \Diamond Timeout(w^f) \Rightarrow$$
$$\Diamond\eta(Remove(w^f)) \wedge \Diamond\eta(\hat{m}!psmp^{suspect}[w^f]). \quad \square \tag{3.27}$$

Lemma 3. *Eventually, the failure of w^f will be detected, causing the PS of w^f becomes unavailable.*

Proof. By definition, $Remove(w^f)$ causes $W^{ps} := W^{ps} - w^f$, thus $w^f \notin W^{ps}$. In addition, since $w^f.nt \in ST^{ps}$ since originally w^f is a community member of PS. By combining (3.1) and (3.9), we have:

$$\Diamond\eta(Remove(w^f)) \Rightarrow \Diamond w^f.nt \in MT^{ps}$$
$$\Rightarrow \Diamond MT^{ps} \neq \phi \Rightarrow \Diamond\neg ServiceAlive(). \tag{3.28}$$

By combining (3.25), (3.26), (3.27), and (3.28), a failure is eventually detected, namely,

$$\eta(Fail(w^f)) \Rightarrow \Diamond\neg ServiceAlive(). \quad \square \tag{3.29}$$

Lemma 4. *Eventually, PSM finds alternative nodes by performing node discovery for the type of w^f.*

Proof. The results is readily obtained from (3.9) and (3.28):

$$\Diamond\neg ServiceAlive() \wedge \Diamond w^f.nt \in MT^{ps}$$
$$\Rightarrow \Diamond\eta(\hat{m}!ssdp^{msearch}[w^f.nt]). \quad \square \tag{3.30}$$

Lemma 5. *Eventually, there must be some alternative nodes of the type $w^f.nt$ that respond to the node discovery.*

Proof. It is easy to observe from (3.23) and (3.30) that the following statement holds:

$$\Diamond\eta(\hat{m}!ssdp^{msearch}[w^f.nt]). \tag{3.31}$$

In addition, (A5) states that $\forall nt \in ST^{ps}$, $\exists w : w.nt = nt$. Since $MT^{ps} \subseteq ST^{ps}$, the following statement can be obtained:

$$\forall nt \in MT^{ps}, \exists w : w.nt = nt. \tag{3.32}$$

Since $w^f.nt \in MT^{ps}$, from (3.32), there must be at least one alternative node w^r such that $w^r.nt = w^f.nt$, in other words, $\exists w^r.nt = w^f.nt$. By combining (3.10), (3.11), (3.12), and (3.32), the following statement holds:

$$\begin{aligned} &\forall nt \in MT^{ps}, \\ &\exists w^r : w^r.nt = nt \wedge \Diamond\eta(\hat{u}!ssdp^{resp}[w^r]). \quad \square \end{aligned} \tag{3.33}$$

From (3.13), (3.23) , (3.33), and the definition of *Add* (see Table 3.4),

$$\begin{aligned} &\Diamond\eta(\hat{u}?ssdp^{resp}[w^r]) \Rightarrow \Diamond\eta(MT^{ps} := MT^{ps} - w^r.nt) \\ &\Rightarrow \Diamond MT^{ps} = \phi \Rightarrow \Diamond\eta(W^{ps} := W^{ps} \cup w^r) \\ &\Rightarrow \Diamond w^r.state = ACTIVE. \end{aligned} \tag{3.34}$$

From (3.2) and (3.33), $\Diamond ServiceAlive()$ holds. By combining (3.29), (3.30), (3.31), (3.33), and (3.34), the following statement holds:

$$\eta(Fail(w^f)) \Rightarrow \Diamond\neg ServiceAlive() \wedge \Diamond ServiceAlive() \quad \square \tag{3.35}$$

Consequently, it can be concluded from (3.35) that PerSAM/PSMP-based Pervasive Services are robust.

3.3.2 Recovery Rate

Several simulations were performed to study the PS recovery rates of Aura PIP (Prism Interaction Protocol) [125] and PSMP under different failure rates of Worker Nodes.

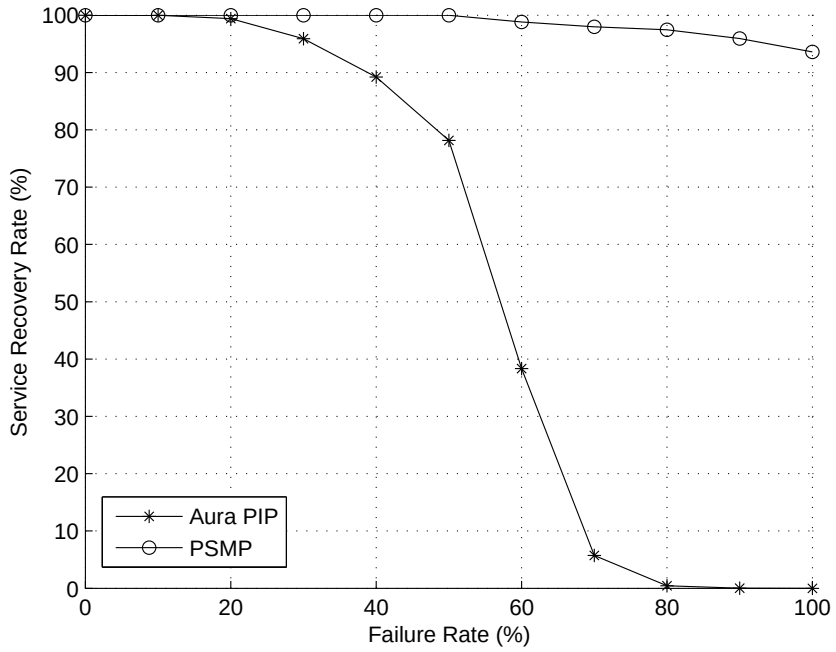


Figure 3.13: The PS recovery rates of Aura PIP and PSMP under various failure rate (NT=25)

Aura PIP is re-implemented based on the FSP specified in [125]. The simulation environment consisted of 100 Service Components, 10 Services, and 3 Hosts. Each Service consisted of 3 Service Components with different node types. Initially, node types were evenly distributed to the Service Templates and to the Service Components. After that, all Services were composed and made available.

In this experiment, it is assumed that each failure is independent, since in MOM, the failures can be isolated. The exponential distribution is used as the failure model for the experiments:

$$f(t) = \begin{cases} \lambda e^{-\lambda t}, \text{ for } t > 0 \text{ and } \lambda > 0 \\ 0, \text{ otherwise,} \end{cases} \tag{3.36}$$

where the experiment starts at time $t = 0$, and $f(t)$ is the probability density function that a node fails at time unit, namely, $t + \Delta t$. The probability that a node fails within

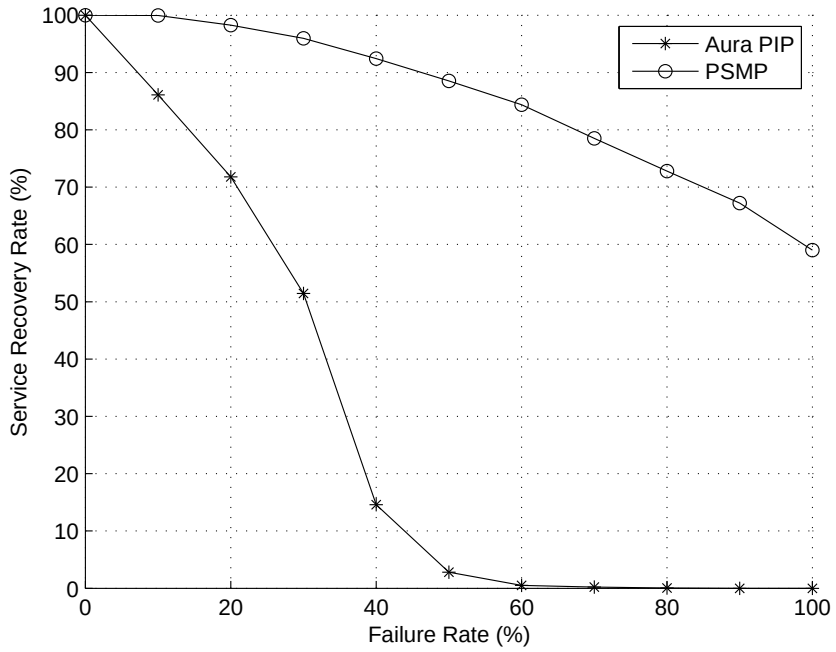


Figure 3.14: The PS recovery rates of Aura PIP and PSMP under various failure rate (NT=50)

the time interval $(0, t]$ can be obtained by integrating f from 0 to t:

$$F(t) = P(T \leq t) = \int_0^t f(u)du, \tag{3.37}$$

where T is a random variable denoting the time to failure. From (3.36), the probability that a node does not fail after time t is therefore

$$R(t) = 1 - F(t) = P(T > t) = \int_t^\infty f(u)du = e^{-\lambda t}, \text{ for } t > 0 \tag{3.38}$$

where $R(t)$ is the reliability function of this experiment.

From (3.37), $f(t)$ can be defined as the difference per time unit t when t is very small:

$$f(t) = \frac{d}{dt}F(t) = \lim_{\Delta t \to 0} \frac{F(t + \Delta t) - F(t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{P(t < T \leq t + \Delta t)}{\Delta t}, \tag{3.39}$$

The failure rate function $z(t)$ is then defined by the probability that a node fails at

$t + \Delta t$ given that the node is still functional at time t.

$$z(t) = \lim_{\Delta t \to 0} \frac{P(t < t \leq T + \Delta t | T > t)}{\Delta t},$$

from (3.39) and (3.38),

$$z(t) = \lim_{\Delta t \to 0} \frac{P(t < T \leq t + \Delta t)}{P(T > t)} \frac{1}{\Delta t} = \lim_{\Delta t \to 0} \frac{P(t < T \leq t + \Delta t)}{\Delta t} \frac{1}{R(t)} = \frac{f(t)}{R(t)}. \quad (3.40)$$

Combining (3.36), (3.38), and (3.40), the failure rate function for this experiment can be derived as follows:

$$z(t) = \frac{f(t)}{R(t)} = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda, \text{ for } t > 1. \quad (3.41)$$

Consequently, the failure rate function $z(t)$ is constant in the experiment. Note that the afore-mentioned definitions of $f(t)$, $F(t)$, $z(t)$ and $R(t)$ are adopted from the classical system reliability theory [112]. Marvin et al. also point out that it is reasonable to use a constant failure rate if an item is in the useful life period of an empirical bathtub curve [112].

In each experiment, Service Components were randomly terminated (crashed) according to a constant failure rate λ. Besides, among the failures, there are 40% of unrecoverable hardware failures. Specifically, 40% of the failures were hardware or network interface failures, and thus were unrecoverable by software-based mechanisms. The experiments were performed 1000 rounds under each failure rate. Then, the average recovery rate is reported. Note that the recovery rate is the percentages of recovered Pervasive Services after no further service composition is observed in the system.

Figure 3.13 and Figure 3.14 show the influences of Service Components failure rate on the service recovery rates when the number of node types is 25 and 50, respectively. The recovery rates of Aura PIP dropped rapidly when the failure rates of Service Components increased. On the contrary, the recovery rates of PSMP decreased much slower. The results suggest that PSMP is superior to Aura PIP in the recovery capability. It is noteworthy that with PSMP, significant portion of Services were able

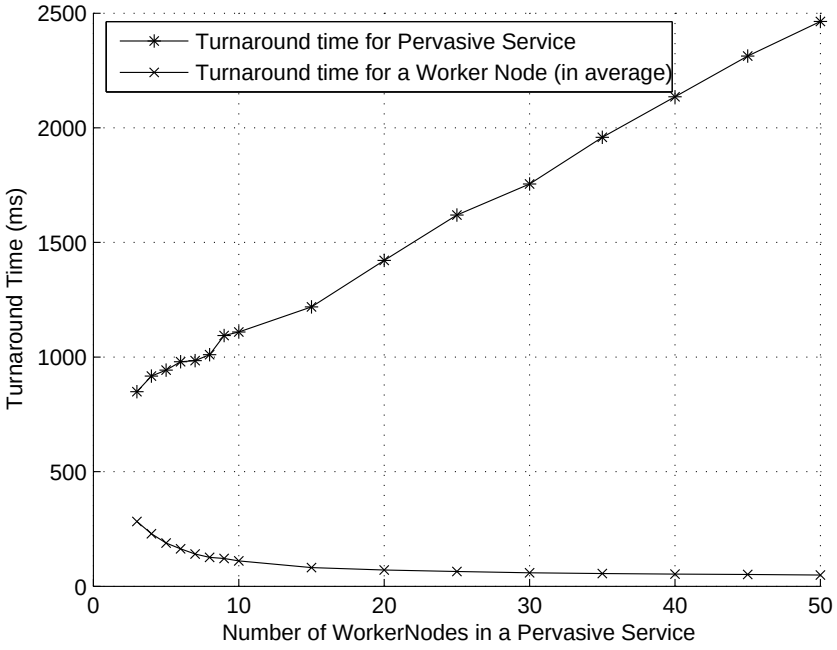


Figure 3.15: Performance of PSMP service composition

to be recovered even when the failure rates arrived 100%. This is because Aura PIP only discovers nodes that are already loaded into memory (i.e. in `DORMANT` or `ACTIVE` state). As opposed to Aura PIP, PSMP is capable of discovering the nodes that are in `INSTALLED` states via PHMs. By comparing Figure 3.13 and Figure 3.14, it can also be concluded that the number of node types (NT) has great impact on the recovery rate since it affects the number of alternative node for each node type.

3.3.3 Performance

This sub-section evaluates the performance of PerSAM/PSMP by conducting experiments in realistic home network. Experiments consisted of two parts: 1) in the first experiment, the objective is to measure the turnaround time of service composition under different service lengths, and 2) Experiments on measuring the recovery time helped us to investigate the trade-offs between eviction threshold k (see Fig. 3.16) and

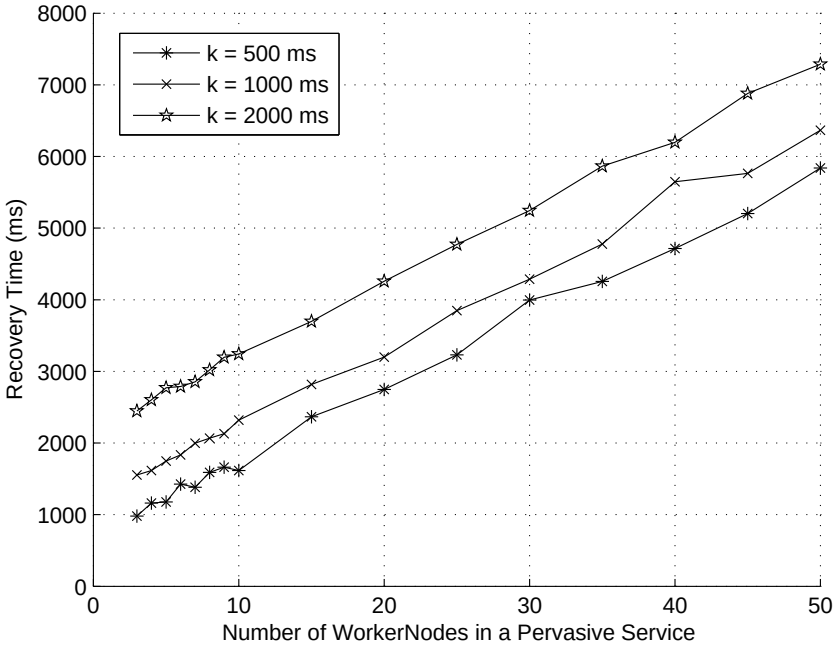


Figure 3.16: Performance of PSMP failure detection and recovery

the recovery time.

All nodes were deployed on Knopflerfish 2.0.1 OSGi servers, which were evenly distributed over three P4 1GHz mini-PCs in the same LAN with 1G bytes memory. The environment consisted of 1 PS and 3 PHs. In each PH we deploy 50 Worker Nodes, whose node types were configured so that all PS can be composed successfully. We obtained the turnaround time of service composition by measuring the time from the PSM was started to the time when all required Worker Nodes were activated. After that, we increased the size of PS and re-performed the tests. The experiments were performed 100 times under each configuration of service length and then the average percentages of recoverable PSs were reported. The results are shown in Figure 3.15. The turnaround time of service composition increased linearly when the number of Worker Nodes in PS increased. In our experiences, most real-world PSs consist of less than 10 nodes. Hence we can observe in Figure 3.15 that most real-world PSs require

less than 1.2 seconds before it is available. It is also interesting to note that due to each node were executed in parallel, the average turnaround time for each node in a PS decreased as the service length increased. The second experiment was performed in similar ways with the previous experiment except that after the PS was composed, one Worker Node was randomly terminated. After that, we recorded the time from the Worker Node failed to the time when the PS is resumed. After that, we increased the size of PS and re-performed the tests. Figure 3.16 indicates the performance of failure detection and recovery. The results show that the eviction threshold k affects the recovery time. This is because k determines the upper bound of failure detection time. If k is set as 500 ms and the service length is less than 10, then the total service unavailable time is less than 2 seconds.

3.3.4 Discussion

This sub-section discusses the cost of facilitating robust service management based on PerSAM/PSMP. First of all, the hierarchical architecture can be a cost because of the inclusion of Manager Nodes. In a hierarchical approach (manager-worker), a system can monitor the status of nodes in a more centralized and effective way. However, it leads to possible single point of failure. On the contrary, decentralized approaches such as consensus protocols are usually decentralized yet not efficient, less accurate, require more overheads, and less scalable. Since the number of Worker Nodes is typically far larger than Manager Nodes. We suggest a hybrid architecture that employs a centralized approach for Worker Node and consensus-based approach for Manager Nodes. This research assumes Manager Nodes do not fail. A semi-consensus protocols that make the failures of Manager Nodes detectable and recoverable without centralized coordinators is recently developed, some initial results can be found in [78]. Second, PSMP is designed by extending SSDP. Obviously, there are interoperability

costs imposed by this approach. However, PSMP does not interfere with the traditional UPnP Devices. This is because the use of "psmp:" headers. According to the UPnP specification, traditional UPnP Devices do not process the headers other than "ssdp:".

3.4 Summary: A Running Scenario

This section summarized the service models and protocols proposed in this chapter by examining a running scenario that goes through service composition, failure detection and recovery procedures of PSMP.

Let us consider a four-node Pervasive Service $ps1$ depicted in Figure 3.5, where ST^{ps1} ={Temperature Sensor, Context Interpreter, Indoor Temperature Control Logic, Air Conditioner}. Initially, $W^{ps1} = \phi$ and $MT^{ps1} = ST^{ps1}$ ={Temperature Sensor, Context Interpreter, Indoor Temperature Control Logic, Air Conditioner}. From Def.3, $ServiceAlive() = false$, so that SC_{psm} is triggered. For example, the statement $\hat{m}!ssdp^{msearch}[TemperatureSensor]$ indicates that an m-search message is emitted to search nodes of the type "Temperature Sensor" (refer to Figure 3.9, Step 1). After that, SS_{psm} is triggered, which listens for the responses from qualified nodes. Let us assume that node A is a "Temperature Sensor" node and that it responds to the discovery request (Figure 3.9, Step 2). Supposing that FCFS selection policy is used, from (3.10) and (3.13), node A is selected (Figure 3.9, Step 6). Hence, MT^{ps1} ={Context Interpreter, Indoor Temperature Control Logic, Air Conditioner} and $W^{ps1} = \{A\}$. In similar way, m^{ps1} is able to discover node C, D, and F with the node type Context Interpreter, Indoor Temperature Control Logic, and Air Conditioner, respectively, and causes $MT^{ps1} = \phi$; $W^{ps1} = \{A, C, D, F\}$. Finally, according to (3.13), SA_{psm} is triggered (Figure 3.9, Step 7-8), and thus $\forall w \in W^{ps1}, w.state = ACTIVE$. Now that $ServiceAlive() = true$ and that $ps1$ is successfully composed.

After $ps1$ is composed, its affiliating Worker Nodes perform heartbeats (3.16) peri-

odically (see Figure 3.10, step 1 and step 2). In (3.3), there is a set T^{ps} used to store the previous timestamps of heartbeat for each node. If node A fails, then node A eventually stops heartbeat, causing (3.17) to emit a suspect message against A and $MT^{ps1} = \{A\}$ (Figure 3.10, step 3). According to (3.19), upon receiving suspect message, $phm1$ removes A from memory (Figure 3.10, step 4). Finally, $ServiceAlive() = false$ and then SC_{psm} is triggered again in order to re-compose $ps1$. Given that node B is a "Temperature Sensor" node and that it responds to the discovery in the first place; then node B is activated and $ps1$ is recovered.

Chapter 4

Efficiency Boosting Schemes for UPnP-based Smart Home Networks

The previous chapter introduces a message-oriented service model for pervasive services and an UPnP/SSDP-based service management protocol, namely, PerSAM and PSMP, to address the flexibility and robustness issues of pervasive systems in the Smart Home. In Chapter 3, it is assumed that all messages are guaranteed to be successfully delivered and that a message does not appear in the network unless a node sends one. Nevertheless, SSDP relies on UDP which is very likely to lose packets under heavy traffic. Thus, causing PerSAM/PSMP becomes invalid when a network is busy. To alleviate this problem, the UPnP specification suggests broadcasting SSDP messages repeatedly for 2 or 3 times. Unfortunately, this approach tends to make network traffic even heavier. Furthermore, the situation is getting worse if the heartbeat rate is increase to achieve higher availability.

To investigate issues mentioned above, network simulations have been conducted using the NS-2 network simulator [74]. In these simulations, all nodes are connected by network links with bandwidth 100Mbps and 5ms delay, and all nodes are connected to a switching device through these links. Drop Tail is used for queue management. The sizes of SSDP packets are normally distributed from 200 to 450 bytes. Random noises are also introduced in the scheduled packet departure times to avoid collisions. Each node emits an SSDP packet every 500 milliseconds. As shown in Figure 4.1, there is a rapid increase in the rate of packet loss when the number of nodes exceeds 50. Also, the system is nearly unusable after the number of nodes exceeds 100. One can alleviate packet loss problem of UDP by reducing unnecessary traffic. More specifically,

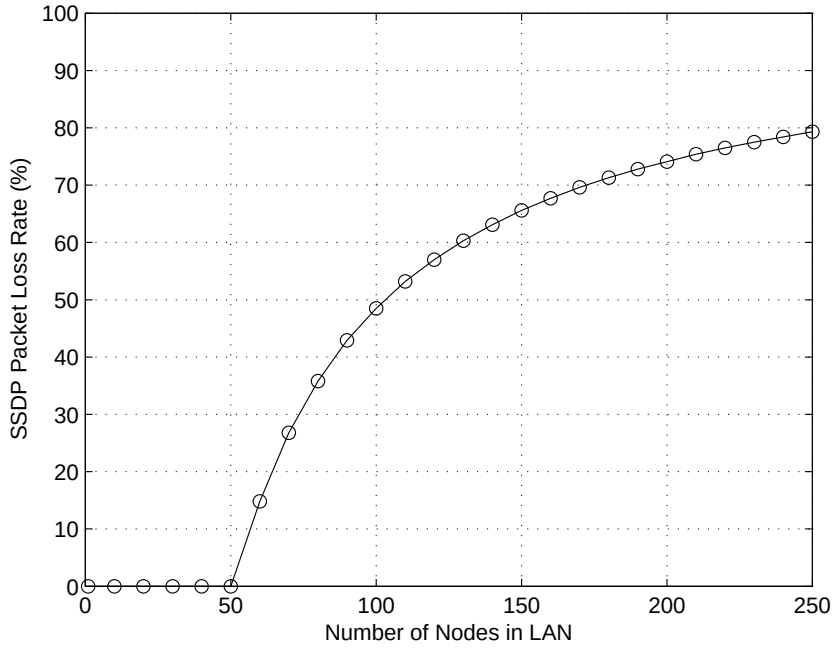


Figure 4.1: Packet loss rate with various number of nodes in a typical UPnP-based local area network

PerSAM is group-based and comprises a 2-layer hierarchy (see Figure 3.1) of nodes. On the contrary, an UPnP network is flat and peer-based, where all peers share a multicast address, so that the broadcasting nature of SSDP tends to flood the network with unnecessary packets. This is because not all nodes need to receive all messages. In fact, more sophisticated traffic dissemination techniques can help to reduce unnecessary messages.

Since UPnP/SSDP is designed for general use, an UPnP network is peer-based, where all peers share a multicast address, so that the broadcasting nature of SSDP tends to flood the network with unnecessary packets. Given the characteristics of an MOM-based pervasive system, we can deal with the packet loss problem by introducing several traffic reduction schemes. MOM-based services are usually composed of a group of nodes that form a "service chain". Hence, most of the traffic is in-group communication. In other words, not all nodes need to receive all messages. In fact, more

sophisticated traffic reduction schemes can help to eliminate unnecessary messages. The objective of this chapter is therefore to investigate the UPnP/SSDP protocol elements in an MOM-based UPnP network and then to devise possible enhancements for them. According to the specification, UPnP/SSDP can be divided into two parts: advertisement and search. Thus, we propose two schemes, Decomposing Multicast Traffic (DMT) and Service-based Node Searching (SNS), to deal with the efficiency issues of the advertisement part and the searching part, respectively. Besides, we also introduce Heartbeat by DMT (DMTH) and On-Demand Heartbeat (ODH) that are able to greatly reduce the heartbeat traffic. The target environment of the proposed schemes is a typical home network, which is an Ethernet-based LAN (Local Area Network) with one router and few switching devices. Besides, it is worthy to point out that although the proposed schemes are implemented based on PerSAM/PSMP, they are also applicable to all service networks where some form of group-based management mechanisms can be enforced.

Note that this chapter involves issues stemming from the network layer and the application layer. Consequently, depending on the context of discussion, we use the terms "packet" and "message" interchangeably, both of which are units of data transmitted over the network. The term "packet" is used when the discussions focuses on the network layer, whereas the term "message" is used in the application layer.

4.1 Assumptions and Term Definitions

Before turning to a closer examination of these techniques, it is helpful to explain assumptions and the definitions of terms used in the following sections. Three assumptions are made in the analysis of communication complexity in this chapter:

1. **The packet size is not taken into account:** The reason for this assumption is that, contrary to media streaming protocols, the packet size of SSDP is small

(typically 200 bytes to 450 bytes), and can be transmitted with single UDP packet. The theoretical limitation of the size of an UDP packet is 65527 bytes.

2. **Each endpoint in a LAN is occupied by exactly one UPnP Device:** UPnP relies on IP Multicast, where UPnP is an application layer protocol whereas IP-multicast is a network layer mechanism. Therefore, it is possible that more than one UPnP Devices reside in the same endpoint which is a network-connected appliance with an IP address. From UPnP's point of view, each message is forwarded to all UPnP Devices whereas from IP Multicast's point of view, an IP Multicast-enabled switching device only forwards received packets to all endpoints, instead of all UPnP Devices residing in these endpoints. Specifically, when one sender sends an SSDP message to more than one UPnP Device residing in the same endpoint, although all these n UPnP devices receive n copies of messages, only one message is actually passed through the network to the endpoint. The endpoint is responsible for dispatching the message to all residing UPnP Devices. To simplify and to clarify the communication complexity analysis, we assume each endpoint is occupied by exactly one UPnP Device, that is, by one PerNode. In addition, to avoid introducing too many constants in the analysis results, we also ignore the message that are dispatched locally, namely, when one sender sends an SSDP message, then it is replicated n times instead of $n-1$ times. This assumption can cause inaccuracy of the predictions when the number of Worker Nodes is small.

3. **All Pervasive Services can be successfully activated eventually and each Worker Node participates in at least one Pervasive Service:** The purpose of this assumption is to ensure the Equilibrium of Load Factors (see Theorem 2) holds so that one can reduce the variables of the results, hence making them more tractable. In fact, the proposed techniques do not depend on this assumption.

The enhanced protocols are still more efficient than the original ones in respect of communication complexity even if the above assumption does not hold. Especially, when there are some Worker Nodes which do not participate in any Pervasive Service, the hereby obtained results are even better. For example, in the original protocols, the Worker Nodes that do not participate in any Pervasive Service still send useless heartbeat messages, whereas in the enhanced ones, these nodes do not send any message at all, and hence causing better results.

To facilitate further analysis and discussions, we define several concepts by extending the service model mentioned in Section 3.1 in the following:

Definition 7. (Cardinality Function) *The cardinality function* $n : X \to \mathbb{N}$ *returns the cardinality of the set* X*.*

For example, the number of all Worker Nodes can be denoted by $n(W)$, where W is the universe of Worker Nodes. Likewise, S is the universe of Pervasive Services and $n(S)$ is the number of Pervasive Services in the system.

Definition 8. (Service Length) *The length of a Pervasive Service* s *is denoted as* $\ell(s)$ *which is the number of Worker Nodes in* s*. The value of* $\ell(s)$ *can be obtained by calculating the cardinality of* W^s*, that is,* $\ell(s) = n(W^s)$*, where* W^s *is the set of Worker Nodes belonging to* s*.*

From Definition 8, the average length of all Pervasive Services in the system, $\bar{\ell}$, can be obtained by:

$$\bar{\ell} = \frac{1}{n(S)} \cdot \sum_{s \in S} \ell(s) = \frac{1}{n(S)} \cdot \sum_{s \in S} n(W^s) \tag{4.1}$$

Definition 9. (Contribution) *The contribution of a Worker Node* w*, denoted as* $\lambda(w)$*, is the number of Pervasive Services in which the Worker Node* w *participates, where* $\lambda \in \mathbb{N}$ *and* $0 \leq \lambda \leq n(S)$*.*

Note that $\lambda(w) = n(S)$ when w participates in all Pervasive Services in the system, indicating that w is highly contributive. On the contrary, $\lambda(w) = 0$ when w does not participate in any Pervasive Service. According to Definition 9, the average contribution of all Worker Nodes is:

$$\bar{\lambda} = \frac{1}{n(W)} \cdot \sum_{w \in W} \lambda(w) \tag{4.2}$$

Intuitively, the contribution of a node w is the labors it supplies, whereas the number of required Worker Nodes is the labors a Pervasive Service demands.

To activate all Pervasive Services, we require at least $n(S) \cdot \bar{\ell}$ labors, since each Pervasive Service requires $\bar{\ell}$ Worker Nodes. The most efficient way to activate all Pervasive Services is to strike a balance between the supplied labors and the demanded labors. The labors provided by all Worker Nodes are therefore $\sum_{w \in W} \lambda(w)$. Thus, we have the following theorem:

Theorem 2. (Equilibrium of Load Factors) *If Assumption 3 holds then all Pervasive Services are activated most efficiently if and only if the following equation holds:*

$$\sum_{w \in W} \lambda(w) = n(S) \cdot \bar{\ell}. \tag{4.3}$$

For example, if there are two Pervasive Services, and each of them is of length 3, then the total load factor in demand is 6. One possible solution is to employ 6 Worker Nodes and each is with $\bar{\lambda} = 1$. Alternatively, we can use 2 Worker Nodes which are equipped with better computing capabilities. In this case, each Worker Node has to work for 3 Pervasive Services, causing $\bar{\lambda} = 3$.

Consequently, from (4.2) and (4.3), one can obtain the following equation:

$$\bar{\lambda} = \frac{n(S) \cdot \bar{\ell}}{n(W)}, \text{ where } 1 \leq \lambda \leq n(S). \tag{4.4}$$

Note that $1 \leq \bar{\lambda}$ due to Assumption 3. Finally, the UPnP specification requires re-sending of messages to deal with UDP packet loss. We therefore define a repetition

factor r to represent the count of messages being re-sent, where $1 \leq r \leq 3$ is suggested by the UPnP specification. Table 4.1 is the summary of notations mentioned above and Table 4.2 summarizes additional terms and abbreviations used in this chapter. Based on the above discussions, the following sub-sections present the core idea of the proposed traffic reduction techniques as well as the analysis on how much traffic can be reduced after applying these techniques.

