PROVISION, DISCOVERY AND DEVELOPMENT OF UBIQUITOUS SERVICES AND APPLICATIONS

ZHU JIAN

Bachelor of Computing (Honors)

National University of Singapore

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2011

Acknowledgments

My first and foremost thank goes to my supervisor Dr. Pung Hung Keng, for his heuristic guidance throughout the duration of my Ph.D. study. His insights and experience in ubiquitous computing have been invaluable for my research. The rigorous attitude towards problems learnt from him makes me think more carefully not only for my research work but also for future life problems. I appreciate all of his suggestions, encouragement and patience made throughout my research as well as the thesis writing. I also want to thank Dr. Gu Tao who inspired me a lot about research ideas through discussions in the initial stage of my study. I specially thank Dr. Leong Ben and Dr. Ooi Wei Tsang for serving on my thesis committee and providing many useful comments in reviewing this thesis.

I would like to thank the Department of Computer Science, School of Computing, National University of Singapore for providing me the opportunity and financial support to pursue my Ph.D.. I also want to thank Dr. Wong Wai Choong who offered me an opportunity to work for him as a research assistant that eases my financial burden in the last year of my research work.

I would like to thank all the members in the Network Systems and Services Lab (now located in the Systems and Networking Research Lab 4) including Chen Peng He, Xue Ming Qiang, Sen Shubhabrata and Zhou Li Feng. In particular, I would like to thank Oliya Mohammad for coauthoring with me to further improve the technical depths and presentation qualities of my papers. It is the above people who make my journey to the Ph.D. not boring but wonderful.

Last but not least, I would like to thank my parents, Zhu Pi Jia and Zhang Mei Jiu for their continual support and encouragement. Without them, I would never have the courage to start and fulfill my study.

CONTENTS

Abbreviations

ACE Application Context Engine

ACM Application Context Model

ASC Application Service Consumer

ASP Application Service Provider

AST Application Scenario Table

DHT Distributed Hash Table

DS Destination Sampling

LASPD Location-Aware Service Provision and Discovery

PSG Physical Space Gateway

SM Service Mediator

SOLE Sharing of Living Experience

SOLE-AS SOLE Application Server

SOLE-EC SOLE Experience Consumer

SOLE-EP SOLE Experience Provider

SS Source Sampling

TS Threshold Sampling

UbiComp Ubiquitous Computing

Abstract

The rapid advancement in the field of hardware and information communication technologies has enabled the emergence of mobile ubiquitous computing as a field of research that would greatly improve people's daily living experience. Tremendous research efforts have been put into this area recently, including the development of embedded system and sensors, the design of system architecture and middleware, and the implementation of software agent and application framework. This dissertation addresses two open issues in mobile ubiquitous computing: (i) the provision and discovery of services in mobile ubiquitous environments, especially those services to be hosted on mobile portable devices; (ii) the development of ubiquitous applications, namely the process of embedding context-awareness into the application design.

This thesis proposes a new framework for the provision, discovery and development of ubiquitous services and applications. The whole thesis consists of three parts. *First*, a three-tier architecture (LASPD) is designed for scalable and effective service provision and discovery. The first tier geographically divides the world into autonomous areas to facilitate local service administration and management. The second tier organizes service providers of an area with adequate computing capability into a structured peer-to-peer network. The locality-preserving property of the Hilbert space filling curve is exploited in these two tiers to achieve location-awareness during service discovery. In addition, an evolutionary link-rewiring mechanism is proposed to make the network navigable and self-organized in mobile environments. To support service provision on mobile devices in the third tier, the mobile service providers are allowed to delegate their services to one of the designated peer nodes of the second tier, known as "proxy". The proxy is introduced to address some of the practical constraints of mobile devices, such as limited processing capability and battery power. Extensive simulations have been carried out to evaluate the navigability, adaptability and the resilience of the proposed

platform in mobile environments. *Second*, to ease the development process of ubiquitous applications, we introduce a specification model which allows context logic (i.e. context-related tasks) required by application to be specified and be bound to the application logic during runtime. The proposed framework (ACE) effectively decouples the processes of developing context-related tasks with that of application logic. Consequently it allows modification of the context logic to be carried without perhaps having to re-implement and re-deploy the application. Three types of context-related tasks have been defined, including application adaptation, constraint enforcement and context flow. We have conducted a case study of a simple and yet realistic ubiquitous application to have a better understanding of the software development process with the proposed framework. Real-time experimental assessments have also been provided to demonstrate the feasibility of the framework. *Third*, to further illustrate and validate the concepts in the previous two proposals, a ubiquitous application (SOLE) has been developed for context-aware information sharing. SOLE is indeed an application framework, as it is extensible and generic in the sense that we do not assume the sharing of information is due to a specific application scenario. In addition, SOLE offers flexibility in storing user data and allows embedding context-awareness such as for intelligent information push. The details of the application prototype implementation and performance measurements are discussed in this thesis.

Publications

Materials in this thesis are mainly revised from the following list of papers published/accepted:

1. **Jian Zhu**, Mohammad Oliya and Hung Keng Pung, "A Pragmatic Approach to Location-Aware Service Organization and Discovery", *In Proceedings of IEEE 28th International Performance, Computing, and Communications Conference (IPCCC)*, pp. 272–279, Arizona, USA, 2009.

2. **Jian Zhu**, Mohammad Oliya, Hung Keng Pung and Wai Choong Wong, "LASPD: A Framework for Location-Aware Service Provision and Discovery in Mobile Environments", *In Proceedings of IEEE 6th Asia Pacific Services Computing Conference (APSCC)*, pp. 218–225, Hangzhou, China, 2010.

3. **Jian Zhu**, Mohammad Oliya, Hung Keng Pung and Wai Choong Wong, "SOLE: Context-Aware Sharing of Living Experience in Mobile Environments", *In Proceedings of 8th International Conference on Advances in Mobile Computing and Multimedia (MoMM)*, pp. 366–369, Paris, France, 2010.

4. **Jian Zhu**, Penghe Chen, Hung Keng Pung, Mohammad Oliya, Shubhabrata Sen and Wai Choong Wong, "Coalition: A Platform for Context-Aware Mobile Application Development", *In Ubiquitous Computing and Communication Journal (UBICC)*, Volume 6, Issue 1, 2011.

5. **Jian Zhu**, Mohammad Oliya, Hung Keng Pung and Wai Choong Wong, "SOLE: Context-Aware Sharing of Living Experiences", *In International Journal of Pervasive Computing and Communications (IJPCC)*, Volume 7, Issue 1, 2011.

6. **Jian Zhu**, Hung Keng Pung, Mohammad Oliya and Wai Choong Wong, "A Context Realization Framework for Ubiquitous Applications with Runtime Support",

In IEEE Communications Magazine, Volume 49, Issue 9, 2011.

7. **Jian Zhu**, Mohammad Oliya and Hung Keng Pung, "Service Discovery for Mobile Computing: Classifications, Considerations and Challenges". *In Handbook of Mobile Systems Applications and Services (Editors: Anup Kumar and Bin Xie)*, CRC Press, Taylor and Francis Group, USA, 2012.

In addition, during my Ph.D. study, I have made efforts in doing researches on Web service matching and context data management. The resulted publications (including those I participated in) are listed as follows:

10. **Jian Zhu** and Hung Keng Pung, "Process Matching: A Structure Behavioral Approach for Business Process Search", *In Proceedings of 1st International Conferences on Pervasive Patterns and Applications (PATTERNS)*, pp. 227–232, Athens, Greece, 2009.

11. **Jian Zhu** and Hung Keng Pung, "A Pragmatic Approach to Context-aware Service Organization and Discovery", *In Enabling Context-Aware Web Services: Methods, Architecture, and Technologies (Editors: Quan Z. Shen, Jian Yu and Schahram Dustdar)*, CRC Press, Taylor and Francis Group, USA, 2010.

12. Hung Keng Pung, Tao Gu, Wenwei Xue, Paulito P. Palmes, **Jian Zhu**, Wen Long Ng, Chee Weng Tang and Nguyen Hoang Chung, "Context-Aware Middleware for Pervasive Elderly Homecare", *In IEEE Journal of Selected Areas of Communications*, Volume 27, Issue 4, 2009.

13. Wenwei Xue, Hung Keng Pung, Shubhabrata Sen, **Jian Zhu** and Daqing Zhang, "Context gateway for physical spaces", *In Journal of Ambient Intelligence and Humanized Computing*, Volume 1, Issue 4, 2010.

CHAPTER 1

INTRODUCTION

1.1 Background: Mobile Ubiquitous Computing As New Paradigm for Distributive Computing

With the emergence of current mobile and wireless computing technologies, mobile devices such as smart phones and tablets are often the choice for organizing personal information and accessing services over the Internet. These devices also provide a new form of platform for people to do business, e.g. marketing and advertising products. A recent study conducted by A-1 Technology[1], a leading New York-based software outsourcing company, estimates that the world market for mobile marketing and advertising revenues will reach nearly $50 billion by 2014, up from about $29 billion in 2009, representing an annual growth rate of 12 percent. Such a trend drives up the pace of application development for these mobile devices, accessible through platforms such as App

[1] http://www.a1technology.com/.

Store[2] and Google Play[3]. Furthermore, due to the adoption of open standards for service integrations, such as Web services[4] in service-oriented computing, the interoperability issue in communicating among services has largely been resolved. Consequently, any mobile user may access Internet services anywhere and at anytime.

To further enhance user experiences, software applications developed for mobile platforms should be aware of the user situation and adapt to changes such as of the surrounding environment and users. The awareness can be of user's location, personal preference, or generally his *contexts*. "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [1]. For instance, a taxi booking application may filter search results based on the passenger's location and return only those taxis within his proximity. Fortunately, with the recent advancement of ubiquitous technologies, such as RFID tagging, GPS and embedded wireless sensors (e.g. accelerometer and gyroscope), most of these information can be captured automatically without any human intervention. Therefore, by leveraging on this information, the resulting applications could act intelligently by understanding the environment settings (*perception*) and reacting accordingly (*adaptability*). This style of computing paradigm, known as *Ubiquitous Computing (UbiComp)*, has been identified as the new emerging paradigm for distributive computing that would greatly improve people's living experience [2]. *Context-awareness*, as one of the key techniques of UbiComp, is adopted to practically realize the "disappearing computer" concept [3] — pushing all the computer and hardware devices into the background and making them seamlessly integrated with people's daily activities.

A typical architecture of UbiComp system consists of five layers as shown in Figure

[2]http://itunes.apple.com/us/genre/ios/id36?mt=8.
[3]https://play.google.com/store.
[4]http://www.w3.org/2002/ws/.

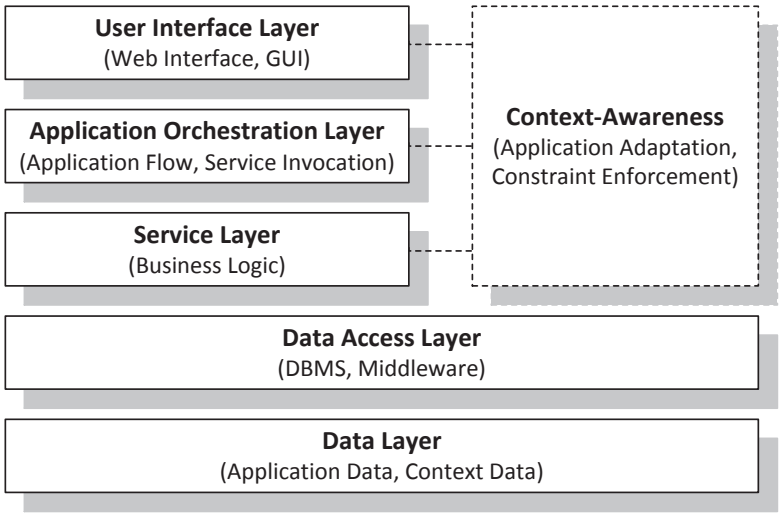


Figure 1.1: Overview of the architecture for ubiquitous computing systems.

1.1. The *User Interface Layer* directly interacts with users and captures their intentions through means such as Web interface and Graphical User Interface (GUI). The *Application Orchestration Layer* decides the application flow and controls the order of service invocations so to complete application tasks. The *Service Layer* refers to the business logic currently fulfilled by hardware devices (e.g. a printer) and human beings (e.g. a salesman). This service is re-usable by an application or another service. The *Data Access Layer* deal with mechanism for data retrieval and subscription, such as database management systems (DBMS) or middlewares. The *Data Layer* physically supplies the actual data required by the application, including the context data used to characterize the situation of entities involved in the application, such as the location of the user. To end-users, context-awareness of ubiquitous applications is mostly realized in the *User Interface Layer*, *Application Orchestration Layer* and *Service Layer*, as illustrated in Figure 1.1. For instance, the presentation style of an application to the user can be dynamically changed by adapting to the device capabilities (e.g. screen resolution) and user preferences (e.g. text display preferred); or in a specific situation, a particular application is triggered due to its relevance to the user contexts (e.g. a shopping recommender

application is started when the user steps into a shopping mall), which may again pick up the most relevant services (e.g. to recommend those shops which match the user's interests). With all the reactive context logic embedded into the design of ubiquitous applications, the user's quality of experience could be greatly improved.

Indeed, the idea of UbiComp was proposed almost two decades ago. However, due to technology restrictions such as high-cost sensor chips and low-speed and unreliable wireless communication, the vision of UbiComp was only realized in laboratories or prototyped in "smart spaces" in early days. Examples can be found in MIT's Project Oxygen[5], Georgia Tech's Aware Home[6], CMU's Project Aura[7] and UCLA's Smart Kindergarten[8]. Theses projects demonstrate the potentials of UbiComp with human-centered computation design except that their testbed environments are on a small scale. In recent years, the rapid progress in microelectronics (e.g. low-cost tiny sensor chips) and the convergence of information and communications technologies (e.g. WiFi, Bluetooth and 3G) have opened up the possibility that when every object in the world is identified and interconnected, there is a good chance that the vision of "a billion people interacting with a million e-businesses with a trillion intelligent devices interconnected" (Lou Gerstner, CEO IBM, 1995) can be achieved. The technology advancement can be best demonstrated by the evolution of the smart phones. In addition to the conventional functionality of voice call, they are now often equipped with cameras, GPS and other sensor chips. Most importantly, they have the near full functionalities of desktop computers. With networking capabilities, smart phones may operate as personal base-stations that flexibly exchange user-specific information with the environments, including among themselves. The widespread use of smart phones and other mobile portable devices essentially has created a large-scale pervasive infrastructure that allows the concept of UbiComp ap-

[5]http://oxygen.lcs.mit.edu/.
[6]http://awarehome.imtc.gatech.edu/.
[7]http://www-2.cs.cmu.edu/ aura/.
[8]http://nesl.ee.ucla.edu/projects/smartkg/.

plications be realized and tested for assisting people's daily living. The combination of mobile and ubiquitous computing, or referred to as *Mobile Ubiquitous Computing* in this thesis, is emerging as a promising future paradigm with the goal to provide adaptive computing services at anywhere, anytime and invisibly to users.

1.2 Challenges in Mobile Ubiquitous Computing

To develop an application in mobile ubiquitous computing, there are many factors to consider. These include the hardware devices to be deployed, the communication protocols, the resources and services, the application contexts and the type of end-users involved. The following summarizes the major challenges to be encountered in mobile ubiquitous computing for application developers.

1.2.1 Hardware Limitations

Hardware devices play the most important role in ubiquitous applications, especially those wearable mobile devices. They are not only become smaller in size but also with better capability. Moreover, there is an emerging trend of embedding sensors into these portable devices, turning them into a very powerful and yet agile data collection, aggregation and primary processing mobile nodes. Despite all these advancements, *power management*, *processing capability*, *memory availability* and *heterogeneity of device platforms* are still the biggest challenges for these devices [4]. In the case of smart phones, battery life is an issue as it can be quickly drained in less than half a day if functions such as GPS and 3G are enabled. The CPU speed and RAM of the device also limit the system software capability such as the ability to handle complex tasks and to support multi-tasking. Moreover, an application developed for a device may not work for another device of different mobile operating system. When developing ubiquitous appli-

cations, developers have to be aware of these issues and minimize the impact of device constraints in their application design, such as by offloading heavy-weight computation tasks to more powerful machines like desktops, and avoiding using low-level codes for better code efficiency, but at the expense of portability of applications to device platforms (hardware and operating system).

1.2.2 Communication Requirements

The standardization of communication technologies for mobile devices (e.g. IEEE 802.11a/b/g/n WiFi standards, Bluetooth, 3G, HSPA and 4G) has greatly enhanced the interoperability, pervasiveness, availability and efficiency of ubiquitous applications. However, different communication technologies have their own characteristics such as maximum transmission rates, costs, and power requirements. If there are more than one communication options available, the ubiquitous application should select the most suitable communication mean based on either user's service level requirements and pre-defined system policies. In addition, the so-called *uneven conditioning* issue in mobile ubiquitous computing must be handled. It is referring to the lack of consistent devices, technologies and services throughout a user's environment. For instance, WiFi connectivity may be present in one room but not in another. Most likely, this issue is caused by the mobility of the device owner, e.g. moving from one place to another. Thus, the challenge will be how to provide seamless mobility support on both the network and application levels so that the interruption to end-user's activity is minimized.