4.2 Decomposing the Multicast Traffic

Whenever an UPnP Device is started, it sends presence announcement (PA) messages to a multicast address to inform other UPnP Devices about its presence. The PA messages that are sent to the multicast address are replicated and then propagated to all UPnP Devices in the UPnP Network. According to the UPnP specification [15], a PA demands $3 + 2d + k$ messages, where 3 messages are used to describe the specific information about the UPnP Device, d is the number of embedded UPnP Devices, and k is the number of UPnP Services. As mentioned in Section 3.2, a PerNode is identical to an UPnP Device with one UPnP Service in an UPnP Network. Thus, $d = 0$ and $k = 1$ because that a PerNode does not have any embedded device and that a PerNode has one UPnP Service (see Section 3.2). Consequently, a PerNode demands 4 (i.e. $3+2\cdot 0+1$) messages for PA. Furthermore, the UPnP specification also suggests re-sending of messages with a pre-determined repetition factor r, which is usually 2 or 3, to deal with UDP packet loss. Let us denote the SSDP multicast address and the 4 PA messages as $\hat{m}^{ssdp}$, x_1, x_2, x_3, and x_4, respectively. Then, the original presence announcement protocol (PA^{orig}) can be formally described as follows:

$$PA^{orig} \triangleq \coprod_{r} \left[\begin{array}{ll} \hat{m}^{ssdp}!x_1 & \rightarrow \hat{m}^{ssdp}!x_2 \rightarrow \\ \hat{m}^{ssdp}!x_3 & \rightarrow \hat{m}^{ssdp}!x_4 \end{array} \right] \rightarrow SKIP. \tag{4.5}$$

Table 4.1: Notations for communication complexity analysis

Notation	Description
s	A Pervasive Service
S	The set of all Pervasive Services in the system
w	A Worker Node
W	The set of all Worker Nodes in the system
W^s	The set of Worker Nodes belonging to Pervasive Service s
$n(X)$	Number of elements in the set X
$\ell(s)$	Length of the Pervasive Service s
$\bar{\ell}$	Average length of all Pervasive Services in the system
$\lambda(w)$	Contribution of a Worker Node w
$\bar{\lambda}$	Average contribution of all Worker Nodes in the system
r	Repetition factor
$\hat{m}$	A multicast address
$\hat{\mu}$	An unicast address
x,y,z,x^*,y^*	messages

Table 4.2: Additional acronyms used in this chapter

Abbreviation	Full Name
DMT	Decomposing Multicast Traffic
SNS	Service-based Node Searching
DMTH	Heartbeat by Decomposing Multicast Traffic
ODH	On-Demand Heartbeat
TRR	Traffic Reduction Ratio

Recall that there is one PSM for each Pervasive Service, in other words,

$$\forall s \in S, n(m^s) = 1, \tag{4.6}$$

so that the quantity of PSM instances is the same as the quantity of Pervasive Services

$$\sum_{s \in S} n(m^s) = \sum_{s \in S} 1 = n(S). \tag{4.7}$$

From (4.7), it can be concluded that there are totally $n(W)+n(S)$ nodes in the network. To perform PA, 4 messages are sent (i.e. x_1, x_2, x_3, and x_4) and totally $4 \cdot r$ messages are sent if the repetition factor r is taken into account. Due to the effect of multicasting, the $4 \cdot r$ PA messages are replicated for $n(W) + n(S)$ times to be forwarded to all nodes in the network. Hence, there are totally $4 \cdot r \cdot (n(W) + n(S))$ messages replicated. To sum up, from (4.5) we can conclude that the communication complexities of sending and replicating PA messages are $4 \cdot r$ and $4 \cdot r \cdot (n(W) + n(S))$, respectively.

It is important to observe that although multicast is believed to be more efficient than broadcast, it is not the case from the UPnP network's points of view since all UPnP Devices in the network share the same multicast address. In other words, communication complexity of the multicast is actually identical to broadcast in a UPnP network. So far as the hierarchical structure of PerSAM is concerned, this design is inefficient. In PerSAM, only PSMs are interested in receiving PA messages, and therefore traffic can be reduced by assigning different multicast addresses for different types of receivers. This technique is called Decomposing the Multicast Traffic, or simply DMT, in the sequel. As a result, two new multicast addresses, $\hat{m}^{psm}$ and $\hat{m}^{w}$, are created for messages to be received by PSMs and Worker Nodes, respectively.

Furthermore, sending 4 PA messages is unnecessary, either. As discussed in Section 3.2, the structures of PerNodes and UPnP Devices are identical. Thus, one message, denoted as x^*, is sufficient to convey the information describing the structure of a

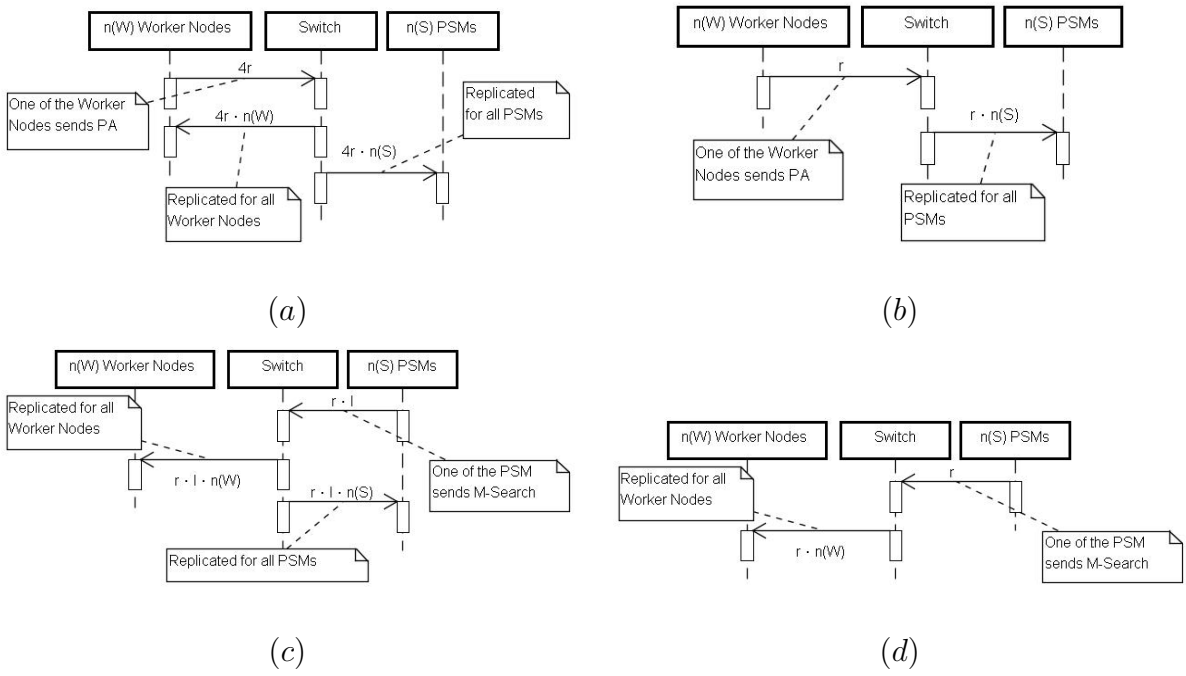


Figure 4.2: Sequence diagrams of PA/LA and node searching protocols: (a) Original PA; (b) PA after applying DMT; (c) Original node searching; (d) Node searching after applying SNS

PerNode. The enhanced protocol (PA^{new}) is shown as follows:

$$PA^{new} \triangleq \coprod_{r} \hat{m}^{psm}!x^* \rightarrow SKIP. \tag{4.8}$$

For PA^{new}, there is only one PA message sent to $\hat{m}^{psm}$, which are then received by all PSMs. As mentioned earlier, there are $n(S)$ PSMs in the network, so that messages are also replicated for $n(S)$ times. By considering repetition factor, the communication complexities of sending and replicating PA messages after applying DMT become r and $r \cdot (n(S))$, respectively. Note that the same results can be obtained in the case of leave announcement.

4.3 Service-based Node Searching

After a Pervasive Service is activated, a PSM first searches for qualified Worker Nodes by sending M-Search messages for each required node type to $\hat{m}^{ssdp}$, where an M-Search message describes a required node type (see Listing 4.1).

Listing 4.1: A typical SSDP M-Search message

```
M-SEARCH * HTTP/1.1
ST: urn:schemas-upnp-org:device:sensor:1
MX: 3
MAN: "ssdp:discovevr"
HOST: 239.255.255.250:1900
```

A Pervasive Service has ℓ members (Definition 8), and each of them has distinct node type, so that totally ℓ M-Search messages are sent. A Worker Node responds to the PSM immediately when its node type is identical to the one given in the M-Search message. After gathering at least one qualified Worker Nodes for each node type, a PSM then selects and activates the best ones among these candidates. The above-mentioned protocol is called node searching (NS^{orig}) which is listed in (4.9). Note that an M-Search message is denoted as y and SA denotes the service selection and activation protocol (see Protocol 5).

$$ND^{orig} \triangleq \coprod_{\ell} \hat{m}^{ssdp}!y \rightarrow SA \tag{4.9}$$

In this protocol, a PSM sends an M-Search message for each required node type. Thus, in order to find all required node types for a Pervasive Service, $\bar{\ell}$ messages are sent in average. These messages are broadcasted to all nodes, so that the messages are replicated for $\bar{\ell} \cdot (n(W) + n(S))$ times. Again, the repetition factor r is taken into account, causing the average communication complexities of sending and replicating messages to be $r \cdot \bar{\ell}$ and $r \cdot \bar{\ell} \cdot (n(W) + n(S))$, respectively.

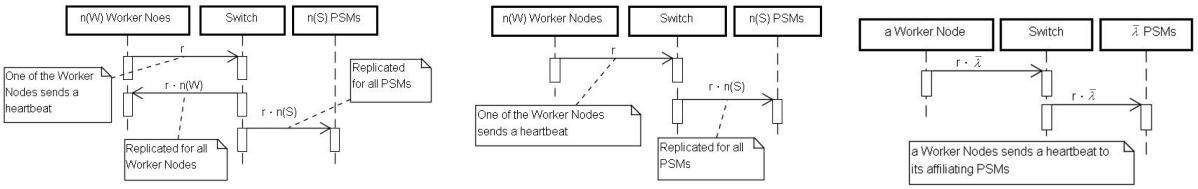
Note that the required node types are known in advance, and hence, instead of sending ℓ M-Search messages individually, the search request belonging to the same service can be sent in a batch. Specifically, all required node types of a Pervasive Service can be bundled into one message by which the message counts are reduced to $1/\bar{\ell}$ in average. For example, the aggregated M-Search message shown in Listing 4.2 is capable of specifying several node types at the same time. In Listing 4.2, the MAN header is changed to "psmp:discover" to prevent non-PSMP devices from processing aggregated M-Search messages. In this scheme, the node types specified in the ST header belong to the same service. Hence, this scheme is called Service-based Node Searching or SNS. Also note that only Worker Nodes need to receive M-Search messages, so that DMT (see Section 4.2) can also be applied. In short, the number of M-Search messages of a Pervasive Service now becomes one, denoted as y^*, and DMT is applied by sending the message to $\hat{m}^w$, which replicates messages only for Worker Nodes. The enhanced protocol, denoted as NS^{new}, is shown below.

$$ND^{new} \triangleq \hat{m}^w!y^* \rightarrow SA \tag{4.10}$$

From (4.10), it can be concluded that to find all required node types for a Pervasive Service, one aggregated message is sufficient, which is then replicated for $n(W)$ times because that the message sent to $\hat{m}^w$ are forwarded to Worker Noes. Finally, if the repetition factor is considered, then r messages are sent and $r \cdot n(W)$ messages are replicated.

Listing 4.2: An aggregated M-Search message

```
M-SEARCH * HTTP/1.1
ST: urn:attentivehome-org:sensor:temperature:1,
    urn:attentivehome-org:logic:aircon:1,
    urn:attentivehome-org:actuator:fan:1
MX: 3
MAN: "psmp:discover"
HOST: 239.255.255.250:1900
```



(a) (b) (c)

Figure 4.3: Sequence diagrams of heartbeat protocols:(a) Original heartbeat protocol; (b) After applying DMTH; (c) After applying ODH.

It is important to point out that the size of y^* depends on $\bar{\ell}$. When $\bar{\ell}$ is too large, the aggregated message can exceeds MTU or even the maximum size of an UDP. However, the proposed scheme works well in most practical cases. To show this, assuming that the average M-Search message size without the ST header is $\bar{\beta}$ and that average size of ST header is $\bar{\tau}$. Then, the average size of y^* is

$$\bar{\beta} + \bar{\ell} \cdot \bar{\tau}. \tag{4.11}$$

In real world, it is reasonable to assume that the average size of M-Search messages without ST header $\bar{\beta}$ is less than 300 bytes, the average size of ST headers $\bar{\tau}$ are less than 100 bytes and the average service length $\bar{\ell}$ are less than 10. In the extreme case where $\bar{\ell} = 10$, $\bar{\beta} + \bar{\ell} \cdot \bar{\tau} = 1300$ bytes, which is still less than MTU (1500 bytes). If y^* exceeds MTU but not the UDP packet limitation (65527 bytes), then it takes more frames to transmit the message in the data link layer. However, the overall traffic is still reduced by $(\bar{\ell} - 1) \cdot \bar{\beta}$ bytes since the total size of contents to be transmitted is $\bar{\ell} \cdot \bar{\beta} + \bar{\ell} \cdot \bar{\tau}$.

4.4 Reducing the Heartbeat Traffic

Although SSDP does not provide heartbeat mechanism, however, it can be simulated by sending one PA messages every few seconds which can be described by the following

CSP statement:

$$HB^{orig} \triangleq \coprod_{r} \hat{m}^{ssdp}!z \rightarrow HB^{orig}, \tag{4.12}$$

where z is used to denote the heartbeat message. Unfortunately, this approach tends to flood the network since the messages are broadcasted to all peers. In each heartbeat, one message is sent to m^{ssdp}, which is then replicated for $n(S) + n(W)$ times, causing the message count of sending and replicating messages to be r and $r \cdot (n(S) + n(W))$, respectively, if the repetition factor is taken into account.

Again, since only PSMs are interested in knowing the status of Worker Nodes, DMT can be employed by sending messages to $\hat{m}^{psm}$ instead of $\hat{m}^{ssdp}$ so that only $n(S)$ messages are replicated per heartbeat. Therefore, the number of sent messages and that of replicated messages can be reduced to r and $r \cdot n(S)$, respectively. The following CSP statement shows the enhanced protocol, which is referred to as Heartbeat by Decomposing Multicast Traffic (DMTH).

$$DMTH \triangleq \coprod_{r} \hat{m}^{psm}!z \rightarrow DMTH \tag{4.13}$$

However, DMTH is still not optimal, since heartbeat messages are forwarded to all PSMs, whereas not every PSM is interested in the status of every Worker Node. For example, when $\bar{\lambda} = 1$, each Worker Node participates in exactly one Pervasive Service, then for each heartbeat, $n(S) - 1$ out of $n(S)$ heartbeat messages are useless. It is desirable to ensure the heartbeat messages being only sent to the needed PSMs. The mechanism that realizes this idea is called On-Demand Heartbeat (ODH). By using ODH, a PSM asks all affiliated Worker Nodes to keep track of its remote reference after they are activated. Note that the remote reference of a PSM is encoded in the messages that are used to activate Worker Nodes, and then the Worker Nodes send heartbeat back according to these references. The resulting protocol is shown in (4.14), where $\hat{\mu}^s$

is a remote reference of a Pervasive Service s in which the Worker Node participates.

$$ODH \triangleq \coprod_{r} \hat{\mu}^{s}!z \rightarrow ODH \tag{4.14}$$

In this protocol, a Worker Node only sends heartbeat messages to the demanding PSMs. Thus, the message count depends on how many Pervasive Services does a Worker Node takes part in. In other words, $\bar{\lambda}$ messages are sent in average for each heartbeat. Upon arriving at the switching device, the messages are also replicated for $\bar{\lambda}$ times and then are forwarded to their destinations. As a result, the numbers of messages sent and replicated are both $r \cdot \bar{\lambda}$ times after the repetition factor is considered. It is important to note that ODH induces overheads when sending messages by a factor of $\bar{\lambda}$, so that ODH is only effective when the average load factor of Worker Nodes $\bar{\lambda}$ is small. More specifically, although ODH reduces the replicated messages to $\frac{\bar{\lambda}}{n(S)+n(W)}$, it also sends more messages than HB^{orig} by $\bar{\lambda}$ times. In the worst case, where $\bar{\lambda} = \bar{\ell} = n(S) = n(W)$, ODH only saves replicated messages by $\frac{1}{2}$, since

$$\frac{\bar{\lambda}}{n(S)+n(W)} = \frac{n(S)}{n(S)+n(S)} = \frac{1}{2}, \tag{4.15}$$

whereas the messages sent by ODH is still $\bar{\lambda}$ times more than HB^{orig}, where $1 \leq \bar{\lambda} \leq n(S)$ (see (4.4)), causing the traffic to be heavier. In this case, the system should use DMTH instead, which reduces the replicated messages by $\frac{1}{2}$, but the number of messages sent is the same as HB^{orig}. Consequently, one solution is to switch between DMTH and ODH depending on which of them is more efficient, i.e.,

$$HB^{new} \triangleq \coprod_{r} [\ ODH\ \diamond\ DMTH\ ], \tag{4.16}$$

where $\diamond$ is a CSP deterministic choice operator which means that one of the two processes will be executed and it can be decided deterministically depending on the system context. In (4.16), the choice is made based on the value of $\bar{\lambda}$ against a threshold, which is calculated based on the ratio between saved messages and the overheads

produced by ODH, namely,

$$\frac{n(W)+n(S)}{\bar{\lambda}} > \bar{\lambda} \Rightarrow \bar{\lambda}^2 < n(S)+n(W), \tag{4.17}$$

where $1 \leq \bar{\lambda} \leq n(S)$. Consequently, it is more efficient to use ODH when $\bar{\lambda} < \sqrt{n(S)+n(W)}$. Otherwise, DMTH is a better alternative.

4.5 Evaluation

This section concentrates on evaluations of the proposed techniques for reducing traffics produced by PerSAM/PSMP. The proposed techniques are first evaluated analytically and then the NS-2 simulation results are presented. Meanwhile, we also validate the consistencies between analysis results and simulation results. Finally, the results of experiments in a small scale network are reported.

Before taking a closer look of the analysis results, we first introduce Traffic Reduction Ratio (TRR), which estimates the traffic reductions of the proposed approaches.

Definition 10. (Traffic Reduction Ratio) *The Traffic Reduction Ratio (TRR) is defined as:*

$$TRR(P_{orig}, P_{new}) = 1 - \frac{\tau(P_{new})}{\tau(P_{orig})}, \tag{4.18}$$

where $\tau : P \rightarrow \mathbb{N}$ returns the number of messages produced by a protocol P, and P_{orig} and P_{new} denote the original protocol and the proposed protocol, respectively. Notice that TRR is negative if the proposed method increases the message count.

It can be observed from Definition 10 that the proposed techniques are more effective when TRRs are higher. For instance, if original 100 messages are produced and TRR equals 50%, then only 50 messages are produced after applying the proposed technique.

4.5.1 Communication Complexity

Table 4.3, 4.4, 4.5 and 4.6 summarize the traffic reductions of several protocols after applying the proposed techniques. Note that the details of calculating message counts of these protocols have been mentioned in Section 4.2, Section 4.3, and Section 4.4. This sub-section focuses only on calculating the TRRs of the proposed techniques.

Presence Announcement and leave Announcement From Table 4.3, it is obvious that the messages sent by original PA protocol are reduced by $\frac{3}{4}$, since $1-\frac{r}{r/4\cdot r}=\frac{3}{4}$. Similarly, the ratio of replicated messages between PA^{orig} (4.5) and PA^{new} (4.8) are

$$\frac{r\cdot n(S)}{4\cdot r\cdot(n(W)+n(S))}.$$

From (4.4), we can substitute $n(W)$ by $\frac{\bar{\ell}}{\bar{\lambda}}\cdot n(S)$, thus we have

$$\frac{1}{4\cdot(\bar{\ell}/\bar{\lambda}+1)}.$$

The TRR of replicated messages is therefore

$$1-\frac{1}{4\cdot(\bar{\ell}/\bar{\lambda}+1)}.$$

Since larger $\bar{\ell}$ or smaller $\bar{\lambda}$ both cause TRR to be larger, the advantage of the enhanced protocol (PA^{new}) is greater when the average length of Pervasive Services ($\bar{\ell}$) increases and when the average contribution ($\bar{\lambda}$) decreases. Empirically, $\bar{\ell}$ ranges from 3 to 5, and $\bar{\lambda}$ is close to 1. Hence, we can expect that the traffic reductions ranges from $\frac{1}{16}$ to $\frac{1}{24}$. In the worst case, where $n(S)=n(W)=\bar{\lambda}=1$, PA^{new} still reduces the replicated messages by $\frac{1}{8}$. Note that the proposed approach and the analysis results also applies to the leave announcement protocol.

Node Searching Based on Table 4.4, in respect of sending messages, the TRR between ND^{orig} (4.9) and ND^{new} (4.10) is:

$$1-\frac{r}{r\cdot\bar{\ell}}=\frac{\bar{\ell}-1}{\bar{\ell}}.$$

Table 4.3: Traffic Reductions after applying the Decomposing Multicast Traffic

	PA^{orig}	PA^{new}	TRR
Sent	$4 \cdot r$	r	$\frac{3}{4}$
Replicated	$4 \cdot r \cdot (n(W) + n(S))$	$r \cdot n(S)$	$1 - \frac{1}{4 \cdot (\bar{\ell}/\bar{\lambda}+1)}$

Table 4.4: Traffic Reductions after applying Service-based Node Searching

	NS^{orig}	NS^{new}	TRR
Sent	$r \cdot \bar{\ell}$	r	$\frac{\bar{\ell}-1}{\bar{\ell}}$
Replicated	$r \cdot \bar{\ell} \cdot (n(W) + n(S))$	$r \cdot n(W)$	$\frac{\bar{\ell}+\bar{\lambda}-1}{\bar{\ell}+\bar{\lambda}}$

On the other hand, the number of replicated messages is reduced by:

$$\frac{r \cdot n(W)}{r \cdot \bar{\ell} \cdot (n(W) + n(S))},$$

which can be further reduced by substituting $n(W)$ with $\frac{\bar{\ell}}{\bar{\lambda}} \cdot n(S)$:

$$\frac{n(W)}{\bar{\ell} \cdot (n(W) + n(S))} = \frac{n(S) \cdot \bar{\ell}/\bar{\lambda}}{\bar{\ell} \cdot (n(S) \cdot \bar{\ell}/\bar{\lambda} + n(S))} = \frac{1}{\bar{\ell} + \bar{\lambda}}.$$

As a result, the TRR for replicated messages is:

$$1 - \frac{1}{\bar{\ell} + \bar{\lambda}} = \frac{\bar{\ell} + \bar{\lambda} - 1}{\bar{\ell} + \bar{\lambda}}.$$

Again, the advantage of SNS is greater both when $\bar{\ell}$ increases or when $\bar{\lambda}$ increases, since larger $\bar{\ell}$ or smaller $\bar{\lambda}$ both cause TRR to be larger. For instance, if $\bar{\ell} = 4$ and $\bar{\lambda} = 1$, then we can expect to reduce the replicated message count by $\frac{4}{5}$. In the worst case, that is, $(n(S) = n(W) = \bar{\ell} = 1)$ and $\bar{\lambda} = 1$, NS^{new} still reduces the message count of replicated messages by $\frac{1}{2}$.

Heartbeat According to the strategy proposed in Section 4.4, ODH is used when $\bar{\lambda} < \sqrt{n(S)}$ and DMTH is used otherwise. Table 4.5 and Table 4.6 summarize the

analysis results of ODH and DMTH, respectively. So far as the messages sent by Worker Nodes are concerned, the TRR of ODH is -1 since it performs worse than the SSDP for $\bar{\lambda}$ times. It is trivial that ODH reduces replicated messages by $1 - \frac{\bar{\lambda}}{n(W)+n(S)}$. Observe that since lower $\frac{\bar{\lambda}}{n(W)+n(S)}$ implies higher TRR, the smaller $\bar{\lambda}$ is, the better ODH performs. It is important to point out that since ODH is unicast-based, the message count of ODH is invariant to the service length ($\bar{\ell}$), and thus the equation (4.4) is not applicable to ODH.

On the other hand, DMTH does not produce additional traffic when it sends messages. Thus the TRR is zero. The TRR for the replicated messages are

$$\frac{r \cdot n(S)}{r \cdot (n(W) + n(S))}.$$

Again, from (4.4), we can substitute $n(W)$ by $\frac{\bar{\ell}}{\bar{\lambda}} \cdot n(S)$, thus we have

$$\frac{1}{\bar{\ell}/\bar{\lambda} + 1}.$$

The TRR of replicated messages is therefore

$$1 - \frac{1}{\bar{\ell}/\bar{\lambda} + 1} = \frac{\bar{\ell}}{\bar{\ell} + \bar{\lambda}}.$$

Similar to PA^{new}, the superiority of DMTH over HB^{orig} is greater when $\bar{\ell}$ increases and when $\bar{\lambda}$ decreases. Since $1 \leq \bar{\lambda} \leq n(S)$, so that in the extreme case mentioned in Section 4.4, where $\bar{\lambda} = n(S)$ and $2 \cdot n(S) = n(W)$, from Table 4.6 we know that the message counts are saved by $\frac{2}{3}$. On the contrary, when $\bar{\lambda} = 1$, where ODH is used, the traffic is reduced by $\frac{n(W)+n(S)-1}{n(W)+n(S)}$. On these bases we can conclude that ODH makes the system more scalable when the average contribution is low, since the traffic is greatly reduced by ODH when number of nodes increases. Consequently, the analysis results shown in this sub-section imply a great reduction in network traffic by applying the proposed techniques to UPnP Networks.

Concluding from the above analysis results, there are great reductions in network traffic by applying the proposed techniques to UPnP Networks.

Table 4.5: Traffic Reductions after applying On-Demand Heartbeat

	HB^{orig}	ODH	TRR
Sent	r	$r \cdot \bar{\lambda}$	$1 - \bar{\lambda}$
Replicated	$r \cdot (n(W) + n(S))$	$r \cdot \bar{\lambda}$	$1 - \frac{\bar{\lambda}}{n(W)+n(S)}$

Table 4.6: Traffic Reductions after applying the Heartbeat by Decomposing Multicast Traffic

	HB^{orig}	$DMTH$	TRR
Sent	r	r	0
Replicated	$r \cdot (n(W) + n(S))$	$r \cdot n(S)$	$\frac{\bar{\ell}}{\bar{\ell}+\bar{\lambda}}$

4.5.2 NS-2 Simulations

To investigate the traffic reductions of the proposed techniques, we simulated a typical home network by using the NS-2 network simulator [74] with two extensions, that is, AgentJ [131] and the IGMP extension [34, 66]. AgentJ enables NS-2 to access Java classes; the IGMP (Internet Group Management Protocol) extension to NS-2facilitates IGMP, which is required to realize the proposed techniques. The simulation parameters are set based on typical local area networks. Specifically, the simulated home network adopts a star topology, where every device and host is connected to a IGMP-capable switching device by a 100 megabits per second link with 5 ms delay. The sizes of packets are normally distributed from 200 to 450 bytes. Drop-Tail is used for queue management. The total simulation time is 120 time units for each scenario. The simulated protocols are implemented as Java classes, which can be accessed via AgentJ wrapper interfaces.

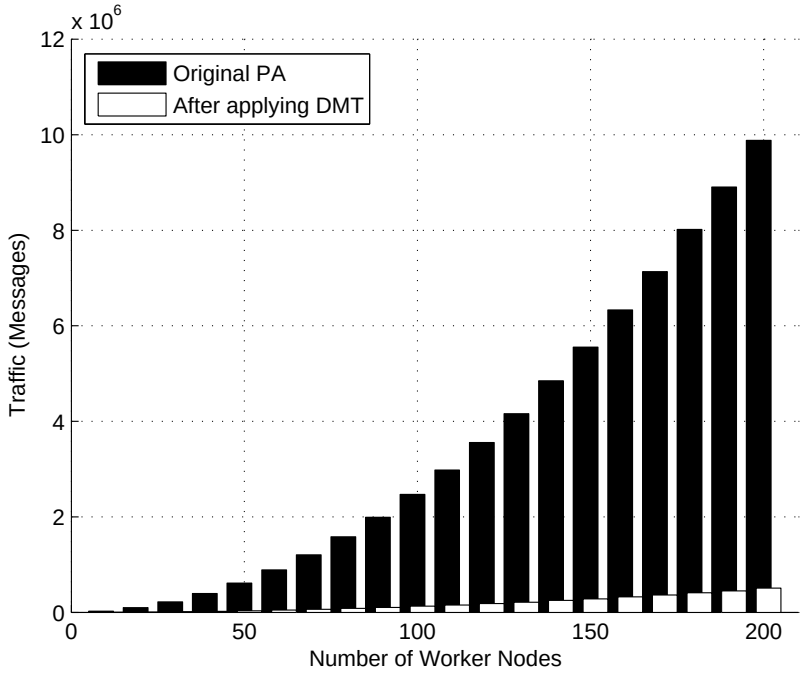


Figure 4.4: Traffic generated by presence announcement, before and after applying DMT ($\bar{\lambda} = 1$ and $\bar{\ell} = 4$)

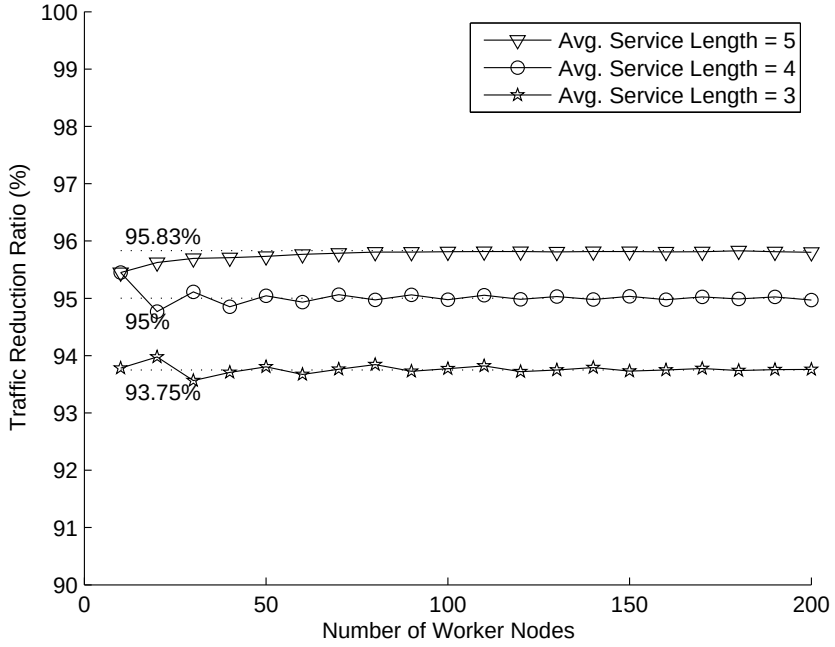


Figure 4.5: Traffic reductions of presence announcement after applying DMT

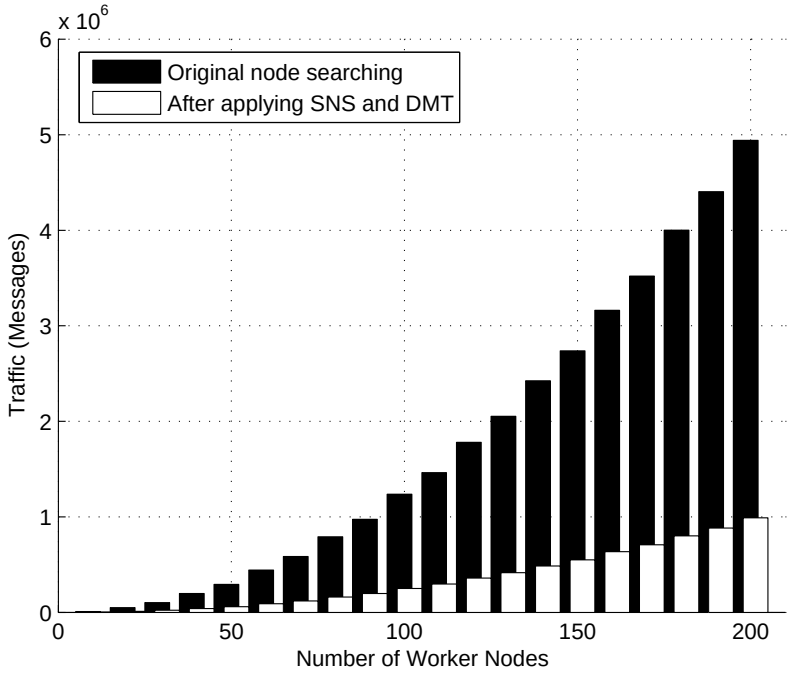


Figure 4.6: Traffic generated by the node discovery protocol, before and after applying SNS and DMT ($\bar{\lambda} = 1$ and $\bar{\ell} = 4$)