1.2.3 Resource and Service Discovery

In ubiquitous environments, sensors and computing devices are part of the resources through which services may be offered. For examples, sensor nodes provide information services and laptops provide computing services. The concept of "Everything as a Ser-

vice" in service-oriented computing also applies here. However, to utilize a service, it must be discovered and identified first, and it is not trivial to discover and select a service among many without some kinds of centralized registry for services. In small-scale environments such as *smart spaces* as described in [5] [6] and [7], protocols such as UPnP[9] are often used to discover local resources. However, complex ubiquitous applications may involve more than one space. For instance, a mobile healthcare application may rely on resources in the clinic as well as to monitor and collect data for the patient at home, office and other places of visiting. How to efficiently discover the surrounding for data or remote services at real-time given a very large scale of search scope is a challenging problem. The issue is further complicated by the dynamism of the respective services, such as their availability in the presence of mobility as mentioned in Section 1.2.2. We will give a more detailed discussion on this problem in Section 1.4.

1.2.4 Context Awareness

Context-awareness, touted as a promising technique to enable UbiComp, has received a lot of attentions by researchers in the last decade. However, to allow applications to effectively make use of these contextual information is not a trivial task. Firstly, contexts must be represented in a machine-readable format so as to support automatic data processing. A well-defined representation should capture the various characteristics of contexts, including data freshness and uncertainty. The technique for representing context is usually referred to as *context modeling*. Indeed, there are many context modeling techniques, for examples, from the simplest key-value approach to the most complicated ones such as ontology-based approach, as surveyed in [8]. Different techniques may have their own advantages and disadvantages, such as in processing efficiency and modeling expressiveness. However, as evaluated in [9], the most promising assets for

[9]http://www.upnp.org/.

context modeling for UbiComp environments are found in ontology-based models. The advantages and challenges are further discussed in their later work as reported in [10]. To summarize, ontology-based approach provides a unified way for specifying concepts and relations of contexts across different application domains, which eases the task for knowledge sharing and reusing; however, it also imposes problems such as unscalable performance due to its centralized data processing and reasoning. Secondly, *context storage and retrieval* are crucial to the performance (e.g. in terms of adaptability efficiency) of applications. In earlier context-aware systems such as CoBrA [11] and CASS [12], the context data acquired from data sources (e.g. sensors) is usually stored in a central place and managed via a DBMS. This approach is useful for efficient data retrieval and reasoning; however, due to the issues of single point of failure, maintenance cost and potentially very large data volume, it is slowly replaced by the distributed approach. The distributed approach leverages the existing Peer-to-Peer (P2P) techniques. It distributes the storage of context data to various peers (usually the peer closer to the data source, such as the sensor gateway) and relies on P2P routing protocols to discover and query the data. The representative systems are SOCAM [13] and Coalition [14]. This approach has a better scalability, but the retrieval efficiency and data completeness remain as a big challenge especially for those time- or mission-critical applications. Lastly, once the context data is acquired from various sources, it is usually being processed as soon as possible. The *context processing* concerns about aggregating and interpreting data to the level as required by the application. For instance, if the application wants to know the current activity of the person (e.g. eating or sleeping), then the data from relevant sensors should not be fed to the application; instead, it should be passed on to a context processor which derives the person's activity. Clearly, the processor's reasoning capability and its processing efficiency are of major concerns.

1.2.5 Application Adaptation and Development

The application adaptation refers to the reaction of the application to the changes of user contexts or events. The reaction can be autonomous without human intervention or with human guidance (e.g. via popping up messages). On one hand, people may leave entirely to the application for decision making; on the other hand, others may wish to make some decision entirely themselves or to have certain level of control over the decision making processes of the application. A good design approach to adaptation of applications should balance the options for user input, especially for mobile users whose attention span to a given task is typically shorter and hence the timing to have direct user input is crucial for a successful mobile application.

The development of ubiquitous applications is yet another issue. The questions is how to efficiently embed context-awareness into the application design so that we can easily re-design existing applications that are not context-aware to be context-aware or modify applications that are already context-aware to satisfy new requirements. Numerous context frameworks have been developed for simplifying the development of context-aware applications by providing low-level context data operations such as acquisitions and simple aggregations in terms of APIs or toolkits. As will be discussed in Section 1.3 and Section 2.2, these frameworks are still falling short in requiring developers to explicitly deal with context-related tasks such as constraint enforcement in the application code, often resulting in a tight-binding application implementation model[10].

1.2.6 Privacy and Security

To allow effective context-awareness in ubiquitous applications such as those assisting daily living of people, the technology often requires entities involved, such as users

[10]When developing an application, if the developers have to implement all the functions/tasks in the code, we refer to this approach as “tight-binding”; on the other hand, if the need for low-level code writing is eliminated with techniques such as model-driven architecture, we refer to it as “soft-binding”.

and location sensors. For instance, end-users may be required to periodically upload their personal information such as locations and health records through the mobile application to an application server for centralized processing. This requirement introduces two potential threats to user privacy. First, the user's information may be exposed to the environment unintentionally during its uploading process to the application server. Second, there is no sure way to prevent personal data being abused or further exploitation for selfish reasons. It seems clear that, effective data protection policies and data storage schemes must be put in placed as infrastructure services for the application; otherwise, the technology of mobile ubiquitous computing could be used to create a undesirable surveillance infrastructure. We will further discuss this issue in Section 2.3.

1.3 Motivation: Mobile Device As A Nomadic Service Provider

While in the design of classical mobile applications, the application services are usually provided by static servers such as Web servers. However, with the recent advancement in mobile devices' capabilities (e.g. to their CPU, RAM and storage), more services would be provided directly by hand-held devices. For instance, a healthcare application running on a handheld can directly provide patient's health status to a legitimate requester on an on-demand basis instead of pushing it through a server periodically. This approach eliminates the need to submit patient's data from the patient's mobile device on a recurrent basis, which is costly and ineffective. In the business domain, this kind of people-centric service provisioning model also opens up new business opportunities for small business operators, such as plumbers or traveling salesmen. Instead of relying on a third-party hosting service for applications, they can now simply use their mobile devices to advertise and offer services. Moreover, to enable context-awareness and per-

sonalization of services, the user contexts (e.g. his location and preferences) captured or kept by the device can be available as a data service to other applications. In summary, the resulting "nomadic mobile services" [15], or referred to as *mobile services* in this thesis, would greatly enhance the effectiveness and flexibility of the mobile applications in the following ways: (i) services are hosted on the mobile device and the up-to-date contexts of the device (including its carrier) can be detected and leveraged on during the service invocation; (ii) devices may play a more active role in applications by invoking services running on each other and real-time device-to-device interactions can be supported without the requirement of a third-party server. The following two scenarios illustrate the potentials of making mobile devices as nomadic service providers.

Scenario 1. *Mobile (M-)Health Application. Bob is a patient suffering from coronary artery disease. His family bought him a mobile phone add-on with auxiliary devices for monitoring his heart status, e.g. heart beat rate and blood pressure. In addition to its function as a mobile electro-cardiogram (ECG) monitor, the phone is also installed with various mobile services potentially beneficial to Bob. For instance, information about his body's vital signs can be made available over the Internet and be retrieved anytime by authorized third parties such as an online wellness care provider or his doctor. Similarly, his personal information such as his historical records of wellness/health can be stored in his phone, and made available as a data service for retrieval by, for example, the authorized health-care service providers such as clinics and hospitals. Furthermore, in the event that an abnormality of his heart is detected signaling a possible heart attack, an in-mobile wellness service could summon the nearest ambulance to Bob's current location to ferry him to the nearest hospital in the shortest time. In addition, it could also look for an emergency care service for any nearby volunteer who may render an urgent assistance, assuming the volunteer has offered his emergency care service via his mobile device.*

Scenario 2. *Interactive Event Sharing Application. Kate is a social media maven. She has installed an interactive information sharing application service on her smart phone, which allows her to share the social events she is attending remotely with people at real-time. The mobile service can be discovered by people located anywhere. Interested people who have appropriate permission may invoke the service and directly interact with Kate (without the requirement of a third-party server). For instance, assume Kate is currently doing a sharing of information regarding the general election in Singapore. Keith, a Singaporean currently on a business trip to USA, is particularly interested in the current campaigning activities of his political party. He can discover the information sharing services in Singapore and finds the service provided by Kate. He could then requests Kate for the latest information of the event in text, picture, or video format.*

"The paradigm shift of smart phones from the role of service consumer to the service provider is a step toward practical realization of various computing paradigms such as pervasive computing, ubiquitous computing, ambient computing and context-aware computing", [16]. However, by taking these mobile services into the design consideration of ubiquitous applications, it adds in more challenges to those discussed in Section 1.2. First, services provisioned on mobile devices tend to be dynamic. A mobile service provider may turn on or shut down his device and hence the service at anytime, and along with the movement of the device, the binding information of the service is also changing. Such a *service dynamism* property makes the conventional centralized approaches (e.g. UDDI[11] and Jini[12]) for service organization inappropriate, as they can have bottlenecks in dealing with service updating and have a potential single point of failure. Second, the network transmission speed for the mobile service provider can be the potential bottleneck in supporting concurrent requests. This is also because in practice wireless communication channels are often affected by environmental factors such

[11] http://www.uddi.org/pubs/uddi_v3.htm.
[12] http://www.jini.org/wiki/Main_Page.

as channel interference. In addition, these mobile devices can be hidden behind the Network Address Translation (NAT) or Firewall, which causes *network addressability* issue for the outside world. Third, issues such as *service administration*, *security* and *privacy* are crucial during the provision of mobile services. For instance, the volunteered caregivers in Scenario 1 may not want their locations to be revealed to all except people in their respective proximity and requiring emergency first-aids.

While the mobile service enhances the flexibility of context data provision (e.g. on-demand basis, ad-hoc style) which potentially can create a low-cost and yet large-scale deployment environment for ubiquitous applications, research gaps have been identified in the development process of these applications ([17], [18], [19] and [20]). Most early ubiquitous applications as demonstrated in [21] and [22] are developed within a "closed" environment, where each of them is designed for a specific application scenario, for example in tourism. In this case, the developers always have to build their applications from scratch, that is to reinvent all aspects related to context-awareness such as context modeling and context management. Although such an approach yields efficiency in building a specific application, it is generally harder to reuse any component or resource inside the application. Furthermore, interoperability will be the major concern if there is any demand for application integration. By considering the above issues, numerous context frameworks including toolkits and middlewares (for examples, Solar [23] and SOCAM [13]) have been proposed to simplify the context-related tasks for ubiquitous application development. Although these middlewares are fairly generic and hence can be used by any application, they are mainly designed for the acquisition of lower level context data and hence do not provide capabilities in dealing with higher level context-related tasks such as runtime constraint enforcement which are desirable from the application developers' point of view. Consequently, developers have to explicitly implement the logic in the application code, for examples, calling the API to retrieve context data (e.g.

Bob and all ambulances' locations in Scenario 1), and enforcing constraints by using if/else statements (e.g. to find the nearest available ambulance for Bob). This results in a tight-binding model for application implementation as mentioned in Section 1.2.5. Consequently for any changes in the requirements or the underlying context frameworks, application developers most likely have to rewrite and redeploy the whole application.

1.4 Problem Statement

In this thesis, we address the problem of the provision, discovery and development of ubiquitous services and applications. As motivated in Section 1.3, the two objectives are (i) to provide a scalable and effective way to manage mobile services in ubiquitous computing, and (ii) to create a mechanism to allow a soft-binding implementation approach for context-related tasks in a ubiquitous application. Hence the scopes of research in this thesis are the service and application orchestration layers of the ubiquitous system architecture as shown in Figure 1.1. The specific problems and their respective challenges are summarized as follows:

Service Provision and Discovery While there are scalable solutions (as will be surveyed in Section 2.1) that utilize P2P techniques such as Chord [24] and CAN [25] to handle the provision and discovery of services, they have seldom considered their application to mobile services. For instance, conventional P2P approaches typically require the maintenance of $O(logN)$ connections among peer nodes (i.e. service providers). But due to their resource constraints such as battery power constraint and intermittent wireless connections, portable devices such as smart phones can barely maintain many P2P connections for an extended period of time. In addition, P2P approaches such as Chord are typically very sensitive to the dynamics of the network [26], and they incur high maintenance cost to pre-

serve their network structures (e.g. $O(log^2 N)$ messages are needed periodically for a node to maintain its routing table in Chord). The challenge is how to design a scalable virtual network platform to manage distributed mobile services for an efficient lookup of a service with high relevancy while minimizing the maintenance cost for the network structure. To elaborate, let consider Scenario 1 for the case when an emergency happens to Bob, such as a heart attack, after summoning the ambulance, the immediate task is to find a caregiver who is in his vicinity so that first-aid can be given in time. Therefore, contexts such as locations of service providers or consumers should be considered during a service discovery so that in the above example, the caregiver closest to Bob could be located. In the design of the service platform architecture, we have incorporated the geographical location information of service providers so that location-based range search can be efficiently supported.[13]

Ubiquitous Application Development As mentioned in Section 1.2.4, context modeling is the first step towards context-awareness in ubiquitous applications. Similarly, to enable a soft-binding approach for ubiquitous application development, context-related tasks as required by the application must first be identified and isolated from the application, and then their logic be defined and modeled by the developers at the design time. Instead of just modeling low-level properties of tasks such as context sources and data types merely for data acquisition purpose, the model should also capture those high-level derived context information processes. For instance, in Scenario 1, to summon an ambulance for Bob, the constraints about the ambulance — "nearest" and "available" — should be specified in the model for runtime evaluation and enforcement. These constraints are derived context produced by context-related tasks that determine the constraints by computing

[13]The main motivation for incorporating location context into the infrastructure is to cater the practical considerations related to service administrative domain. Further discussion can be found in Section 3.1.1.

"nearest" and "available" based on the current location of ambulances with respect to Bob's, and their availability. As will be surveyed in Section 2.2, less work has been done on how to isolate and separate context-related tasks from the application business tasks as an application design process. Most context frameworks supporting ubiquitous application development only consider context modeling at a lower level. They leave the implementation of context-related tasks to application developers, which imposes a lot of programming effort and also reduces the application flexibility, especially in dealing with context requirement changes. In addition to the design of the context model to represent various context-related tasks, another challenge is the development of an engine to automatically realize the specified tasks. To develop an effective engine, we have to consider many aspects, including the context data management scheme, the context reasoning mechanism and the engine deployment strategy.

1.5 Approaches and Contributions

In this section, we briefly describe our approaches to the problems as mentioned in Section 1.4. The overview of the approaches is shown in Figure 1.2. As compared to Figure 1.1, two layers are added in between the Service Layer and the Application Orchestration Layer, namely the *Service Management Layer* and the *Context Realization Layer*. The Service Management Layer is about both the provision and discovery of distributed services including those provisioned on mobile portable devices. We have developed a *Location-Aware Service Provision and Discovery* platform, referred to as *LASPD*, to support the desired tasks. The Context Realization Layer incorporates a context realization engine which we call it as *Application Context Engine (ACE)* to provide high-level development support for ubiquitous applications. The ACE is capable of cater-

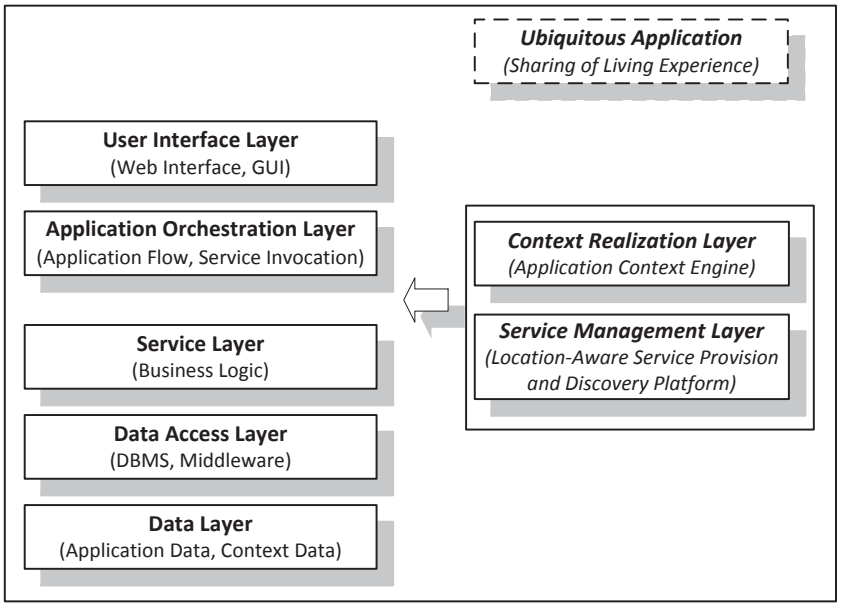


Figure 1.2: Overview of the approaches for the provision, discovery and development of ubiquitous services and applications.

ing context-related tasks as specified by the developers at design time, and automatically realizing them at application runtime. In summary, we have developed a software development framework consisting these two layers for the provision, discovery, and development of ubiquitous services and applications. As a proof of concept, we demonstrate the use of the framework for the development of an information sharing application, known as *Sharing of Living Experience (SOLE)*. In the following, we summarize our proposed techniques and contributions for the context-aware application development framework as well as the SOLE application.