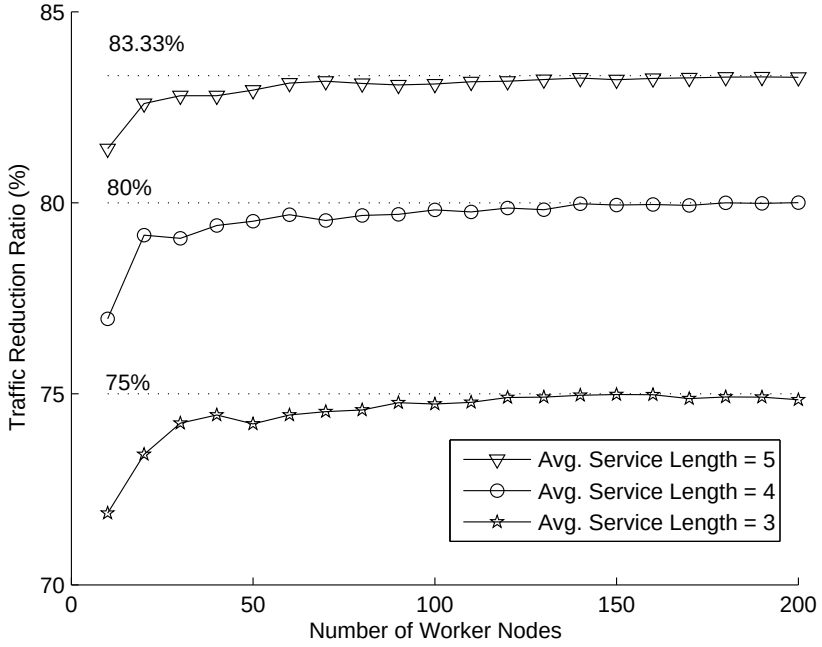


Figure 4.7: Traffic reductions of node discovery after applying SNS and DMT

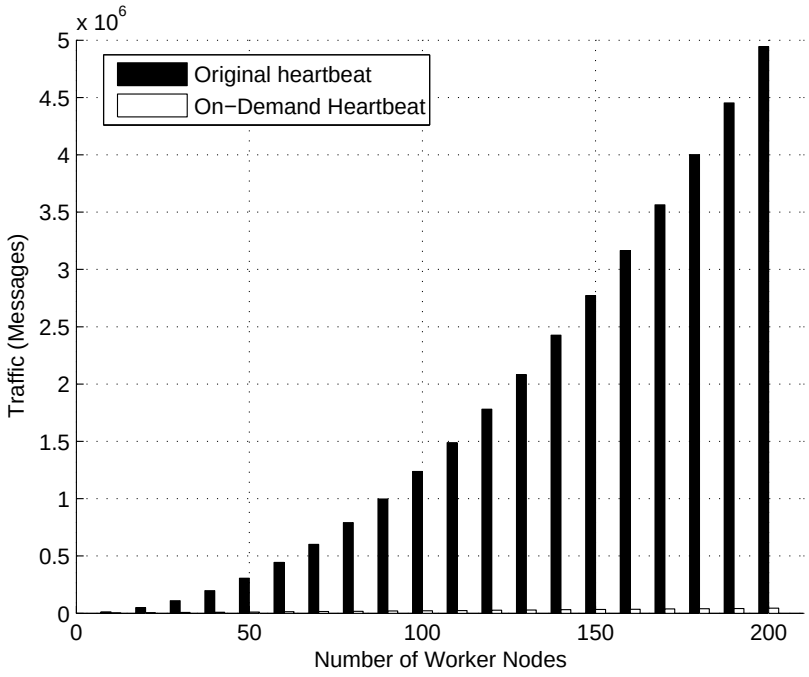


Figure 4.8: Heartbeat traffic in a light-loaded system, before and after applying ODH ($\bar{\lambda} = 1$ and $\bar{\ell} = 4$)

Presence Announcement and Leave Announcement Figure 4.4 shows the traffic generated by SSDP-based PA and by applying DMT when $\bar{\lambda} = 1$ and $\bar{\ell} = 4$. The results show that traffic can be greatly reduced after applying DMT. Similar results can be obtained when $\bar{\ell}$ equals to 3 or 5, and when DMT is applied to leave announcement. All simulations were performed under varying numbers of Worker Nodes, and then we calculated the message counts from the traces generated by NS-2. The numbers of PSMs are determined by the average service length ($n(S) = n(W)/\bar{\ell}$). Hence, if $n(W)$ is fixed in each round of simulation, then $n(S)$ decreases when $\bar{\ell}$ increases. The traffic reductions of replicated messages after applying DMT to PA under different service lengths are depicted in Figure 4.5, from which we can observe that more than 90% of replicated messages can be saved. The dotted lines in Figure 4.5 indicate the expected TRRs under different service lengths. Note that the analysis and the simulation results

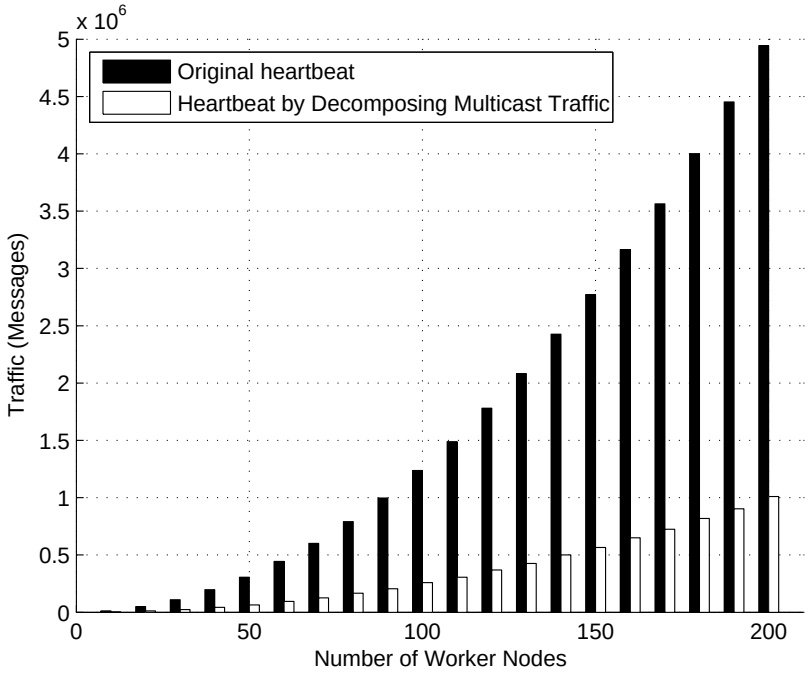


Figure 4.9: Heartbeat traffic in a light-loaded system, before and after applying DMTH ($\bar{\lambda} = 1$ and $\bar{\ell} = 4$)

are quite consistent. The results are more consistent with the analysis results when $n(W)$ is greater, since we introduce randomness to the message departure time. Similar consistencies can be perceived for the TRRs of sent messages when $\bar{\ell} = 3$, 4 and 5, which are all approaching 75%.

Node Searching In a similar fashion, we set up simulations for evaluating the enhancements after applying SNS to the node searching. As mentioned in Section 4.3, DMT is also used after aggregating messages. We can perceive from Figure 4.6 and from Figure 4.7 that the traffic can be reduced by more than 70% after applying the proposed techniques. In Figure 4.7, the dotted lines indicate the expected TRRs under different service lengths. The results are more consistent with analysis results when $n(W)$ is greater then 50. As for the TRRs of sent messages, the TRRs approach 66%, 75%, and 80% when $\bar{\ell}$ equals to 3, 4 and 5, respectively.

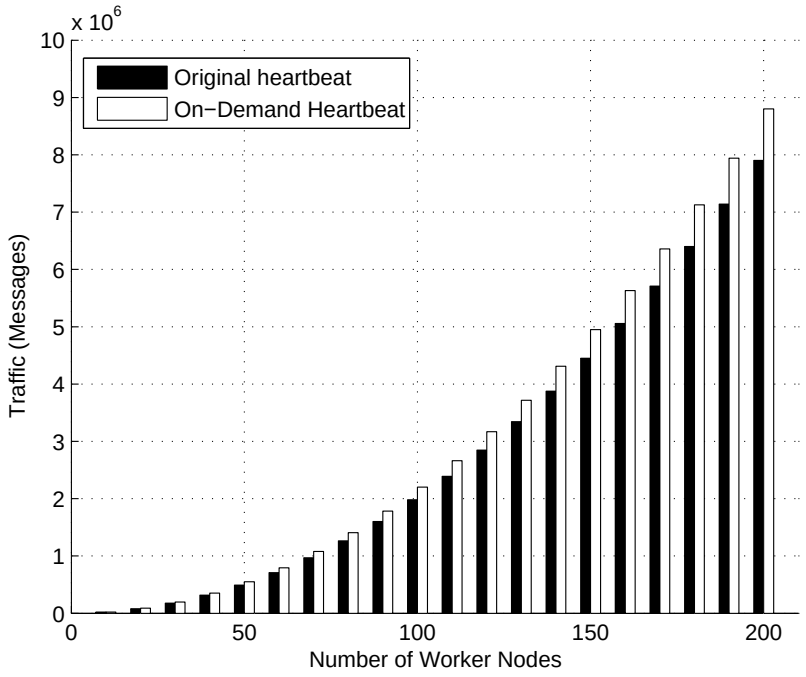


Figure 4.10: Heartbeat traffic in a heavy-loaded system, before and after applying ODH ($\bar{\lambda} = n(S)$ and $\bar{\ell} = 4$)

Heartbeat Two sets of simulations are conducted to evaluate traffic reductions of HB. Because of $1 \leq \bar{\lambda} \leq n(S)$, $\bar{\lambda}$ was set to 1 for the first set of simulations and was set to $n(S)$ for another set. Two proposed heartbeat efficiency enhancement techniques, that is, ODH and DMTH, were both applied to the original heartbeat protocol under different $\bar{\lambda}$ values. By comparing Figure 4.8 and Figure 4.9, we can observe that in a light-loaded system ($\bar{\lambda} = 1$), DTH performed much better than DMTH. On the contrary, in a heavy-loaded system, where $\bar{\lambda} = n(S)$, ODH performs worse than the original protocol (see Figure 4.10), whereas DMTH was still capable of reducing traffic by approximately 50% (see Figure 4.11). As mentioned in Section 4.4, the reason is that ODH sent additional messages when $\bar{\lambda} > 1$. Based on these results we can conclude that ODH is more suitable when $\bar{\lambda}$ is low and DMTH is more suitable for high $\bar{\lambda}$, which is consistent with the analysis presented in Section 4.5.1.

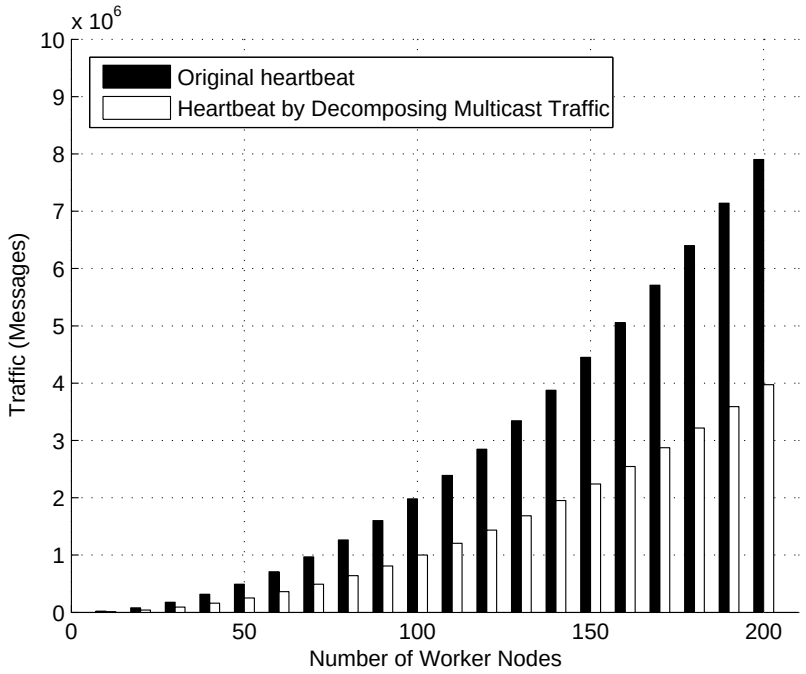


Figure 4.11: Heartbeat traffic in a heavy-loaded system, before and after applying DMTH ($\bar{\lambda} = \bar{\ell} = n(S) = n(W)$)

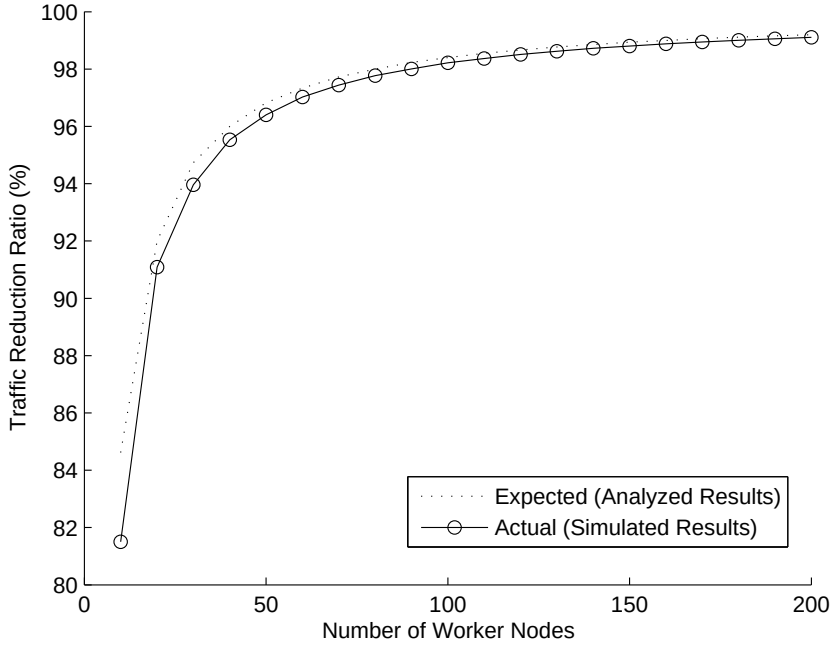


Figure 4.12: Traffic reductions of heartbeat after applying ODH when $\bar{\ell} = 4$ and $\bar{\lambda} = 1$

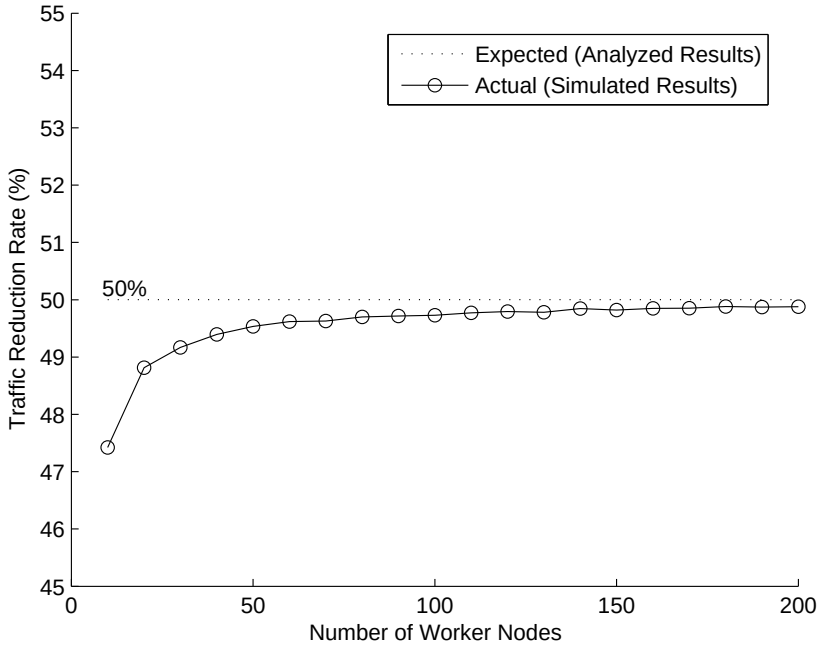


Figure 4.13: Traffic reductions of heartbeat after applying DMTH when $\bar{\lambda} = \bar{\ell} = n(S) = n(W)$

Figure 4.12 depicts the TRRs of message counts after applying ODH. In these experiments, we set $\bar{\ell} = 4$ and $\bar{\lambda} = 1$, similar results were obtained when $\bar{\ell} = 3$ and $\bar{\ell} = 5$. The traffic can be reduced by more than 95% after number of nodes exceeds 50. The dotted lines indicate the expected values of TRR obtained by analysis. Unlike in PA/LA and in ND, the expected TRRs in these simulations are fixed. The expected values of TRR increase when the number of nodes in the system grows. Figure 4.12 also reveals that the expected results are consistent with simulated results.

According to (4.16), the heartbeat protocol switches to DMTH when $\bar{\lambda}$ is greater than $\sqrt{n(S)}$. Hence, we set $\bar{\lambda} = \bar{\ell} = n(S) = n(W)$ to ensure that DMTH is chosen. Figure 4.13 shows the TRRs after applying DMTH. The results show that even the system was heavy-loaded, the traffic was still reduced by more than 47%. These results are also consistent with the analysis results, and they are more coherent when $n(W)$

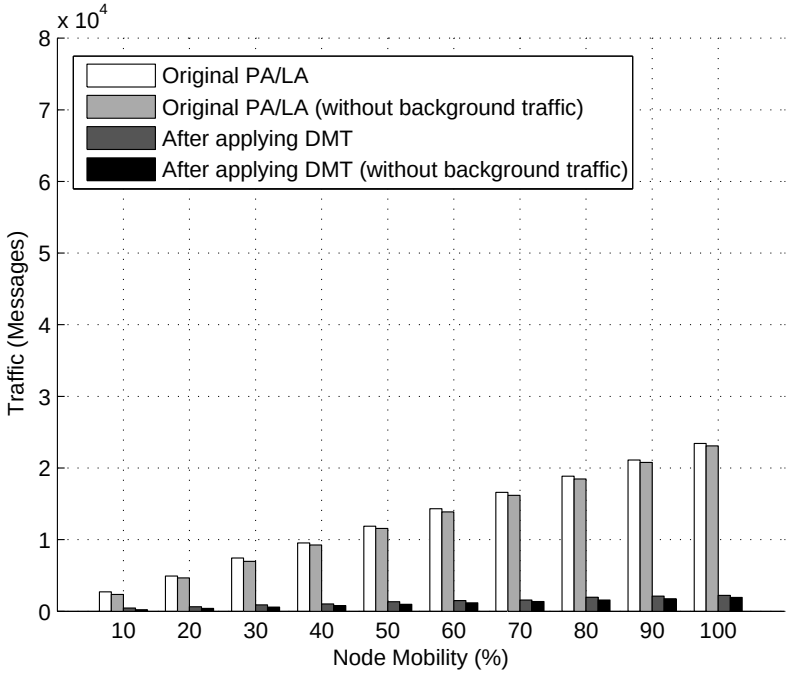


Figure 4.14: Evaluating the proposed schemes in a real home network, where $\bar{\lambda} = 1$ and $\bar{\ell} = 2$, when only PA and LA are enabled.

is greater then 100.

4.5.3 Experiments

To investigate the effectiveness of the proposed approaches when deploying in a real environment, we conducted prototype-based experiments in a small-scale switched home network. In these experiments, the original and proposed schemes are implemented and integrated. Then, we evaluate these protocols under different node mobility, that is, the frequency of leaving and joining the network, in a home network.

The home network consists of two PSMs and four Worker Nodes. Each PSM is installed on an IBM X61 notebook with Intel Core 2 Duo 1.8 GHz CPU and 2G RAM, whereas each Worker Node is installed on an IBM X31 notebook with Intel Pentium-M 1.6 GHz CPU and 512MB RAM. The machines are interconnected by an IGMP-capable

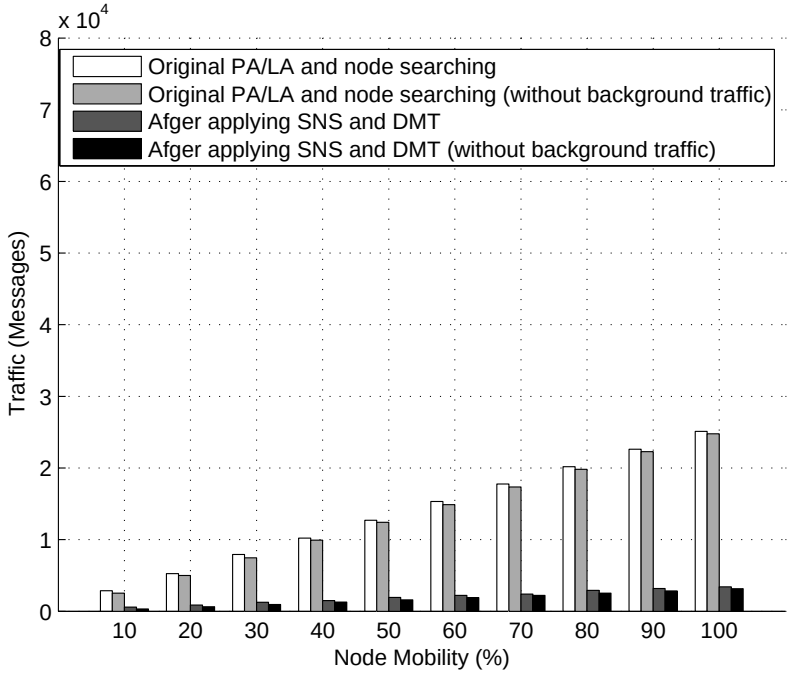


Figure 4.15: Evaluating the proposed schemes in a real home network, where $\bar{\lambda} = 1$ and $\bar{\ell} = 2$, after enabling PA, LA and node searching.

switch (D-Link DES-3526), and are able to access the internet via a router (DrayTek Voyger 2104). On each machine, an instance of Wireshark packet sniffer [1] is installed in order to capture and analyze the network traffic. In addition, to facilitate all nodes to start execution at approximately the same time and to dispatch the parameters to each node more efficiently, each PerNode is instrumented so that it receives multicast control messages sent by a centralized experiment controller. The experimental environment is configured so that there are two Pervasive Services and each of them is with $r = 1$, $\ell = 2$, and $\lambda = 1$. After being executed, a Worker Node runs for 120 time units, which is equal to the heartbeat period. According to the assigned value of node mobility, an action vector that indicates when a node should join or leave the network is generated. For example, 50% mobility causes a node to leave and then to re-join the network in 60 time units, which are randomly distributed over 120 time units. In addition, the

[1]The Wireshark packet sniffer, available at http://www.wireshark.org

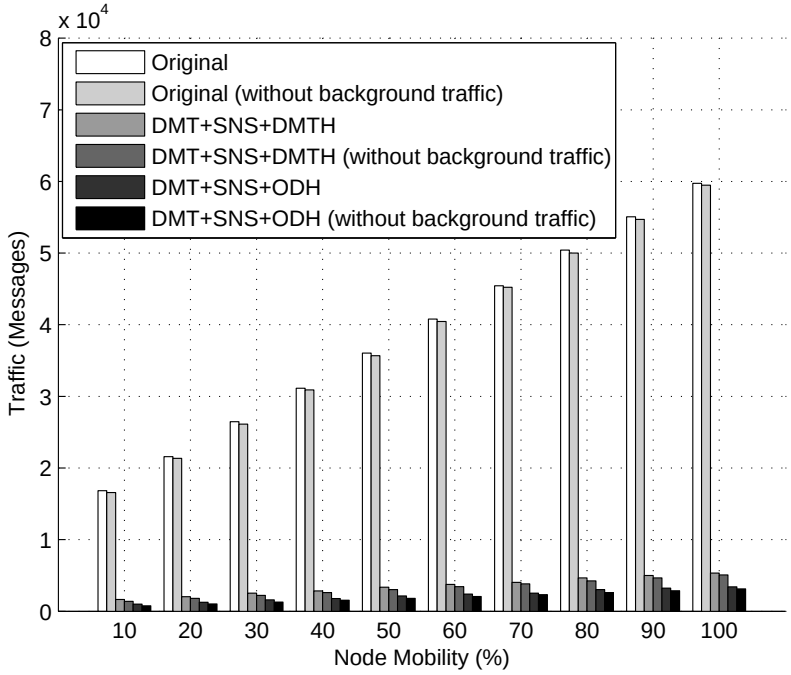


Figure 4.16: Evaluating the proposed schemes in a real home network, where $\bar{\lambda} = 1$ and $\bar{\ell} = 2$, after enabling all protocol capabilities.

heartbeat process is implemented as a separate thread in order to prevent the heartbeat periods from being interfered by the main process.

Figure 4.14, Figure 4.15, and Figure 4.16 depict the message counts when the node mobility increases. In the first experiment (Fig. 4.14), the heartbeat and node searching capabilities are turned off. The results show that even in a small scale network, the PA/LA traffic can still be greatly reduced by applying DMT. Also note that the PA/LA message count increases when the node mobility is higher, since a node performs PA/LA whenever it joins/leaves the network. In Fig. 4.15, the node searching capability is turned on, so that the network traffic slightly increases. Again, higher node mobility increases the message counts of node searching, since when PA/LA messages are observed, a PSM either finds that one of its affiliating Worker Node leaves or some nodes re-join the network. In either case, the PSM will re-compose the service

by performing node searching. Thus, the message count of node searching is roughly in proportion to the node mobility. In Fig. 4.15, the traffic does not increase much since the size of network is small. In the third experiment (Fig. 4.16), all capabilities are enabled, so that the traffic is greatly increased. In this experiment, the re-joining time is relatively short due to the scale of the network. As a result, the message counts of heartbeat are less sensitive to node mobility. On the other hand, the traffic of heartbeats decreases when the average time to re-join the network of a node is too long, since an absent node does not send any heartbeat message.

4.5.4 Discussion

This sub-section discusses costs and limitations of the techniques we have proposed so far. Currently, the proposed techniques are tightly coupled with UPnP networks. However, the core ideas of these techniques are still applicable to other service models. Specifically, DMT is useful when the protocol is based on single IP multicast channel and some unbalanced communication patterns can be observed. When information is known in advance, it is helpful to aggregate several search requests into one as long as the package size is less than a unit of transportation (theoretical size of an UDP packet is 65527 bytes). Finally, the comparisons of ODH and DMTH give us the insight that as a heartbeat mechanism, unicast is much more efficient than multicast when the number of monitors/managers are related much fewer than the number of service entities (i.e. Worker Nodes in PerSAM).

As mentioned in 4.3, when applying SNS, the size of an aggregated message can exceed MTU or even the maximum size of an UDP when $\bar{\ell}$ is too large. As a result, SNS may not work properly when the size of an aggregated message exceeds the UDP packet limit (65527 bytes). From (4.11) we can obtain the theoretical upper bound of the $\bar{\ell}$, which is approximately 652. In other words, the theoretical limitation for

applying SNS is when the average service length is smaller than 652.

Mobility is one of the most important issues in a pervasive network, since nodes can join or leave the network at anytime. While this issue has been addressed in Chapter 3, we assume that all nodes are always available in the analysis to simplify and clarify the presentation. In a highly dynamic network, where $n(W)$ and $n(S)$ change drastically, one has to add an additional time parameter, and then performs a summation over a time period. On the other hand, if $n(W)$ and $n(S)$ are relatively stable, where the number of nodes does not change significantly, then the analysis results are still good approximations of the message reductions. In these cases, given that the advertising and searching messages are emitted constantly, DMT and SNS are still able to reduce more message counts. However, the TRR (see Definition 10) is not affected. Likewise, to make the network more sensitive to the mobility, one has to increase the heartbeat rate and therefore the heartbeat traffic. As a result, DMTH and ODH are able to reduce more messages while TRR does not affected.

Compatibility can also be an issue, since the proposed approaches are extensions of UPnP/SSDP. According to the UPnP specification, UPnP Devices do not process the headers other than "ssdp:". So that we can avoid the interferences between legacy UPnP devices and PerNodes by introducing additional headers in SSDP MAN header. More specifically, PSMP uses an unique "psmp:" header to distinguish from "ssdp:" headers used by SSDP. As a result, PerSAM nodes and traditional UPnP Devices are able to co-exist in the same network without interfering with one another.

Observe that the DMT technique is only effective when the router or home gateway supports IGMP[34]. Otherwise, the routers broadcast packets to all endpoints instead of sending packets only to the listeners of multicast addresses. Recently we notice that there is an increasing number of low-end IGMP-capable routers available in the consumer markets. To name a few, D-Link DES-1228 and DrayTek Vigor 2110

are examples of IGMP-capable routers, for which the prices are less than 200 USD. Therefore, we believe that the proposed techniques will be realizable in most home networks in the near future. If there are more than one switch in the network, it will result in greatest traffic reduction if each switch is IGMP-capable. If only a few IGMP-capable switches are available, then the one with the highest performance and being IGMP-capable should be deployed at the root of the tree.

Finally, the correctness of the analysis results can be interfered when the system suffered from extremely high loads, where the message arrival rate is larger than the message consumption rate. The Worker Node starts dropping messages when the size of un-processed messages exceeds the buffer size, causing the accuracy of analysis results being affected. However, in an MOM-based pervasive system, it is reasonable to assume short data processing time since most of the tasks are I/O bound and MOM-based I/O is asynchronous (fire-and-forget). Besides, since the average service length $\bar{\ell}$ is used throughout the analysis process, the accuracy can be interfered when the variation of service lengths is large. This is because the service length affects the message count of node searching. As mentioned earlier, the service lengths typically range from 3 to 5 in practice so that the variation of $\bar{\ell}$ is limited. Likewise, the average contribution $\bar{\lambda}$ is used in the analysis. λ has great impact to the traffic of heartbeat. Therefore, it can be difficult to determine whether to use ODH or DMTH when the variation of λ is significant. In this case, DMTH is a better choice, since it does not cause negative effect, as discussed in Section 4.4.

4.6 Summary

Many popular service management protocols uses broadcast-based or multicast-based group communication mechanisms, which, if not carefully designed, tend to flood the network with unnecessary messages. Therefore, a compact service management pro-

tocol should be designed to minimize the downtime of a system while maintaining low message counts. In this chapter, we present the design, analysis, simulations, and experiments of several techniques for boosting the network efficiency - Decomposing Multicast Traffic, Service-based Node Searching, Heartbeat by Decomposing Multicast Traffic and On-Demand Heartbeat - based on PerSAM and PSMP. Both analysis results and simulation results reveal that the proposed approaches can reduce message counts of presence and leave announcement, node searching, and heartbeat by more than 93.75%, 66%, and 50%, respectively, in average service lengths.

Chapter 5

Consistent Service Composition in a Smart Home

Service composition is the process of discovering, selecting, and activating service components that best fit pre-specified criteria. It has been an object of study in Service Computing discipline for a long time [147]. Although it has been reported that service composition is also one of the most significant issues of Pervasive systems [33], existing enterprise service composition techniques are not suitable for such systems due to the following challenges that do not take place in enterprise environments:

1. **User-centricity:** As mentioned earlier, enterprise services are usually composed based on developer-defined Service Templates that are specified to fit business requirements, whereas the goal of Pervasive service composition is to compose services that meet maximum satisfaction of users.

2. **Representing and unifying user preferences:** In a Pervasive environment such as the Smart Homes, users should be able to describe their preferences according to which the service template is updated before the composition starts. Complexities arise when multiple parties submit conflicting preferences simultaneously. For example, the energy saving policy is likely to conflict with the user's comforts.

3. **Impromptu consistency:** Enterprise services are relatively static and well-planned whereas Pervasive services are usually dynamic and ad hoc. Since service components can come and go at any time, a Pervasive service must be able to search for alternatives when a required service components suddenly disappears.

Contrary to enterprise services, Pervasive services are usually deployed in ad hoc ways. Service components work perfectly individually do not guarantee that they can still work perfectly when several of service components co-exist in the same environment. Compatibility issues arise due to resource competing and interferences among different effects of service components.

To sum up, the core issue of Pervasive service composition is threefold: 1) how users specify their preferences precisely, 2) how to unify preferences submitted by multiple parties, and 3) how to detect and to avoid undesired interactions among service components. This chapter concentrates on issues mentioned above by improving the framework introduced in Chapter 3. The objective of this chapter is therefore to devise a set of techniques that facilitates the composition of consistent services, where a service is consistent if all conflicting user preferences can be unified and the service components are chosen so that the interferences among them are minimized from the users' points of view.

The following sections first introduces a formal expression language, Preference Expression (PE), that is able to precisely specify user preferences based on the CC/PP (Composite Capability/Preference Profiles) standard [84]. Note that PE denotes user preferences from two perspectives: the enumerability and the necessity. Next, a set of rules for unifying different types of potentially conflicting preferences are also presented. Lastly, since the degree of interference is usually determined by user's subjective feeling, which is usually vague, we propose a fuzzy-based approach to estimate the degree of interference among service components and a quality evaluation scheme for integrating all techniques mentioned above. Details of these mechanisms will be elaborated in the subsequent sections.

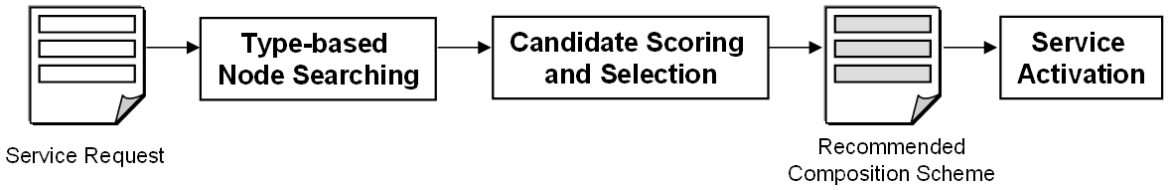


Figure 5.1: A general service composition architecture

5.1 Overall Architecture

Figure 5.1 illustrates the general architecture of a classical service composition. Classical service composition mechanisms are usually driven by developer-specified Service Templates that describe criteria (e.g. type or attributes) for selecting qualified service components. Typically, a service composition framework consists of three phases. In the phase of Type-based Node Searching, a service composition manager (i.e. a PSM), which is responsible for composing the service according to the Service Request, searches for qualified candidates either from a centralized service registry or by first broadcasting the type information as the searching criteria and then waiting for responses. The actual procedure in this phase largely depends on the underlying service discovery protocol (see Section 2.2). Usually, there are more than one qualified candidates, so that the manager needs to select the best one among candidates in the phase of Candidate Scoring and Selection, where the selection will proceed according to pre-specified preference expressions. Note that the selection mechanisms can greatly affect the quality of composed services, and sophisticated selection mechanisms usually involve evaluation and scoring of candidates based on their attributes. After all of the most appropriate candidate nodes are determined, each node is then activated in the Service Activation phase.

5.1.1 Capabilities and Preferences for Service Composition

Let us now examine how PerSAM/PSMP presented in Chapter 3 fits into this architecture. In PerSAM/PSMP, PSM plays the role of the service composition manager. After the service composition is started, PSM first issues M-SEARCH messages to a multicast channel and then waits for responses from Worker Nodes in the Type-based Node Searching phase (see Protocol 2, 3, and 5). In the Candidate Scoring and Selection phase, FCFS (First Come, First Select) is adopted as the default selection policy. Finally, by (3.14) in Protocol 5, a Pervasive Service is then activated.

Observe that the Service Template in PSMP only consists of a set of node types demanded by a Pervasive Service: it does not support attribute-based lookup. However, in Pervasive environments such as Smart Homes, service components with the same type do not imply that they are interchangeable. For instance, the contexts obtained from sensors in the living room are different from these observed in the bed room so that the sensors in two rooms are not interchangeable. In addition, PSMP can not adapt to user's needs dynamically, since a user does not have a chance to modify Service Templates. Finally, when there are more than one candidate nodes found, PSMP chooses the best node among these candidate nodes based on FCFS policy, which however usually leads to low user satisfaction. To improve quality of the composed services, more sophisticated selection or ranking algorithms that take the attributes of nodes into account are required in the phase of Candidate Scoring and Selection.

It follows from the above discussions that further enhancements of PerSAM and PSMP are required. In the following, several new data structures that support a PSM to search for Worker Nodes based on both node type as well as attributes are proposed. Originally, a Worker Node is merely described by the node type (i.e. *nt* in Figure 5.2-(a)), which, as mentioned above, is too restrictive for Pervasive service composition. Hence, it is desirable to describe Worker Nodes in a more general way. This research

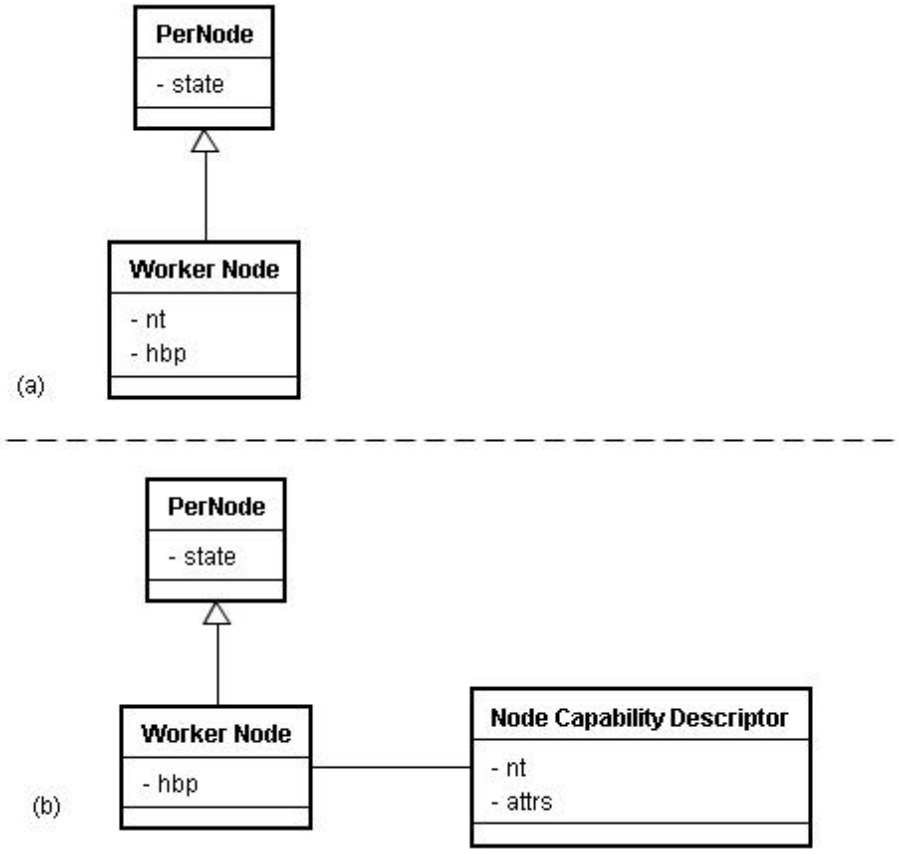


Figure 5.2: Modifying Worker Node structure to facilitate more sophisticated Pervasive service composition: (a) Original Worker Node structure, (b) Enhanced Worker Node structure

work therefore proposes to model the capabilities and the selection criteria for Worker Nodes by extending CC/PP, which is a W3C standard for specifying device capabilities and user preferences [84]. The capability of a Worker Node w is described by its type τ and a set of attributes $A = \{\alpha_1, \alpha_2, .., \alpha_k\} = \{\alpha_i\}_{i=1}^{k}$, where $\alpha_i = (n_i, v_i)$ is a name-value pair, based on which the Worker Node can be described by arbitrary attributes. The description of the capability of a Worker Node is called Node Capability Descriptor and is formally defined as follows.

Definition 11. (Node Capability Descriptor) *The Node Capability Descriptor of a Worker Node w is a pair: $C(w) \triangleq (\tau, A)$, where τ is the type of w, and $A = \{\alpha_i\}$ is the attribute set of w. For the attribute α_i, (n_i, v_i) is its associated name-value pair, where n_i is the attribute name and v_i is the attribute value.*

For example, a Worker Node w_1 that controls 37 inch LCD monitor which is located at room-1 can be described as follows:

$$(LCD, [(size, 37), (location, "room - 1")]).$$

In the original service model, the node type nt of a Worker Node is matched by a Service Template. Since in the enhanced version, a Worker Node is described by its Node Capability Descriptor (see Definition 11), a replacement for Service Template capable of matching Node Capability Descriptors is hence required. As a result, a new data structure, called Service Request, is proposed to replace Service Template for this purpose (see Figure 5.3-(b)). A Service Template ST^{ps} of a Pervasive Service ps is essentially a set of node types (cf. Definition 4). Likewise, a Service Request is a set of Node Preference Descriptors, of which each specifies the preferable property of one of the desired Worker Node $\widetilde{w}$ so that a PSM is able to search for qualified nodes according to the set of descriptors. The structure of a Node Preference Descriptor is defined below.

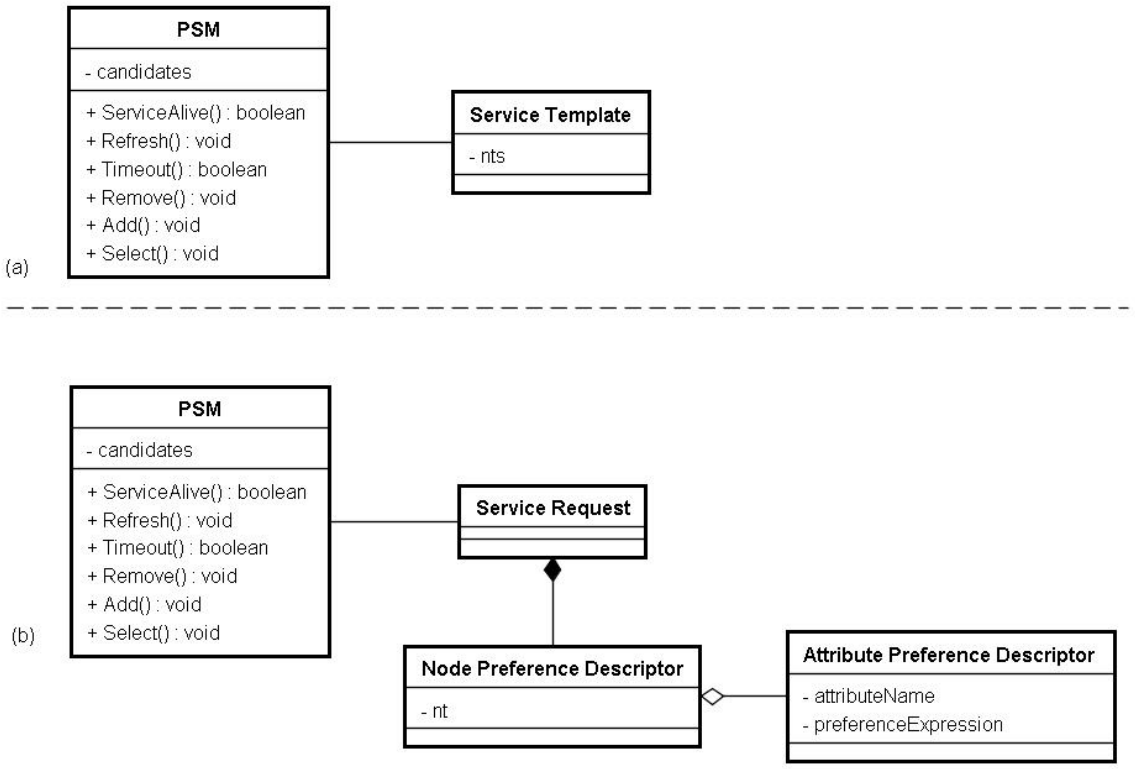


Figure 5.3: Modifying PSM structure to facilitate more sophisticated Pervasive service composition: (a) Original PSM structure, (b) Enhanced PSM structure

Definition 12. (Node Preference Descriptor) *The Node Preference Descriptor of a desired Worker Node $\widetilde{w}$ is a pair: $P(\widetilde{w}) \triangleq (\tau, \widetilde{A})$, where τ is the type of $\widetilde{w}$, and $\widetilde{A} = \{\widetilde{\alpha}_i\}$ is a set of Attribute Preference Descriptors $\widetilde{\alpha}_i = (n_i, \epsilon_i)$. Note that n_i is the attribute name and ϵ_i is a Preference Expression that specifies selecting criteria for attribute values.*

For example, the following Preference Descriptor $P(\widetilde{w}_1)$ directs the PSM to search for an LCD monitor that is more than 30 inches and is located at room-2:

$$(LCD, [(size, (\geq 30)), (location, (== "room - 2")]).$$

An Attribute Preference Descriptor facilitates users to participate in the selection process when composing services so that the service composition adapts to users' needs. In other words, when users express their preferences for a specific attribute, they typically override the default Preference Expressions in Attribute Preference Descriptors.

Depending on the characteristics of service composition mechanisms, the syntax of ϵ is usually different. In non-critical services, a user does not always insist on their preferences. Instead, there is usually room for negotiation when some preferences can not be met. Therefore, differentiation of mandatory preferences from negotiable ones is useful if we want to cover wider service compositions, especially when there are multiple users whom will be concerned by the services of interest. To facilitate consistent services, the aforementioned Preference Expression proposes an unifiable and negotiable expression. Based on the evaluation results of Preference Expressions, PSM is able to filter out the unqualified nodes and to rank the qualified ones. The details of Preference Expression and their unification rules will be elaborated in Section 5.2.

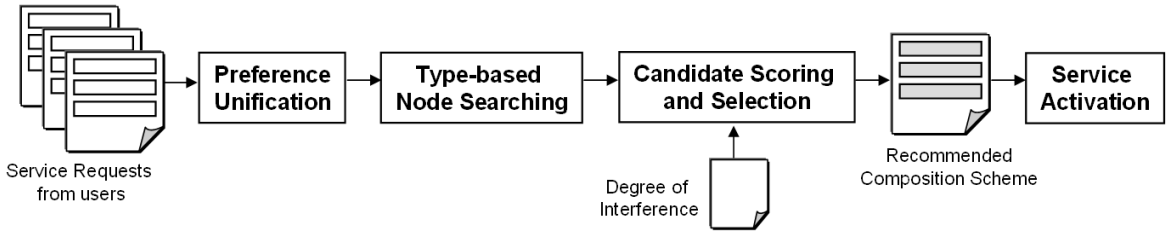


Figure 5.4: Refined service composition architecture for Pervasive environments

5.1.2 The Enhanced Architecture for Pervasive Service Composition

Figure 5.4 depicts the refined version of Pervasive service composition architecture. In the refined architecture, a Service Request of a Pervasive Service is essentially a subset of the power set of Node Preference Descriptors. Similar to Service Template, the Service Request is typically pre-defined by the developer of a Pervasive Service. However, users can express their own preferences by overriding the default ones.

When there are multiple users, the preferences can be conflicting. As a result, in the Preference Unification phase (see Figure 5.4), a PSM unifies multiple possibly conflicting preferences according to unification rules presented in Section 5.2. Next, in the Type-based Node Searching phase, a PSM also issues M-SEARCH messages to a multicast channel and then waits for responses from Worker Nodes. The major difference from the original architecture is that the candidates whose attributes do not satisfy the constraints specified in Preference Expressions are dropped in this phase. The remaining candidates are ranked based on what have been specified in negotiable Attribute Preference Descriptor and a scoring scheme presented in Section 5.4. In the following sections, the details of all phases except Service Activation phase as shown in Figure 5.4 will be elaborated. The Service Activation stage is identical to that in the original architecture, namely, the selected nodes are activated by using (3.14) in Protocol 5

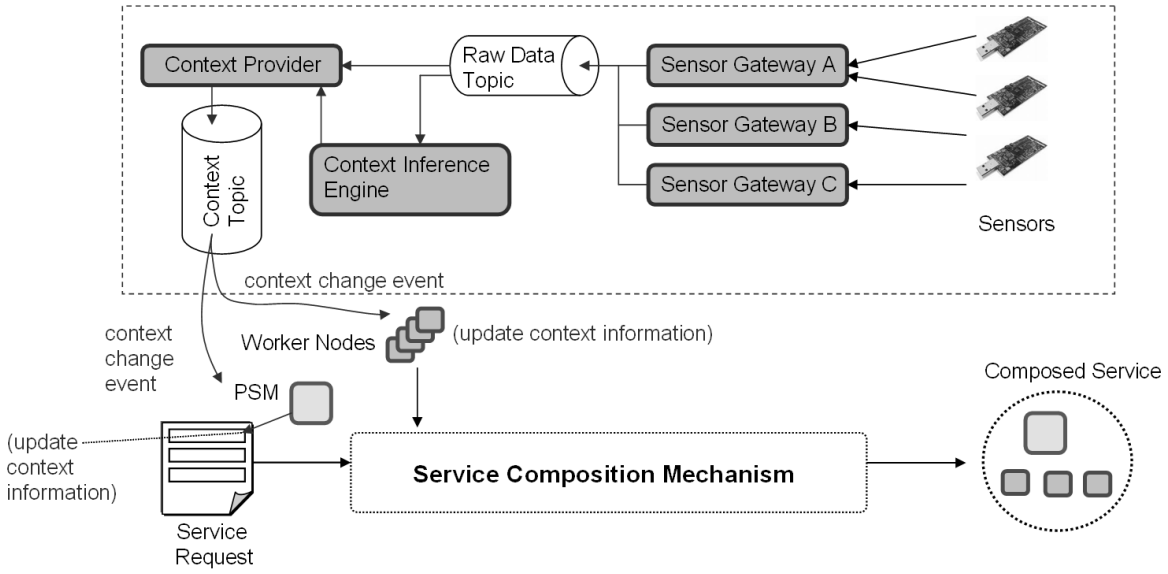


Figure 5.5: Dynamic contextual node re-binding

5.1.3 Dynamic Contextual Node Re-binding

One of the distinguishing characteristics of the Pervasive system is that a Pervasive environment is highly dynamic. As a result, environmental contexts are changing frequently so that the Pervasive system has to adapt to context changes. So far as a Pervasive Service Composition mechanism is concerned, it has to constantly probe for better candidate nodes. Upon a better node is found, the old one is replaced to enhance the quality of service. Bottaro *et al.* [32] identify these situations that will trigger dynamic re-binding of nodes: 1) presence of new nodes, 2) absence of existing nodes that have participated in one or more services, and 3) changes of contexts.

In fact, PSMP has built-in capabilities for detecting the presence and absence of Worker Nodes so that the situations 1) and 2) can be detected. The last situation, changes of contexts, can be considered from two aspects: the detection of environmental context change as well as the detection of the changes of contexts in a Worker Node. The platform is able to be aware of environmental context change such as human activities by interpreting the raw data gathered by sensors. As shown in Fig.

5.5, the raw data are interpreted by the Context Inference Engine [94] and then the Service Request is updated accordingly by the PSM, which reflects the fact that the the users' preferences are changing dynamically. On the other hand, the changes of contexts within a Worker Node are reported by the Worker Node itself in PerSAM. As a result, PerSAM/PSMP is aware of context changes either from the environment or within Worker Nodes. After the changes are detected, the Service Composition mechanism then re-selects the nodes that are affected by the contexts according to the new preferences as well as the new context information.

5.2 Specifying and Unifying User Preferences

As discussed in the previous section, specifying user preferences is a non-trivial task. The approach taken by this research is first to distinguish different kinds of preferences from various aspects and then proposes approaches to deal with them separately. First of all, preferences can be categorized by their necessity. For example, if a user is fond of playing a classic music composed by either J. S. Bach or by W. A. Mozart, but prefers W. A. Mozart better, then the preference associated with the attribute "Composer" can be expressed by the following Attribute Preference Descriptor:

$$(\text{"}Composer\text{"}, (== \text{"}W.A.Mozart\text{"} \rightarrow \text{"}J.S.Bach\text{"})). \tag{5.1}$$

On the contrary, suppose that the expression is changed as follows:

$$(\text{"}Composer\text{"}, (== \text{"}W.A.Mozart\text{"} \rightarrow == \text{"}J.S.Bach\text{"} :! = \text{"}A.Vivaldi\text{"})), \tag{5.2}$$

where the colon means "otherwise", then it means that this expression is negotiable. Specifically, all composers except A. Vivaldi can be an option for consideration if neither Bach's nor Mozart's music is available. Second, depending on the nature of attribute values, an attribute can be enumerative (such as "J. S. Bach" or "W. A. Mozart") or numeric (such as the size of LCD is 17 or 22 inches). The syntax of enumerative

attributes is different from that of numeric ones and so are their unification rules. Consequently, this section will present the syntax of Preference Expression and its auxiliary unification rules to deal with the challenges mentioned above. It follows from the above discussions that the Preference Expressions and their unification rules can be designed from two aspects: 1) a preference can be specified either numerically or enumeratively, and 2) from the user's points of view, the preference is either mandatory or negotiable.

In the Preference Unification phase (see Figure 5.4), if there are conflicting preferences specified by different users, then the PSM integrates these preferences according to unification rules (as will be presented in Section 5.3). If the unification process fails because that the conflicting preferences are mutually exclusive, then the PSM reports errors to users for further correction; otherwise, the composition process proceeds to the next phase. The following subsections are going to elaborate different types of Preference Exprssions and their unification rules in detail.

5.2.1 Enumerative Preference Expressions

The Enumerative Preference Expression is used to describe preferences for an enumerable attribute by specifying a list of preferred values. A candidate node is considered unqualified immediately if its corresponding attribute value matches none of the values listed in the expression. Otherwise, a matching score is calculated and stored for further selection (see Section 5.4). This type of expression is called a Mandatory Enumerative Preference Expression (MEPE). For instance, the preference to the composers of a music can be specified as

$$(== \text{"J. S. Bach"} \rightarrow == \text{"W. A. Mozart"}). \tag{5.3}$$

The syntax of MEPE is presented in Listing 5.1 in BNF (Backus-Naur Form) [22]. In an MEPE, the preferred values are a listed of strings, delimited by arrowheads.

Listing 5.1: The BNF of MEPE

```
MEPE ::= PtList | NegationExpr
PtList ::= '(' == STRING PtListTail* ')'
PtListTail ::= →== STRING
NegationExpr ::= '(' != STRING(∧!= STRING)* ')'
```

Alternatively, one can enumerate the undesired values by a list of conjunctions (e.g. != "J. S. Bach"∧!= "A. Vivaldi"). The list is ordered by preferences in descending order so that one can easily conceive that the first qualified service component is the most preferable one. The expression is evaluated to be true as soon as the PSM finds a service component whose attribute value meets the criteria specified in the expression. Alternatively, one can enumerate the undesired values by a list of conjunctions (e.g. != "*J.S.Bach*"∧!= "*A.Vivaldi*"), as mentioned earlier, and then the expression is evaluated to be true if the attribute value of a service component matches none of the undesired values.

In a Preference Expression, the term without an operator is called a "preference term", or simply a "p-term". A set of p-terms is called a "preference term set", or called a "pt-set", which is denoted as $pt(\varepsilon)$, where ε is the Preference Expression. The formal definitions of p-term and pt-set are given below.

Definition 13. (P-term and pt-set) *A term without an operator in a Preference Expression ε is called a "preference term", or simply a "p-term". A set of p-terms is called a "preference term set", which is denoted as $pt(\varepsilon)$. Assume that there are k possibly conflicting Preference Expressions $\{\varepsilon_i\}_{i=1}^{k}$, then the pt-sets of these expressions are denoted as $\{pt(\varepsilon_i)\}_{i=1}^{k}$.*

For example, if ε = (== "*J.S.Bach*" →== "*W.A.Mozart*"), then "J. S. Bach" and "W. A. Mozart" are p-terms and $pt(\varepsilon) = \{$"*J.S.Bach*","*W.A.Mozart*"$\}$.

In MEPE, p-terms that are associated with the notations "==" and "!=" operators

are respectively called positive p-terms (denoted ε^+) and negative p-terms (denoted ε^-). Positive and negative p-terms are formally defined below.

Definition 14. (Positive and Negative p-terms) *A p-term associated with an operator "==" is called a positive p-term; a p-term is a negative p-term if its operator is "!=".*

For example, == "*J.S.Bach*" implies "*J.S.Bach*" is a positive p-term, whereas ! = "*A.Vivaldi*" shows "*A.Vivaldi*" is a negative p-term. An MEPE is either composed of a set of positive p-terms, called Positive MEPE, or a set of negative p-terms, called Negative MEPE, but not a mixture of them. The pt-set of a Positive MEPE and a Negative MEPE are denoted as $pt(\varepsilon^+)$ and $pt(\varepsilon^-)$, respectively.

If there are more than one specified preference expressions, then these expressions have to be unified. The core idea of unifying Preference Expressions is to construct a new expression such that for all p-terms in the new expression satisfy all involved original expressions. Before the unification rules are presented, the following definition will be useful for further discussions. If there is at least one Positive MEPE, then a set of possibly conflicting MEPEs can be unified as a single Positive MEPE, which is formally defined as follows.

Definition 15. (Positively Unified MEPE) *Assume that* $\{\varepsilon_i\}_{i=1}^k$ *are a set of possibly conflicting MEPEs, where* $\exists\varepsilon^+ \in \{\varepsilon_i\}_{i=1}^k$ *such that* ε^+ *is a Positive MEPE. Then, the set* $\{\varepsilon_i\}_{i=1}^k$ *can be integrated into a Positively Unified MEPE, denoted as* ε^{u+}*, where* $\forall t \in pt(\varepsilon^{u+})$, $\wedge_{i=1}^{k_1}[t \in pt(\varepsilon_i^+)]$ *and* $\wedge_{j=k_1+1}^{k}[t \notin pt(\varepsilon_j^-)]$.

As mentioned earlier, the core idea is to construct an expression ε^{u+} such that all p-terms in ε^{u+} satisfy all of the involved original expressions.

Let us start from a simple case, in which all MEPEs to be unified are positive. Intuitively, in order to fulfill all preferences, the unifying outcome should be an inter-

section of all pt-sets. The unification rules for deriving the pt-set from a set of possibly conflicting Positive MEPEs is formally specified below.

Theorem 3. (Deriving the unified pt-set of Positive MEPEs) *The pt-set of a Positively Unified MEPE* $pt(\varepsilon^{u+})$ *can be obtained by the intersection among a set of Positive MEPEs, denoted as* $\{pt(\varepsilon_i^+)\}_{i=1}^k$*, namely,*

$$pt(\varepsilon^{u+}) = \cap_{i=1}^k pt(\varepsilon_i^+). \tag{5.4}$$

If all MEPEs are positive, then according to Definition 15 the possibly conflicting Positive MEPEs $\{\varepsilon_i^+\}_{i=1}^k$ can be unified by finding ε^{u+} such that $\forall t \in pt(\varepsilon^{u+})$, we have $\wedge_{i=1}^k [t \in pt(\varepsilon_i^+)]$. Based on this observation, the proof of Theorem 3 is shown below.

Proof. From (5.4),

$$pt(\varepsilon^{u+}) = \cap_{i=1}^k pt(\varepsilon_i^+) = pt(\varepsilon_1^+) \cap pt(\varepsilon_2^+) \cap ... \cap pt(\varepsilon_k^+).$$

Thus, we have

$$[pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_1^+)] \wedge [pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_2^+)] \wedge ... \wedge [pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_k^+)].$$

Consequently,

$$\forall t \in pt(\varepsilon^{u+}), [t \in pt(\varepsilon_1^+)] \wedge [t \in pt(\varepsilon_2^+)] \wedge ... \wedge [t \in pt(\varepsilon_k^+)]$$

is true, that is,

$$\forall t \in pt(\varepsilon^{u+}), \wedge_{i=1}^k [t \in pt(\varepsilon_i^+)].$$

□

The next theorem deals with a more general case in which at least one of the possibly conflicting MEPEs is positive.

Theorem 4. (Deriving the unified pt-set of a mixture of Positive and Negative MEPEs) *If there is a mixture of several possibly conflicting Positive and Negative*

MEPEs, then the pt-set of the Positively Unified MEPE $pt(\varepsilon^{u+})$ *can be obtained by the following operations:*

$$pt(\varepsilon^{u+}) = \cap_{i=1}^{k'} pt(\varepsilon_i^+) - \cup_{j=k'+1}^{k} pt(\varepsilon_j^-), \tag{5.5}$$

where there are k' *positive MEPEs and* $k - k'$ *negative MEPEs.*

Proof. Based on the De Morgan's laws and the set difference operation, that is, $A - B = A \cap \bar{B}$, the equation (5.5) can be transformed to intersections of pt-sets, specifically,

$$\begin{aligned} pt(\varepsilon^{u+}) &= \cap_{i=1}^{k'} pt(\varepsilon_i^+) - \cup_{j=k'+1}^{k} pt(\varepsilon_j^-) \\ &= \cap_{i=1}^{k'} pt(\varepsilon_i^+) \cap \overline{\cup_{j=k'+1}^{k} pt(\varepsilon_j^-)} \\ &= \cap_{i=1}^{k'} pt(\varepsilon_i^+) \cap \cap_{j=k'+1}^{k} \overline{pt(\varepsilon_j^-)} \\ &= pt(\varepsilon_1^+) \cap pt(\varepsilon_2^+) \cap ... \cap pt(\varepsilon_{k'}^+) \cap \overline{pt(\varepsilon_{k'+1}^-)} \cap \overline{pt(\varepsilon_{k'+2}^-)} \cap ... \cap \overline{pt(\varepsilon_k^-)}. \end{aligned}$$

Thus, we have:

$$\begin{aligned} &[pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_1^+)] \wedge [pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_2^+)] \wedge ... \wedge [pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_{k'}^+)] \wedge \\ &[pt(\varepsilon^{u+}) \subseteq \overline{pt(\varepsilon_{k'+1}^-)}] \wedge [pt(\varepsilon^{u+}) \subseteq \overline{pt(\varepsilon_{k'+2}^-)}] \wedge ... \wedge [pt(\varepsilon^{u+}) \subseteq \overline{pt(\varepsilon_k^-)}], \end{aligned}$$

which can be rewritten as

$$\begin{aligned} &[pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_1^+)] \wedge [pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_2^+)] \wedge ... \wedge [pt(\varepsilon^{u+}) \subseteq pt(\varepsilon_{k'}^+)] \wedge \\ &[pt(\varepsilon^{u+}) \nsubseteq pt(\varepsilon_{k'+1}^-)] \wedge [pt(\varepsilon^{u+}) \nsubseteq pt(\varepsilon_{k'+2}^-)] \wedge ... \wedge [pt(\varepsilon^{u+}) \nsubseteq pt(\varepsilon_k^-)]. \end{aligned}$$

As a result, $\forall t \in pt(\varepsilon^{u+})$, we see that

$$\begin{aligned} &[t \in pt(\varepsilon_1^+)] \wedge [t \in pt(\varepsilon_2^+)] \wedge ... \wedge [t \in pt(\varepsilon_{k'}^+)] \wedge \\ &[t \notin pt(\varepsilon_{k'+1}^-)] \wedge [t \notin pt(\varepsilon_{k'+2}^-)] \wedge ... \wedge [t \notin pt(\varepsilon_k^-)]. \end{aligned}$$

In other words, $\forall t \in pt(\varepsilon^{u+})$, we have

$$\wedge_{i=1}^{k'} [t \in pt(\varepsilon_i^+)] \wedge \wedge_{j=k'+1}^{k} [t \notin pt(\varepsilon_j^-)].$$

□

If all possibly conflicting MEPEs are negative, then the unification rule for generating Negatively Unified MEPEs is needed. Before turning to a closer examination of the unification rule for unifying Negatively MEPEs, let us define the Negatively Unified MEPE precisely first.

Definition 16. (Negatively Unified MEPE) *Assume that* $\{\varepsilon_i^-\}_{i=1}^k$ *is a set of possibly conflicting Negative MEPEs. Then,* $\{\varepsilon_i^-\}_{i=1}^k$ *can be integrated into a Negatively Unified MEPE, denoted as* ε^{u-}*. If such* ε^{u-} *exists, then* $\forall t \in \overline{pt(\varepsilon^{u-})}$ *implies* $\wedge_{i=1}^k [t \notin pt(\varepsilon_i^-)]$.

Based on Definition 16, the rule for deriving the pt-set of Negatively Unified MEPE from a set of Negative MEPEs is presented below.

Theorem 5. (Deriving the pt-set of the Negatively Unified MEPE from a set of Negative MEPEs) *The pt-set of the Negatively Unified MEPE* ε^{u-} *can be obtained by the union among pt-sets of Negative MEPEs* $\{pt(\varepsilon_i^-)\}_{i=1}^k$*, namely,* $pt(\varepsilon^{u-}) = \cup_{i=1}^k pt(\varepsilon_i^-)$.

Proof. The equation $pt(\varepsilon^{u-}) = \cup_{i=1}^k pt(\varepsilon_i^-)$ implies that

$$\overline{pt(\varepsilon^{u-})} = \overline{\cup_{i=1}^k pt(\varepsilon_i^-)} = \cap_{i=1}^k \overline{pt(\varepsilon_i^-)},$$

in other words,

$$\overline{pt(\varepsilon^{u-})} = \overline{pt(\varepsilon_1^-)} \cap \overline{pt(\varepsilon_2^-)} \cap ... \cap \overline{pt(\varepsilon_k^-)}.$$

Thus, we have

$$[\overline{pt(\varepsilon^{u-})} \nsubseteq pt(\varepsilon_1^-)] \wedge [\overline{pt(\varepsilon^{u-})} \nsubseteq pt(\varepsilon_2^-)] \wedge ... \wedge [\overline{pt(\varepsilon^{u-})} \nsubseteq pt(\varepsilon_k^-)],$$

namely, $\forall t \in \overline{pt(\varepsilon^{u-})}$, we have

$$[t \notin pt(\varepsilon_1^-)] \wedge [t \notin pt(\varepsilon_2^-)] \wedge ... \wedge [t \notin pt(\varepsilon_k^-)] = \wedge_{i=1}^k [t \notin pt(\varepsilon_i^-)]. \quad \square$$

If follows that the pt-set of an Unified MEPE can be derived from either Theorem 3, 4, or 5 and that the unification fails if the pt-set of the Unified MEPE is an empty set, that is, $pt(\varepsilon^u) = \phi$.

After deriving unified pt-sets, the order of p-terms have to be determined if the result is a Positively Unified MEPE. In this work, the user has to designate one Positive MEPE as the master expression according to her/his preferences. On the contrary, the master expression is not required for Negatively Unified MEPEs since the order of negative p-terms does not affect the result of service composition.

The following examples demonstrate the concrete procedures of unifying MEPE based on the rules mentioned above.

Example 1. *Derive the Positively Unified MEPE from the following MEPEs, where* ε_1^+ *is the master expression:*

$\varepsilon_1^+ = (== "J.S.Bach" \rightarrow== "W.A.Mozart" \rightarrow== "A.Vivaldi")$

$\varepsilon_2^+ = (== "W.A.Mozart" \rightarrow== "J.S.Bach" \rightarrow== "A.H.Haydn" \rightarrow== "A.Vivaldi")$

$\varepsilon_3^- = (! = "A.Vivaldi").$

Solution. First of all, the pt-sets of these expressions can be obtained based on Definition 13, as shown below.

$$pt(\varepsilon_1^+) = \{"J.S.Bach", "W.A.Mozart", "A.Vivaldi"\}$$

$$pt(\varepsilon_2^+) = \{"W.A.Mozart", "J.S.Bach", "A.H.Haydn", "A.Vivaldi"\}$$

$$pt(\varepsilon_3^-) = \{"A.Vivaldi"\}$$

Note that ε_1^+ and ε_2^+ are positive, whereas ε_3^- are negative. Therefore, after applying Theorem 4, the pt-set of the Positively Unified MEPE can be derived, namely,

$$\begin{aligned} pt(\varepsilon^{u+}) &= pt(\varepsilon_1^+) \cap pt(\varepsilon_2^+) - pt(\varepsilon_3^-) \\ &= \{"W.A.Mozart", "J.S.Bach", "A.Vivaldi"\} - \{"A.Vivaldi"\} \\ &= \{"W.A.Mozart", "J.S.Bach"\}. \end{aligned}$$

Next, $pt(\varepsilon^{u+})$ is ordered according to ε_1^+ which is chosen as the master expression. Hence, the Positively Unified MEPE, denoted as ε^{u+}, can be obtained after attaching the operators:

$$\varepsilon^{u+} = (== "J.S.Bach" \rightarrow == "W.A.Mozart").$$

Example 2. *Derive the Negative Unified MEPE from the following MEPEs:*

$$\varepsilon_1^- = (! = "J.S.Bach")$$

$$\varepsilon_2^- = (! = "W.A.Mozart" \wedge ! = "A.H.Haydn")$$

Solution. Again, the pt-sets have to be found first, as shown below.

$$pt(\varepsilon_1^-) = \{"J.S.Bach"\}$$

$$pt(\varepsilon_2^-) = \{"W.A.Mozart", "A.H.Haydn"\}$$

In this example, Theorem 5 is applied since ε_1^- and ε_2^- are both negative MEPEs. As a result, the pt-set of the Negative Unified MEPE can be derived as follows:

$$\begin{aligned} pt(\varepsilon^{u-}) &= pt(\varepsilon_1^-) \cup pt(\varepsilon_2^-) \\ &= \{"J.S.Bach"\} \cup \{"W.A.Mozart", "A.H.Haydn"\} \\ &= \{"J.S.Bach", "W.A.Mozart", "A.H.Haydn"\}. \end{aligned}$$

Finally, the Negatively Unified MEPE, denoted as ε^{u-}, can be obtained after attaching the operators:

$$\varepsilon^{u-} = (! = "J.S.Bach" \wedge ! = "W.A.Mozart" \wedge ! = "A.H.Haydn").$$

Listing 5.2 summarizes the procedure of unifying MEPEs mentioned above.