1.5.1 The Service Management Layer: LASPD

A three-tier architecture is designed for LASPD, which caters all the practical considerations (as will be discussed in Section 3.1.1) while minimizing the platform implementation complexity. In the first tier, the world is geographically divided into autonomous areas to facilitate local service administration and management. In the second tier, ser-

vice providers of an area with adequate computing capability are organized into a structured peer-to-peer network. The third tier supports mobile service providers equipped with less capable devices such as smart phones or tablets. These providers are allowed to delegate their services to one of the designated peer nodes of the second tier, known as "proxy". One major feature of LASPD is that location-awareness (i.e. during a service discovery) can be easily augmented to its structured peer-to-peer network, whereas in the conventional structured P2P approaches this property is usually hard to achieve (as will be mentioned in Section 2.1.2). Besides, we propose an evolutionary rewiring mechanism to make the network navigable and self-organized with minimum maintenance effort. The followings summarize all the detailed features of our LASPD:

1. LASPD distinguishes service administrative areas and considers a *superpeer* for each of them. The service providers within each area are organized separately according to their capabilities. More specifically, relatively powerful service providers may take the role of a *proxy* to host services delegated by the resource-constrained mobile devices. The combination of the P2P network structure and the proxy design solves the scalability issue and communication constraints of mobile service providers while catering its practical concerns such as the need for service migration and the requirement of privacy protection (Section 3.3);

2. LASPD deploys Hilbert space filling curve [27] on the geographical locations of service providers in the first two tiers of the architecture. The benefits are twofold: first, due to its fractal (self-similar) property, the curve defines a flexible way for organizing (i.e. adding/deleting) service administrative areas and setting up connections among service providers (Section 3.1.3). Second, location-aware service discovery including range search can be supported by exploiting the locality-preserving property of the curve (Section 3.2.4);

3. LASPD exploits the fractal property of the Hilbert curve to achieve network navigability: the recursively constructing manner of the curve exhibits a hierarchical property; therefore, Kleinberg's small world network model on the hierarchical structure [28] can be applied to the peer nodes on the curve. As the default probability model from Kleinberg assumes a completed tree-structure, which is infeasible for its deployment in practice, we have modified the model by introducing another two parameters (Section 3.2.1). The effectiveness about our modifications is investigated through simulation studies (Section 3.4.2);

4. LASPD develops a mechanism for evolutionary rewiring of long-range links among peer nodes. As a result, the network maintenance cost is minimized. In addition, an indexing scheme for the long-range links is developed to make the platform more resilient to the failure of superpeers. Extensive simulations have been carried out to study the routing behaviors of peer nodes in the platform, and the results show that our approach is more resilient to network topology changes when compared with other similar approaches (Section 3.4).

1.5.2 The Context Realization Layer: ACE

In this work, *context logic* is defined as the set of context-related tasks required by a ubiquitous application. They concern about three types of tasks: *application adaptation* — the situation or event when the application is triggered/terminated; *constraint enforcement* — the checking on the validity of application entities involved and their context status; *context flow* — the transition of the context data among application entities. In previous works (as will be introduced in Section 2.2), the context logic is explicitly dealt with by application developers during implementation; while in the proposed ACE, the realization of these tasks is facilitated automatically. ACE allows the context logic of an application to be specified in an *Application Context Model (ACM)* at application design

time. The ACM of the application is then registered with the ACE and instantiated for automatic context realization at application runtime. The full life cycle — initialization, execution and termination — of each ACM instance is handled by the ACE. One major feature of ACE is that the ACM is formulated independently of the underlying context data management frameworks, and therefore it could be built upon any of the existing solutions. As a proof of concept, we showcase a ubiquitous application and illustrate how ACE can be used to simplify embedding context logic into the application design (Section 4.4). Besides, experimental results derived from the prototype also indicate the feasibility of ACE. The followings are major contributions of ACE:

1. ACE allows the formulation of context logic to be shifted to the application design time rather than during the implementation phase. The proposed ACM identifies and specifies the requirements of a ubiquitous application over different context-related tasks. The Model-Driven Approach (MDA) reduces the complexity of application implementation and further provides flexibility in handling requirement changes for evolving applications (Section 4.4);

2. ACE develops a set of architecture components to enable automatic context realization of context logic at application runtime. The whole process is transparent to application developers. If there is any change in context logic (e.g. context constraints), the developers are only required to update the respective ACM, but without rewriting and redeploying their applications.

1.5.3 The Ubiquitous Application: SOLE

SOLE is an application developed for context-aware experience sharing. It considers user's location, preferences, and other useful information for a seamless user experience in mobile ubiquitous computing. It leverages on LASPD for the provision and discovery

of its application services. It uses ACE for easy incorporation of context logic. Furthermore, it relies on our previously developed context middleware — Coalition [14] — for data access and data layer functionality. SOLE facilitates location-based selection of entities for associating or retrieving experiences. Other contexts of users are also embedded in the discovery and delivery of experience information. We use SOLE to validate our framework including LASPD and ACE. As compared to other information sharing applications or systems (Section 2.3), SOLE has the following advantages:

1. SOLE is generic and extensible, by not assuming a specific application scenario. It allows users to share and retrieve experiences about any things at anywhere, anytime (without requiring RFID tags). In addition, third-party applications and services can be integrated with SOLE easily;

2. SOLE is flexible, by allowing the user to choose where to store the experience data and to specify his audience. When the data is stored on mobile devices, it could be offered through services provided by these devices;

3. SOLE is context-aware, by considering the contexts for the user and his surrounding environments. With the support of Coalition, the context-awareness is no longer restricted to attributes such as user's location and social relationship.

1.6 Thesis Outline

The rest of the thesis is organized as follows:

- Chapter 2 reviews the related work. The topics to survey include platforms for service provision and discovery, context frameworks for ubiquitous application development, and applications/systems for information sharing;

- Chapter 3 presents our three-tier architecture for location-aware service provision and discovery (LASPD). The characteristics of LASPD are demonstrated by simulation studies;
- Chapter 4 describes the context realization framework that we proposed for ubiquitous applications (ACE). The use of ACE and its real-time performance are discussed through a case study;
- Chapter 5 introduces our application framework for context-aware sharing of living experience (SOLE). The concepts of SOLE (including LASPD and ACE) are illustrated and validated by prototype implementation;
- Chapter 6 concludes the thesis and highlights future research directions.

CHAPTER 2

BACKGROUND

This chapter surveys the related work in three areas, including *Service Provision and Discovery*, *Context Frameworks for Ubiquitous Application Development* and *Context-Aware Information Sharing*. The respective approaches will be compared to our proposals as mentioned in Section 1.5, namely the *Service Management Layer (LASPD)*, the *Context Realization Layer (ACE)* and the *Ubiquitous Application (SOLE)*, to justify the motivations behind this thesis work.

2.1 Service Provision and Discovery

Any typical platform for service provision and discovery will have two participating entities: *service provider* and *service consumer*. In the early days, the roles of the two entities are clearly separated. However, with the recent adoption of the Peer-to-Peer (P2P) concept, the physical separation of these two roles is weakened. In most cases, the entity can be served as both the service provider and the consumer. To develop a platform

for service provision and discovery, three processes have to be defined, namely *service advertisement*, *service discovery* and *service invocation*. The service advertisement is usually carried out by deploying a third participating entity also known as a *service directory*. This "yellow page" directory is responsible for hosting partially or entirely all the service information published by service providers and used for service matching in the discovery phase. The service discovery is the process of finding services, where the mechanisms applied are strongly dependent on the logical network structure used for service provision. Finally, the service access refers to the invocation method supported to utilize the service if it is a software service.

There have already been several survey papers for service provision and discovery published in learning literatures, such as [29], [30], [31], [32] and [33]. In this thesis, we focus on the classification of architecture support (i.e. the distribution of service information) in the surveyed platforms. The methods of service advertisement, discovery and invocation are discussed. In addition, the recent researches on mobile service provision and discovery are surveyed and compared with LASPD.

2.1.1 Centralized Architecture

In this type of architecture, there is a clearly directory server to which service providers are requested to upload their service information for their service provision. Service discovery is done by directly sending requests to the server and getting candidate services. This architecture is good for administration control and resource management, but it lacks scalability and may cause a single point of failure. Besides, the reliability of the retrieved service information is usually not guaranteed, as the server does not maintain any state for the registered services. This could degrade user experience for service discovery especially in highly-changing network topologies such as mobile ad-hoc networks (MANETs).

The industry initiative UDDI protocol[1] is one of the representatives. It defines a service directory named as *Universal Business Registry (UBR)* for all publicly available Web services. UBR stores four types of information, including *business information*, *service information*, *binding information* and *information about specifications for services (tModel)*. tModel further describes Web services in terms of attributes and metadata such as taxonomies, transports and digital signatures. To support automatic service registration and discovery, UDDI provides a set of APIs for Web service providers and consumers to interact with the registry. To minimize the issue of a single point of failure, UDDI supports data replication among two or more registry nodes as for registry affiliation. It deploys a logical ring structure for all sites involved in the replication process. A propagation process is started periodically to propagate all updates since the last propagation. In summary, UDDI has attributed to the ease of administration control and service management, which is superior for service discovery in local and static environments. The disadvantage, however, is the lack of scalability in the design and has a potential single point of failure. Though the latest version of UDDI recognizes the need for affiliations among registries, lacking of an efficient mechanism to locate the desired registry would result in the same scalability issue as for service discovery. Another issue is that UDDI is solely designed for Web services. As a consequence, users such as service providers must have prior knowledge regarding Web services standards. Lastly, as investigated in [34], many problems have been found in current UDDI implementations, such as lack of guarantee for the availability of the registered services.

The Jini framework[2] is yet another industrial standard specifying the way for Java-enabled devices to find services on each other. The whole architecture is built on top of the Java Virtual Machine (JVM), and thus is platform and protocol independent. Services are considered as Java classes and each is identified with a *Universally Unique Identifier*

[1] http://www.uddi.org/pubs/uddi_v3.htm.
[2] http://www.jini.org/wiki/Main_Page.

(UUID). The instance of the service is described as a *service item* (i.e. Java object) which assigns the value for each attribute defined in the service class. Like UDDI, in Jini there is a *lookup server* acting as the registry for all the service items. The lookup service maps service requests described by template attributes to service items that implement interfaces for the matched services. The service items (or their RMI stubs for remote access) are then downloaded by the consumer for service invocation. In Jini, the lookup server can be reactively discovered by replying requests multicasted by service providers or consumers; or, the lookup server can proactively announce its presence to the network by multicasting. As compared to other service discovery solutions, a special feature of Jini is the mobile Java codes, which can be moved among providers, registries and consumers. Besides, it also supports leases to help remove those services no longer available. However, the problem with Jini is that the JVM is required to be installed on all the entities. This can be hard in practice due to the device heterogeneity[3]. Resource-constrained devices may not have enough memory to support JVM. In addition, Jini has the same scalability issue as UDDI, such as the use of multicast protocol for message exchange.

The centralized architecture is mostly adopted in industry due to its simplicity of implementation and management; while in the research arena, researchers are more focusing on the scalability and efficiency issues of the developed platforms. Distributed architectures such as those based on P2P overlays are designed.

2.1.2 Distributed Architecture

In this type of architecture, there is no dedicated directory server to maintain all the service information. P2P overlay networks, such as Gnutella[4], Chord [24], CAN [25], Pastry [35], and Tapestry [36] have been deployed as the underlying platforms for service

[3]One representative example is from iOS-based mobile devices, which do not support JVM.
[4]http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html.

discovery. Service information is distributed to peers in the network. Depending on the type of the P2P overlay used (a detailed survey on P2P systems is referred to [33]), the approach can be further classified into two categories: unstructured and structured.

Unstructured P2P-based

The approach inherits the flexibility from unstructured P2P overlays such as Gnutella. Each service provider joins the platform in a peer form and maintains minimum information about the network topology as well as information regarding services provided by other peers. To ensure peer reachability, schemes for network conductivy, i.e. the selection of neighbors, must be designed beforehand. For instance, there are four schemes proposed in [37] — *high-degree biased*, *proximity biased*, *connectivity biased* and *proximity-connectivity hybrid*. The interesting points about this type of architecture include implementation simplicity, support for partial match queries, and relative resilience (i.e. less maintenance overhead) to peer leave or failures. Besides, it allows complex service matching mechanisms to be implemented and provides better privacy protection. However, due to the lack of network knowledge, the discovery of services is blind in that it is independent of the query and usually is based on flooding, resembling broadcasting on the network layer; therefore, they tend not to scale for wide area networks and queries may not always get resolved if techniques for overhead reduction are applied, e.g. by setting TTL in the query header.

The representative platform is WSPDS [38], which utilizes Gnutella as the underlying protocol and defines a set of application protocols for neighbor selection and Web service matching. In the initial version, each service peer, or so-called *servent*, maintains a list of the most recently active servents of the network, denoted as *servent cache*. Each time a servent is activated, it probes the servents listed in the servent cache to find k nodes that are still active and designates them as its neighbors. The servent cache also contains

access points of a few WSPDS servents that are almost always active in case it is the first time for the servent to activate. The routing of queries is probabilistic-based flooding. To further reduce the overhead of query dissemination, the recent version adopts their studies in querical data network [39], in which servents of similar "identity" are selected as neighbors with higher probability. The similarity is measured by adopting the Match-Maker algorithm in [40] on the inputs/outputs of their provisioned Web services. Service queries are then routed to the neighbor with the most similar identity.

Indeed, service provision and discovery in MANETs can also be considered as unstructured P2P-based. As for MANETs, two practical issues are limited topology knowledge due to wireless communication range and changing topology due to node mobility. Similarly, broadcasting and multicasting are used as the main communication techniques for service advertisement and discovery. The representatives can be found in Bluetooth[5], DEAPspace [41], Mobiscope [42], PDP [43], Mobi-Dik [44] and HESED [45]. To reduce the communication overhead and improve the energy efficiency, different optimization techniques are applied. [32] has summarized four major approaches: *advertisement range bounding/scoping*; *selective, probabilistic and intelligent (advertisement/request) forwarding*; *P2P information caching*; and *intermediate node responding to service requests*. The authors have also done a survey study on the respective approach, which we will not repeat here. There are also cross-layer design for service provision and discovery in MANETs. This type of approach piggybacks service information into routing messages, and service discovery is done on the network layer so as to reduce unnecessary message exchange at the application level. Examples can be found in AODV-SD [46] and MZRP [47]. To summarize, all these cross-layer approaches have shown their efficiency in terms of network throughput, service acquisition time and energy consumption, as demonstrated in [46] and [48]. However, this kind of approach destroys the protocol

[5]https://www.bluetooth.org/Technical/Specifications/adopted.htm.

stack in the Internet OSI model and tends to be protocol-specific, which may limit the interoperability of the discovery platform, especially when they are deployed for large-scale and heterogeneous networks.

Structured P2P-based

Unstructured approach has two major issues: firstly they may result in queries that are not always resolved; secondly, they do not guarantee anything about performance while scaling. On the other hand, platforms relying on structured P2P overlays solve these issues. Most of them rely on the Distributed Hash Table (DHT) concept and assign key ownerships in a pre-determined manner: each node in the overlay is responsible for maintaining a part of the hash range which represents an index to available resources; queries are input to a hash function and then are looked up in the nodes responsible for the resulting keys. For these approaches, query resolution path lengths are usually increasing logarithmically with the number of nodes in the network, i.e. in *O(logN)* scale. Nevertheless, DHT-based P2P systems have their own limitations, as discussed in [49], [26], and [50]. Firstly, most hashing techniques adopted do not consider data semantics; thus, partial or range search is not supported. Secondly, DHT-based overlays incur large overhead for the maintenance of its network structure, which usually requires $O(logN)$ messages to be exchanged for each peer to join or leave. Lastly, the constructed overlay does not consider practical Quality of Service (QoS) parameters such as physical distances among peers and query response time in real-life situations.

The representative is INS/TWINE protocol from MIT [51], which is based on Chord structure and supports partial matching by hashing partitions of the resource description (i.e. attribute-value pairs). In [52], the authors propose Lanes, which adopts the overlay structure based on CAN. The overlay has two dimensions: one for propagations of service advertisements through a lane in a top-bottom manner on the y axis; and one for

distributions of service requests between lanes on the x axis. Nodes in the same lane share the same anycast address which allows anycast routing for sending messages from one lane to another lane. Other Similar DHT-based service discovery approach can be found in [53].