As mentioned earlier, representing preferences by mandatory expressions is decisive, that is, users either come to an agreement or no service is provided at all. However, users are usually willing to negotiate: they do not always insist on the criteria and may want to give up some desired service quality if the criteria can not be met in the

Listing 5.2: The algorithm for unifying MEPEs

Procedure Unify_MEPE
Input
$\{\varepsilon_i\}_{i=1}^{k}$: PreferenceExpression[] { *A set of MEPEs* }
ε^{master}: PreferenceExpression
{ *The Master MEPE, where* $\varepsilon^m \in \{\varepsilon_i\}_{i=1}^{k}$ }
Return
ε^u: PreferenceExpression { *The Unified MEPE* }
Begin
If ($\exists \varepsilon \in \{\varepsilon_i\}_{i=1}^{k}$ such that ε is a Positive MEPE) **Then**
$pt(\varepsilon^u) := \cap_{i=1}^{k_1} pt(\varepsilon_i^+) - \cup_{j=k_1+1}^{k} pt(\varepsilon_j^-)$
Sort($pt(\varepsilon^u)$,ε^{master}) { *Ordered by the Master MEPE* }
Else { *All expressions are negative* }
$pt(\varepsilon^u) := \cup_{i=1}^{k} pt(\varepsilon_i^-)$
ε^u:= NewMEPE($pt(\varepsilon^u)$)
End.

Listing 5.3: The BNF of NEPE

```
NEPE ::= PtList? (: '(' NegationExpr ')' )?
PtList ::= '('== STRING PtListTail* ')'
PtListTail ::= == → STRING
NegationExpr ::= '(' != STRING (∧ != STRING)* ')'
```

first place. The Negotiable Enumerative Preference Expression (NEPE) is designed for this purpose, which is used to specify the "good to have" criteria for an enumerative attribute. The BNF of NEPE is presented in Listing 5.3.

An NEPE has the form $\mathbb{P} : \mathbb{N}$, where the $\mathbb{P}$ segment is a list of positive p-terms, whereas the $\mathbb{N}$ segment is a set of negative p-terms. For example, in the expression:

$$(== "W.A.Mozart" :! = "A.Vivaldi"),$$

the $\mathbb{P}$ segment is $== "W.A.Mozart"$, and the $\mathbb{N}$ segment is $! = "A.Vivaldi"$. The p-terms in $\mathbb{P}$ specify all "good to have" options. If the p-terms in $\mathbb{P}$ can not be satisfied, then the expression can be considered satisfied as long as the criteria specified in $\mathbb{N}$ are evaluated to be True. If follows that if one of the Preference Expressions to be unified is NEPE, then the expressions are first treated as Positive MEPEs that are composed

of the p-terms in $\mathbb{P}$. If the unification fails, namely, $pt(\varepsilon^u) = \phi$, then the p-terms in $\mathbb{P}$ are replaced by those in $\mathbb{N}$ and then they are unified again. In this way, the NEPE provide an additional chance for unification, since $\mathbb{N}$ has weaker constraint than $\mathbb{P}$. The following example explains the overall unification process mentioned above.

Example 3. *Derive the Positively Unified MEPE from the following MEPEs and NEPEs, where ε_1 is the master expression:*

$$\varepsilon_1 = (== "J.S.Bach" \rightarrow== "A.Vivaldi" \rightarrow== "A.H.Haydn")$$

$$\varepsilon_2 = (== "W.A.Mozart" :! = "A.Vivaldi").$$

Solution. First of all, ε_2 is converted into an MEPE based on $\mathbb{P}$, so that we have $\varepsilon_2^{\mathbb{P}} = (== "W.A.Mozart")$, where $\varepsilon_2^{\mathbb{P}}$ is an MEPE that are composed of all p-terms in $\mathbb{P}$. However, the unification between ε_1 and $\varepsilon_2^{\mathbb{P}}$ fails, since

$$\begin{aligned} pt(\varepsilon^u) &= pt(\varepsilon_1) \cap \varepsilon_2^{\mathbb{P}} \\ &= \{"J.S.Bach", "A.Vivaldi", "A.H.Haydn"\} \cap \{"W.A.Mozart"\} \\ &= \phi. \end{aligned}$$

Next, because ε_2 is negotiable, $\varepsilon_2^{\mathbb{P}}$ is replaced by $\varepsilon_2^{\mathbb{N}}$ so that the unification is performed again. Hence,

$$\begin{aligned} pt(\varepsilon^u) &= pt(\varepsilon_1) - \varepsilon_2^{\mathbb{N}} \\ &= \{"J.S.Bach", "A.Vivaldi", "A.H.Haydn"\} - \{"A.Vivaldi"\} \\ &= \{"J.S.Bach", "A.H.Haydn"\}. \end{aligned}$$

Finally, $pt(\varepsilon^u)$ is ordered according to ε_1 which is chosen as the master expression. Hence, the Positively Unified MEPE can be obtained after attaching the operators:

$$\varepsilon^u = (== "J.S.Bach" \rightarrow== "A.H.Haydn").$$

It is important to observe that the result of unifying a set of MEPEs and NEPEs must be an MEPE. The reason is that the outcome has to be a consensus (and also the

most constrained). If there is at least one of them which is not negotiable, the outcome must not be negotiable. But for a special case where all expressions to be unified are NEPEs, the outcome will be an NEPE. When unifying NEPEs, the p-terms in $\mathbb{P}$ and in $\mathbb{N}$ segments are converted into MEPEs and are unified correspondingly. The following example explains the unification process mentioned above.

Example 4. *Derive the unified NEPE from the following NEPEs, where ε_1 is the master expression:*

$$\varepsilon_1 = (== "J.S.Bach" \rightarrow== "A.Vivaldi" :! = "A.H.Haydn")$$

$$\varepsilon_2 = (== "J.S.Bach" :! = "A.Vivaldi").$$

Solution. First, different segments of ε_1 and ε_2 are converted into MEPEs.

$$\varepsilon_1^{\mathbb{P}} = (== "J.S.Bach" \rightarrow== "A.Vivaldi")$$

$$\varepsilon_1^{\mathbb{N}} = (! = "A.H.Haydn")$$

$$\varepsilon_2^{\mathbb{P}} = (== "J.S.Bach")$$

$$\varepsilon_2^{\mathbb{N}} = (! = "A.Vivaldi")$$

Next, the MEPEs derived from $\mathbb{P}$ and from $\mathbb{N}$ are unified separately, that is,

$$\begin{aligned} pt(\varepsilon^{u_1}) &= pt(\varepsilon_1^{\mathbb{P}}) \cap pt(\varepsilon_2^{\mathbb{P}}) \\ &= \{"J.S.Bach", "A.Vivaldi"\} \cap \{"J.S.Bach"\} \\ &= \{"J.S.Bach"\} \\ pt(\varepsilon^{u_2}) &= pt(\varepsilon_1^{\mathbb{N}}) \cup pt(\varepsilon_2^{\mathbb{N}}) \\ &= \{"A.H.Haydn"\} \cup \{"A.Vivaldi"\} \\ &= \{"A.H.Haydn", "A.Vivaldi"\}. \end{aligned}$$

Hence, the unification result of $\mathbb{P}$ segment is $\varepsilon^{u_1} = (== "J.S.Bach")$ whereas that of $\mathbb{N}$ segment is $\varepsilon^{u_2} = (! = "A.H.Haydn" \wedge ! = "A.Vivaldi")$. The unified Preference Expression can be obtained by concatenating ε^{u_1} and ε^{u_2}, namely,

$$\varepsilon^u = (== "J.S.Bach") : (! = "A.H.Haydn" \wedge ! = "A.Vivaldi").$$

Listing 5.4: The unification algorithm for MEPEs/NEPEs

Procedure Unify_MEPE_NEPE
Input
  $\{\varepsilon_i^{mandatory}\}_{i=1}^{k_1}$: PreferenceExpression []
  $\{\varepsilon_j^{negotiable}\}_{j=1}^{k_2}$: PreferenceExpression []
  ε^{master}: PreferenceExpression { *The master expression* }
Return
  ε^u: PreferenceExpression { *The unified expression* }
Begin
  If $(|\{\varepsilon_i^{mandatory}\}_{i=1}^{k_1}| > 0)$ **Then** { *There is at least one MEPE* }
    ε^u:= Unify_MEPE($\{\varepsilon_i^{mandatory}\}_{i=1}^{k_1} \cup \{\mathbb{P}(\varepsilon_j^{negotiable})\}_{j=1}^{k_2}$, ε^{master})
    If $(pt(\varepsilon^u) := \phi)$ **Then**
      ε^u:= Unify_MEPE($\{\varepsilon_i^{mandatory}\}_{i=1}^{k_1} \cup \{\mathbb{N}(\varepsilon_j^{negotiable})\}_{j=1}^{k_2}$, ε^{master})
  Else { *All expressions are NEPE* }
    ε^u:= NewNEPE(Unify_MEPE($\{\mathbb{P}(\varepsilon_j^{negotiable})\}_{j=1}^{k_2}$, ε^{master}),
      Unify_MEPE($\{\mathbb{N}(\varepsilon_j^{negotiable})\}_{j=1}^{k_2}$, ε^{master}))
End.

If either the unification result of $\mathbb{P}$ segments or that of the $\mathbb{N}$ segments is ϕ, then the final result have to be transformed to an MEPE. For example, if ε_2 is replace by $(== "W.A.Mozart" :! = "A.Vivaldi")$, causing $pt(\varepsilon^{u_1}) = \phi$, then the final result becomes $\phi : (! = "A.H.Haydn" \wedge ! = "A.Vivaldi")$, which can be rewritten as a negative MEPE $(! = "A.H.Haydn" \wedge ! = "A.Vivaldi")$.

Listing 5.4 summarizes the unification rules for a mixture of MEPEs and NEPEs discussed above, where the functions $\mathbb{P}(\varepsilon)$ and $\mathbb{N}(\varepsilon)$ return the $\mathbb{P}$ segment and the $\mathbb{N}$ segment of ε, respectively.

5.2.2 Numeric Preference Expressions

Numeric attributes are different from enumerative ones in that they are numerically comparable and that they can be constrained by specifying intervals (i.e. upper and lower bounds). As a result, numeric expressions must support more operators than that are supported in enumerative ones. Specifically, there are only two operators

supported in Enumerative Preference Expressions: "==" and "!=", whereas Numeric Preference Expressions uses additional operators such as ">", "<", "≤", "≥", and "¬".

Again, Numeric Preference Expressions can also be mandatory or negotiable. A Mandatory Numeric Preference Expression (MNPE) is a numeric preference expression whose criteria must be met. For example, the MNPE:

$$((> 20 \wedge \leq 30) \vee < 10) \tag{5.6}$$

can be used to specify the selection criteria of size of an LCD display whose size is either between 20 to 30 inches or smaller than 10 inches.

On the other hand, the Negotiable Numeric Preference Expression (NNPE) is the numeric preference expression that contains "negotiable" semantics. Similar to NEPE, an NNPE is also composed of a $\mathbb{P}$ segment and an $\mathbb{N}$ segment which are delimited by a colon mark. For example, the following expression

$$((> 20 \vee < 10) \wedge != 25 : \gg) \tag{5.7}$$

is capable of specifying a selection criteria and a negotiable expression for an LCD display, where the former is that the size should be either larger than 20 inches or less than 10 inches and must not equal 25 inches, whereas the later is the expression after the colon mark (":"), i.e. the $\mathbb{N}$ segment with the notation "$\gg$", which means the size is the greater the better. Note that the $\mathbb{N}$ segment of an NNPE is useful when the user only wants to specify a vague constraint for an attribute. For instance, $\gg$, $\ll$, and $\approx$ denote "the greater the better", "the less the better", and "the closer to a specified value the better". Taking the expression (5.7) as an example, assume that three kinds of display are available, and their sizes are 25, 18, and 12 inches, respectively. The order of preference should be 25, 18 and 12. This is because all of them do not match the $\mathbb{P}$ segment of (5.7), so that according to the $\mathbb{N}$ segment, their preferences will be

Listing 5.5: The BNF of Mandatory Numeric Preference Expression (MNPE)

```
MNPE ::= BinaryExpr | UnaryExpr | NegateExpr
BinaryExpr ::= '('MNPE BinaryOp MNPE ')'
NegateExpr ::= ¬ BinaryExpr
UnaryExpr ::= UnaryOp NUMBER
BinaryOp ::= ∧ | ∨
UnaryOp ::= == | != | ≥ | ≤ | > | <
```

Listing 5.6: The BNF of Negotiable Numeric Preference Expression (NNPE)

```
NNPE ::= LogicExpr? (: NegotiationExpr)?
NegotiationExpr ::= ≈ NUMBER
                  | !≈ NUMBER
                  | ≫
                  | ≪
LogicExpr ::= BinaryExpr | UnaryExpr | NegateExpr
BinaryExpr ::= '('LogicExpr BinaryOp LogicExpr ')'
NegateExpr ::= ¬ BinaryExpr
UnaryExpr ::= UnaryOp NUMBER
BinaryOp ::= ∧ | ∨
UnaryOp ::= == | != | ≥ | ≤ | > | <
```

ranked in descending order. The BNF of MNPE and NNPE are shown in Listing 5.5 and Listing 5.6, respectively.

Again, unification rules are required if there are more than one Numeric Preference Expressions. Not surprisingly, the unification rules for Numeric Preference Expressions are different from enumerative ones because they are now integration of numerical interval as well as comparative operators rather than lists of strings. However, it can be shown that the integration of numerical intervals and operators can actually be reduced to a few types of compact forms so that specific unification rules for these compact forms can still be derived to integrate Numeric Preference Expressions efficiently.

The first step of unifying MNPE is to convert the expressions into Conjunctive Normal Forms (CNF) which is a conjunction of clauses, where a clause is a disjunction of logical terms (e.g. $> 30 \vee < 20$). Theoretically, every logical expression can be

converted into an equivalent CNF expression by repeatedly applying distributive law and De Morgan's laws.

Let us denote an MNPE and a logical term as ζ and ρ, respectively, then any ζ can be converted into the following CNF, denoted as $\hat{\zeta}$:

$$\hat{\zeta} = \bigwedge_{i=1}^{k_1}(\bigvee_{j=1}^{k_2}\rho_{ij}) = (\rho_{11}\vee\rho_{12}\vee...\vee\rho_{1k_2})\wedge(\rho_{21}\vee\rho_{22}\vee...\vee\rho_{2k_2})\wedge...\wedge(\rho_{k_11}\vee\rho_{k_12}\vee...\vee\rho_{k_1k_2}).$$

Here, the logical terms such as > 20, ≤ 30, and < 10 are also called p-terms. The clause that consists of a set of p-terms connected by $\vee$ is called a **disjunctive clause** (i.e. $\bigvee_j \rho_j$).

The purpose for converting expressions into CNF is that both $\cap$ and $\cup$ satisfy the associativity property so that the logical terms can be unified pairwise. Specifically, after an MNPE is converted into a CNF, all clauses are connected by $\cap$, and all logical terms are connected by $\cup$. Therefore, logical terms within a clause can be unified pairwise, and the order in which they are unified does not affect the outcome. For example,

$$\begin{aligned}&(\rho_{11}\vee\rho_{12}\vee...\vee\rho_{1k_2})\\ &\equiv(((((\rho_{11}\vee\rho_{12})\vee\rho_{13})\vee\rho_{14})...\vee\rho_{1k_2}))\\ &\equiv(((((\rho_{1k_2}\vee\rho_{1k_2-1})\vee\rho_{1k_2-2})\vee\rho_{1k_2-3})...\vee\rho_{11})).\end{aligned}$$

The same principle holds for clauses within an MNPE. Taking the MNPE in (5.6) as an example, it can be converted in to the following CNF by applying De Morgan's laws, that is,

$$((>20\wedge\leq 30)\vee<10)\equiv(((>20\vee<10)\wedge(\leq 30\vee<10)). \tag{5.8}$$

Now let us turn to the second step. The purpose of this step is to derive the most compact form for each disjunctive clause. In fact, all disjunctive clause can be reduced to one of the eight compact forms shown in Table 5.4 by repeatedly applying the

Table 5.1: Possible pairwise combinations between two numeric p-terms

	$< x$	$> x$	$== x$	$! = x$
$< y$	$< x \vee < y \ldots (1)$	$> x \vee < y \ldots (2)$	$== x \vee < y \ldots (4)$	$! = x \vee < y \ldots (7)$
$> y$		$> x \vee > y \ldots (3)$	$== x \vee > y \ldots (5)$	$! = x \vee > y \ldots (8)$
$== y$			$== x \vee == y \ldots (6)$	$! = x \vee == y \ldots (9)$
$! = y$				$! = x \vee ! = y \ldots (10)$

Table 5.2: Reduction rules for deriving compact forms

No.	Case	Rule
(1)	$< x \vee < y$	if $x \geq y$ then $< x$ else $< y$
(2)	$> x \vee < y$	if $x \leq y$ then $True$
(3)	$> x \vee > y$	if $x \geq y$ then $> y$ else $> x$
(4)	$== x \vee < y$	if $x \leq y$ then $< y$
(5)	$== x \vee > y$	if $x \geq y$ then $> y$
(6)	$== x \vee == y$	if $x == y$ then $== x$
(7)	$! = x \vee < y$	if $x < y$ then $True$ else $! = x$
(8)	$! = x \vee > y$	if $x > y$ then $True$ else $! = x$
(9)	$! = x \vee == y$	if $x == y$ then $True$ else $! = x$
(10)	$! = x \vee ! = y$	if $x == y$ then $! = x$ else $True$

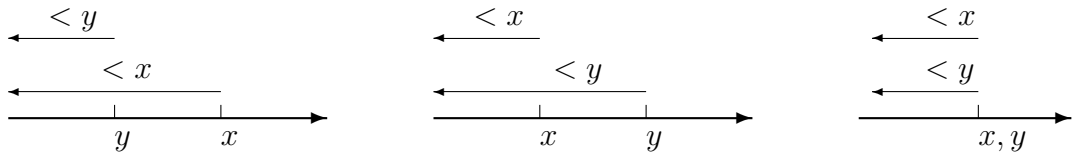


Figure 5.6: Reducing $< x \vee < y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

reduction rules shown in Table 5.2. It can be observed from Listing 5.5 that there are six different operators defined in MEPE, namely, $>$, $<$, $\leq$, $\geq$, $==$, and $!=$. Among these operators, $\leq$ and $\geq$ is semantically equivalent to $(< \vee ==)$ and $(> \vee ==)$, respectively. In this way, the number of different operators can be reduced to four: $>$, $<$, $==$, and $!=$. Consequently, all possible pairwise combinations among numeric p-terms in a disjunctive clauses are $4^2 = 16$. However, as show in Table 5.1, there are actually 10 distinct combinations because of the commutativity of $\vee$.

Because the logical operator for connecting p-terms in distinctive clauses is $\vee$, the outcomes of unifications should be with fewer constraints, that is, with greater possible coverage. Taking case (1) shown in Table 5.1 as an example, Figure 5.6-(a) reveals that the coverage of "$< x$" is greater than that of "$< y$" when $x > y$, whereas the coverage of "$< y$" is greater that of "$< x$" when $x < y$ (Figure 5.6-(b)). In case that x is equal to y, one can reduce "$< x \vee < y$" to either "$< x$", or "$< y$", which is reduced to "$< x$" in this work, as shown in Figure 5.6-(c). The same procedure for obtaining the rules applies to other cases as well. Specifically, the reduction rules for the cases (1) to (10) listed in Table 5.2 can be diagrammatically and intuitively derived, as illustrated from Figure 5.6 to 5.15, respectively. The deriving procedures for the remaining cases are therefore not elaborated in further detail.

Now let us prove that all disjunctive clauses can be reduced to one of the eight compact forms shown in Table 5.4 (i.e. Theorem 6).

Lemma 6. *If a p-term of the form* $!= s$ *appears in a disjunctive clause, then either*

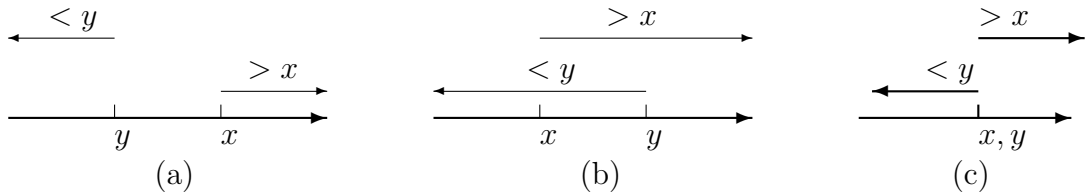


Figure 5.7: Reducing $> x \vee < y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

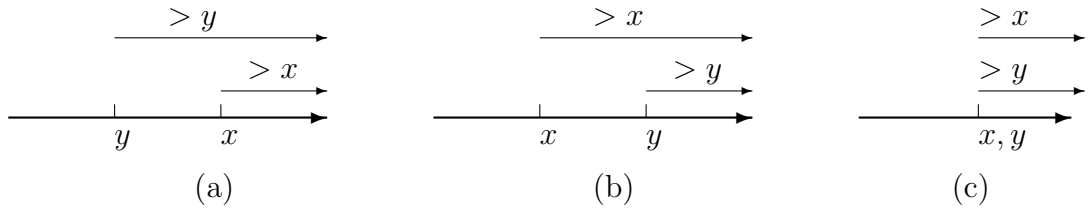


Figure 5.8: Reducing $> x \vee > y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

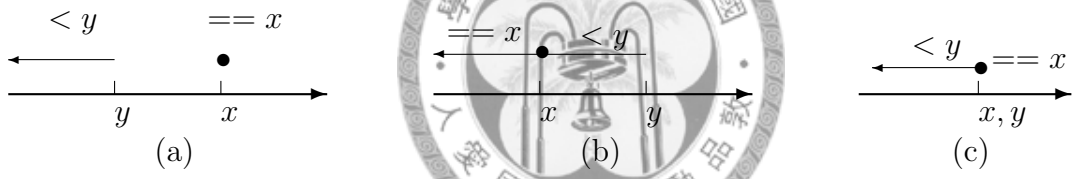


Figure 5.9: Reducing $== x \vee < y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

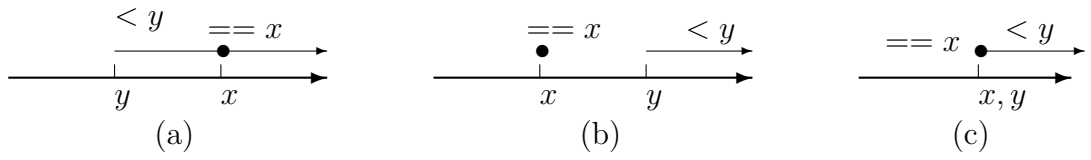


Figure 5.10: Reducing $== x \vee > y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

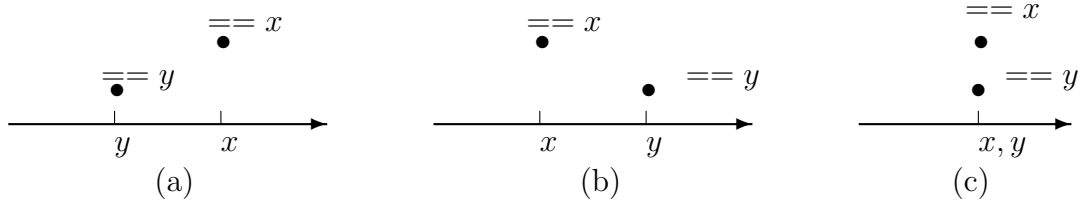


Figure 5.11: Reducing $== x \vee == y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

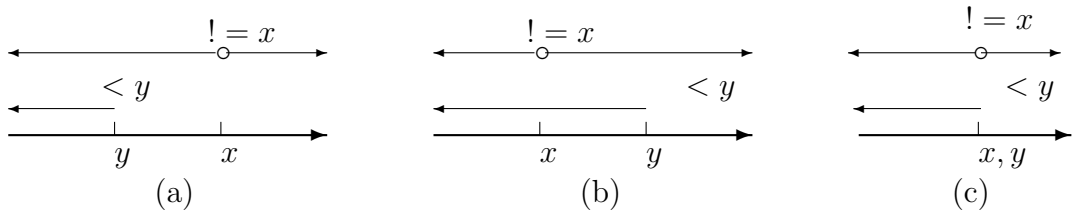


Figure 5.12: Reducing $! = x \vee < y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

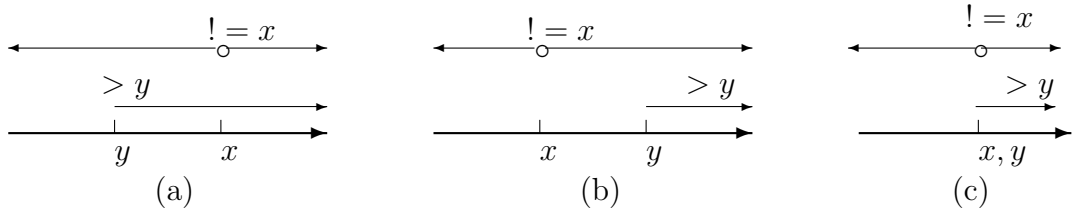


Figure 5.13: Reducing $! = x \vee > y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

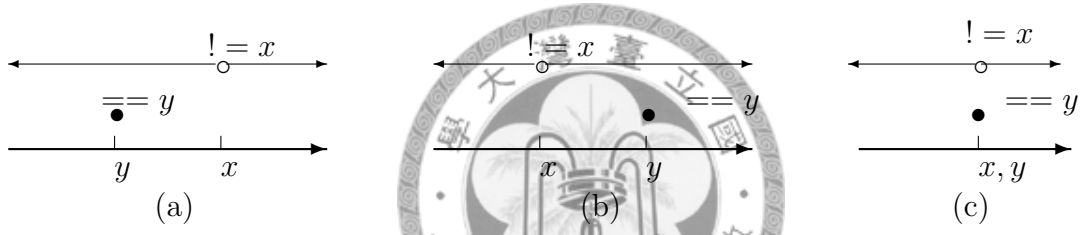


Figure 5.14: Reducing $! = x \vee == y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

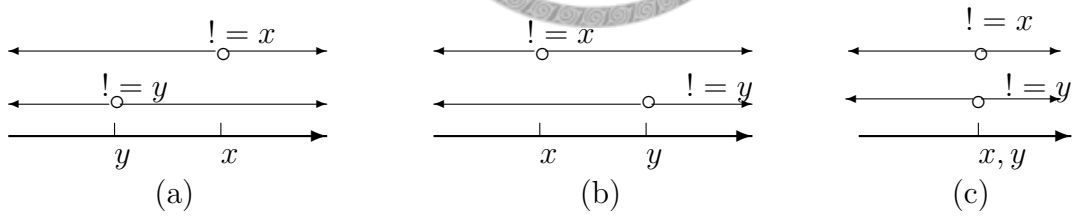


Figure 5.15: Reducing $! = x \vee ! = y$ when (a) $x > y$, (b) $x < y$, and (c) $x = y$.

Table 5.3: General forms for disjunctive clauses

No.	General Form
(1)	$! = s$
(2)	$> a \vee < b \vee \bigvee_i (== x_i)$, where $a \neq b$

$!= s$ *is the only p-term in the disjunctive clause or the clause is resolved to be True.*

Proof. Recall that the unification is performed pairwise throughout the disjunctive clause. According to Case (7)-(10) shown in Table 5.2, the results of integrating "$!= s$" with another p-term is either $True$ or "$!= s$". If the result is $True$, then the whole disjunctive clause are immediately evaluated as being $True$; otherwise, only "$!= s$" is derived and then it is integrates with the next term in the disjunctive clause. Finally, the clause contains either solely "$!= s$" or the whole clause is evaluated as being $True$. □

Lemma 7. *There is at most one p-term of the form* $> a$ *and at most one p-term of the form* $< b$ *in a disjunctive clause.*

Proof. This lemma can be directly proved by using case (1) and case (3) shown in Table 5.2. Assuming there are two different p-terms of the form $> a$, (for instance, $> x$ and $> y$), then according to case (1) in Table 5.2, the p-terms can be integrated into single term, namely, either $> x$ or $> y$. Similar results hold for for $< b$ according to case (3). □

Lemma 8. *All disjunctive clauses can be reduced to the general forms shown in Table 5.3, namely, either* $!= s$ *or* $> a \vee < b \vee \bigvee_i (== x_i)$*, where* $a \neq b$.

Proof. From the definition of MNPE (Listing 5.5), the resulting general form of a disjunctive clauses should be

$$\bigvee_{i_1} (!= s_{i_1}) \vee \bigvee_{i_2} (> a_{i_2}) \vee \bigvee_{i_3} (< b_{i_3}) \vee \bigvee_{i_4} (== x_{i_4}).$$

The general form can be reduced to one of the following form based on Lemma 6:

$$\bigvee_{i_2} (> a_{i_2}) \vee \bigvee_{i_3} (< b_{i_3}) \vee \bigvee_{i_4} (== x_{i_4}), \tag{5.9}$$

Table 5.4: Compact forms for disjunctive clauses

No.	Type	Compact Form
(1)	Negation	$! = s$
(2)	Disjoint Intervals or Disjunctions of Positive Enumerations	$> a \vee < b \vee \bigvee_i (== x_i)$, where $a \neq b$
(3)	Disjoint Intervals or Disjunctions of Positive Enumerations	$> a \vee \bigvee_i (== x_i)$
(4)	Disjoint Intervals or Disjunctions of Positive Enumerations	$< b \vee \bigvee_i (== x_i)$
(5)	Disjoint Intervals	$> a \vee < b$, where $a \neq b$
(6)	Disjoint Intervals	$> a$
(7)	Disjoint Intervals	$< b$
(8)	Disjunctions of Positive Enumerations	$\bigvee_i (== x_i)$

or

$$\vee(! = s). \tag{5.10}$$

Note that (5.9) can be further reduced to the following form based on Lemma 7:

$$> a \vee < b \vee \bigvee_i (== x_i). \tag{5.11}$$

As a result, this lemma can be proved by combining (5.10) and (5.11). □

Theorem 6. (The compact forms of disjunctive clauses) *All disjunctive clauses can be reduced to one of the eight compact forms shown in Table 5.4.*

Proof. The compact forms (1) and (2) listed in Table 5.4 can be directly obtained from the two general forms derived in Lemma 8, namely, (5.10) and (5.11). The compact forms (3) to (8) are special cases of general forms, which can be obtained by

Table 5.5: Compact forms derived form $> a \vee < b \vee \bigvee_i (== x_i)$

Condition	Derived Compact Form
$b = -\infty, (i.e. < b = False)$	$> a \vee \bigvee_i (== x_i)$
$a = \infty, (i.e. > a = False)$	$< b \vee \bigvee_i (== x_i)$
$i = 0$	$> a \vee < b$, where $a \neq b$
$b = -\infty$ and $i = 0$	$> a$
$a = \infty$ and $i = 0$	$< b$
$a = \infty$ and $b = -\infty$	$\bigvee_i (== x_i)$

assigning ∞, $-\infty$, and 0 to the variable a and b in (5.11) and i in (5.10), respectively, as shown in Table 5.5. □

After each disjunctive clauses are reduced to the most compact forms, the final step involves connecting disjunctive clauses by conjunctive logical operator ($\wedge$) and applying unification rules to each pair. From Lemma 8, there are three possible combination:

$$! = s \wedge ! = t, \tag{5.12}$$

$$! = s \wedge [> a \vee < b \vee \bigvee_i (== x_i)], \text{ and} \tag{5.13}$$

$$[> a \vee < b \vee \bigvee_i (== x_i)] \wedge [> c \vee < d \vee \bigvee_j (== y_j)], \tag{5.14}$$

where $a > b$. The following paragraphs will derive reduction rules for each of them.

First, let us consider (5.12). It is important to observe that the statement can not be reduced further unless s equals t, in which case the statement can be reduced to either $! = s$ or $! = t$. In this work, $! = s$ is chosen.

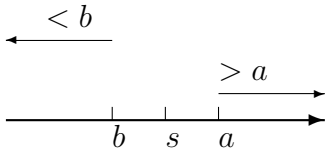


Figure 5.16: Reducing the first term of (5.15): $! = s \wedge (> a \vee < b)$.

As for the case of (5.13), it can be expanded as follows:

$$! = s \wedge [> a \vee < b \vee \bigvee_i (== x_i)]$$

$$\equiv [! = s \wedge (> a \vee < b)] \vee [! = s \wedge \bigvee_i (== x_i)]. \tag{5.15}$$

According to Figure 5.16, the first term of (5.15) can be reduced to $> a \vee < b$ if $s \geq b \wedge s \leq a$, where $a > b$, since when $s \geq b \wedge s \leq a$, $! = s$ is redundant. As for the second term, by the definition of subtraction ($B \cap A = A - B$), it can be reduced to $\bigvee_j [(== z_j)]$, where $z_j \in (X - s)$, $X = \{x_1, x_2, ...\}$.

To reduce (5.14), we first apply De-Morgan's laws:

$$[> a \vee < b \vee \bigvee_i (== x_i)] \wedge [> c \vee < d \vee \bigvee_j (== y_j)] \equiv$$

$$[(> a \vee < b) \wedge (> c \vee < d)] \vee \tag{5.16}$$

$$[(> a \vee < b) \wedge \bigvee_j (== y_j)] \vee \tag{5.17}$$

$$[\bigvee_i (== x_i) \wedge (> c \vee < d)] \vee \tag{5.18}$$

$$[\bigvee_i (== x_i) \wedge \bigvee_j (== y_j)]. \tag{5.19}$$

To merge (5.16), one should use a more restrictive boundary. In other words, (5.16) can be reduced to $> e \wedge < f$, where $e = max(a, c)$ and $f = min(b, d)$. Semantically, the term $\bigvee_i (== x_i)$ in (5.17) positively enumerates acceptable values. The term $(> a \vee < b)$ further constraints the acceptable values, and therefore (5.17) can be reduced to $\bigvee_k (== z_k)$, $z_k \in Y$, where $Y = \{y_1, y_2, ...\}$, and $z_k < b$ or $z_k > a$. Finally, it can

Table 5.6: Unification rules for *NegotiationExpr*

Master	Slave	Outcome
$\gg$	$\gg$	$\gg$
$\ll$	$\ll$	$\ll$
$\gg$	$\ll$	ϕ
$\ll$	$\gg$	ϕ
other cases	other cases	Master *NegotiationExpr*

be inferred that if statement (5.19) is true, than there exists a non-empty set W such that $W \subseteq X$ and that $W \subseteq Y$, namely, $W \subseteq (X \cap Y)$, where $W = \{w_1, w_2, ...\}$, $X = \{x_1, x_2, ...\}$, and $Y = \{y_1, y_2, ...\}$. Consequently, (5.19) can be reduced to $\bigvee_l (== w_l)$, where $w_l \in (X \cap Y)$.

The unification procedure of NNPE is the same as that of MNPE except for the $\mathbb{N}$ segment. The rules for unifying $\mathbb{N}$ segments are listed in Table 5.6. If the semantics of terms in $\mathbb{N}$ are the same, then the segment $\mathbb{N}$ is directly adopted. On the contrary, the terms are removed if any conflict exists. If it is neither of the two cases, then the segment $\mathbb{N}$ of the master expression is chosen.

5.3 Type-based Node Searching

After the Preference Unification phase (see Figure 5.4), there is exactly one unified Preference Expression that specifies the criteria for each attribute. Based on the unified Preference Expression, the PSM issues M-SEARCH messages to a multicast channel and then waits for responses from Worker Nodes. In fact, the M-SEARCH messages are responsible for conveying the Node Preference Descriptor (see Definition 12) which consists of a type criterion and Attribute Preference Descriptors for selecting nodes. As revealed in Listing 5.8, the type criterion is encoded as the value for the *ST* header,

Listing 5.7: Embedding an Node Preference Descriptor in an M-SEARCH message

```
M-SEARCH * HTTP/1.1
ST: urn:schemas-upnp-org:device:sensor:a
MX: 3
MAN: "psmp:discover"
CRITERIA: (location,(=="livingroom"→=="bedroom"))
          (sensor_type, (=="thermo":!="humidity"))
HOST: 239.255.255.250:1900
```

whereas the Attribute Preference Descriptors are embeded in the $CRITERIA$ section.

After receiving M-SEARCH messages, Worker Nodes will judge whether their types satisfies the ones specified in the Node Preference Descriptors embedded in the M-SEARCH messages and respond to the PSM if a match is found.

To enhance the interoperability, an Ontology for Smart Home system is proposed to standardize the representation of node types. Advantages of using Ontology techniques in Pervasive systems have been extensively discussed in several literatures, which can be summarized as follows: 1) Knowledge sharing between agents and services, 2) Supports of the hierarchical inference, and 3) Reuse of previously defined ontology models. As a result, many researchers have been committed to develop the ontology of Pervasive environments [41, 137]. In this work, we define concepts of sensors and actuators by using OWL [8]. Note that the ontology can be easily incorporated into other well-known ontology such as SOUPA [41]. In order to decide if a node type is the sub-type of another node, PSM consults the ontology repository by issuing an SPARQL [14] statement. Assuming that there is a class called $VideoDisplay$, the following statement searches for all sub-classes of $Display$, which can be used for performing type matching.

To decide whether their attributes satisfies the criteria specify in the Attribute Preference Descriptor, Worker Nodes evaluate their attribute values against the Preference Expressions that specify constraints for the attributes having the same names. Note that some Worker Nodes may be implemented in embedded devices (hosts) causing im-

Listing 5.8: The SPARQL statement for searching sub-classes of a class called *VedioDisplay*

```
PREFIX home:<http://www.attentivehome.org/ontologies/
            pernode/2010/10/Core.owl#>
SELECT ?name
WHERE {
  ?appliance home:hasType home:Display.
  ?appliance home:hasName ?name.
  }
```

plementation of evaluating mechanisms for Preference Expression infeasible. In these cases, these Worker Nodes do not check attributes. Instead, the attribute evaluating tasks are delegated to PSM.

As mentioned above, to be compatible with hosts having weak computing capabilities, for each response from a Worker Node, the PSM first evaluates the attributes of the Worker Node against the Preference Expressions specified in the Attribute Preference Descriptor to filter out the un-qualified Worker Nodes. Next, the PSM adds the node into a list of candidates for a specific node type and then selects the best ones among them based on the results of interference estimation and a scoring mechanism which are going to be presented in the next section.

5.4 Candidate Scoring and Selection

The objective of the Candidate Scoring and Selection phase is first to assess the possible interference of the candidate node with other existing nodes, that is, the degree of interference, and then to grade each candidate node depending on how well the capability of the candidate node matches the Preference Expressions.

5.4.1 Estimating the Degree of Interference

Let us first introduce how we estimate the interference degree among nodes. The interference degree $\iota(w)$ is a measure that indicates the degree of user's dissatisfaction caused by the interferences among the Worker Node w and other nodes, where $0 \leq \iota(w) \leq 1$. The point to observe is that the estimated outcomes usually depend on "feelings" of users, which are rather subjective, vague, and context dependent. As a result, Fuzzy sets are used to represent the factors that affect the interference degree, which are inputs to the Fuzzy reasoner. Currently, three factors are taken into consideration:

1. **The physical distance between two devices:** The shortest physical distance between two devices respectively controlled by two candidate nodes in the smart home. The closer two devices are to each other, the more interferences between them there can be.

2. **The similarity of effects:** The similarity between two different effects produced by the physical devices controlled by two different nodes. For instance, the effects of playing music and text-to-speech, respectively, are similar : both of them produce sounds. In this thesis, the effects of devices are represented by ontological concepts (see Fig. 5.17) which can be obtained in their Capability Descriptors.

3. **The intensity of the effect:** The intensity, ranging from 0 to 1, of the effect is an attribute pre-defined by the system administrator. The higher the intensity of the effect is, the more likely that the device that produce the effect to interfere with other devices.

To calculate the distance between devices (nodes), an undirected graph is created based on a given floor plan, which considers each room as a vertex and edges two vertices if the corresponding two rooms are adjacent. As a result, the distance between

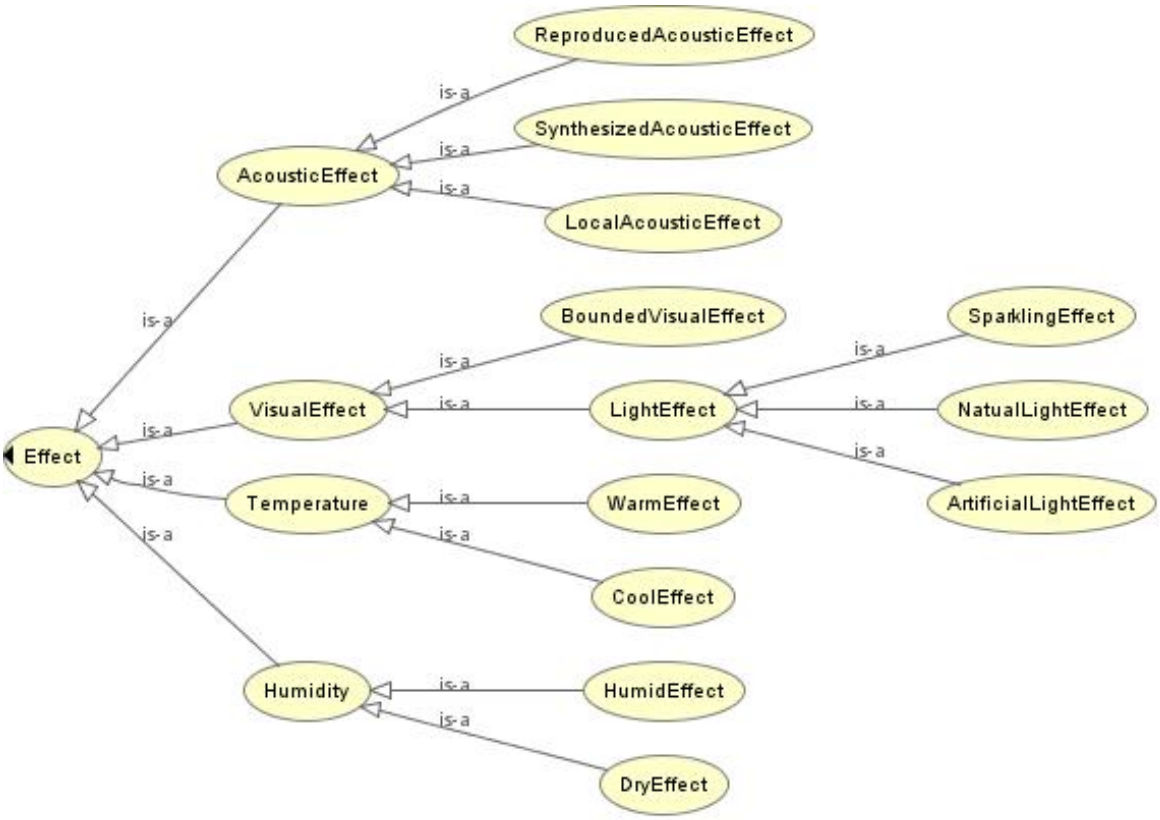


Figure 5.17: The Effect ontology in a Smart Home

device a and device b, denoted $D(a, b)$, can be obtained by calculating the number of edges contained in the shortest path between two vertices. Note that it is assumed that the floor plan is given and that the location of nodes can be obtained by examining the attributes in the Capability Descriptors. The distance is ∞ if the node does not control any devices (i.e. sensors or actuators) at all. For each candidate node, the device is calculated against all activated nodes. After that, the least value is chosen as the representative value since the nearest node is more likely to cause interference.

Meanwhile, the similarity values of effects among nodes are obtained by calculating the semantic relevance of effects. This work adopts the approach proposed by Yu *et al.* [144] to estimate the semantic relevance of effects. In this approach, the semantic

relevance between two effects, which are ancestor/descendant, is defined by a function,

$$S(e_x, e_y) = \frac{\hat{d}(Ancestor(e_x, e_y))}{\hat{d}_{max}},$$

where the function $Ancestor(e_x, e_y)$ returns the ancestor node of the two, $\hat{d}_{max}$ is the maximum depth of the ontology, and $\hat{d}(e)$ is the depth of the effect e. Otherwise, if the effects do not have ancestor/descendant relationships, then

$$S(e_x, e_y) = \frac{\hat{d}(NCA(e_x, e_y))}{\hat{d}_{max}},$$

where $NCA(e_x, e_y)$ is the Nearest Common Ancestor (NCA) of e_x and e_y. Note that the depth of the root is defined as 1.

Taking the "Effect" ontology in Figure 5.17 as an example, the semantic similarity between $VisualEffect$ and $ArtificialLightEffect$ is

$$S(VisualEffect, ArtificialLightEffect) = \frac{\hat{d}(VisualEffect)}{\hat{d}_{max}} = \frac{2}{4} = 0.5.$$

Likewise, the semantic similarity of $BoundedVisualEffect$ and $DryEffect$ is

$$S(BoundedVisualEffect, DryEffect) = \frac{\hat{d}(NCA(BoundedVisualEffect, DryEffect))}{\hat{d}_{max}} = \frac{\hat{d}(Effect)}{\hat{d}_{max}} = 0.25,$$

since $BoundedVisualEffect$ and $DryEffect$ do not have a ancestor/descendant relationship. After the similarity values are calculated, the highest value is chosen as the representative value since the most similar node is more likely to cause interference.

In this work, Fuzzy sets are modeled by Sigmoid functions and Gaussian distribution functions (see Fig. 5.18, 5.19, and 5.20). The Sigmoid function is of the form

$$A_{L_1}(x) = \frac{1}{1 + e^{t \cdot (x-u)}}, \tag{5.20}$$

where L_1 is the label of a Fuzzy set; t and u are parameters pre-defined for a specific membership functions of the Fuzzy set L_1. t is used to adjust the shape of function, where as u is used to adjust the offset from the origin. Likewise, the Gaussian

Table 5.7: Membership functions for Fuzzy sets of "distance" and default parameter values

Label	Membership Function	Default Parameter Values
Close	Sigmoid (5.20)	$t = -17$ and $u = 1.4$
Average	Gaussian (5.21)	$\tilde{m} = 2$ and $\sigma = 0.25$
Far	Sigmoid (5.20)	$t = 10$ and $u = 2.5$

Table 5.8: Membership functions for Fuzzy sets of "intensity" and default parameter values

Label	Membership Function	Default Parameter Values
Small	Sigmoid (5.20)	$t = -30$ and $u = 0.3$
Medium	Gaussian (5.21)	$\tilde{m} = 0.5$ and $\sigma = 0.1$
Large	Sigmoid (5.20)	$t = 30$ and $u = 0.7$

distribution functions is of the form

$$A_{L_2}(x) = exp(\frac{-(x - \tilde{m})^2}{\sigma^2}), \tag{5.21}$$

where L_2 is the label of the membership function, and $\tilde{m}$ is the mean and σ is the standard deviation of the distribution. Tables 5.7, 5.8, and 5.9 show respectively how the Fuzzy sets of "distance", "intensity", and "similarity" are defined. The default parameter values are pre-defined based on empirical experiences of the system designer, which can be adjusted afterwards by users or by automatic fuzzy learners such as ANFIS [75].

Figures 5.18, 5.19, and 5.20 depict respectively the Fuzzy sets for "distance", "intensity", and "similarity" diagrammatically after the membership functions and parameter values are applied.

Table 5.9: Membership functions for Fuzzy sets of "similarity" and default parameter values

Label	Membership Function	Default Parameter Values
Dissimilar	Sigmoid (5.20)	$t = -25$ and $u = 0.4$
Average	Gaussian (5.21)	$\tilde{m} = 0.5$ and $\sigma = 0.1$
Similar	Gaussian (5.21)	$\tilde{m} = 0.75$ and $\sigma = 0.75$
Nearly The Same	Sigmoid (5.20)	$t = 35$ and $u = 0.8$

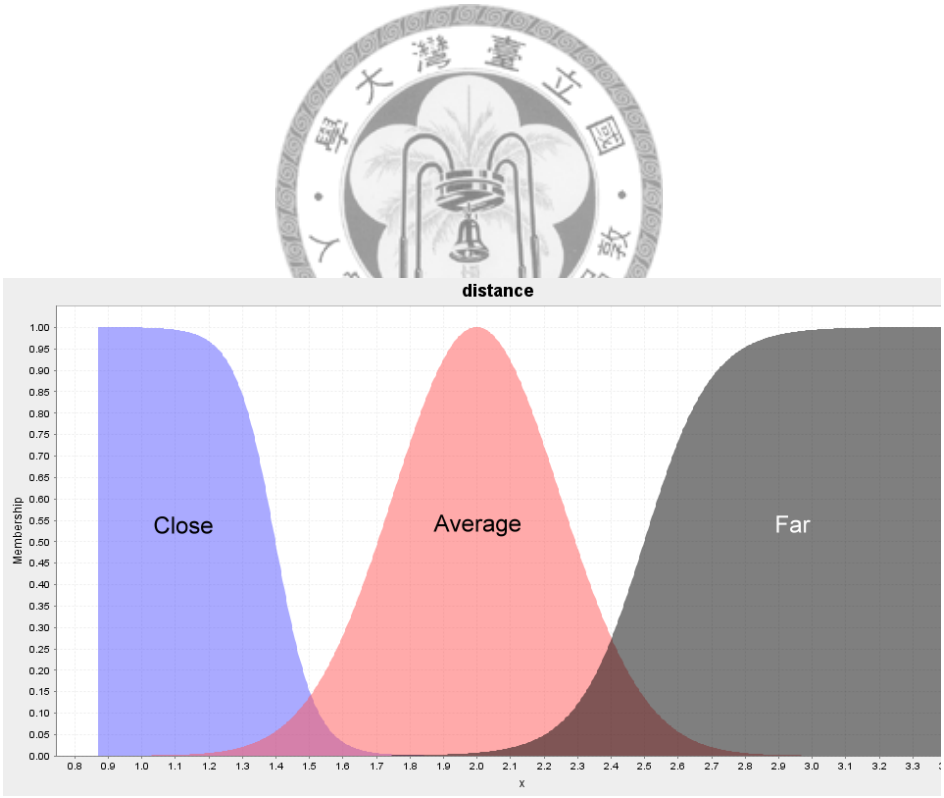


Figure 5.18: Fuzzy sets of "distance"

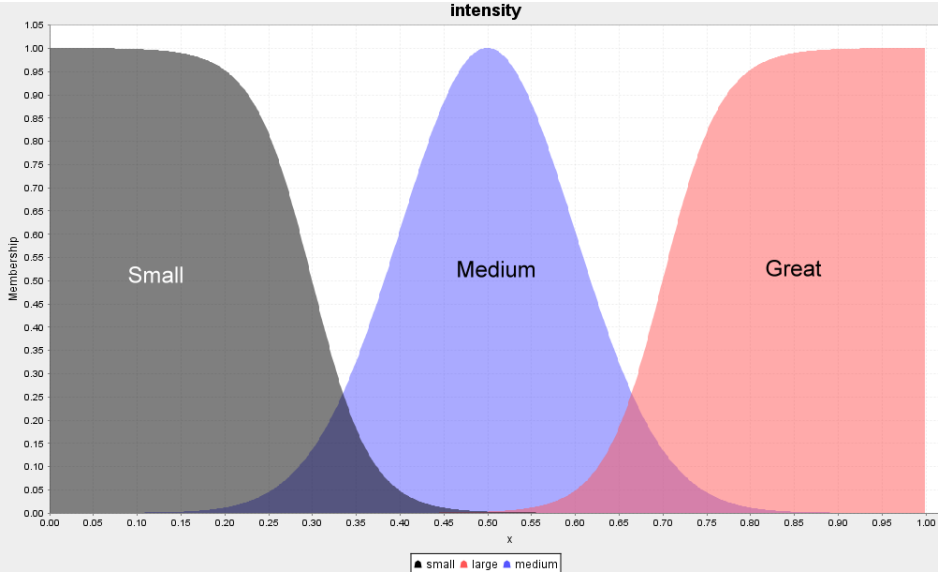


Figure 5.19: Fuzzy sets of "intensity"

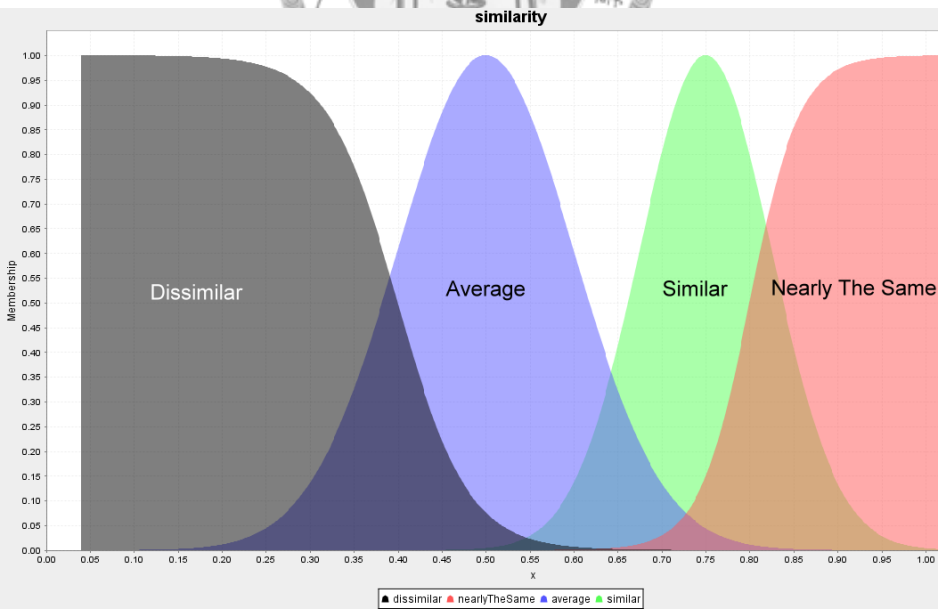


Figure 5.20: Fuzzy sets of "similarity"

Table 5.10: Membership functions for Fuzzy sets of "interference" and default parameter values

Label	Membership Function	Default Parameter Values
Critical	Sigmoid (5.20)	$t = 40$ and $u = 0.8$
Serious	Gaussian (5.21)	$\tilde{m} = 0.75$ and $\sigma = 0.75$
Average	Gaussian (5.21)	$\tilde{m} = 0.4$ and $\sigma = 0.1$
Insignificant	Sigmoid (5.20)	$t = -40$ and $u = 0.2$

Similarly, the output of Fuzzy inference is also represented by Fuzzy sets, as shown in Table 5.10 and Figure 5.21.

Listing 5.9 shows the algorithm used to estimate the interference degree of a candidate Worker Node w. This algorithm use Mamdani's approach for Fuzzy inference [95], where the MIN-MAX model is used for aggregation and accumulation and the results are defuzzified by calculating COG (Center Of Gravity). To ensure the completeness and consistency of rules, one rule is defined for each combination of Fuzzy variables. Consequently, there are totally $3 \times 3 \times 4 = 36$ rules defined (3 for both variables of distance and intensity, and 4 for similarity).

5.4.2 Scoring Candidate Worker Nodes

As mentioned in Section 5.3, the Type-based Node Searching phase is essentially to search for the sets of Worker Nodes whose types are identical to the ones specified in Preference Descriptors. Specifically, assuming that there are k desired Worker Nodes specified in a Service Request, then there are k Preference Descriptors, that is, $\{P(\widetilde{w}_i)\}_{i=1}^{k}$, and k sets of candidate Worker Nodes whose types are the same with the ones specified in $\{P(\widetilde{w}_i)\}_{i=1}^{k}$, namely, $\{W^{\tau_1}, W^{\tau_2}, ..., W^{\tau_k}\} = \{W^{\tau_i}\}_{i=1}^{k}$, where τ_i is the type specified in $P(\widetilde{w}_i)$.

Figure 5.21: Fuzzy sets of "interference"

Listing 5.9: Estimating the degree of interference

Procedure Estimate_Interference_Degree

Input

A^*: A list of attribute sets of all activatedWorker Nodes

A^w: The attribute set of the candidate Worker Node

Local

$\Delta_{distance}, \Delta_{similarity}, \delta_{intensity}$

Return

$\iota \in [0,1]$: The interference degree

Begin

For each $A \in A^*$

$\Delta_{distance}[A] \leftarrow D(A^w[location], A[location])$

$\Delta_{similarity}[A] \leftarrow S(A^w[effect], A[effect])$

End;

$\delta_{intensity} \leftarrow A^w[intensity]$

$\iota \leftarrow$Mamdani$(\min(\Delta_{distance}), \max(\Delta_{similarity}), \delta_{intensity})$

End;

The objective of the Candidate Scoring and Selection phase is therefore to calculate a score for each $w \in W^{\tau_i}$, and then to pick out the one with the highest score as the recommended candidate. The score is calculated based on how well the attribute values of a candidate satisfies the Attribute Preference Descriptors (cf. Definition 12). We quantify the issue mentioned above by defining the concept of Delta Value for attributes.

Definition 17. (Delta Value) *The Delta Value δ is a measure of difference between an attribute value α of the desired Worker Node $\widetilde{w}$ and the value of the same attribute of the candidate Worker Node w.*

Listing 5.10 is the default algorithm for calculating Delta Value δ, where $0 \leq \delta \leq 1$. If the value of α at least satisfies one p-term in $pt(\epsilon)$, then δ is calculated based on the rank of the matched term for an enumerative expression. For numeric expressions, if the value of α at least satisfies one p-term in the compact form of ϵ, then δ is assigned depending on the number of satisfied p-terms. In the cases mentioned above, δ must be less than or equal to 0.5. Otherwise, if the negotiation part of ϵ is matched, then $0.5 < \delta < 1$. Finally, $\delta = 1$ if there is no match or if w does not have the corresponding α. To calculate the rank r in the negotiation part of numeric expressions, attribute values have to be sorted according to the negotiation operator. For $\ll$ and $\gg$, the values are sorted in ascending and descending orders, respectively. In case of $\approx$, the values are sorted according to $min(|b_u - \alpha.v|, |b_l - \alpha.v|)$, where b_u and b_l are upper bound and lower bound of the terms specified in the numeric expression, respectively.

The score of a Worker Node w given a desired node $\widetilde{w}$ can be obtained by accumulating $1 - \delta$ for all α in $\widetilde{w}$. Since the importance of each attribute can be different, we also use a weight vector $\vec{\psi} = (\psi_1, \psi_2, ..., \psi_n)$ to specify the relative importance of attributes, with $\sum_{\psi_i} = 1$, where i is the number of attributes of $\widetilde{w}$. Therefore, the score of w given $\widetilde{w}$ can be formally defined as follows.

Listing 5.10: Algorithm for calculating the Delta Value

Procedure Calculate_Delta_Value
Input
ϵ: The Preference Expression
α: The attribute
$p^- \in (0, 0.5)$: The negotiation penalty
Return
$\delta \in [0, 1]$: The Delta Value for the attribute α
Begin
If α is enumerative **Then**
If v matches the r-th p-term in $pt(\epsilon)$ **Then**
$\delta \leftarrow \frac{r}{2|pt(\epsilon)|}$
Else If v matches the negotiation part of ϵ **Then**
$\delta \leftarrow 0.5 + p^-$
Else $\delta \leftarrow 1$
End If;
Else If α is numeric **Then**
If v at least satisfies partial of ϵ **Then**
$\delta \leftarrow \frac{|pt(\epsilon)| - |\text{satisfied p-terms}|}{2|pt(\epsilon)|}$
Else If v matches the negotiation part of ϵ with rank r **Then**
$\delta \leftarrow 0.5 + \frac{r}{2|pt(\epsilon)|}$
Else $\delta \leftarrow 1$
End If;
End;

Definition 18. (Score of a candidate Worker Node) *The score of a candidate Worker Node w given a desired Worker Node $\widetilde{w}$, denoted as $score(w|\widetilde{w})$, is the weighed sum of all $1-\delta_{\alpha_i}$, where δ_{α_i} is the Delta Value of the i-th attribute specified in $\widetilde{w}$, namely,*

$$score(w|\widetilde{w}) = \vec{\psi} \cdot \vec{\delta^{-}}, \tag{5.22}$$

where $\vec{\psi} = (\psi_1, \psi_2, ..., \psi_n)$, $\vec{\delta^{-}} = (1-\delta_{\alpha_1}, 1-\delta_{\alpha_2}, ..., 1-\delta_{\alpha_n})$, α is the attributes of $\widetilde{w}$, and n is the number of attributes in $\widetilde{w}$.

Finally, based on (5.22) and the interference degrees (see Section 5.4.1), we can obtain the best candidate Worker Node w^{best} as the one with the highest score, namely,

$$w^{best} = \underset{w_j, 1 \le j \le k}{\arg\max} \{(1-\iota(w_j)) \cdot score(w_j|\widetilde{w})\}, \tag{5.23}$$

where k is the number of candidate Worker Nodes.

5.5 Evaluation

This section presents an example scenario that shows possible applications of the techniques proposed in this chapter, namely, negotiable expression, preference unification, and interference estimation. Then, these techniques are evaluated based on a set of quality metrics, namely, Success Rate of Composition (SRC), Success Rate of Matching (SRM), Precision of Composition (PoC), and User Satisfaction Index (USI).

5.5.1 Application Scenario

Bob comes home today after work at 7:00 P.M. Usually, the first thing he wants to do is to watch TV. After identifying Bob's ID at the front door, the pervasive system in the smart home initiates a Watching TV Service. There are three instances where televisions are available; one is with 31 inches, one is 17 inches, and the other is 28

inches. According to Bob's preference $(size, (\geq 30 :\gg))$, the system learns that the one with 31-inch screen is preferred. Unfortunately, the TV with largest screen is now broken. As a result, the 28-inch television is chosen since Bob has specified that it is okay if his preference can not be satisfied, but the size should be the larger the better.

At 8:00 P.M., John, Bob's room-mate, comes home. John is used to watch movie via an on-line media service when he is at home so that he also prefers a display with larger size. There is a device with 17-inch screen being available in the living room. However, the system soon finds that this device has the same effect (see Fig 5.17) with the one used by Bob. Moreover, these devices located at the same room. The ι value of the 17-inch display is high, causing its score to be low. As a result, the system select an alternative display located at the study room so that the interferences between Bob's and John's services are avoided.

After finishing dinner, John and Bob browse magazines in the living room. They used to listen to classical music when reading in the living room but they prefer different composers. Their preferences on composers are listed below:

$$\varepsilon^{Bob} = (== \text{"J. S. Bach"} \rightarrow == \text{"A. Vivaldi"})$$

$$\varepsilon^{John} = (== \text{"M. S. Mozart"} :! = \text{"A. Vivaldi"}).$$

Although Bob's and John's preferences appear to be conflicting, the preferences are still unifiable since John's preference expression specifies that any composer is acceptable except A. Vivaldi. As a result, Bach's music is played since the unified result is $==$ {"J. S. Bach"}.

We can learn from the above application scenario that, due to the negotiable expression, Bob's Watching TV service can still be composed even when the most preferable component is broken. The price is that Bob has to specify a negotiable criterion and the quality of service may be degraded. Also, despite the interference degree is taken into account when selecting nodes, Bob's and John's services are free from interfering

with each other. Finally, when Bob and John are located at the same place, their preferences can be negotiated and then be unified according to the proposed unification mechanisms.

5.5.2 Quality Metrics

According to a recent survey of 24 existing service composition frameworks in pervasive environments [33], 17 of them are categorized as Type-based Service Composition (TBSC), since they only compose the service by simply matching node types; the remaining 7 of them match the values of attributes against a set of user-specified expression, which are called Expression-driven Service Composition (EDSC). In the following, the negotiable and unifiable service composition approach is called the Negotiable-Expression-driven Service Composition (NESC). This work evaluates the quality of the above mentioned approaches based on several metrics. The detailed discussions and experimental results of these metrics are reported below.

All experiments are conducted on P4 1GHz CPU PCs with 1GB memory and all input data are randomly generated to simulate the real world situation. In each experiment, the number of Service Request is set to 1000, the lengths of services are randomly distributed from 3 to 5, and there are totally 15 node types in the system. Each composition method is performed to select candidates among a group of Worker Nodes ranging from 500 to 1500 instances and each node consists of 7 to 11 attributes. Among these attributes, 50% of them are constrained by user preferences. By default, the number of mandatory preferences is equal to the number of negotiable preferences.

Success Rate of Composition (SRC)

One simple way to evaluate the quality of a service composition mechanism is to calculate the Success Rate of Composition (SRC) [80]. SRC is defined as follows:

$$SRC = \frac{n(S^{success})}{n(S)}, \tag{5.24}$$

where $S^{success}$ is the set of services that are successfully composed, and S is the set of services to be composed. Recall that $n(S)$ is the cardinality of the set S (see Definition 7).

As depicted in Figure 5.22, along with increasing of the number of nodes, the success rate for EDSC and NESC (the proposed approach) are also slightly increased, whereas TBSC is steady at approximately 75%. TBSC always has the best SRC score since its selection criteria are less restrictive. More specifically, only the type information is used as a constraint. On the other hand, EDSC has poor SRC, since its Node Preference Descriptor is more restrictive. It is worthy to point out that the SRC of the NESC is higher than that of EDSC because the inclusion of negotiation capability.

The major issue of SRC metric is that it is a coarse-grained measure of the success rate of service composition. Specifically, a service is successfully composed if and only if all of the desired Worker Nodes are found. Assuming that there are n desired Worker Nodes, then the composition fails even when $n - 1$ out of n valid nodes are found. Hence, SRC largely depends on desired Worker Nodes specified in the Service Request so that it can be inaccurate to evaluate the quality of a service composition mechanism by using only the SRC metric.