2.1.3 Hybrid Architecture

The hybrid architecture aims to combine the merits of the above introduced architectures, including centralized architecture, unstructured P2P-based architecture and structured P2P-based architecture, so that their respective strengths are maximized while the effects of their undesired properties are minimized. The followings describe the two mostly adopted approaches.

Distributed Centralized

To minimize the chance of a single point of failure and to relieve the workload on the global directory server in the centralized approach, distributed centralized approach is proposed. In this approach, service information is shared among several directory servers and each server is only in charge of a partial scope. The advantage of such platforms is that they can achieve a certain level of scalability; meanwhile, they can take practical concerns such as service administration and privacy protection into consideration. Local service discovery can be achieved in the same manner as that in centralized one by approaching the local directory server. However, global discovery is hard to support unless there is a backbone of these local directory servers. Existing distributed centralized approaches can be mainly divided into two groups according to the backbone architecture deployed: topology-specific and non-topology-specific.

In topology-specific architectures, the backbone of directory servers is predefined by using structures such as hierarchy, ring or grid. Useful criteria to define such a topology

include those based on administrative domains (company divisions), network topology (network hops), network metrics (bandwidth or delay) and geographic location (physical distance). The followings summarize the four major topologies: hierarchy, ring, grid and hybrid.

- *Hierarchy Topology.* This topology reduces the number of resources to a manageable size, and it also allows autonomy for different parts of the system. The hierarchical structure usually follows the DNS[6] or LDAP[7] model, where directory servers are organized in a tree structure. Service queries are propagated up or down through the hierarchy. The representatives are Bonjour[8], NOMAD [54], Ad-UDDI [55] and SDS [56]. Apple's Bonjour is based on the Multicast Domain Name System (mDNS) for service name to address translation. The European project NOMAD combines UDDI and LDAP information model to build a distributed service discovery platform in mobile networks. The Ad-UDDI platform extends UDDI by updating service information to different registries in an active manner. It organizes all the UDDI registries according to the hierarchy defined in Global Industry Classification Standard. Registries in the same industry classification are established with neighboring relationship. Service queries are routed to the registry which is in charge of the classification. If they cannot be resolved locally, they will be routed to the neighboring and upper level registries. The Secure Service Discovery Service (SDS) from the Berkeley Ninja Project allows directory servers to be organized into multiple hierarchies. To guarantee a query can reach all SDS servers, one particular hierarchy (primary hierarchy) must be supported by all servers, e.g. hierarchy based on administrative domains.

- *Ring Topology.* This kind of approach links directory servers in a ring structure,

[6]http://tools.ietf.org/html/rfc1034.
[7]http://tools.ietf.org/html/rfc4512.
[8]http://developer.apple.com/opensource/.

and it relies on existing P2P techniques to help efficient query routing. In the platform presented in [57], the authors build the backbone of service directories (i.e. peers) on top of the P2P protocol Chord. Service registration is performed implicitly by first embedding semantic information into the peer identifiers, then grouping peers by service categories, and lastly forming islands on the ring topology. Service discovery is performed by sending queries and anycast messages to peers registered in the appropriate islands. The routing mechanism in Chord is adopted to achieve $O(logN)$ scale for efficient service discovery. Similar approach can also be found in SPiDeR [58].

- *Grid Topology.* This approach is mostly used in MANETs and the backbone of service directories is formed in a grid network. In [59], the authors propose a Geography-based Content Location Protocol (GCLP) for service discovery. Service advertisement occurs along a crisscross trajectory with four directions: 1) east; 2) west; 3) north; and 4) south. The provider selects four relay nodes that are the farthest node in each direction, i.e. within its wireless communication range. These nodes are picked up to cache the advertisement and further propagate it in the respective direction until there is no more relay node available. Service discovery happens in the same manner. However, this approach imposes heavy storage and communication consumption requirements. The recent SGrid platform [60] addresses the issue by splitting the public area into a hierarchical grid. The proposed scheme registers the information of available services to a specific location along a predefined trajectory, which is to avoid the sparse node network topology as for GCLP. The service description is registered to the center line of the network (i.e., the maximum grid level). When a requestor wants to access a service, he discovers the service toward the maximum grid level. The discovery is thus confined to a quarter area of the network. The experiments show SGrid outperforms GCLP

in terms of higher discovery success ratio and lower control overheads.

- *Hybrid Topology.* The approach represents a hybrid of the above introduced topologies. In [61], the Service Rings platform establishes a hierarchical ring architecture. A ring is a group of devices that are physically close to each other and offer similar services. Each ring has a designated Service Access Point (SAP) for storing local service information. SAPs are also linked to form high-level rings, which have SAPs that store summaries of services they provide.

In non-topology specific architectures, there is no predefined topology structure for directory servers. It is therefore important for directory servers to discover each other for global service discovery. In UPnP[9] and SLP[10][11] protocols, IP multicast is extensively used in the design, such as discovery of *control points* in UPnP and *Directory Agents (DAs)* in SLP. In Meteor-S [62], an unstructured P2P infrastructure of registries, based on JXTA protocol[12], is built for semantic publication and discovery of Web services. Nevertheless, there are exceptions that directory servers are not required to discover each other and form links, such as that in [63]. The platform treats mobile users as messengers, and they can submit cached Web service information to the UDDI registry in range. In this way, other mobile users which are in the vicinity of the registry can discover services in other places. But, precisely speaking, such a mechanism does not support global discovery, and also, data reliability is an issue.

P2P Cluster-based

Clustering techniques are applied to the conventional P2P systems to further improve the system scalability and efficiency. In addition, properties such as support for partial

[9]http://upnp.org/sdcps-and-certification/standards/.
[10]http://www.ietf.org/rfc/rfc2165.txt.
[11]http://tools.ietf.org/html/rfc3224.
[12]http://jxta.kenai.com/.

match queries and resilience to peer leave or failures as in unstructured P2P overlays can be possessed. The followings are the three mostly applied clustering criteria:

- *Semantic-Based Clustering.* The approach is to cluster semantic-close services to support semantic search and reduce routing effort. In [64], the authors map the sequence of service keywords in the d-dimensional CAN space to 1-dimensional linear space by using the Hilbert SFC technique. Points that are close on the curve are mapped from close points in the d-dimensional space. However, such a mapping destroys the properties of consistent hashing, and thus, not ensuring load balance. An ad-hoc load balancing technique is further devised in the paper. ERGOT [65] solves the load balancing issue by combining DHTs and Semantic Overlay Networks (SONs) to enable distributed and semantic based service discovery on the Grid. The system involves a module to compute similarity among services based on their Category Ontology (CO). Semantic links are built among those semantic close service peers in addition to the conventional DHT links in Chord. Other semantic-based clustering approaches have also been proposed in [66] and [67].

- *QoS-Based Clustering.* To take practical QoS (e.g. search efficiency) into consideration, QoS-based clustering techniques may be used. In [50], service peers are organized into clusters, in which every node may reach others within a given time frame. This is similar to the mechanism used in WSPDS [38]; thus, a certain quality of service can be provided. Algorithms for identifying such clusters are also discussed in the paper and demonstrated through simulations.

- *Superpeer-Based Clustering.* DHT-based P2P systems require high maintenance cost for their overlay structures. As a result, they may apply the same technique as in the distributed centralized approach by narrowing down the maintenance scope and forming clusters. Each cluster has a representative peer, known as a super-

peer or ultrapeer, which has an overview of the resources (e.g. services) kept in the local cluster. The superpeers in different clusters form connections among themselves so that if the desired resource cannot be found locally, the request is forwarded to other clusters. The backbone architecture for organizing superpeers is similar to that in the distributed centralized approach. For instance, hierarchy topology is adopted in GloServ [68] and VIRGO [69]. While for the organization of peers inside a cluster, it can be either unstructured-based (e.g. FastTrack[13] and Gnutella2[14]) or structured-based (e.g. GloServ and VIRGO). One major problem in this approach is that a lot of workload is imposed on superpeers, and their presence makes the network more vulnerable to the failure of superpeers [33].

2.1.4 Service Provision and Discovery for Mobile Services

Most of the platforms introduced in the previous sections (Section 2.1.1 to Section 2.1.3) seldom consider service provisioning on mobile devices in Internet scale, as mobile and wireless computing technologies were not ready for such application at that time. With the recent advancement of these technologies, using portable devices such as smart phones as service providers (e.g. context data provision) is feasible and is identified as a step toward practical realization of UbiComp [16]. Although the above mentioned approaches can still be applied for service provision and discovery on mobile devices, they are not specially tailored for the paradigm, as the challenges and problems discussed in Section 1.3 and Section 1.4. In the following, we present platforms specifically designed for mobile services and compare them with LASPD.

The feasibility and performance of deploying Web services on mobile devices have been investigated in [70], [71], and [72]. [72] claims that the total Web services processing time on mobile devices is only a small fraction of the total request-response time

[13] http://en.wikipedia.org/wiki/FastTrack.
[14] http://g2.trillinux.org/index.php?title=Main_Page.

($< 10\%$). Various frameworks for mobile service provisioning have also been proposed since 2005. For instance, [73] presents a lightweight SOAP server design; [74] describes the Mobile Host concept for Web service providers on smart phones; [75] designs a lightweight component architecture for hosting Web services on mobile devices; and [76] proposes an adaptable Web service provision/consumption architecture for mobile service providers/consumers. However, these works only examine hosting Web services on mobile devices; the issues of service organization and discovery for such mobile services have rarely been discussed. As far as we know, only the following platforms have taken in the design of mobile service provision for their large-scale service discovery.

There are platforms leveraging on the Jini Surrogate Architecture[15] (JSA) for the provision and discovery of mobile services. The representatives can be found in the Mobile Service Platform [15] [77] and the Mobile Service Provisioning Middleware [78]. The JSA specifies a software architecture to allow a device (hardware or software components) that is not capable of directly participating in a Jini network to join a Jini network with the aid of a third-party host, known as *surrogate host*. The surrogate host is a framework residing in a machine that is already in the Jini network. It retrieves and activates the device's surrogate so to behavior on behalf of the device. Interconnect protocols are defined for discovery, retrieval of the surrogate and liveness between the surrogate host and device. For instance, the HTTP interconnect is implemented in both representatives. As compared to our approach LASPD, the major difference is that the JSA requires the environment of Jini, which is purely Java-based platform and requires multicasting for message exchange and service discovery/invocation. Despite of the issues discussed in Section 2.1.1 for Jini, in [79] the author has argued that the JSA may be not suitable for large-scale deployment due to device heterogeneity (e.g. vendor-specific device APIs) and different platform requirement (e.g. Web services desired). In addition, the device

[15] http://www.jini.org/files/specs/surrogate/sa.pdf.

service provider has to explicitly provide the surrogate (i.e. Java codes) to be loaded by the surrogate host. LASPD, on the other hand, applies the emerging Web services technology for service invocation. Instead of restricting the platform to be Java-based, the Web service technology allows services to be developed and interoperated under different programming environments. The detailed discussion on the architecture for service provision in LASPD can be found in Section 3.1.

The Mobile Web Services Mediation Framework (MWSMF) [80, 81] is extended from the Mobile Host concept in [74] and acts as an intermediary between the mobile Web service clients and the mobile hosts so to form the Mobile Enterprise [82, 83]. The platform is responsible to convert and route the mobile Web service's messages to the respective mobile hosts with QoS considered. Mobile hosts are organized as JXTA peers in a peer-to-peer network [84]. To address the integration problem across mobile enterprises, the platform leverages on the *Enterprise Service Bus (ESB)* technology [85] to allow services on mobile hosts to be requested by different enterprise networks. The recent efforts have focused on shifting components in the platform to the new cloud computing paradigm to further improve the scalability [16], as well as adding context-awareness into the service discovery process [86]. Although the authors have done various load tests for the platform in [16], the scalability study for mobile service discovery in [84] is limited by the number of JXTA peers simulated. Indeed, JXTA is suitable for stable peer-to-peer networks, and due to its random walk routing protocol, it does not guarantee search results even if there are matched services. LASPD deploys the superpeer-based clustering approach. By distinguishing service administrative areas (e.g. enterprise networks) and considering a *superpeer* for each of them, the scalability can be achieved. Besides, instead of relying on a message broker (the *bus* in the ESB) to redirect all messages among different service administrative areas, a consistent P2P routing scheme is designed for both local-area and cross-area scenarios. Moreover, LASPD deploys the Hilbert space

filling curve in its network structure and applies the so-called source sampling technique to help construct a small world network in an autonomous manner. While achieving network navigability, location-awareness during service discovery is also supported. The detailed discussion on the network model and the routing scheme of LASPD can be found in Section 3.2. The simulation studies are demonstrated in Section 3.4.

Other related work can be found in Nokia's Mobile Web Server project [87] and Splendor [88]. The former approach relies on a gateway in the fixed network to redirect all the HTTP requests to the HTTP servers hosted on mobile phones; while the latter one adopts a *client-service-directory-proxy* model to support secure service discovery for nomadic users and services in public environments. Nevertheless, both platforms have scalability issue: the gateway is inherently a bottleneck as mentioned in [87], and Splendor is mainly designed for small-scale environments (e.g. a shopping mall scenario) while the set of security protocols adopted incur high overhead during the discovery process. For mobile service provision, LASPD adopts a similar idea by deploying the role of *proxy*. However, the design of the proxy is more distributed: rather than having a fixed static machine acting as the gateway, the functions of the proxy are implemented as services and can be hosted on any capable peer in LASPD. The use of the proxy is also flexible: the mobile service providers may explicitly choose their trusted proxies for service provision. The design of the proxy and its potential usage will be discussed in Section 3.1.5 and Section 3.3.

2.2 Context Frameworks for Ubiquitous Application Development

This section gives an overview of the related work done in context frameworks for ubiquitous application development, which serves as the motivation for our proposed

ACE framework for context realization. In the discussion, we focus on the aspects of *programming support*, *development support*, *target domain*, and *technique detail* for the surveyed frameworks. The programming support considers what functions (e.g. APIs) are supplied for the application developers to utilize, such as those for context data retrieval and for application triggering. The development support discusses in which phase these frameworks are integrated with the application development, e.g. at design time or at implementation stage. The target domain concerns about for which application domain these frameworks are used, such as activity-based applications. The technique detail describes what techniques are applied to help achieve application context-awareness, such as those discussed in Section 1.2.4.

Over the last decade, many researchers have been working on development support for context-aware applications. Their primary focus has been on the simplification of context-related tasks such as context management and retrieval so that developers could focus more on the application itself. Numerous context frameworks including toolkits and middlewares have been developed.

Context Toolkit [89] [90], as one of the pioneer toolkits for context programming, presents a reusable solution for making use of context information by leveraging on the notion of *Widget*. The various Widgets abstract the details for context acquisition from each context source and also allow applications to do event subscription. A set of other abstractions including *Aggregator*, *Interpreter*, *Service*, *Discoverer* are provided to further ease the task for applications. The toolkit itself is developed in Java environment and a set of base Classes are supplied for developers to extend.

Solar [23] is an infrastructure for context collection, aggregation and dissemination. Similar to Context Toolkit, it provides a collection of distributed, reusable *planet* to facilitate the retrieval and subscription of context events. Depending on the application needs, the developers may compose a directed acyclic graph of event operators including

filter, *transformer*, *merger* and *aggregator*. These operators are then subscribed to the respective planet by a centralized *star* process.

iQL [91] is a non-procedural programming language defined for higher-level context composition. It eases the task for application developers by automated binding and rebinding of heterogeneous data sources. By specifying context requirements on *Composer* (i.e. the central element of the iQL programming model), a pervasive composition system is capable of discovering appropriate data sources, binding them and rebinding when the values of data sources change. The language has powerful operators useful in composition, including operators to generate, filter, and abstract streams of values.

a CAPpella [92] is a system designed to empower end-users in building context-aware applications. The motivation is that end-users have a better position in knowing what they want under a particular situation. The system allows the user to do programming by demonstration. By using the graphical interface, the user may indicate what portions of the demonstration (i.e. scenario) are relevant to the desired behaviors, e.g. turning on the lights. With such kinds of training, the system can perform the demonstrated action whenever it detects the demonstrated situation.

CoBrA [11] is an agent architecture to support context-aware systems. It leverages on the Semantic Web language to define ontologies for context modeling, sharing and reasoning. The rule-based reasoning engine uses a combination of Jena reasoner for ontology inference and Jess reasoner otherwise. Application programmers develop software agents which cater for ontology sharing and rule subscription. Once the rule is triggered, the agents receive notification from the broker architecture, and fulfill their tasks, e.g. sending command to the respective presentation service in a meeting room. The architecture is demonstrated by a smart meeting room system called *EasyMeeting* [93].

SOCAM [13] [94] is an ontology-based, service-oriented middleware architecture

for rapid prototyping of context-aware applications. All the architecture components including *context provider*, *context interpreter*, *context database*, *context-aware application* and *service-locating service* are designed as independent services distributed over the network. Java RMI mechanism is used for communication. The middleware offers a set of APIs for application developers to do context query and event subscription. Rules specifying context-aware behaviors (e.g. application methods invoked) upon conditions satisfied can be registered with the context interpreter which runs Jena reasoner.

iCAP [17] is a high-level toolkit for rapid context-aware application prototyping. It aims to allow end-users to design applications visually, without writing any code. To achieve so, it provides a set of graphical interface to let users create people and artifacts, and specify the context-aware behaviors associated with these objects. The three types of applications supported — *simple if-then rule*, *relationship-based actions* and *environment personalization* are relying on a rule engine to manipulate the actual object contexts populated from Context Toolkit.

Software Engineering Framework [18] takes an object-role approach to context modeling. The proposed *Context Modeling Language (CML)* is capable of capturing context requirements of a context-aware application, including the abstraction of high-level situations by using predicate logic. Application personalization is catered by its preference model. In addition, the framework supports two programming models to allow efficient software engineering in the process of context-aware application development, whereas the *branching* model is to assist flexible context-dependent decision making and the *triggering* model is to launch an application under a particular situation.

VisualRDK [95] is a high-level toolkit for prototyping context-aware applications. It offers a visual programming environment for the developers to incorporate context-awareness into their application flows. Developers may define hardware components with a set of commands they accept or emit, and these components together with the

application logic commands (e.g. specifying the situation) are composed in the editor for application specifications. VisualRDK is responsible for generating executable application code for the actual deployment.

Hydra [19] is a framework to help the development of context-aware applications for mobile devices. The detailed functionalities to handle mobile application related tasks are encapsulated into different layers to achieve reusability. This includes *Mobile Communication* layer for networking tasks, *Framework Component* layer for mobile user information retrieval task, *Mobile Applications* layer for application development task. The awareness of user profiles and his location can be embedded in the mobile applications by calling the specific framework services.

Coalition, extended from SOCAM, is an open, programmable and reusable infrastructure for context data management [14] (the middleware was previously known as CAMPH [96] and has been uprated with more functions such as the support of callbacks in the recent version). It applies a more scalable solution — the "space" concept for modeling context data. The various context sources are organized based on the *context space* they belong to and form the respective peer-to-peer overlays. With a declarative, SQL-based context query interface implemented on the top of the middleware, context queries could be issued to do context data retrieval and subscription. As the prototype of our proposed SOLE application leverages on Coalition for context data retrieval and subscription, we will give a more detailed introduction in Section 5.5.

OPEN [20] is a programming framework emphasizing on the cooperation among users with different technical skills in the development of context-aware applications. It consists of a set of middleware components including *Context Providers* for low-level context data retrieval, *Context Manager* for context query and reasoning. The programming toolkits for application development support three programming modes to satisfy diverse user technical skills and the resources (e.g. application rules) may be shared

among the users so that they can learn from each other and do fast customization.

There are also tools developed for prototyping a specific type of context-aware applications, such as Topiary [97] for location-aware applications, CRN Toolbox [98] and ActivityDesigner [99] for activity-based applications and CMM [100] for multimedia related applications.

To compare with our ACE, Table 2.1 summarizes the key properties of the above introduced frameworks over the aspects we discussed in the beginning of this section. As shown in the table, most of these frameworks only consider programming support at a low level: context data retrieval and application triggering. Frameworks such as a CAPpella, iCAP and VisualRDK, although give graphical user interfaces to let developers implement their applications visually, they are typically restricted by the functions allowed for applications and only meant for prototyping. In frameworks such as Context Toolkit, SOCAM and Coalition, various APIs are provided to simplify context-related tasks for developers, such as the retrieval of user's context and the subscription of an event. For instance, in Scenario 1 of Chapter 1 by subscribing the event of heart attack for Bob (through a set of criteria over his heart status such as heart beat rate and blood pressure), the emergency care application can be triggered in time. Nevertheless, these frameworks only consider the triggering of an application at the right context but without catering for its runtime context requirements. The application developers have to explicitly integrate those low-level APIs in the application code. For example, after the emergency care application starts, to find the nearest ambulance service for Bob, in the application logic the developer has to first get locations of all the available ambulances (through the provided APIs), and then compare with Bob's location (through a set of if/else statements) to derive the best candidate. As mentioned in Section 1.3, this approach results in a tight-binding model during the application implementation, and it imposes a lot of programming effort on context-related tasks. Meanwhile, the developed

	Programming Support	**Development Phase**	**Target Domain**	**Technique Detail**
Context Toolkit, iQL, Solar	context retrieval, event subscription	implementation (hard integration)	general	context modeling (attribute-value model), context composition (iQL)
a CAPpella	application prototyping, application triggering	design (UI-based)	general	context modeling (attribute-value model), context reasoning (machine learning)
CoBrA	context retrieval, application triggering	implementation (hard integration)	general	context modeling (ontology-based model), context reasoning (rule-based)
SOCAM, Coalition	context retrieval, event subscription, application triggering	implementation (hard integration)	general	context modeling (OWL-based model), context reasoning (rule-based)
iCAP	application prototyping, application triggering	design (UI-based)	general	based on Context Toolkit, context reasoning (rule-based)
Software Engineering Framework	application triggering, context branching at runtime	design (CML-based), implementation (soft integration)	general	context modeling (object-role model), context reasoning (rule-based), context branching (preference model)
VisualRDK	application prototyping, application execution at runtime	design (UI-based)	general	context modeling (attribute-value model), context branching (event-based)
Hydra	context (mobile user information) retrieval	implementation (hard integration)	mobile application	context modeling (attribute-value model)
OPEN	application triggering	design (for normal user), implementation (for expert)	general	context modeling (OWL-based model), context reasoning (rule-based)
Topiary	application prototyping, application triggering	design (UI-based)	location-aware application	context modeling (spatial relation model), context retrieval (back tracking search)
CRN Toolbox	application prototyping, application triggering	design (UI-based)	activity-based application	context modeling (attribute-value model), context reasoning (machine learning)
ActivityDesigner	application prototyping, application triggering	design (UI-based)	activity-based application	context modeling (attribute-value model), context retrieval (SQL query)
CMM	context retrieval	implementation (hard integration)	multimedia application	context modeling (OWL-based model)
ACE	application triggering, context constraint enforcement at runtime	design (OWL-based), implementation (soft integration)	general	context modeling (application context model), context reasoning (rule-based)

Table 2.1: Summary of the context frameworks for context-aware application development.

application is not flexible. In case there are changes in the context requirements or the underlying context frameworks, application developers have to rewrite and redeploy the whole application. For instance, if special care facilities are required on the ambulance, the application logic has to incorporate these new requirements in its code implementation. Our proposed framework ACE is targeting to address this issue. It aims to help application developers to specify their context requirements easily at design time and automatically realize them at runtime in a soft integration fashion. With this approach, developers have not to worry about context logic during application implementation.[16] The case study shown in Section 4.4 demonstrates the differences for ubiquitous application development with and without ACE.

The closest framework to our proposal is the *Software Engineering Framework* from Henricksen and Indulska [18]. However, their context model is application dependent; hence, context facts describing the same entity for different applications may be duplicated, which results in large-size context repository and incurs additional maintenance overhead. In our approach, we allow the mapping of entities in the application level to entities in the context level through abstraction, so that the same set of context facts can be used in reasoning for different application triggering. The detailed discussion on the two-level application context management can be found in Section 4.3.2. In addition, their approach does not consider application life cycle in the management of context facts. Indeed, context facts can be generated or determined during the application execution, i.e. by the application itself; therefore, context realization at application runtime is necessary. In spite of the proposed branching model in their approach, they have not addressed the problem adequately. While our proposed ACE framework caters the full life cycle — initialization, execution and termination — of each application instances. The details of the ACE support during application runtime are referred to Section 4.3.3.

[16]The programming effort on context-related tasks can be minimized by the ACE framework, but there are potential issues related to our proposed approach. Further discussion can be found in Section 4.4.

2.3 Context-Aware Information Sharing

This section gives an overview of the state of the art work in context-aware information sharing, which is related to our proposed SOLE application.

The concept of information sharing had long been developed, and become a crucial part of our daily living since the available of networked computing systems. Examples of application are the FTP, network-file-systems, databases, and Webs for files/data sharing between servers and clients through the Internet. For context-aware information sharing, the advancement of technologies is closely related to the following development of distributed computing environment: (i) mobile computing, which is an extension of the traditional desktop networked computing with the clients being mobile. With additional features such as embedded GPS receiver on the mobile device, the client's context data can be utilized by applications such as mobile Webs; (ii) ubiquitous computing, which benefits from the recent ubiquitous technologies such as RFID tags. The developed system is capable of identifying the objects around the user by reading their tag information, and then their associated information is smartly delivered to the user. Table 2.2 summarizes those representative context-aware information sharing applications/systems. In the following, we compare them with SOLE in greater detail.

Existing Web 2.0 applications such as Flickr[17], Youtube[18] provide a worldwide platform for people to contribute content (e.g. photos and videos) and interact with each other (e.g. comments). While this type of applications are more focusing on the sharing of the information itself (e.g. content uploading, viewing and embedding in other Websites), the emerging Web applications such as Facebook[19], Twitter[20], Foursquare[21]

[17] http://www.flickr.com/.
[18] http://www.youtube.com/.
[19] http://www.facebook.com/.
[20] http://www.twitter.com//.
[21] http://foursquare.com/.

	Target Domain	Context Support	Context-Aware Application	Privacy Protection	Additional Feature
Flickr, Youtube	Web application for photo, video sharing	user's location, social relationship	location-based content search, content sharing and comment notification	user controlled settings such as public, friend, private and guest pass	allow dynamic badge to be embedded in other sites to display content; provide tools and APIs for manipulating people and content
Facebook, Twitter, Foursquare, Gowalla	Web application for social networking	user's location, social relationship	location-based people and information search, notification about people's activity such as place check-in and comment	user controlled settings such as public, protected and private	design with a game-oriented approach with badges and titles; involve business to reward users for checking in to their locations; provide APIs for other applications to interact with the platform
Sentient Graffiti	Web application for sharing of object annotations	user and object's locations, time, user's preference	location-based information search, browsing and push	user controlled settings such as public and restricted	support multi-model interactions to enable location-awareness such as by RFID tags, GPS and Bluetooth; involve rules for smart information push
Cyberguide, RevisiTour	tour guide system	user's location, time	location-based content search and browsing	—	keep the history of the places visited for the tourist
C-Map	exhibition tour system	user and exhibit's locations, time, user preference, exhibit site status	location/semantic-based content search and browsing, exhibit recommendation based on user contexts such as his interest	—	combine with a virtual exhibition space including user avatars
eXspot	museum exploration system	user's identity time	exhibit bookmarking by triggering automatic cameras	—	keep the bookmarking of the exhibits visited so that the tourist can explore more later
LORAMS, APriori, SharedLife	living experience sharing system	object's identity, user's location and profile (SharedLife)	learn people's experience on the objects around the user, find relevant sharing partner (SharedLife)	personal memory for private usage and community memory for public access (SharedLife)	link experience with movies (LORAMS); propose dynamic product rating criteria (APriori); differentiate content data into different memory models and manage them with distinct characteristics (SharedLife)
SOLE	living experience sharing system	user and object's contexts (supported by Coalition)	location-based information search, browsing and push	three kinds of data storage schemes: public, protected, private	support information provision directly from mobile device; utilize context frameworks for context modeling, retrieval and reasoning

Table 2.2: Summary of the context-aware information sharing applications/systems.

and Gowalla[22] emphasize on the social networking perspective along with the recent advance of social computing. Relationships such as friends and families are used as the major communication links for information sharing and propagation. As mentioned before, with the widespread adoption of mobile technologies in people's daily life, more and more such applications are allowing access via people's mobile devices, e.g. smart phones. Through functions such as embedded camera, the mobile device provides a convenient way for people to create and share experience. Furthermore, context-aware information sharing can be achieved by utilizing the sensors attached on these devices. For instance, in Foursquare and Gowalla, the *place check-in* concept is adopted: whenever a user checks in a place (in the application), his friends will get notified about his location, and the tips (e.g. comments) created about the place will be shared. The user may also discover nearby places and friends and interact with them. Such kind of location-awareness is adopted by many mobile applications. However, these Web-based context-aware information sharing applications have their own problems. First, these applications require the user to upload his content to a central repository. A user may not be able to do so due to technical issues such as low bandwidth, or intermittent availability in network connectivity. Second, their context-awareness is restricted to attributes such as user's location and social relationship. They cannot support more advanced tasks such as automatic photo tagging (for people involved) and intelligent information push due to lack of context infrastructure support. The issue is also raised in [101] and [102]. Indeed, the Sentient Graffiti framework in [101] resembles our approach of SOLE by considering contextual attributes of user-created graffitis (associated to a location or object). In the back-end, there is a *Graffiti Rulebase* to infer suitable graffitis for a user and filter out those unlikely to be of interest. The information is then pushed to the user. Nevertheless, the contextual attributes of graffitis are all static, with values assigned during

[22] http://gowalla.com/.

the creation time, such as locations and periods of availability. The context model does not support dynamic context attributes, such as crowd level of a graffiti associated with a shop service. Therefore, the information push is done in an intuitive manner, i.e. by matching distance and tags interested by the user. Compared with the framework, SOLE is built on top of Coalition for context data retrieval and subscription, and therefore those dynamic attributes of graffitis can also be supported in the process of inference. Furthermore, with the ACE framework, SOLE does not need to consider how to interact with Coalition. The push mechanism can be simply achieved by specifying a set of rules over the contextual attributes of the user and the graffitis involved. The details about context-awareness of SOLE will be presented in Section 5.5. Third, privacy is the major concern[23][24] for these Web applications, as the user may not trust the hosting service for the storage of his private data. For instance, the recent data collected by Inside Facebook Gold[25] shows that Facebook has lost 6 million users in the U.S. during May 2011, which is the first time U.S. numbers have dropped in more than a year. Reasons have been analyzed in Websites such as PCMAG.COM[26], and one of the main reasons is that these Websites overuse people's private data such as their profiles without permissions. Although all these applications provide user controlled privacy settings, straightly speaking, all the data is still under the control of the application but not the user. Compared to these applications, SOLE is more flexible by allowing the user to choose where to store the experience data and to specify his audience. Three storage schemes have been designed for different levels of privacy protection. When the data is stored on mobile devices, it could be offered through services provisioned by these devices. The details will be described in Section 5.2.

In UbiComp, there are context-aware systems making use of ubiquitous technolo-

[23]http://www.prweb.com/releases/facebook/privacysurvey/prweb4057284.htm.
[24]http://news.blogs.cnn.com/2011/06/08/gotta-watch-facebook-privacy-concerns/.
[25]http://gold.insidenetwork.com/facebook/.
[26]http://www.pcmag.com/article2/0,2817,2386884,00.asp/.

gies such as Infrared (IR) sensors, RFID tags and Near Field Communication (NFC) technology to detect user contexts (e.g. location) and identify objects around him. In case the objects are associated with information contributed by other users, the information can be searched, browsed and utilized by the user. The early representatives are Cyberguide [103] and C-Map [104]. In C-Map, several criteria are used for the recommendation of exhibition, such as the similarity between the interest vector of the user and each exhibit's keyword vector, the geographical distances between exhibit sites and the user location, and the exhibit site attendance. Like Web 2.0 applications, the recent context-aware information sharing systems are also encouraging users to contribute their own data. In RevisiTour [105], the system enables visitors to organize photos taken from tour sites and share them with others. Visitors are identified with RFID tags and the pictures taken are sorted by location and time. The whole system is integrated with Flickr, so others including the visitor himself could see the sorted pictures on the Web. Similar ideas by tagging people or objects for information sharing can also be found in systems eXspot [106], LORAMS [107], and APriori [108]. However, one major limitation of these systems is that they are designed and deployed for specific application domains, e.g. for museums, exhibitions and conference rooms, and hence tend to be smaller in scale. They assume all the entities are identified with RFID tags, which may be restrictive in practice. In addition, privacy protection is seldom discussed in these systems (An exception can be found in SharedLife system, which differentiates user data into several "memory" categories — personal, community, object and application, and thus managing them with distinct characteristics). Compared to these context-aware information sharing systems, SOLE is more generic and extensible. The widespread use of smart phones and other mobile portable devices has effectively created a large-scale testbed for UbiComp. The user may be identified by the mobile device carried with him, and through services running on the device, the user's context data such as his location

can be easily retrieved. Moreover, like other Web 2.0 applications, SOLE provides a set of APIs for third-party applications and services to utilize. For instance, a shopping recommendation application can make use of people's experience rating on shops from SOLE.