Success Rate of Matching (SRM)

To deal with the issue mentioned above, a finer-grained metric called the Success Rate of Matching (SRM) is proposed below:

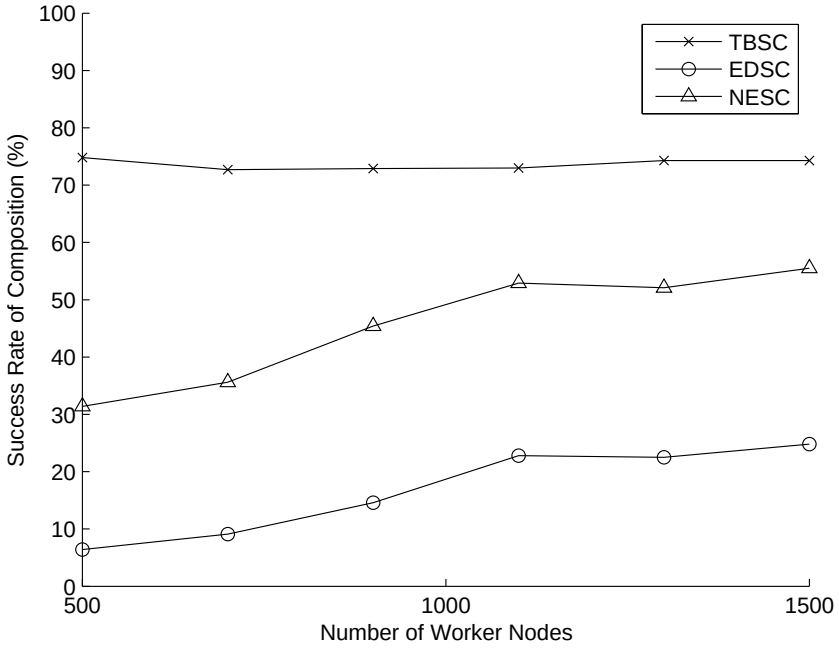


Figure 5.22: Success Rate of Composition (SRC)

$$SRM = \frac{\sum_{s \in S} n(W_s^{valid})}{\sum_{s \in S} n(W_s^{requested})} = \frac{\sum_{s \in S} n(W_s{}^{valid})}{\sum_{s \in S} \ell(s)}, \tag{5.25}$$

where W_s^{valid} is set of Worker Nodes that are successfully found and matched for the service s, S is the set of services to be composed, and $\ell(s)$ is the length of s which is defined in the Definition (8).

The core idea of SRM is to measure the success rate based on the number of successfully found nodes instead of the number of successfully composed services. When a service needs to be composed and if there are $n-1$ out of n nodes which are found, then $\frac{n-1}{n}$ is given to SRM instead of 0. Figure 5.23 shows SRM with different number of Worker Nodes. Both the SRMs of EDSC and NESC slightly increase when the number of nodes is increased. Again,TBSC has the highest SRM. It is interesting to point out that all evaluated approaches have higher score in SRM than in SRC, since SRC is an "all or nothing" type metric. Also note that the SRM scores of both EDSC and NESC

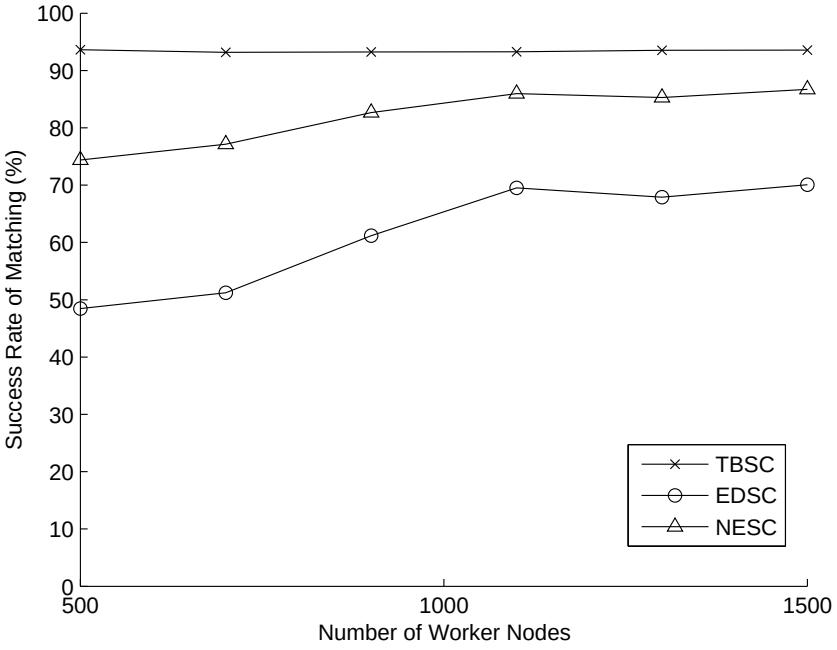


Figure 5.23: Success Rate of Matching (SRM) with different number of Worker Nodes

are higher than that of SRC.

A composition mechanism with high SRM does not guarantee high quality of service. In an extreme case, a composition mechanism can achieve high SRM by simply reporting that all nodes are candidates. As a result, a metric that measures the precision of the timing for reporting candidates is required.

Recently, many researchers found that "recall" and "precision", which are common evaluation measures in the information retrieval field [98], are very useful metrics for evaluating the quality of service composition [121, 117]. According to (5.23), NESC only returns the best node so that $n(W_s^{requested})$ represents the total number of relevant nodes of a Service Request. Therefore, the semantics of SRM is identical to the concept of "recall" which is the number of relevant items retrieved (i.e. $\sum_{s \in S} n(W_s^{valid})$) over the number of total number of relevant items (i.e. $\sum_{s \in S} n(W_s^{requested})$). In the following, the metrics for measuring the precision of a composed service will be presented.

Precision of Composition (PoC)

Precision is the number of relevant items retrieved over the number of total retrieved items [98]. Hence, the Precision of Composition (PoC) can be defined as the number of valid nodes retrieved over the number of total retrieved nodes, namely,

$$PoC = \frac{\sum_{s \in S} n(W_s^{valid})}{\sum_{s \in S} n(W_s^{found})}, \tag{5.26}$$

where $n(W_s^{valid})$ is the number of nodes that fulfills the corresponding Node Preference Descriptors and $n(W_s^{found})$ is the number of nodes found by the PSM.

Figure 5.24 illustrates the PoC of the three approaches with increasing number of nodes. The PoCs of TBSC, EDSC, and NESC are steady at 10%, 78%, and 86%, respectively. It is important to note that the although TBSC gets high SRC/SRM in the previous experiments, it suffers from extremely low PoC. In other words, TBSC tends to retrieve too many candidates causing the precision being extremely low. On the contrary, EDSC is too restrictive so that, although it gets the highest PoC, the success rate (SRC/SRM) is poor. Figure 5.23 and 5.24 show that NESC is able to maintain high score both in SRM/SRC and PoC. Specifically, the NESC is precise enough so that it is able to compose high quality services while maintaining reasonable success rate of composition.

Also, from Fig. 5.24, given that the number of constraints on node attributes are the same, PoC is independent of the number of the Worker Nodes. Therefore, additional experiments are performed to observe the relationship between PoC and the ratios of constrained attributes. The outcomes are shown in Figure 5.25. Observe that the PoCs of TBSC drop rapidly whereas PoCs of other approaches increase gradually. The results show that TBSC is more inappropriate if there are more constraints on attributes.

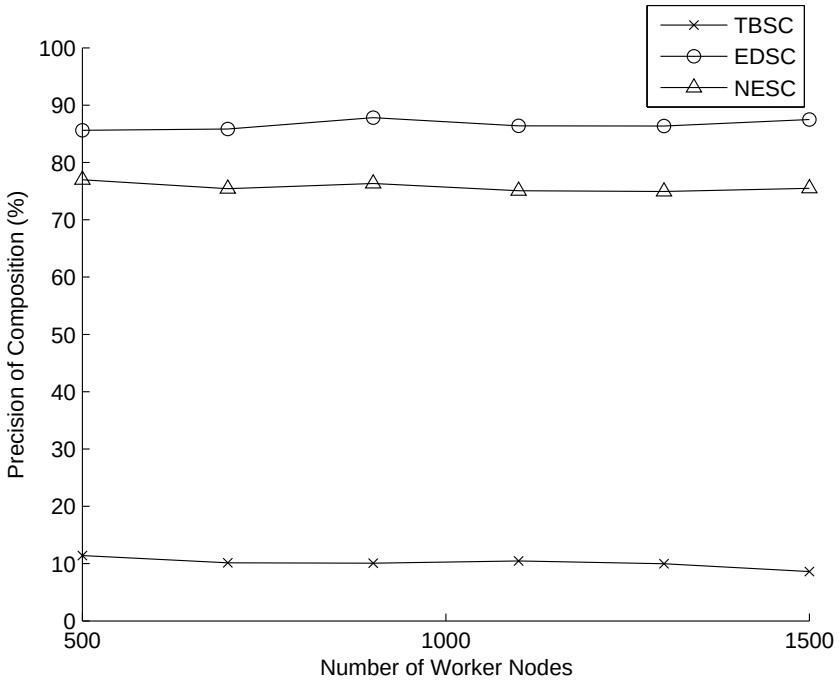


Figure 5.24: Precision of Composition (PoC) with different number of Worker Nodes

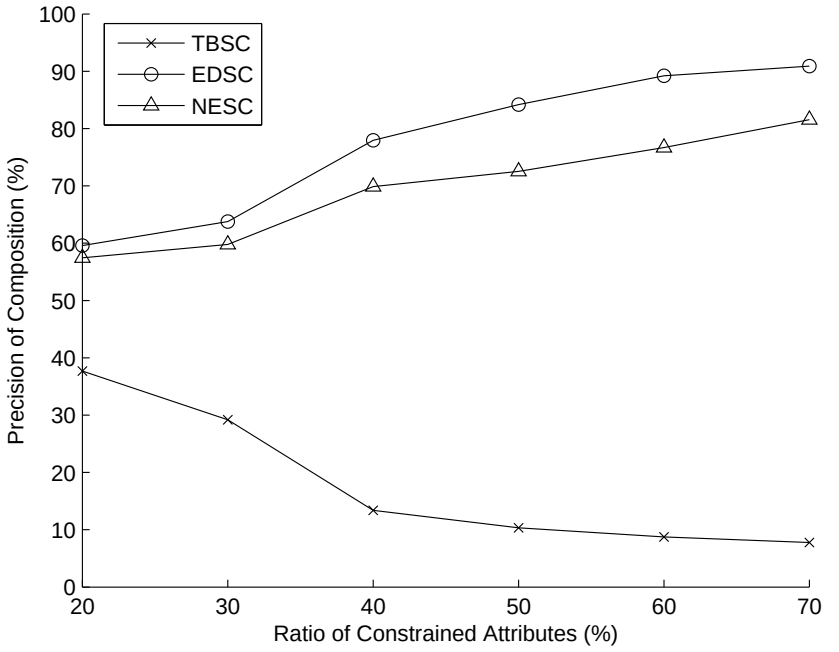


Figure 5.25: Precision of Composition (PoC) with different ratio of constrained attributes

User Satisfaction Index (USI)

In this work, the F_β-score is used to represent the overall quality of composition by integrating SRM and PoC. F_β-score is a popular method used to combine the precision and recall metrics [113], where β is a weight parameter used to adjust the importance between precision and recall. In fact, F_1-score is the harmonic mean of precision and recall. The following equation defines a F_β-based metric called User Satisfaction Index (USI) by integrating the outcomes of SRM and PoC:

$$USI(F_\beta) = (1 + \beta^2) \cdot \frac{PoC \cdot SRM}{(\beta^2 \cdot PoC) + SRM}. \tag{5.27}$$

In real cases, a composition method with low success rate (SRM) usually leads to frustrating user experiences. Users' preferences are usually adjustable, dynamic, and vague so that they are usually willing to negotiate, that is, to adjust their preferences, in order to prevent the composition from failing. When evaluating composition methods, one can put more emphases on success rate (SRM) by increasing β.

Figures 5.26 and 5.27 show the USIs of the three methods when $\beta = 1$ and $\beta = 2$, respectively. The results show that the proposed approach obtains the highest score both in $QoC(F_1)$ and $QoC(F_2)$. When $\beta = 2$, where the metric is in favor of the approaches with higher success rate, The USI score of NESC is obviously much higher than that of EDSC.

It can be concluded that the proposed approach, namely, NESC, is able to achieve high composition precision and maintains reasonable success rate of composition at the same time so that it outperforms the other methods in both $USI(F_1)$ and $USI(F_2)$ metrics.

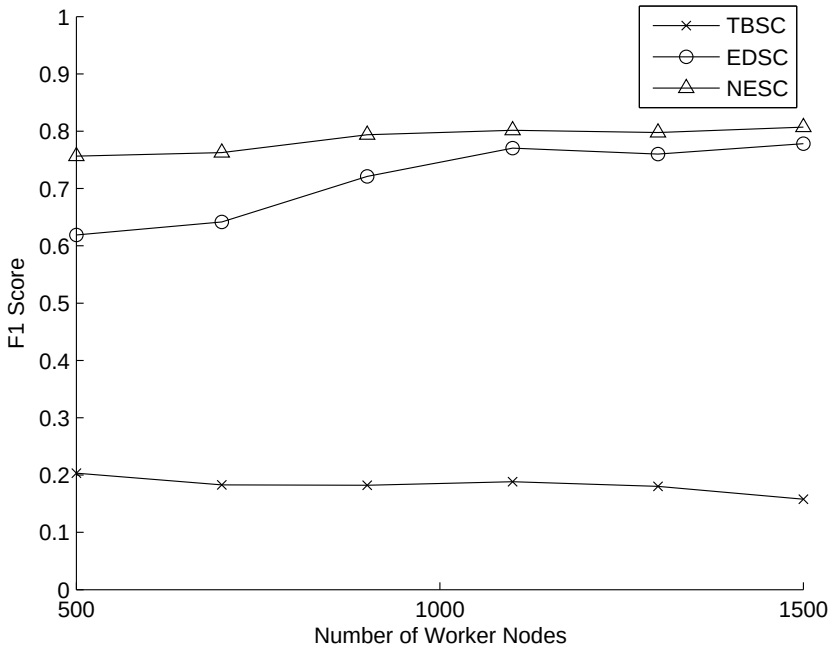


Figure 5.26: F_1 Score with different number of Worker Nodes

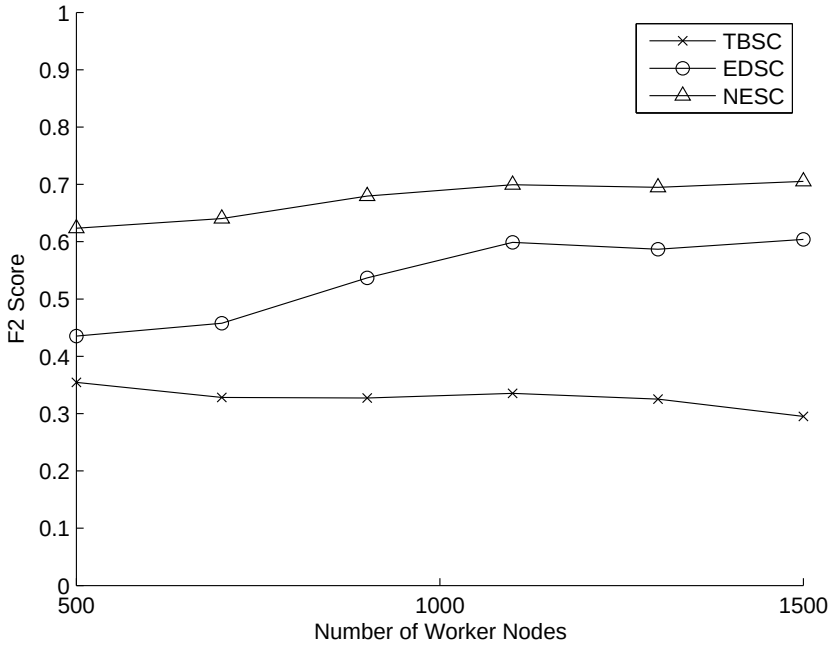


Figure 5.27: F_2 Score with different number of Worker Nodes

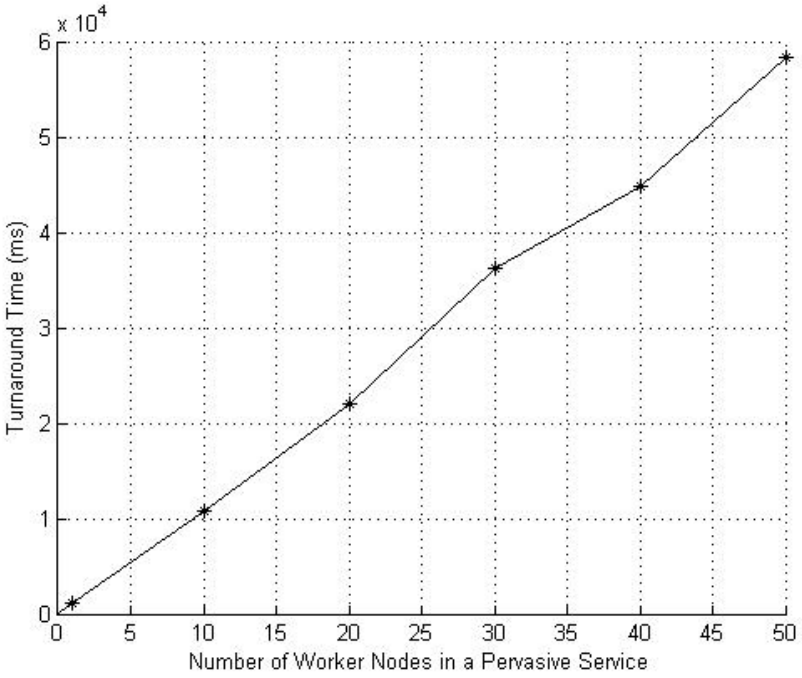


Figure 5.28: Turnaround time of service composition

5.5.3 Performance

To evaluate the performance of the proposed service composition approach, the nodes and their Node Capability Descriptors are first generated and then the turnaround time is measured. In this experiment, the turnaround time is defined as the duration from the time when the Service Request is submitted to the time when all Worker Nodes are activated. After each round, the average length of Pervasive Services are increased and the tests are re-performed, where each round is performed 100 times. The results are shown in Figure 5.28. The turnaround time of service composition increases linearly when the number of nodes increases. In the real world, most Pervasive Services consist of nodes whose quantity is less than 10 nodes. Hence, one can observe from Figure 5.28 that most real-world Pervasive Services require 10 seconds or less before they are available based on the proposed approach.

5.6 Summary

In a Pervasive system such as the Smart Home, the criteria for selecting and ranking services are usually specified by users, which tend to be vague and subjective. Moreover, the deployment of Pervasive services here is usually not as well-planned as that in traditional enterprise environments. Hence, the criteria can be contradictory and the activated services can interfere with one another. This chapter proposes an integrated preference-guided and interference-aware service composition system for composing applications in a smart home. The primary contributions include: 1) we propose a negotiable and unifiable expression language, namely, Preference Expression, along with a set of unification rules for integrating conflicting preferences. Moreover, the expression represents both enumerative and numeric preferences. 2) We propose fuzzy-based mechanism for estimating the degrees of interferences between service components. 3) A scoring scheme is proposed to seamlessly integrate the techniques mentioned above. Experimental results show that the proposed approach is able to greatly increase the success rate of composition especially under strict constraints.

Chapter 6

Implementation

The mechanisms proposed in this thesis are realized as a platform that provides infrastructural support for pervasive services. The Manager Nodes and some Worker Nodes are implemented mainly by using Java language on JDK 6. Other Worker Nodes are implemented with C# and others in C++. This platform uses ActiveMQ [1], an open source MOM, as the message exchanging backbone. ActiveMQ adopts a cross-platform messaging protocol, and supports several programming languages such as C, C++, C#, and Java.

Therefore, a Pervasive Service can be composed of Worker Nodes implemented by means of heterogeneous technologies. This ability is of critical importance in developing pervasive services. UPnP functionalities is implemented based on Intel UPnP SDK [2] for C# and C++ based PerNodes, while Java-based PerNodes are developed by Cyberlink UPnP for Java [3]. There are many CSP-based libraries or toolkits available such as JCSP [139] and PAT [128]. These tools are valuable in the design and validation phase of a protocol. However, these libraries typically built on top of primitive network API, therefore they inevitably lack of the abilities of performing low level control on packets (such as SSDP packet modification). In addition, these implementations are usually alternatives to the primitive thread model; it can be dangerous to mix them with other libraries. Consequently, these libraries are not used in the production release.

An application framework called PerNode SDK (Software Developer's Kits) that supports rapid-prototyping of PerNodes is also developed. PerNode SDK is a Java-

[1]Available at http://activemq.apache.org/

[2]Intel Software for UPnP Technology, available at http://software.intel.com/en-us/articles/intel-software-for-upnp-technology-download-tools/

[3]Available at http://www.cybergarage.org/twiki/bin/view/Main/ CyberLinkForJava

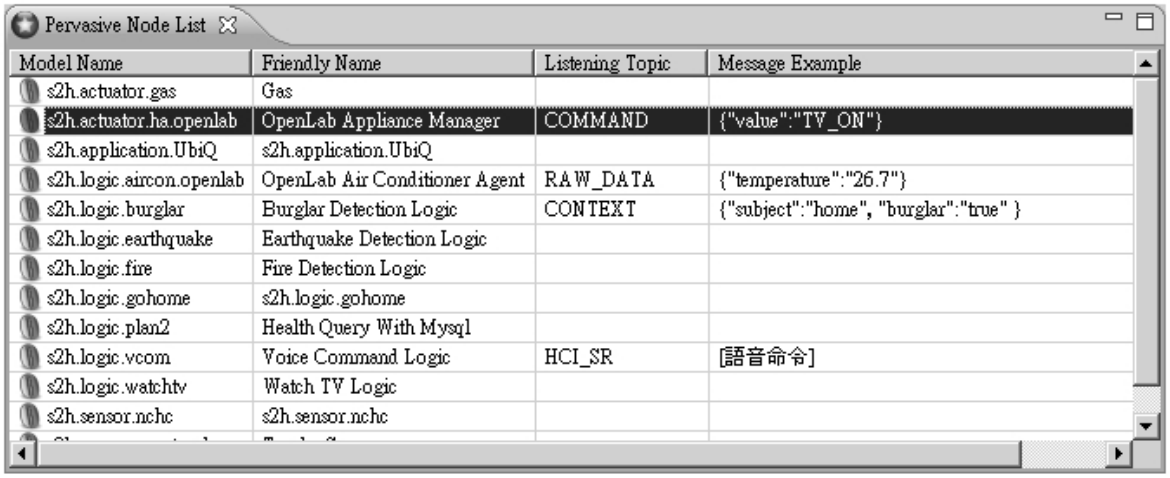


Figure 6.1: The drag-and-drop code generating service

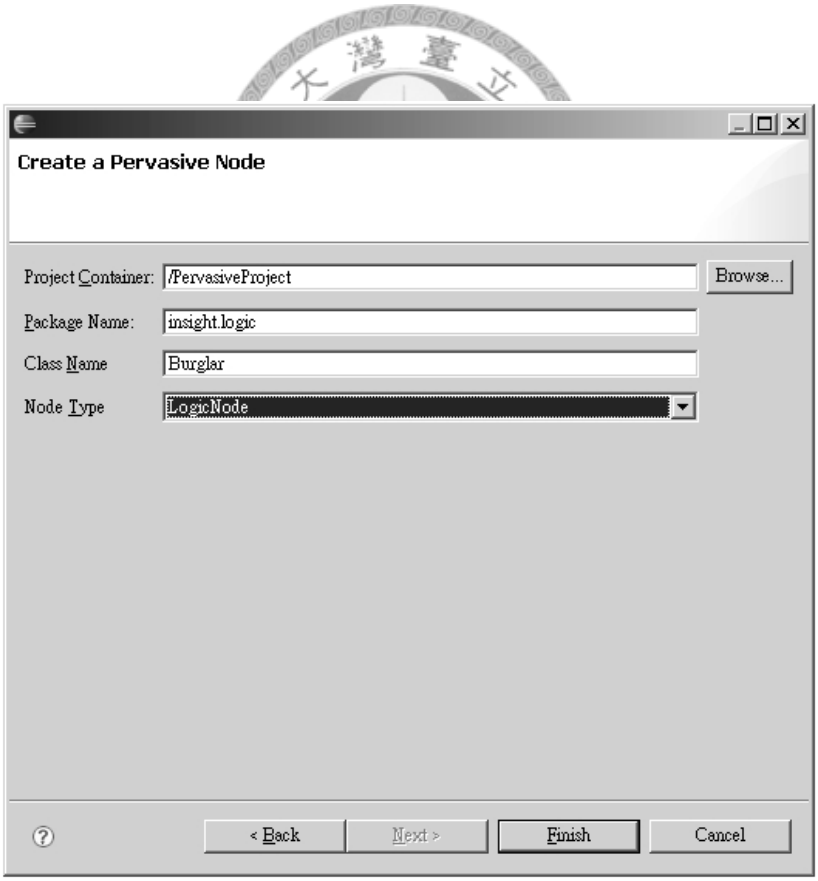


Figure 6.2: The code template generating wizard

Listing 6.1: The code segment that implements a "Media Follow Me" service

```
@MessageBus("failover:(tcp://192.168.4.100:61616)")
@MessageFrom(PlatformTopic.CONTEXT)
@PSMP
public class MediaFollowMeLogic extends LogicNode {
   protected void processMessage(PlatformMessage message)
   { ... }
}
```

based object-oriented application framework that provides design time supports with a set of reusable libraries, interfaces, and default implementations. One of the distinguishing features of PerNode SDK is that it supports attribute-based programming [36]. Therefore, the code becomes intuitive and more comprehensible. For example, the code segment in Listing 6.1 describes a node that provides "Media Follow Me" service. The developers can setup the MOM and the listening topic by using @MessageBus and @MessageFrom, respectively. Also note that @PSMP directs the framework inject PSMP protocol mechanisms into the node. On the other hand, the developers are free to switch to other service management protocols by using other protocol annotations. Currently we support @SSDP or @PSMP. The PerNode SDK provides template-based as well as drag-and-drop code generation services by a set of "Interactive Wizards", which are realized as plug-in modules of the Eclipse IDE [4]. Figure 6.1 is a node browser, from which the developers can drag-and-drop existing PerNodes and then the code will be generated. Figure 6.2 is an interactive wizard that generate a template according to the node attributes specified by developers.

The overall process and the toolchain for constructing a PerNode is depicted in Fig. 6.3. In the Code Generation step, the developer specifies information required for code generation such as the name, version, node type and listening topic in a configuration file (see Fig. 6.4). The code generation is driven by a script file which is executed by

[4]Available at http://www.eclipse.org/

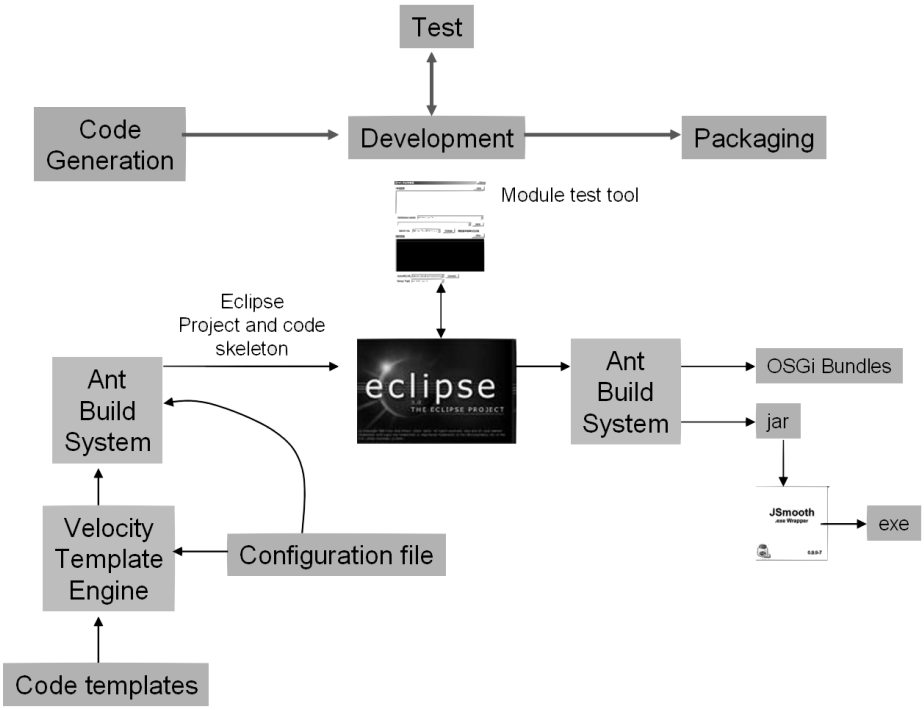


Figure 6.3: The toolchain for constructing PerNode

the Ant build system [5]. The Ant build system generates code skeleton and Eclipse project files based on the specified configuration file and code template. Then, in the Development and Test phase, the developer modifies the generated code in the Eclipse IDE. There is also a testing tool called MQ Simulator which is a useful tool to validate the logic of a PerNode in development time. Finally, in the Packaging step, the code and the related resource files are packaged into a single executable or an OSGi bundles.

Based on the toolchain mentioned above, several Pervasive Services are constructed, some of them are listed in see Table 6.1, and Table 6.2 [6]. These services are deployed in two dissimilar demo sites (Figure 6.5 and Figure 6.6). These sites are different in size (NTU Attentive Home: 400 square feet; NTU INSIGHT Living Lab: 1080 square feet.), partition (NTU Attentive Home is with 1 living room, 1 kitchen and 1 bedroom; NTU INSIGHT Living Lab is with 1 living room, 1 kitchen, 1 toilet, 1 dining room and

[5]Available at http://ant.apache.org/

[6]see http://www.attentivehome.org/video.html for the demo video

```
#  PerNode SDK 3.0
#  Eclipse-based PerNode project generator configuration file
#
#  Chun-Feng Liao
# (2010) National Taiwan Univ. CSIE Dept.
#

# Specify PerNode name
# Note that the name should consists of lower case english characters without any delimeters (conventionally)
# Ex: helloworld , instead of hello-world, helloWorld or hello_world
pernode-name=helloworld

# Specify your name in the following
pernode-author=Chun-Feng Liao

# Specify version of the PerNode
pernode-version=1.0.0

# The generated project depends on the year the project is built, the ${pernode-name} and ${pernode-version}
# Ex:2010-pernode-helloworld-1.0.0

# Specify package name and class name of the PerNode
pernode-package-name=org.attentivehome.pernode.hello
pernode-class-name=HelloPerNode

# Specify Broker URL of ActiveMQ
pernode-mom=failover:(tcp://localhost:61616)
# Specify the topic name from which message comes
pernode-listeningTopic=ssh.CONTEXT
```

Figure 6.4: PerNode Code/Project generator configuration file

Table 6.1: Implemented Pervasive Services

ID	Name	Member Type ID
PS1	Web-based Control and Monitoring	S1, A1, A2
PS2	Media Follow Me	S2, P1, A31, A32, A33
PS3	Fall Detection Alert	S3, L2, A2, A4
PS4	Adaptive Air Conditioner	S1, L3, A2
PS5	Burglar Detection Alert	S1, L4, A2, A4

Figure 6.5: The NTU Attentive Home

2 bedrooms), appliances, and furnishing. Due to their dissimilarities, the developers modified XML-based configuration files for each site in order to deploy the services. However, the source codes need not to be changed.

Table 6.2 shows all services deployed in these environments. These nodes are located in three different hosts (H1, H2 and H3), each host has a PHM. Table 6.1 lists required service types and criteria of Pervasive Services. Notice that there are five Pervasive Services and three Pervasive Hosts. Since several node instances are with the same node types, the PSM can choose among one of them. For instance, the "adaptive air conditioner" service requires node types of S1, P3 and A2. There are two nodes that are with type S1 (PL-2303 and Taroko), hence the PSM can activate one of them when

Figure 6.6: The NTU INSIGHT Living Lab

performing service activation.

It is noteworthy that PerSAM can consist of nodes implemented by means of heterogeneous programming languages. The interoperability of PerSAM makes it a highly extensible integrating platform for pervasive environments. For instance, the real-time image-processing components are better implemented with C or C++ while server-side components are usually implemented with Java language. The cross-platform interoperability is an inherited nature of the MOM, which is very hard to achieve in process-centric architectures as well as Tuple Spaces. Besides, to avoid the possibility of single-point-of-failure, most available MOM supports load-balancing as well as fail-over mechanisms. Consequently, the features discussed above also make this platform both flexible and reliable.

Table 6.2: Implemented PerNodes

Name	Type ID	Type and Criteria	Host ID
PL-2303 Sensor Adapter	S1	Wireless Sensor	H1
Taroko Sensor Adapter	S1	Wireless Sensor	H1
Ekahau Position Engine	S2	Location Sensor	H2
Smart Floor Adapter	S2	Location Sensor	H1
AXIS 207MW Network Camera Adapter	S3	Image Sensor	H1
Control and Monitoring Web Application	A1	Web Application Server	H2
Home Appliance Controller	A2	Home Appliance Controller	H2
Smart Display A	A31	Smart Display, location=livingroom	H1
Smart Display B	A32	Smart Display, location=studyroom	H2
Smart Display C	A33	Smart Display, location=kitchen	H3
Short Message System Gateway	A4	SMS	H2
Media Follow Me Logic	L1	Logic, name=Media Follow Me	H3
Fall Detection Logic	L2	Logic, name=Fall Detection	H3
Air Conditioner Logic	L3	Logic, name=Air Conditioning	H3
Burglar Detection Logic	L4	Logic, name=Burglar	H3

Chapter 7

Conclusion and Future Work

As pointed out in Chapter 1, the objective of this thesis is to investigate approaches for realizing flexible, robust, consistent, and efficient Pervasive service management in a Smart Home. This chapter first explains the contributions of the present research and how these contributions achieve the desired goals as proposed in Chapter 1. Then, several issues that can be explored in the future are suggested.