In summary, SOLE is a context-aware information sharing application developed for mobile ubiquitous computing. It combines the technologies (e.g. mobile portable devices) and techniques (e.g. context infrastructure) applied in the two computing paradigms. We use SOLE to validate the technical feasibility of LASPD and ACE for enabling the provision, discovery and development of ubiquitous services and applications.

CHAPTER 3

LASPD: A PLATFORM OF LOCATION-AWARE SERVICE PROVISION AND DISCOVERY

This chapter presents the detailed design of LASPD — a general platform to support efficient and effective mobile service provision and discovery. The solutions to issues and challenges as mentioned in Section 1.4 are described, including how to support mobile service provision and how to enable location-aware service discovery. We will use the two application scenarios illustrated in Section 1.3 of Chapter 1 as the case studies to help explain concepts and mechanisms in LASPD.

The rest of the chapter is organized as follows: Section 3.1 describes the motivation and structure of the three-tier service provision architecture in LASPD. Section 3.2 presents the network model of LASPD and its location-aware routing mechanism for service queries. Section 3.3 discusses the potential features of LASPD for practical ser-

vice provision and discovery. Section 3.4 shows the performance results derived from simulation studies. Finally, Section 3.5 summarizes the chapter.

3.1 Three-Tier Service Provision Architecture

3.1.1 Motivation

Service oriented computing is becoming the de facto method for efficient development and integration of software in different application domains. The resulting interoperability, due to the adoption of open standards such as Web services, enables us to move closer to the vision of *Internet of Services*. Furthermore, with the emergence of mobile services, the model of service provisioning is moving from the conventional legacy server-centric approach to a people-centric one. To handle such a scale (to the size of the world) of service provision and discovery, conventional centralized solutions are inappropriate due to the issues as discussed in Section 1.3 and Section 2.1.1. The distributed approaches such as those P2P-based (Section 2.1.2) are more promising in terms of scalability when dealing with the situations such as a single point of failure, frequent joining and leaving of service providers. Therefore, we have adopted P2P techniques in designing LASPD for large-scale service provision.

On the other hand, in real-life situations, each service can be under the control of an administrative domain and governed by a set of policies. The area of administration is usually defined by the geographical boundary of for example a shopping mall or an enterprise. In the P2P-based approaches as discussed in Section 2.1.2 and Section 2.1.3, most of them have not addressed this practical concern; while we believe by taking service administrative domains into the design consideration for the architecture of service provision, the benefits are threefold: first, the scale of service management is reduced, with each area hosting a manageable number of service providers; second, services are

controlled by the specific domain, which allows policies such as for service access control and for service data storage to be enforced; third, each administrative domain can be a potential autonomous area so that failing or disruption in other areas does not affect service provision and discovery in the current area.

Furthermore, as mentioned in Section 1.4, mobile portable devices such as smart phones have limited memory, processing power and battery life, and are mostly mobile. Consequently their availability as a service provider cannot be guaranteed. Besides, these devices usually contain personal information which is vulnerable to security threats and breaches of data privacy. Therefore, when organizing the actual service providers, we distinguish normal devices to mobile portable devices, and thus managing those mobile service providers separately for better customization of security and privacy protections.

3.1.2 Architecture Design

As motivated in Section 3.1.1, LASPD is designed with a three-tier architecture for worldwide service provision (Figure 3.1). In the first tier, service administrative areas are identified and some of which may be further divided into smaller sub-areas according to some criteria such as boundaries of service authority. The boundary for each area is geography-based and is associated with a semantic name such as ION Shopping Center. In addition, the area hierarchy (i.e. whether or not an area is contained by another one) is specified in a *geographical tree* structure. All the service providers are assumed to originate in one of the specified areas and reside there except those mobile services. As illustrated in Figure 3.1, we are currently modeling each area as a rectangle for simplicity in prototyping while noticing an irregular area could be approximated by multiple rectangles in practice. The complete specification of each area is maintained by a special peer node known as a *superpeer*. Though the superpeer represents the geographical center of the area, the computing hardware enabling the functionalities of this superpeer does

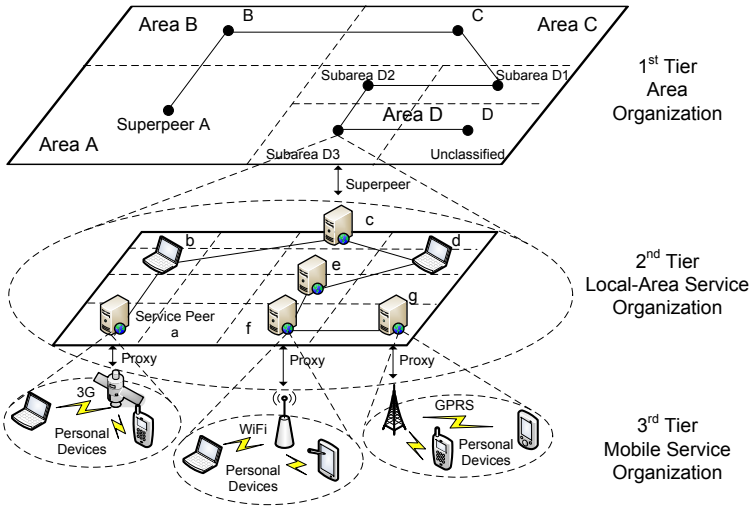


Figure 3.1: Three-tier service provision architecture.

not need to be physically present in that area. In fact, the superpeer in Figure 3.1 can in principle be implemented in a networked server says of a remote data center. The same arrangement is also applicable to other peer nodes shown in the second tier (i.e. defined as *service peers* later). To simplify our discussion from hereon, we assume a superpeer of an area and its service peers are all residing in physical hosts of that area. Each superpeer performs the following two fundamental operations: it helps new service providers to register (i.e. join) to the area and facilitates cross-area queries by maintaining links with superpeers of neighbor areas (Section 3.1.3). The role and functions of a superpeer for an area can be hosted on a designated server or on capable service provider's server in the area. While the former has full administrative control, the latter may only perform the basic essential operations of superpeers due to server ownership issues. With our current indexing mechanism (Section 3.2.3), we have found through simulations that the processing workload of a superpeer is quite light and is manageable by most modern desktop computers.

After dealing with area/sub-area organization, the service providers of a specific area/sub-area (referred to as *local area*) are managed in the second and the third tier

based on their capabilities. Service providers in the second tier tend to be resource richer and highly available, such as the server of a shopping mall or an enterprise. In this tier, P2P concepts are applied to achieve scalability and to mitigate the negative effects of joining and leaving of service providers. We refer to all service providers in this tier as *service peers* which share common computation tasks such as service indexing and query routing. Note that the superpeers are also service peers, and the role and functions of the latter can similarly be implemented in hosts not geographically located in the area.

The third tier is to enable mobile service provisioning through portable devices such as smart phones. We assume they can connect to the Internet in some way, such as through GPRS, WiFi, or 3G. Each device in this tier shares its service through a *proxy* in the second tier, which is a peer of supposedly higher availability and resourcefulness (Figure 3.1). For instance, in the M-Health Application (Scenario 1 in Chapter 1), if a doctor wishes to be the volunteer to provide the emergency care service to nearby people, he can run the application on his mobile phone which announces his presence. When entering a specific area, the mobile application will look for a proxy which will in turn register the mobile service in the P2P network of the area for discovery and invocation. We choose to have the role of proxy in LASPD due to the following considerations: (i) the resource limitation of mobile devices as discussed in Section 1.2.1 and Section 1.2.2 restricts the scalability and availability for the services hosted on them. It is inappropriate for these mobile services to support large number of requests, such as for video streaming from the embedded cameras. The issue is further complicated when dealing with multiple concurrent requests. In addition, the availability of the service is largely affected by the facts such as battery level of the device and its mobility. Therefore, to ensure a better quality of service for these mobile services, they should be distinguished from the conventional services and be taken care of with special support; (ii) considering the P2P concepts deployed in the second tier of LASPD, it is not feasible for these

mobile devices to perform the peer operations such as facilitating P2P connectivity and routing of queries. Also, their frequent moves impose a big challenge to set up and maintain the P2P connectivity for others and themselves. As a result, to help the provision of these mobile services in LASPD, additional components are required. Indeed, the issue presented in (i) is also discussed in [79] and [109]. They have also carried out survey studies on existing approaches to handle the issue, and found that the intermediary-based approaches such as those by deploying JSA (Section 2.1.4) are more effective by comparing with those by using direct communications between clients and mobile services. Our proxy concept is much like the surrogate host concept in JSA; however, as discussed in Section 2.1.4, one major difference between JSA and LASPD is that we are applying the emerging Web services technology for service invocation, which is more flexible for choices of the underlying programming environment (by considering the heterogeneity of mobile devices). The detailed modeling for the proxy and its support of mobile service provisioning are presented in Section 3.1.4 and Section 3.1.5. Additional features supported by the proxy are discussed in Section 3.3.

3.1.3 Location-Aware Identifier Allocation and Connectivity Setup for Service Peers

As mentioned in Section 1.4, the relevance of the discovered services is critical for the quality of service of the developed platform. The relevance can be determined by contexts such as the locations of the service provider and consumer, the type of the service provisioned and the interest of the consumer. In fact, most information (service may also be considered as a type of information) sharing applications/systems utilize the user location (Section 2.3). Therefore, we have incorporated the geographical location information of service providers in their provision process so that location-aware service discovery can be efficiently supported for service consumers.

In practice, the location of an entity is usually represented in a two-dimensional coordinate space[1], with one dimension known as *latitude* and the other as *longitude*. Nevertheless, by simply relying on the geographic coordinates, it has the following limitations for organizing service peers and providing location-aware service discovery in LASPD: (i) the information of the service's administrative area is not reflected, and therefore area-based service discovery cannot be supported; (ii) there lacks an effective way for managing service peers and maintaining their connectivity in an administrative area. If geographical distance is used as the metric to organize different service peers (e.g. each service peer is connected to the nearest neighbor on the geographical map), it would result in an unstructured network and make efficient routing difficult (e.g. by flooding queries for distance-based range search). Furthermore, if there is any changes to the network topology, e.g. due to joining/leaving of a service peer, the maintenance overhead would become high, since variable number of service peers could be affected for their connectivity setup.

To cope with the above issues, we have chosen to encode the location information of each service peer in a one-dimensional identifier. More specifically, the identifier is represented in a binary form and is composed of two parts: $areaID\ peerID$. As an example, the identifier for service peer c in Figure 3.1 is $1110\ 1001$. The areaID is to differentiate peers in different areas so to enable cross-area routing. Its length is $d \cdot \lceil log_2(bf_{max}) \rceil$, where d is the depth of the geographical tree in the first tier and bf_{max} is the maximum branching factor of the tree. Figure 3.2 illustrates the allocation of areaID for the areas in the first tier of Figure 3.1. Once the areaID is determined, the connections among superpeers that represent these areas are settled. Each superpeer is connected to the first superpeer whose areaID is greater than that of its own, i.e. in terms of decimal value. This maximizes the flexibility of area organization; most importantly,

[1]Some geographic positioning system may also add a third dimension to state the entity's altitude.

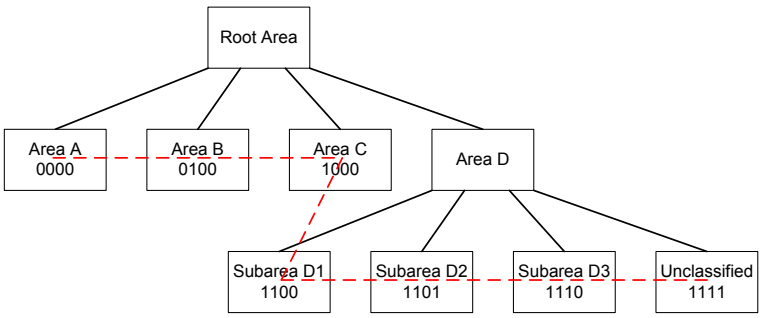


Figure 3.2: Labeling areaIDs for a geographical tree ($d = 2$, $bf_{max} = 4$) defined for the first tier in Figure 3.1.

it ensures a fixed number of connections is maintained by a superpeer irrespective of changes in area definitions.

While the areaID reflects the coarse-grained location information of a service peer, the peerID is supposed to contain the fine-grained location information; meanwhile, it should allow a simple yet effective way for managing service peers in a local area. When surveying the techniques for peerID assignment, we have found the *Hilbert space filling curve* (*Hilbert curve* for short) [27] best fits our requirements due to its locality preserving and fractal (self-similar) properties. As illustrated in Figure 3.3 (top), the Hilbert curve is a continuous fractal curve that can cover a d-dimensional space (in our case $d = 2$) through several iterations. The area shown in the figure consists of seven service peers as demonstrated in the second tier of Figure 3.1. Initially, the curve consists of lines which lay over a few coarse-grained regions. Then it is recursively refined until only one service peer remains in each cell (in two iterations for this case). The peerID of a service peer will then be the ID of the cell it resides in. In fact, the set of cell IDs generated in different iterations can be represented in a hierarchy which we call *Hilbert construction tree (Hilbert tree for short)* (Figure 3.3 bottom). The peerID length is $2r$, where r is the depth of the Hilbert tree for the area. In practice, we target a predefined cell size of $1m^2$ to determine the number of iterations required, as we assume a density of one service peer per square meter is deemed acceptable for most applications.

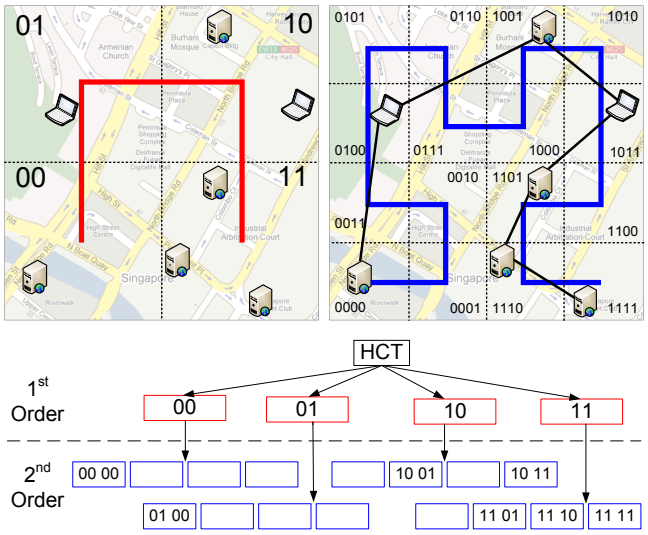


Figure 3.3: Using Hilbert curve for assigning identifiers to service peers (top). The IDs generated in different iterations form a Hilbert construction tree (bottom).

Applying the Hilbert curve has two main advantages: (i) the curve allows one to reduce higher-dimensional proximity problems, e.g. distance-based range search, to a one-dimensional problem. The neighboring cells (in terms of their decimal values of IDs) on the Hilbert curve are always mapped from points that are closest on the geographical map. Although the converse cannot always be true (which is inevitable when mapping from a 2-dimensional space to 1-dimensional space), the Hilbert curve has been found as the best mapping technique for maintaining locality in both directions compared with other space filling curves [110]; (ii) the connectivity among service peers can be simply based on peerID, i.e. if two peers follow each other on the curve, they maintain a connection to each other. With this mechanism, each service peer maintains a fixed number of connections. This allows easy management of service peers especially when the area definition is changed. For instance, if the cell labeled with 01 in Figure 3.3 (top left) is classified as a new area of service administration (due to the construction of a new shopping mall), only the service peers with peerID 0000, 0100 and 1001 are affected for their connectivity. More specifically, service peer 0100 could become the superpeer

for the new area, and service peers 0000 and 1001 will become neighbors. In Section 3.2, we would like to give more details on how to utilize peerID to achieve efficient location-aware service discovery.

Indeed, the method for connecting service areas in the first tier of LASPD resembles the mechanism to construct the Hilbert curve, i.e. by considering the geographical tree as a "Hilbert tree". Though the geographical tree may be incomplete in terms of area definitions (i.e. certain area may be undefined in the tree), the resulted connectivity among service areas exhibits the two properties as discussed for the Hilbert curve. For instance, Area B shown in Figure 3.2 can be further defined into B1 (0100) and B2 (0101), and then the dot line will cross each area in the order <A1, B1, B2, C, D1, D2, D3, Unclassified>, which preserves the area locality. As a result, we will use the term "Hilbert curve" for both tiers in our later discussion, and when referring to the curve for a specific area on the second tier (i.e. local area), we use the term "local Hilbert curve".

3.1.4 Functional Components of Service Peer

Each service peer in LASPD can perform two basic tasks: *service provision* and *service discovery*. The service provision relates to the publication and indexing of services; while the service discovery deals with lookup and invocation of the desired service upon a request. Figure 3.4 shows the core functional components of a service peer and how it relates to services. The functions of each component will be highlighted below.

As a service peer may host multiple services, *service management* is essential. It controls the start/termination of a service and may support service migration as will be discussed in Section 3.3.2. The *service registration* component handles service registration to LASPD with its specifications such as name and description (e.g. WSDL file for Web services), in which we consider three types of services: local service, remote service and mobile service. The latter two types of services are deployed on separate ma-

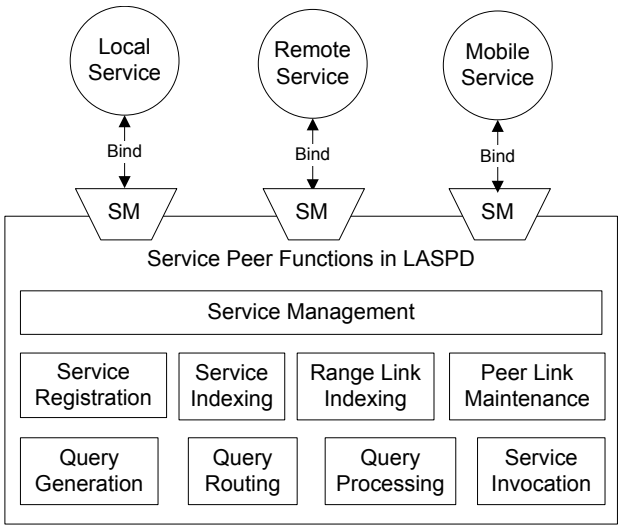


Figure 3.4: Illustration of the functional components of a service peer and its relationship with services. (SM stands for the service mediator)

chines that do not co-host any service peer. For instance, in an enterprise environment, there can be only one machine (e.g. the enterprise server) implementing the functions of a service peer, while the rest of the machines which have services to offer are operating as remote service providers (to the enterprise server). There can also be service provisioned on mobile devices which have limited resources to run a service peer. With the help of the *service mediator* component, the service peer may operate as a proxy for these remote and mobile services. The details about service mediator and how the proxy functions will be discussed in Section 3.1.5. To improve the efficiency of the process of service discovery, all service information registered are indexed using the DHT technique over the respective P2P network of service peers for each local area. The process is completed by the *service indexing* component, and we will discuss in details in Section 3.1.6. The components of *range link indexing*, *peer link maintenance*, *query generation*, *query routing* and *query processing* are all related to the network model of LASPD and the protocol for service discovery, for which we would like to present their functions in Section 3.2. Once the desired service is looked up, the requester may invoke the service directly through the *service invocation* component.

3.1.5 Mobile Service Provisioning

To enable mobile service provision, the mobile service providers have to first find a proxy (as motivated in Section 3.1.2) in the second tier of LASPD. In the prototype, we have designated a port for each service peer capable of being a proxy to listen to, so that if the mobile service provider and the proxy are in close proximity, i.e. in the same WiFi network, the proxy can be discovered via WiFi broadcasting. Alternatively, a list identifying the addresses of potential proxies can be retrieved from a dedicated Web server; or due to security and privacy concerns, the mobile service provider may explicitly choose his own trusted proxy (Section 3.3.3). Once the proxy is discovered, a TCP connection is established between the mobile service and the service peer. More specifically, the connection is maintained by the *service mediator* (SM component in Figure 3.4) which is created by the service peer and is bound to each mobile service provider. The connection between the mobile service provider and the service mediator acts as a control channel and fulfills three tasks:

- *Utilization of Service Peer Function.* Through the service mediator, the mobile service provider registers its service in LASPD by sending the request with necessary service information (e.g. name, description and WSDL file). The service mediator is responsible for interacting with the underlying functions of the service peer such as service registration and service indexing, so that the mobile service can be discovered in LASPD.

- *Invocation of Mobile Service.* To the outside world, the service mediator running on top of the service peer acts on behalf of the mobile service. It registers the service for the mobile service provider using its own IP address and a port as the service reference. Thereafter, any request for invocation of that service will first be routed to the corresponding service mediator designated by the IP address and

the port. By receiving the request (in Web Services SOAP[2] format), the service mediator will pass the message to the mobile service provider through the connection between them. To support concurrent requests, for each request received another TCP connection is set up between the mobile service provider and the service mediator which serves as the data channel to transmit the reply. The number of concurrent requests supported by the mobile service can be configured by the provider.

- *Keepalive.* Despite of the above two types of control messages, a dummy keepalive message will be sent periodically from the service mediator to the mobile service provider. It is to ensure the liveness of the mobile service. In case the mobile service provider has turned off the service or changed network connection due to the movement, the service mediator will detect the event and deregister the mobile service with LASPD and then terminate itself. In addition, the message is also used to prevent disconnection due to network inactivity when the provider is behind a NAT proxy or a Firewall.

It is worth noting that the presence of service mediator has eliminated the need for the service registration to differentiate between local, remote and mobile service provisions, which simplifies the process of registration. The network addressability issue as discussed in Section 1.3 is also solved so that mobile service providers do not necessarily possess public IP addresses for their service provision. At this moment we are assuming each proxy has a public address, but solutions such as Traversal Using Relay NAT (TURN)[3] and Interactive Connectivity Establishment (ICE)[4] protocols can be incorporated into LASPD to remove the assumption.

[2]http://www.w3.org/TR/soap12-part1/.
[3]http://tools.ietf.org/html/rfc5766.
[4]http://tools.ietf.org/html/rfc5245.

3.1.6 Service Keyword Indexing

For validation purpose, the current prototype of LASPD supports keyword-based service discovery; that is, the set of service keywords extracted from the service's description file (e.g. WSDL file) for its name and service description are filtered with a list of stop words (e.g. "and", "the"), and then indexed in the Hilbert curve of the local area by using the SHA-1 hashing function. We have truncated the key size so that each keyword is hashed to a $2r$-bit key (i.e. according to the length of peerID in the area) and is assigned to the first service peer on the Hilbert curve whose $peerID$ is equal to or greater than the key value. The manner to distribute keys is exactly the same as that in the consistent hashing algorithm proposed in [111] and that adopted by P2P systems such as Chord. Meanwhile, the concept of *service scope* is considered in the process of keyword indexing. It defines the target areas (i.e. areas/sub-areas in the first tier of LASPD) for which the service is supposed to serve. The rationale is that for services such as Web services, their hardware facilities do not have to be physically present in their target areas. In such cases, the information of their services should be correctly indexed. For instance, consider a weather report Web service which is hosted in USA but returns weather information of Singapore. To make this service locally discoverable to Singaporeans, this service should be indexed in the Hilbert curve of Singapore but not that of USA.

Of course, by simply using keyword-based service discovery, it has two trivial problems: (i) only exact matching is supported, which restricts the service consumer to issue exactly the same keyword used for indexing in order to discover the service; (ii) for Web services which contain process flows for business logic, the matching is only done at shallow level: the process structure of the service is not reflected in the matching process, and therefore the incompatibility issue may raise up during the integration with the discovered service. There are semantic-based approaches as introduced in Section 2.1.3

to address issue (i); while for issue (ii), our previous work has developed a structure behavioral approach in [112].

3.2 Location-Aware Service Discovery

Due to area classifications, a service consumer and a desirable service provider may not reside in the same area. This mandates the need for differentiating between local-area and cross-area queries. For local area queries, they are routed along the Hilbert curve of the local area until they reach the destination service peer. However for a cross-area query, it could first be directed to the superpeer of the area where the query was originated, and then be routed to the superpeer of the target area along the Hilbert curve of the first tier, and finally be routed to the destination service peer via the Hilbert curve of the target local area. Unfortunately, such a method (which we will refer it as *superpeer-based routing approach* in Section 3.2.3 and Section 3.4.2) is neither efficient nor fault tolerant, as the routing path may involve all the service peers in the two areas and if any of them fails, the destination peer will be unreachable. Moreover, the frequent cross-area queries through the query source's superpeer may stress that superpeer and the overloading or failure of a superpeer would jeopardise the efficiency or operation of cross-area query routing. Therefore, we apply the concept of *long-range links* (i.e. shortcuts) in the small world model to help construct the underlying network and to achieve network navigability and fault tolerance.

3.2.1 Small World Model

Small world phenomenon was first demonstrated by [113], showing that pairs of people in a society can be connected by short chains of acquaintances, or so-called six degrees of separation. Since then, a number of network models have been proposed to

exploit the phenomenon. The most successful one is proposed by [114], in which they construct the small world by modifying a regular network with a small portion of links randomly rewired to form long-range links. The model achieves small world properties in terms of low network diameter and high cluster coefficient. Later, [115] generalizes their model to an infinite family of network models and provides a way of rewiring long-range links so that a decentralized greedy algorithm can be applied to achieve poly-logarithmic search time. Such a *navigability* property is first studied in a 2-dimensional grid and later extended to the hierarchical and group structure models [28]. While Kleinberg's work largely characterizes the static properties of a network that needs to be navigable, recent studies on the small world model are more focussed on its emergence in real world settings. Among those, the rewiring or augmentation process in the evolution of a network attracts the most attention, as it potentially makes a network autonomous. [116] proposes a process that if the target cannot be reached after T_{thresh} steps, a long-range link is rewired from the source peer to the current peer. The threshold T_{thresh} is set based on the expected number of routing steps in the topology. [117] suggests emerging small world by random walks; that is, each peer in the k-dimensional lattice possesses a token. These tokens move mutually independently according to random walks. If the token stops at a certain point of time, a long-range link is rewired from the peer possessing the token to the current one. A link forgetting mechanism is also included in their proposal which reflects natural behavior of the individuals. [118] proposes a so-called *Destination Sampling* mechanism. The mechanism iteratively selects two peers (source and destination) from the peer set uniformly, and constructs the path based on greedy routing from source to destination. For every peer on the path, it has a chance p to create a long-range link to the destination peer. It has been shown that the mechanism adapts robustly to a wide variety of realistic situations [119].

Source Sampling

Among the three evolutionary rewiring proposals, the approaches in [116] and [118] are more efficient and feasible by possibly involving the rewiring process into the process of query routing. However, both of them assume certain information of the destination peer (e.g. distance to the destination peer in [116] and contact information of the destination peer in [118]) is known in advance. Such an assumption can easily be emulated in simulations, but in practice the information of the destination peer can hardly be known until it is discovered first. We hereby propose the Source Sampling mechanism as defined in Definition 1. The rationale is that the information of the source peer (e.g. IP address) is always contained in the query message and can be known to every peer on the routing path; thus it is more practical to rewire long-range links from the source peer to any peer on the path. In addition, the maintenance effort of each peer is minimized as the network structure is constructed and maintained in an autonomous manner. P2P systems such as Chord and CAN usually have fixed network structures and require strong maintenance over the connectivity with the peers in the routing table; while in LASPD, only the connectivity as mentioned in Section 3.1.3 requires strong maintenance to ensure network reachability (more details refer to Section 3.2.5).

Definition 1. *Source Sampling. For a given network graph $G = (V, E)$ composed of a set of vertices V and edges E, let R be the predefined routing scheme and $E(v)$ be the set of edges going out of $v \in V$. A path $X(v_0, t) = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} v_2 \dots v_n \xrightarrow{e_n} t$ can be constructed from vertex $v_0 \in V$ to vertex $t \in V$ based on R, where $v_i \in V$ and $e_i = R(v_i, t, G) \in E(v_i)$. For each vertex v_i ($i \geq 1$) on the path, including t, an independent sampling process is performed. With probability p a shortcut edge e is created from v_0 to v_i. In addition, a random shortcut edge e' is replaced if the current number of shortcut edges reaches the limit for v_0. At the end, $G \Rightarrow G' = (V, E \cup \{e\} - \{e'\})$.*

Figure 3.5 illustrates the source sampling mechanism in a local-area topology. In the

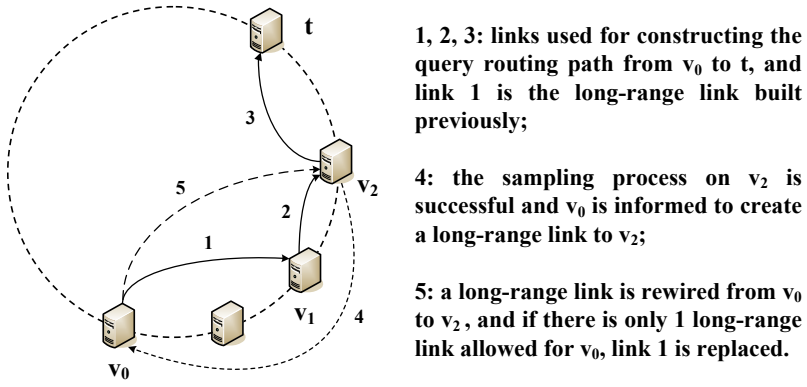


Figure 3.5: Illustration of the source sampling mechanism for a query from v_0 to t.

actual setting, we allow each service peer to have up to k ($O(1)$) number of long-range links. The rewiring probability p is derived based on the formula applied in Kleinberg's hierarchical small world model [28]:

$$p_b(v,u) = \frac{b^{-dist(v,u)}}{Z_v} \tag{3.1}$$

In Equation 3.1, v and u represent the leaf nodes in the hierarchical model (i.e. a complete tree); b is the branching factor of the tree (b-ary tree); the function $dist(v,u)$ is set to measure the height (i.e. the length of the longest downward path to a leaf node) of the least common ancestor between v and u in the tree; Z_v is a normalization factor for v, which is equal to $\sum_{x \neq v} b^{-dist(v,x)}$. The probability $p_b(v,u)$ calculates the chance that there is a long-range link from v to u. Indeed, such a probability ensures the chance to have a long-range link to each group of leaf nodes in the tree (the group is determined by the value measured by $dist(v,x)$, where x is any leaf node except v) to be the same, and a decentralized algorithm is thus allowed to achieve searching in polylogarithmic time with $O(logN)$ number of long-range links assigned for each leaf node [28]. When applying the probability model to the Hilbert tree (as illustrated in Figure 3.3 (bottom)), v and u represent the service peers; b is equal to 4; and $dist(v,u)$ can be measured directly from the two peers' IDs, i.e. $peerID$ for local-area scenarios and $areaID$ for cross-area

scenarios. Since we allow $O(1)$ rather than $O(logN)$ number of long-range links for each service peer, the actual routing efficiency is expected to be $O(log^2 N)$ (as verified[5] in the network navigability test in Section 3.4.2). Nevertheless, in practice, deploying the original probability model faces two issues. Firstly, the default model applies on complete balanced trees only, which may not be true for the case of the Hilbert tree as illustrated by the empty leaf nodes in Figure 3.3 (bottom), and therefore the probability model can be inaccurate and results in routing performance issue as will be demonstrated in Section 3.4.2. Secondly, the process of constructing the small world network (i.e. the convergence speed for the routing performance) is extremely slow when the model is applied to areas with large size, which is due to the large $dist(v, u)$ value applied. As a result, we have introduced two other parameters in the default model:

$$\tilde{p}_b(v, u) = \frac{b^{-dist(v,u)}}{Z_v} \cdot cov_{id}(u) \cdot \alpha \tag{3.2}$$

In Equation 3.2, parameter $cov_{id}(u)$ measures the ID "coverage" of service peer u, and is used to remove the effect of empty leaves in the Hilbert tree, which may frequently happen in large areas. It is derived by halving the ID distance between the two neighbors of u on the Hilbert curve. The constant parameter α, on the other hand, helps to accelerate the convergence speed for source sampling, i.e. by increasing the chance that a shortcut is created. It varies the mean shortcut distance (i.e. the link length in terms of hop count) for all the long-range links rewired. In Section 3.4.2, we will justify the effectiveness of applying the two parameters in performance convergence for source sampling as compared to the original probability model.

[5]The complete formal proof of the $O(log^2 N)$ performance bound still requires further investigation. It is hard to analyze the network model mathematically, which is due to the dependencies between long-range links rewired in the evolutionary process. Nevertheless, to prove the performance convergence of the process, Sandberg and Clarke's approach [118] can be applied here.

Centralized Sampling

An alternative way to construct the small world network is what we called Centralized Sampling. The mechanism requires a centralized coordinator (i.e. the superpeer in this case) to help assign long-range links (i.e. the computation of $\tilde{p}$) for all the peers in the network. For instance, when there is a new service peer registered, the following two sampling processes are involved: (i) sampling for the registered service peer (i.e. v in Equation 3.2) with each peer in the network (i.e. u); (ii) sampling for every service peer in the network (i.e. v) with the registered peer (i.e. u). In (ii), if the process successes, the coordinator will send a message to the respective peer with the information of the registered peer (for the creation of long-range link). However, compared with source sampling, centralized sampling suffers from two drawbacks: (i) the computation and assignment of long-range links are all done by the centralized coordinator, which could be the potential bottleneck and become a single point of failure; (ii) for any topology change, all the service peers have to be re-sampled, and if there is any change to the long-range link assignment, messages have to be sent out by the coordinator, which causes large maintenance overhead.