7.1 Summary of Contribution

The contributions of this thesis and how these contributions achieve the desired goals, that is, flexibility, robustness, consistency, and efficiency, are summarized below.

1. **Flexibility:** The flexibility issues, which include extensibility and interoperability, are addressed in Chapter 2. After reviewing and comparing several representative Pervasive systems, it can be concluded that due to the relief of performance and interoperability issues, MOM is a good choice that benefits from the data-centric architecture while keeps good performance and interoperability at the same time.

2. **Robustness:** Despite the advantages of the MOM architecture, there are still several challenges when designing pervasive systems based on the MOM architecture. Specifically, it lacks a robust service management mechanism that maintains and keeps track of the relationship between services and service components. As a result, Chapter 3 proposes a service application model, called PerSAM, and its auxiliary protocol, called PSMP, for facilitating autonomous service composition, and failure detection and recovery in Message-Oriented Pervasive Systems.

The proposed model and protocols are formally defined by using Process Algebra. Based on these formulations, PerSAM/PSMP has been proved to be robust. The experimental studies show that PSMP has much higher recovery rate than SSDP and is able to recover significant portions of PSs even when the failure rate reaches 100%. The performance evaluations show that for real-world PSs, service composition and failure recovery can be performed within 2 seconds and 0.5 seconds, respectively.

3. **Efficiency:** This research proposes efficient enhancement mechanisms that help PSMP minimizes the downtime of a system while maintains low communication complexity. In Chapter 4, the design, analysis, simulations, and experiments of several techniques for boosting the network efficiency - Decomposing Multicast Traffic, Service-based Node Searching, Heartbeat by Decomposing Multicast Traffic and On-Demand Heartbeat - based on PerSAM and PSMP are presented in detail. Both analyses and simulations reveal that the proposed approaches can reduce message counts of presence and leave announcement, node searching, and heartbeat by more than 93.75%, 66%, and 50%, respectively, in average service lengths.

4. **Consistency:** This research proposes an integrated negotiable and unifiable service composition framework. First, this framework proposes a formal expression notation, namely, the Preference Expression, which is capable of representing negotiable preferences, along with a set of unification rules for merging conflicting preferences. Second, a Fuzzy-logic-assisted technique for interference estimation is proposed. By integrating the proposed techniques, a user-centric service composition framework can be realized. According to the evaluation results, the proposed approach outperforms other methods in the USI (User Satisfaction Index) metric, which means that the proposed approach is able to achieve high

Table 7.1: Enhancements of service model and service management

Comparing Aspect	UPnP	This work
Architectural Style	Process-centric	Data-centric
Service Semantics	-	✓
Expressiveness of Capability	Type	Type and attribute
Expressiveness of Preference	Type	Type and Preference Expression
Discovery Coverage	Active and Dormant nodes	Active, Dormant, and Installed nodes
Recovery Capability	-	✓
Efficiency Improvement	-	✓
Basic Service Composition	-	✓
Consistent Service Composition	-	✓

composition precision and maintains reasonable success rate of composition at the same time.

The above-mentioned mechanisms are realized by constructing a developer's toolkit, called the PerNode SDK, which enables rapid developments of services in MOPS. The toolkit consists of a reusable object-oriented application framework as well as toolkits that enable wizard-based/drag-and-drop styles code generation. The feasibility of the toolkit is demonstrated by developing several Pervasive Services based on the above-mentioned toolkits.

As mentioned in Chapter 3, this work is designed based on the service model and service discovery protocols of UPnP. Table 7.1 summarizes the afore-mentioned contributions by listing the enhancements over UPnP.

7.2 Future Work

Future research could explore the following issues. In PerSAM, the hierarchical architecture can be a cost because of the inclusion of Manager Nodes. The reason for this design is because decentralized failure detection and recovery such as consensus protocols are usually not efficient and are less scalable. In the future, PerSAM/PSMP will be enhanced by a hybrid architecture that employs a centralized approach for Worker Node and a consensus-based approach for Manager Nodes. This approach will be more cost effective since the number of Worker Nodes is much larger than that of Manager Nodes. More concretely, a consensus-based failure detection and recovery protocol will be integrated into PerSAM/PSMP to enhance its robustness.

The most important advantage of forming a Pervasive Host Community is that a PHM is able to accurately detect the presence/absence of a node belonging to the same Pervasive Host. In current design of PSMP, the detection of presence/absence of nodes is carried out by using a distributed mechanism (i.e. heartbeat). As discussed in Chapter 4, in order to achieve higher accuracy of presence/absence detection, the heartbeat mechanism usually produce heavy network traffic. Therefore, one way to enhance the presence/absence detection mechanism is to delegate the job to PHMs since they are able to accurately and efficiently detect the status of local nodes without causing any network traffic. As a result, presence and leave announcements are issued by PHM on behalf of Worker Nodes, so that no heartbeat is needed. This feasibility of this approach depends on the consensus-based failure detection and recovery protocol for Manager Nodes mentioned in the previous paragraph.

As mentioned in Chapter 4, UPnP/SSDP relies on UDP, which is unreliable since UDP loses packets under heavy traffic, and thus causing the management mechanisms become invalid. On the contrary, although TCP is reliable, managing services based on TCP is an overkill since service management packets are usually short and thus do

not require additional functionalities provided by TCP such as congestion control and re-sequencing mechanism. The Wireless Application Protocol (WAP) [2] specification is an industrial standard that is more efficient and therefore useful for wireless applications. The Wireless Transaction Protocol [5], which is part of WAP, is a transport-layer protocol that is able to support reliable communications based on current UDP/IP infrastructure. The reliability is achieved by using unique IDs, acknowledgements, duplicate removal, and re-transmissions. In addition, WTP also support message aggregation so that the traffic can be further reduced. On top of WTP is a specification called Wireless Session Protocol [4], which is essentially an efficient version of HTTP/1.1 in the sense that it uses a binary encoding scheme for headers and data. Hence, WTP appear to be a good starting point for designing an efficient reliable transport protocol based on UDP. Meanwhile, WSP can also be a more efficient replacement of HTTP which is currently adopted by PSMP. As a result, the study about how to improve the efficiency of home network by integrating WTP and WSP into PSMP is under way.

Currently, user preferences are represented and unified in a concrete way. As mentioned earlier, users' preferences are usually vague. Therefore, the syntax and semantics of the Preference Expression can be further extended to facilitate Fuzzy preference representation and unification.

Many services in Smart Homes contain "contents", that is, digitized media such as texts, images, videos and voices that are able to be processed by computers. From a user's point of view, services with different digital contents should be distinguished from one another. For instance, a media player playing different movies provide different user experiences. In other words, the information of contents should be taken into account when selecting and ranking services besides types and QoS attributes of services. Further research is also under way to investigate this type of content-based services.

Besides, although fuzzy-based approaches is used to estimate the interference degree, the selection criteria of membership function is arbitrary (pre-defined by the system designer). After the system is deployed, the fuzzy rules and the parameters of membership function should be adjusted autonomously to reflect user's preferences. This can be achieved by integrating fuzzy-based learning algorithm such as ANFIS [75]. One possible approach is to take the advantages of collective intelligence and to download learning results from a cloud-based service platforms. The interference issue is currently integrated into the node selection process by estimating the possibility of being interfered. In the future, the proposed service composition framework will be enhanced by runtime interference detection capability. The concept of interfering intensity is required which is used to estimate how these interferences affect users. When the intensity of interference is high, a PSM should replace portion of its members to alleviate the interference.

BIBLIOGRAPHY

[1] *Common Object Request Broker Architecture Specifications*. Object Management Group (OMG), 1994.

[2] *Wireless Application Protocol Architecture Specification*. 1998.

[3] *Salutation Architecture*. Salutation Consortium, 1999.

[4] *Wireless Application Protocol Wireless Session Protocol Specification*. 1999.

[5] *Wireless Application Protocol Wireless Transaction Protocol Specification*. 1999.

[6] *CORBA Trading Object Service Specification, Version 1.0*. Object Management Group (OMG), 2000.

[7] *Bluetooth Service Discovery Application Profile, v.1.1*. The Bluetooth Special Interest Group, 2001.

[8] *OWL Web Ontology Language Overview*. W3C Recommendation. World Wide Web Consortium, 2004.

[9] *Web Services Security: SOAP Message Security 1.1 (WS-Security)*. 2004.

[10] *FIPA: The Foundation for Intelligent Physical Agents*. IEEE, 2005.

[11] *OSGi Service Platform Release 4*. OSGi Alliance, 2007.

[12] *Simple Object Access Protocol (SOAP) version 1.2, W3C Recommendation*. WWW Consortium, 2007.

[13] *FIPA Agent Management Specification*. IEEE, 2008.

[14] *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium, 2008.

[15] *UPnP Device Architecture 1.1, ISO/IEC DIS 29341*. UPnP Forum, 2008.

[16] *ZigBee Specification 053474r17*. 2008.

[17] G. D. Abowd. Software engineering issues for ubiquitous computing. In *Proc. 21st International Conference on Software Engineering (ICSE '99)*, pages 75–84, 1999.

[18] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proc. 17th Symposium on Operating System Principles*, 1999.

[19] F. K. Aldrich. *Inside the Smart Home*. Springer-Verlag London Limited, 2003.

[20] K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification*. Addison-Wesley, 1999.

[21] J. C. Augsto and C. D. Nugent. *Designing Smart Homes*. Springer-Verlag, Berlin, 2006.

[22] J. W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. In *Proc. International Conference on Information Processing*, pages 125–132, 1959.

[23] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems. *Communications of the ACM*, 46(2), 2003.

[24] G. Banavar, T. Chandra, R. Strom, and D. Sturman. A case for message oriented middleware. In *Proc. 13th International Symposium on Distributed Computing (DISC'99)*, 1999.

[25] W. C. Barker. *NIST 800-67 Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*. 2008.

[26] A. Bedrouni, R. Mittu, A. Boukhtouta, and J. Berger. *Distributed Intelligent Systems: A Coordination Perspective*. Springer, 2009.

[27] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Ltd., 2007.

[28] J. A. Bergstra, A. Ponse, and S. A. Smolka. *The Handbook of Process Albegra*. Elsevier, 2001.

[29] C. Bettini and D. Riboni. Profile aggregation and policy evaluation for adaptive internet service. In *Proc. IEEE International Conference on Mobile and Ubiquitous Systems*, pages 290–298, 2004.

[30] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[31] G. Booch, I. Jacobson, and J. Rumbaugh. *the Unified Modeling Language Specification, Version 1.3*. 2000.

[32] A. Bottaro and R. S. Hall. Dynamic selection and ranking in context-aware service composition. In *Proc. of the 6th International Conference on Software Composition (LNCS 4829, SC 2007)*.

[33] J. Bronsted, K. M. Hansen, and M. Ingstrup. Service composition issues in pervasive computing. *IEEE Pervasive Computing*, 9(1):62–70, 2010.

[34] B. Cain, S. Deering, and I. Kouvelas. *Internet Group Management Protocol, Version 3, RFC 3376*. 2002.

[35] M. Calder, M. Kolberg, E. H. Magil, and S. R. Marganiec. Feature interaction: a critical review and considered forecast. *Computer Networks*, 41(1):115–141, 2003.

[36] V. Cepa. *Attribute Enabled Software Development*. VDM Verlag Dr. Mueller, 2007.

[37] R. Cerqueira, C. Cassino, and R. Ierusalimschy. Dynamic component gluing across different componentware systems. In *Proc. International Symposium on Distributed Objects and Applications (DOA'99)*, pages 362–371, 1999.

[38] D. Chappel. *Trouble with CORBA*. 1998.

[39] H. Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, 2004.

[40] H. Chen, T. Finin, and A. Joshi. Semantic web in in the context broker architecture. In *Proc. IEEE International Conference on Pervasive Computer and Communications (PerCom'04)*, 2004.

[41] H. Chen, T. Finin, and A. Joshi. *The SOUPA Ontology for PervasiveComputing*. Springer-Verlag, 2005.

[42] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco. Mobile data collection in sensor networks: The tinylime middleware. *Journal of Pervasive and Mobile Computing*, 4(1):446–469, 2005.

[43] C. Dabrowski and K. Mills. Understanding self-healing in service discovery systems. In *Proc. Workshop on Self-healing systems*, 2002.

[44] C. Dabrowski, K. Mills, and S. Quirolgico. Understanding failure response in service discovery systems. *The Journal of Systems and Software*, 80(6):896–917, 2007.

[45] J. Daemen. *The design of Rijndael: AES-the advanced encryption standard*. 2002.

[46] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, 2000.

[47] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 1(5), 2001.

[48] A. K. Dey, T. Sohn, S. Streng, and J. Kodama. icap: Interactive prototyping of context-aware applications. In *Proc. of International Conference on Pervasive Computing (Pervasive'06)*. Springer, 2006.

[49] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1, RFC 4346".* 2006.

[50] S. Dixit and R. Prasad. *Home Networking Challenges*. Wiley-Inderscience, 2008.

[51] D. Eastlake and P. Jones. *RFC 3174 - US Secure Hash Algorithm 1 (SHA1)*. 2001.

[52] W. K. Edwards. Discovery systems in ubiquitous computing. *IEEE Pervasive Computing*, 5(2), 2006.

[53] W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. In *Proc. 3rd International Conference on Ubiquitous Computing (UbiComp'01)*, pages 256–272, 2001.

[54] C. Ellison. *UPnP Security Ceremonies Design Document*. 2003.

[55] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec. The many faces of publish-subscribe. *ACM Computing Survey*, 35(2), 2003.

[56] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proc. IEEE Workshop on Hot Topics in Operating Systems*, 2001.

[57] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6), 1997.

[58] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: Towards distraction-free pervasive computing. *IEEE Pervasive Computing*, 21(2), 2002.

[59] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1), 1985.

[60] L. Gong. Jxta: a network programming environment. *IEEE Internet Computing*, 5(3):88 – 95, 2001.

[61] R. Grimm. One.world: Experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, 3(3), 2004.

[62] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. Systems directions for pervasive computing. In *Proc. 8th Workshop on Hot Topics in Operating Systems*, 2001.

[63] R. Grimm, J. Davis, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. System support for pervasive applications. *ACM Trans. on Computer Systems*, 22(4), 2004.

[64] T. Gu, H. K. Pung, and D. Q. Zhang. Toward an osgi-based infrastructure for context-aware applications. *IEEE Pervasive Computing*, 3(4), 2004.

[65] T. Gu, H. K. Pung, and D. Q. Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28, 2005.

[66] S. Guan. *IGMP-extension User Manual*. 2009.

[67] E. Guttman. Service location protocol: automatic discovery of ip network services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[68] R. Harper. *Inside the Smart Home*. Springer-Verlag, London, 2003.

[69] M. Henning. The rise and fall of corba. *ACM Queue*, 2006.

[70] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), 1978.

[71] G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. Addison Wesley, MA, 2004.

[72] J. I. Hong and J. A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2):287–303, 2001.

[73] C. L. Hu, Y. J. Huang, and W. S. Liao. Multicast complement for efficient upnp eventing in home computing network. In *Proc. IEEE International Conference on Portable Information Devices (PORTABLE'07)*, 2007.

[74] T. Issariyakul and E. Hossain. *Introduction to Network Simulator NS2*. Springer, 2008.

[75] J. S. R. Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man and Cybernetics*, 22(3):665–685, 1993.

[76] B. Johanson. *Application Coordination Infrastructure for Ubiquitous Computing Rooms*. PhD thesis, 2002.

[77] B. Johanson and A. Fox. The event heap: A coordination infrastructure for interactive workspaces. In *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, 2002.

[78] Y. W. Jong, C. F. Liao, and L. C. Fu. A rotating roll-call-based adaptive failure detection and recovery protocol for smart home environments. In *Proc. 7th International Conference On Smart homes and health Telematics (ICOST'09)*, 2009.

[79] M. B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS*. Packt Publishing, 2 edition, 2006.

[80] S. Kalasapur, M. Kumar, and B. Shirazi. Evaluating service oriented architecture (soa) in pervasive computing. In *Proc. IEEE International Conference on Pervasive Computing and Communications (PerCom'06)*, 2006.

[81] D. O. Keck and P. J. Kuehn. The feature and service interaction problem in telecommunications system: A survey. *IEEE Transactions on Software Engineering*, 24(10):779–796, 1998.

[82] T. Kindberg and A. Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1), 2002.

[83] M. Klusch and A. Gerber. Fast composition planning of owl-s services and application. In *Proc. European Conference on Web Services*, 2006.

[84] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0*. W3C Recommendation. 2004.

[85] S. Knauth, R. Kistler, D. Kaslin, and A. Klapproth. Upnp compression implementation for building automation devices. In *Proc. 5th IEEE International Conference on Industrial Informatics*, 2007.

[86] M. Kolberg, E. H. Magill, and M. Wilson. Compatibility issues between services supporting networked appliances. *IEEE Communications*, 41(11):136–147, 2003.

[87] G. Kotz and D. Solar: Towards a flexible and scalable data-fusion infrastructure for ubiquitous computing. In *Proc. ACM International Conference on Ubiquitous Computing (UbiComp'01)*, 2001.

[88] S. Kumar, P. R. Cohen, and H. J. Levesque. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proc. 4th International Conference on Multi-Agent Systems*, pages 159–166, 2000.

[89] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3), 1994.

[90] G. Lee, P. Faratin, S. Bauer, and J. Wroclawski. A user-guided cognitive agent for network service selection in pervasive computing environments. In *Proc. of IEEE International Conference on Pervasive Computing and Communications*, 2004.

[91] Y. Li, J. Huai, H. Sun, T. Deng, and H. Guo. Pass: An approach to personalized automated service composition. In *Proc. of IEEE International Conference on Service Computing*, pages 283–290, 2008.

[92] S. Loke. *Context-Aware Pervasive Systems - Architectures for a New Breed of Applications*. Auerback Publications, Taylor & Francis Group, 2007.

[93] H. K. Low, D. Chieng, A. K. Mustapha, Y. C. Ngeow, and E. Goh. A feature interaction conflicts detection engine for pervasive networked environment. In *Proc. International Conference on Multimedia and Ubiquitous Engineering*, pages 891–896, 2007.

[94] C. H. Lu and L. C. Fu. Robust location-aware activity recognition using wireless sensor networks in an attentive home. *IEEE Transactions on Automation Science and Engineering*, 2008.

[95] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal on Man Machine Studies*, 7(1), 1975.

[96] M. I. Mandel, G. E. Poliner, and D. P. W. Ellis. Support vector machine active learning for music retrieval. *ACM Journal of Multimedia System*, 21(1):3–13, 2005.

[97] W. C. Mann and B. R. Milton. *Home Automation and Smart Environments to Support Independence*. John Wilery & Sons, 2005.

[98] C. D. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Reterival*. Cambridge University Press, 2009.

[99] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. M. Dermott, S. Mcllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. *OWL-S: Semantic Markup for Web Services*. 2004.

[100] Y. Mazuryk and J. J. Lukkien. Analysis and improvements of the eventing protocol for universal plug and play. In *Proc. IASTED Conference on Communications, Internet and Information Technology*, 2004.

[101] J. McCarthy. Circumscription a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[102] E. Meshkova, J. Riihijarvi, M. Petrova, and P. Mahonen. A survey on resource discovery mechanisms, peer-to-peer, and service discovery frameworks. *Computer Networks*, 52(11):2097–2128, 2008.

[103] N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[104] T. P. Moran and P. Dourish. *Human-Computer Interaction*, volume 16. Lawrence Erlbaum Associates, 2001.

[105] A. L. Murphy, G. P. Picco, and G. C. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology*, 15(3):279–328, 2006.

[106] K. Nakamura, M. Ogawa, T. Koita, and K. Sato. Implementation and evaluation of caching method to increase the speed of upnp gateway. In *Proc. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC'08)*, 2008.

[107] M. Nakamura, H. Igaki, and K. Matsumoto. Feature interactions in integrated services of networked home appliances -an object-oriented approach. In *Proceedings of International Conference on Feature Interactions in Telecommunication Networks and Distributed Systems*, pages 236–251, 2005.

[108] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proc. of International Semantic Web Conference (ISWC)*.

[109] V. Poladian, D. Garlan, and M. Shaw. Selection and configuration in mobile environments: A utility-based approach. In *Proc. Fourth Workshop on Economics-Driven Software Engineering Research*, 2002.

[110] S. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. Icrafter: A service framework for ubiquitous computing environments. In *Proc. 3rd International Conference on Ubiquitous Computing*, pages 56–75, 2001.

[111] A. Ranganathan, S. Chetan, J. A. Muhtadi, R. H. Campbell, and M. D. Mickunas. Olympus: A high-level programming model for pervasive computing environments. In *Proc. 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'05)*, pages 7–16, 2005.

[112] M. Rausand and A. Hoyland. *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley, 2 edition, 2004.

[113] C. J. V. Rijsbergen. *Information Retrieval*. Butterworth, 2 edition, 1979.

[114] M. Roman. *An Application Framework for Active Space Applications*. PhD thesis, 2003.

[115] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4), 2002.

[116] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proc. International Conference on Human Factors in Computing Systems (CHI '99)*, 1999.

[117] M. Sathya, M. Swarnamugi, P. Dhavachelvan, and G. Sureshkumar. Evaluation of qos based web-service selection techniques for service composition. *International Journal of Software Engineering*, 1(5):73–90, 2010.

[118] M. Satyanarayanan. Mobile information access. *IEEE Personal Communication*, 3(1), 1996.

[119] C. S. Shankar, A. Ranganathan, and R. Campbell. An eca-p policy-based framework for managing ubiquitous computing environments. In *Proc. IEEE International Conference on Mobile and Ubiquitous Systems*, pages 33–42, 2005.

[120] R. Sharp. *Principles of Protocol Design*. Springer-Verlag, 2008.

[121] E. Silva, L. F. Pires, and M. v. Sinderen. A framework for the evaluation of semantics-based service composition approaches. In *Proc. of 7th IEEE European Conference on Web Services (ECOWS)*.

[122] H. A. Simon. *The Sciences of the Artificial*. 1996.

[123] E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.

[124] M. E. Smid and D. K. Branstad. Data encryption standard: past and future. *Proceedings of the IEEE*, 76(5):550–559, 1988.

[125] J. P. Sousa. *Scaling Task Management in Space and Time:Reducing User Overhead in Ubiquitous-Computing Environments*. PhD thesis, 2005.

[126] J. P. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw. Task-based adaptation for ubiquitous computing. *IEEE Transactions on Systems, Man, and Sybernetics - Part C: Applications and Reviews*, 36(3), 2006.

[127] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz. A message-oriented middleware for sensor networks. In *Proc. International Workshop on Middleware for Ubiquitous and Ad-Hoc Computing*, 2004.

[128] J. Sun, Y. Liu, J. S. Dong, and C. Q. Chen. Integrating specification and programs for system modeling and verification. In *Proc. International Symposium on Theoretical Aspects of Software Engineering*, 2009.

[129] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet Computing*, 8(3):66–73, 2004.

[130] H. Takeda, P. Veerkamp, T. Tomiyama, and H. Yoshikawam. Modeling design processes. *AI Magazine*, 11(4):37–48, 1990.

[131] I. Taylor, B. Adamson, I. Downard, and J. Macker. Agentj: Enabling java ns-2 simulations for large scale distributed multimedia applications. In *Proc. 2nd International Conference on Distributed Frameworks for Multimedia Applications*, pages 1–7, 2006.

[132] R. Thiagarajan, M. Stumptner, and W. Mayer. Semantic web service composition by consistency-based model refinement. In *Proc. IEEE Asia-Pacific Services Computing Conference*, pages 336–343, 2008.

[133] D. T. Tran and E. Choi. A reliable udp for ubiquitous communication environments. In *Proc. WSEAS International Conference on Computer Engineering and Applications*, 2007.

[134] S. Tsang and E. H. Magil. Learning to detect and avoid run-time feature interactions in intelligent networks. *IEEE Transactions on Software Engineering*, 24(10), 1998.

[135] V. K. Vaishnavi and W. K. Jr. *Design Science Research Methods and Patterns - Innovating Information and Communication Technology*. Auerbach Publications, Taylor & Francis Group, 2008.

[136] K. Vanthournout, G. Deconinck, and R. Belmans. A taxonomy for resource discovery. *Personal and Ubiquitous Computing*, 9(2):81–89, 2005.

[137] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using owl. In *Proc. of the IEEE Conference on Pervasive Computing and Communications Workshops*, 2004.

[138] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[139] P. Welch and J. Martin. Formal analysis of concurrent java systems. In *Proc. Communicating Process Architectures*, 2000.

[140] B. Whetten, S. Kaplan, and T. Montgomery. A high performance totally ordered multicast protocol. In *Proc. Of INFOCOMM'95*, 1995.

[141] T. Winograd. Architectures for cotext. *Human-Computer Interaction*, 16(2-4):401–419, 2001.

[142] C. L. Wu, C. F. Liao, and L. C. Fu. Service-oriented smart home architecture based on osgi and mobile agent technology. *IEEE Transactions on Systems, Man and Cybernetics - Part C*, 37(2), 2007.

[143] K. Yaghmour, J. Masters, G. B. Yossef, and P. Gerum. *System Monitoring*, page 85. O'reilly Media, Inc., 2008.

[144] Z. Yu, Y. Nakamura, D. Zhang, S. Kajita, and K. Mase. Content provisioning for ubiquitous learning. *IEEE Pervasive Computing*, 7(4):62–70, 2008.

[145] A. Zeidler. *Event-based Middleware for Pervasive Computing: Foundations, Concepts, Design*. VDM Verlag Dr. Muller, 2007.

[146] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Transaction on Software Engineering*, 30(5), 2004.

[147] L. J. Zhang, J. Zhang, and H. Cai. *Service Computing*. Springer and Tsinghua University Press, 2007.

[148] F. Zhu, M. W. Mutka, and L. M. Ni. Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, 4(4):81–90, 2005.

PUBLICATION LIST

*: Thesis-Related Works

International Journal Papers

1. Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu, "Toward Reliable Service Management in Message-Oriented Pervasive Systems," in *IEEE Transactions on Service Computing*, 2010. (to appear, `http://doi.ieeecomputersociety.org/10.1109/TSC.2010.59`) [*]

2. Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu, "Toward a Message-Oriented Application Model and its Middleware Support in Ubiquitous Environments", *International Journal of Hybrid Information Technology*, vol.1, no.3, July 2008. [*]

3. Chao-Lin Wu, Chun-Feng Liao, and Li-Chen Fu, "Service-Oriented Smart Home Architecture based on OSGi and Mobile Agent Technology", *IEEE Transactions on Systems, Man and Cybernetics - Part C*, vol.37, no.2, 2007. [*]

4. Ching-Hu Lu, Chun-Feng Liao, Chao-Lin Wu, and Li-Chen Fu, "Real-Time Fine-Grained Multiple-Target Tracking on a Extensible Virtual Fab Architecture Using Multi-Agents", *International Journal of Electronic Business Management*, vol.5, no.1, 2007.

International Journal Papers (Work In Progress)

1. Chun-Feng Liao, Hsin-Chih Chang, and Li-Chen Fu, "A Preference-Driven Composition System for Consistent Smart Home Applications, " to be submitted to *IEEE Transactions on Systems, Man and Cybernetics - Part C*, 2011. [*]

2. Chun-Feng Liao, Hsin-Chih Chang, and Li-Chen Fu, "Message-Efficient Service Management Schemes for MOM-based UPnP Networks, " submitted to *IEEE Transactions on Service Computing*, 2011. (Conditionally Accepted 2011.6.8) [*]

3. Ya-Wen Jong, Chun-Feng Liao, Hsin-Chih Chang, and Li-Chen Fu, "A Rotating Roll-call based Adaptive Failure Detection and Recovery Protocol for Ambient Services, " submitted to *International Journal of Automation and Smart Technology*, 2011. (Under Review) [*]

Domestic Journal Papers

1. Chih-Ming Chen, Chun-Feng Liao, Ya-Wen Jong, Li-Chen Fu, and Ching-Nian Chang, "Message-Oriented Service Technologies for Digital Homes, " in *TL Technical Journal*, vol.39, no.5, Oct 2009. [*]

2. Chun-Feng Liao, Hsin-Chih Chang, and Li-Chen Fu, "An Intelligent Guideline-based Home Health Care Service Platform, " in *TL Technical Journal*, vol.39, no.5, Oct 2009. [*]

3. Li-Chen Fu, Chao-Lin Wu, Ching-Hu Lu, Chun-Feng Liao, Yu-Chieh Ho, and Yong-Cheng Liu, "The NTU Attentive Home, " in *Automation*, vol.20, no.4, pp.18-35, 2009. [*]

International Conference Papers

1. Hsin-Chih Chang, Chun-Feng Liao, and Li-Chen Fu, "Unification of Multiple Preferences and Avoidance of Service Interference for Service Composition in Context-Aware Pervasive Systems," in *Proc. of 7th ACM International Conference on Pervasive Services* (ACM SIGAPP ICPS'10), Berlin, Germany, 2010. [*]

2. Chun-Feng Liao, Hsin-Chih Chang, and Li-Chen Fu, "A Guideline Execution Engine for Healthcare Services in Smart Home Environments," in *Proceedings of 8th International Conference On Smart homes and health Telematics* (ICOST 2010), Seoul, Koera, 2010. (Springer LNCS 6159) [*]

3. Chun-Feng Liao, Hsin-Chih Chang, and Li-Chen Fu, "Boosting the Efficiency of the Reliable Service Management Protocol for Message-Oriented Pervasive Systems," in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications* (IEEE SOCA'09), Taipei, Taiwan, 2009. [*]

4. Hsin-Chih Chang, Chun-Feng Liao, Yong-Cheng Liu, and Li-Chen Fu, "A Spontaneous Preference Aware Service Composition Framework for Message-Oriented Pervasive Systems," in *Proceedings of the 4th International Conference on Pervasive Computing and Applications* (ICPCA'09), Taipei, Taiwan, 2009. [*]

5. Ya-Wen Jong, Chun-Feng Liao, and Li-Chen Fu, "A Rotating Roll-call-based Adaptive Failure Detection and Recovery Protocol for Smart Home Environments," in *Proceedings of 7th International Conference On Smart homes and health Telematics (ICOST 2009, Springer LNCS 5597)*, Tours, France, 2009. [*]

6. Chi-Pang Lam, Wei-Jen Kuo, Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu, " An Efficient Hierarchical Localization for Indoor Mobile Robot with Wireless Sensor and Pre-Constructed Map ," in *Proceedings of the 5th International Conference on Ubiquitous Robots and Ambient Intelligence* (URAI 2008), Korea, 2008.

7. Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu, "PSMP: A Fast Self-Healing and Self-Organizing Pervasive Service Management Protocol for Smart Home Environments," in *Proceedings of 2008 IEEE Asia-Pacific Services Computing Conference* (IEEE APSCC 2008), Yilan, Taiwan, 2008. [*]

8. Ya-Wen Jung, Chun-Feng Liao, and Li-Chen Fu, "An Efficient Autonomous Failure Recovery Mechanism for UPnP-based Message-Oriented Pervasive Services," in *Proceedings of 2008 IEEE International Conference on System, Man, and Cybernetics* (IEEE SMC 2008), Singapore, Oct 2008. [*]

9. Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu, "Community-based Autonomous Service Activation and Failure Recovery in a Message-Oriented Pervasive Middleware," in *Proceedings of 2008 International Workshop on Context-Aware Pervasive Communities: Infrastructures, Services and Applications* (CAPC 2008, Held in Conjunction with Pervasive 2008), Sydney, Australia, 2008. [*]

10. Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu, "Toward a Message-Oriented Application Model and its Middleware Support in Ubiquitous Environments," in *Proceedings of 2008 International Conference on Multimedia and Ubiquitous Engineering* (MUE 2008), Busan, Korea, 2008. [*]

11. Wan-rong Jih, Jane Yung-jen Hsu,Chao-Lin Wu, Chun-Feng Liao, and Shao-you Cheng, "A Multi-Agent Service Framework for Context-Aware Elder Care," in *Proceedings of Workshop of Service-Oriented Computing and Agent-Based Engineering* (SOCABE'2006), Hakodate, JAPAN, 2006. [*]

Domestic Conference Papers

1. Ching-Hu Lu, Chun-Feng Liao, Chao-Lin Wu, and Li-Chen Fu, "Real-Time Fine-Grained Multiple-Target Tracking on A Virtual Fab Architecture Based on Multi-Agents," in *Proceedings of 2005 Taiwan Artifical Intelligence and Application Conference* (TAAI 2005), Kaohsiung, Taiwan, 2005.

2. Chun-Feng Liao, Cheng-Rong Yu, Zhi-Yang Chen, Da-Wei Chan and Li-Chen Fu, "Behavior Injector: An Architectural Pattern for Rapid Prototyping the

Reactive Intelligent Robots," in *Proceedings of 2005 Taiwan Software Engineering Conference* (TSEC 2005), Taipei, Taiwan, 2005.