Nevertheless, in practice source sampling also has its own issues: (a) not every peer has a chance to become the source peer, so the peer can have no long-rang link assigned (due to the rewiring process as described in Definition 1); (b) source sampling requires time to converge for its routing performance (as will be demonstrated in Section 3.4.2). Therefore, a more practical approach is to combine the two mechanisms, by first applying centralized sampling for the newly registered service peer (i.e. the sampling process (i) as described in the last paragraph), and then relying on source sampling for the maintenance of its long-rang links. The advantage by combining the two mechanisms has been observed through simulation studies and will be demonstrated in Section 3.4.2.

3.2.2 Network Routing

For query routing, a simple greedy algorithm based on the identifiers assigned to each service peer is adopted. To further improve the routing efficiency, we close the Hilbert curve to form a loop by connecting the service peer with the smallest identifier to the one with the largest identifier. In this way, queries can be routed in both directions on the Hilbert curve. Definition 2 describes the greedy routing process.

Definition 2. *Greedy Routing R. For a given network graph $G = (V, E)$ with each vertex labeled with areaID of length $2d$ and peerID of length $2r$, let $E(v)$ be the set of edges going out of $v \in V$ and $v_i = e_i(v)$ be the vertex connected through $e_i \in E(v)$ from v. Suppose t is the destination vertex, the greedy routing R selects the next edge $e = R(v, t, G)$ from v based on the criteria that $e = \text{argmin}_{e_i \in E(v)} (\tilde{dist}(e_i(v), t))$, where*

$$\tilde{dist}(v_i, t) = \begin{cases} min\{|peerID(v_i) - peerID(t)|, \\ \qquad 2^{2r} - |peerID(v_i) - peerID(t)|\} \\ min\{|areaID(v_i) - areaID(t)|, \\ \qquad 2^{2d} - |areaID(v_i) - areaID(t)|\} \end{cases}$$

for local-area and cross-area routing respectively.

3.2.3 Area-based Service Discovery and Long-Range Link Indexing

The greedy routing algorithm mentioned above is based on a target key contained in the query header. For local-area queries, the keywords specified by the requester are first hashed to the respective keys according to the hash function used in Section 3.1.6. Each key is then looked up on the local Hilbert curve. During the query generation, the $areaID$ of the target service peer in the query header is set to -1; while the $peerID$ is set to the integer value of the key. The greedy routing mechanism is then applied to find

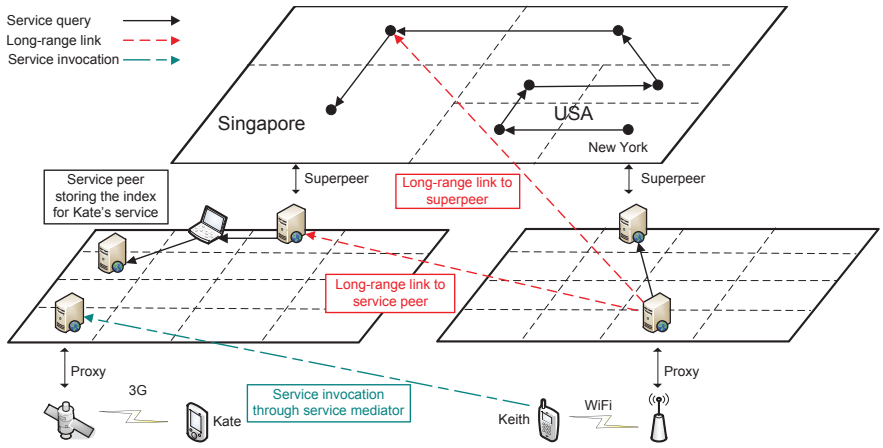


Figure 3.6: Illustration of the source sampling mechanism for the Interactive Event Sharing Application (Scenario 2 in Chapter 1).

the service peer storing the key.

For cross-area queries, the target area is specified by the user and indicated by the $areaID$ in the query header. The superpeer-based routing algorithm as mentioned in the beginning of Section 3.2 can be applied. For instance, consider the Interactive Event Sharing Application illustrated in Scenario 2 of Chapter 1 (Figure 3.6): the proxy of Keith's mobile device may send the service query (i.e. asking for services about sharing of information for the general election) to the local superpeer of New York directly, which in turn will route it to Singapore by the Hilbert curve on the first tier (Note that it is possible to have shortcuts among superpeers on the first tier). Once the query reaches Singapore, the local-area greedy routing algorithm is applied and the query is routed to the service peer storing the index for relevant services. In this case, Kate's mobile service is discovered. After receiving the reply from the service peer, Keith may invoke Kate's service through the service mediator running on Kate's proxy.

However, as mentioned before, the above method would stress superpeers from the workload for cross-area routing and the system robustness is also an issue in case some of the superpeers fail. Therefore, we have decided to make use of the long-range links created through Source Sampling by service peers in an area. Continuing the application

example, now assume there are also people in New York who want to discover other services in Singapore (not necessarily to be Kate's service), their requests would then be efficiently routed to Singapore by leveraging on those shortcuts (red dot lines) created due to the routing of Keith's request. Nevertheless, to let other service peers within the area know the existence of long-range links of the current service peer, *an index for the long-range link* should be kept, which maps the target $areaID$ of the long-range links to the local service peer IP address. Two options are available here for the storage of the index: (i) let each local superpeer take the role; (ii) distribute the index to the service peers in the area. Option (i) obviously contradicts our intention (i.e. make superpeers stressful again); therefore, we have chosen the second option also due to its scalability in dealing with index maintenance. The distribution and maintenance of the index are similar to that of service keyword indexing except that this time we use the target $areaID$ of the long-range link as the key. As a result, using the modified cross-area routing mechanism, the query is first routed to the local service peer that stores the index of the target $areaID$. If there are already long-range links created for the target area, a random service peer with such a link is contacted for the routing of the query; but, if the index is empty, the local superpeer is contacted and used for routing. Such an approach introduces some overhead in the routing of cross-area queries, especially in the initial stage of Source Sampling. Nevertheless, after some rounds we expect that our approach would require significantly less routing effort on the superpeers while the routing performance is not degraded compared to the superpeer-based routing approach. Indeed, the above phenomenons have been observed in our simulation whose details could be found in Section 3.4.2.

3.2.4 Distance-based Range Search

Consider queries like "discover services of type t within x meters" or "browse all services within x meters". These are examples of distance-based range search queries that are extremely useful in real life. It is hard to provide such features via conventional P2P platforms unless the query is flooded to all peers of the area. In our platform, such queries can be easily handled due to the usage of locality-preserving Hilbert curve. The query is only matched with those relevant service peers in the range but not to all local peers. In our approach, we first discover segments of the Hilbert curve involved in the range search (as illustrated in Figure 3.7). Segments are identified with $< startID, endID >$ and are stored in the query in ascending order based on $startID$. The query is then sent to discover the service peers falling within those segments. A temporal segment $< currentStartID, currentEndID >$ is kept to trace the current segment to be discovered, initially assigned with the value of the first segment. Once the query reaches the destination peer which has the least greater $peerID$ compared to $currentStartID$, the query is passed through and matched with all service peers along the curve until the current peer has a $peerID$ exceeding $currentEndID$. The discovery on this segment is then considered finished and the temporal segment is assigned with the value of the next segment. The process continues until all segments are finished.[6]

For the detailed derivation of segments, we have developed an R-tree style search algorithm. Algorithm 1 represents the procedure to get segments. It requires parameters of the area boundary, the number of bits for the peerID and the search bound. After creating and filling the segment list, it will assemble those continuous ones to reduce later process overhead. The actual segments are found by calling the procedure $findSegments()$ in Algorithm 2, which specifies a recursive procedure to find all the relevant segments that

[6]In practice, the routing efficiency of the process can be improved by sending range queries for each segment simultaneously. With this approach, each query only carries the information of a single segment $< startID, endID >$. The major routing overhead occurs when looking for the service peer with $peerID$ equals to $startID$, i.e. to discover the initial peer on each segment.

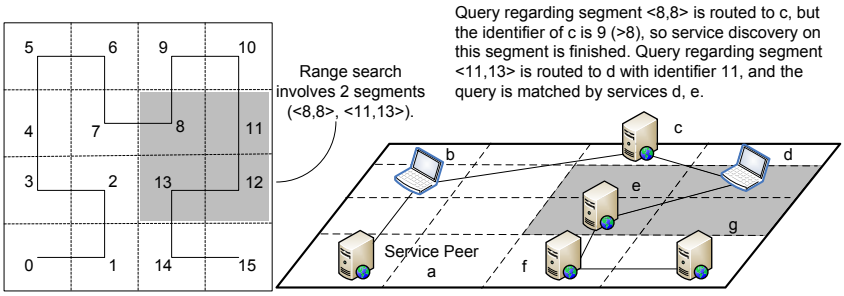


Figure 3.7: Illustration of the distance-based range search mechanism.

are contained in the search bound. It utilizes the process for the Hilbert curve construction. Each sub-area ($r1$,$r2$,$r3$,$r4$) is checked with the $bound$ in ascending order of their startIDs (line 10–18) so that they can be added into the list in an orderly manner. Besides, if the sub-area is already contained by $bound$, a segment can be directly constructed and added to the list. Algorithm 2 presents the recursive procedure, in which the processing of one area is illustrated.

3.2.5 Bootstrapping and Connectivity Maintenance

Like many other P2P networks, a new service peer who intents to join LASPD must first discover contact information for another service peer in the existing network. As LASPD deploys superpeer for each local area (i.e. maintaining area information including the local Hilbert curve), the bootstrapping process of a service peer mainly involves the discovery of its local superpeer. The following methods have been implemented in the current prototype to handle different cases:

- If the service peer is first time joining LASPD, it can contact our Web server to get its local superpeer information (i.e. based on its service scope as discussed in Section 3.1.6). Once the service peer successfully contacts the superpeer, its peerID is assigned based on geographical location and its areaID is directly copied from the superpeer. Besides, the contact information of the two immediate neighbors of this

Algorithm 1 List getSegments(Rectangle $areaBound$, int $areaIDLength$, Rectangle $bound$)

```
1: Create an empty List segs;
2: x = areaBound.x; y = areaBound.y;
3: w = areaBound.width;
4: leftBits = areaIDLength − 2;
5: findSegments(x, y, w, 0, 0, 0, leftBits, bound, segs);
6: for i = 0 to segments.size() − 1 do
7:     seg = segs.get(i);
8:     nextSeg = segs.get(i + 1);
9:     if seg.endID = nextSeg.startID − 1 then
10:        newSeg = Segment(seg.startID, nextSeg.endID);
11:        segs.setElementAt(newSeg, i);
12:        segs.removeElementAt(i + 1);
13:        i = i − 1;
```

Algorithm 2 findSegments(int x, int y, int w, int i1, int i2, int idHead, int leftBits, Rectangle bound, List segs)

```
Require: w > 1
1: w = w/2; id1 = 0; id2 = 0; id3 = 0; id4 = 0;
2: size = 2^leftBits; leftBits = leftBits − 2;
3: if i1 = 0 AND i2 = 0 then
4:     id1 = idHead ∗ 4 + 1; id2 = idHead ∗ 4 + 0;
5:     id3 = idHead ∗ 4 + 3; id4 = idHead ∗ 4 + 2;
6:     r1 = Rectangle(x + i1 ∗ w, y + i1 ∗ w, w, w);
7:     r2 = Rectangle(x + i2 ∗ w, y + (1 − i2) ∗ w, w, w);
8:     r3 = Rectangle(x + (1 − i1) ∗ w, y + (1 − i1) ∗ w, w, w);
9:     r4 = Rectangle(x + 1 − i2) ∗ w, y + i2 ∗ w, w, w);
10:    if bound.intersects(r2) then
11:        if bound.contains(r2) then
12:            seg = Segment(id2 ∗ size, (id2 + 1) ∗ size − 1);
13:            segs.add(seg);
14:        else
15:            findSegments(x + i2 ∗ w, y + (1 − i2) ∗ w, w, i1, 1 − i2, id2, leftBits, bound, segs);
16:    if bound.intersects(r1) then ...
17:    if bound.intersects(r4) then ...
18:    if bound.intersects(r3) then ...
19: else if i1 = 0 AND i2 = 1 then ...
20: else if i1 = 1 AND i2 = 0 then ...
21: else ...
```

new service peer on the local Hilbert curve is also passed by the superpeer. Subsequently, the two neighbors shall be updated with the new service peer information due to the connectivity requests sent from the service peer. In addition, long-range links to other service peers in the same area can be assigned by the superpeer at this moment, if the centralized sampling mechanism (Section 3.2.1) is applied;

- If the service peer is the first peer appearing in the area, it may take the role of superpeer. In that case, the area information including the areaID and area size is retrieved from the Web server to aid it in the construction of local Hilbert curve. In addition, the new superpeer has to set up two extra connections with the two neighbors on the Hilbert curve in the first tier of LASPD in favor of cross-area query routing;

- Each service peer caches a list of other service peers in LASPD (i.e. their contact information) after it successfully joins LASPD, e.g. its local superpeer and the two neighbors. For the case of rebooting or rejoining, the cached service peer may be contacted to acquire the local superpeer information. The procedure is completed by sending a superpeer discovery message to the cached service peer. The message is then routed to the target area (where the new service peer is supposed to register) similarly to the service discovery message, and any peer in the area can reply with its superpeer information.

All the service peers in LASPD maintain their connectivity with other peers by periodically sending *ping* messages to detect peer leavings or failures. More specifically, the maintenance involves two types of connectivity :

- *Neighboring link connectivity*, which is assigned based on the mapping of geographical locations of service peers on the Hilbert curve. Note that all superpeers have two more neighboring links (i.e. for the Hilbert curve on the first tier) as

compared to service peers on the second tier. The superpeer does not explicitly maintain connectivity with all the local-area service peers; rather, it operates in a passive manner such as by relying on reports sent from services peers to deduct the unavailability of a particular service peer. For instance, if a service peer leaves without notice, its two neighbors would discover the fact as they cannot receive any reply from their ping messages. If this situation (i.e. ping without reply) occurs more than a certain threshold (e.g. 3 times as set in the prototype), the two neighbors will conclude that the service peer has gone, and a link repair request will be sent to the local superpeer from their respective peer link maintenance component as shown in Figure 3.4. Once the request is received, the superpeer will remove the service peer from its local-area peer list, and new neighboring connectivity will be assigned to the requested service peer;

- *Long-range link connectivity*, which is assigned by the process of centralized sampling or source sampling. Since long-range link is one-way connection (as defined in Definition 1), its maintenance is only done by the assigned service peer. More specifically, if the remote peer of the long-range link becomes unavailable (i.e. detected with the same mechanism as for neighboring link), the long-range link will be simply removed from the connectivity list of the service peer.

However, we realize that the above approach for neighboring link maintenance would have problems when the superpeer fails. In that case, service peer would not be able to get neighboring link assigned either for their registrations or for link repair requests. We are currently working on an autonomous approach, which involves a protocol for service peer discovery (similarly to the case for superpeer discovery). The protocol leverages on the existing peers in the network to discover the destination peer, i.e. the neighboring peer. With the protocol, service peers may register or repair their links without the help of the superpeer. But there are further challenges to take care of, such as the segmentation

of network overlay due to simultaneous failures/leavings of service peers.

3.3 Further Discussion

This section presents and discusses potential features supported by LASPD, which is for practical considerations of service provision and discovery.

3.3.1 Location Determination

Location determination of service providers and requesters is essential for location-aware service discovery. In fact, providing the location information of mobile devices can itself be considered as a service deployed in LASPD. The physical location of the service provider could be determined by using either the Global Positioning System (GPS), the WiFi networks, or other suitable localization technologies. The coordinates are then transformed to the semantic location (area name) identified by areaID and peerID during the registration of a service peer by the superpeer. For mobile service providers, their proxy will join the corresponding network of service peers according to the location of the service provider. However, if the mobile devices do not support GPS or are with GPS function disabled (e.g. due to large power consumption), they could use the physical location of the proxy to approximate their coordinates, assuming they are using WiFi to connect the proxy.

3.3.2 Data Replication, Caching and Service Migration

Data replication is extremely important in mobile and dynamic environments since nodes may join or leave at anytime. For the first and second tiers, failure or leaving of a peer without notice may result in key loss. In LASPD, we could replicate the keys by setting the length of a key to less than $2r$ bits as derived by the order of local Hilbert

curve. For instance, with key length equal to $2r - 3$, the last 3 bits of the peerID are not considered when distributing keys, and the same set of keys may be stored by up to 2^3 service peers, which are indeed geographically close on the local Hilbert curve.

For third-tier peers, caching of WSDL files and SOAP responses on the proxy can lessen the workload on the mobile device. Another option is to do service migration as in [75]; that is, to deploy the Web service to the proxy and letting it temporarily host the Web service. The two options are also quite useful in dealing with the mobility of the service provider. For instance, if the mobile service provider (e.g. Kate) is moving, the services provided by her may be interrupted when she switches proxies, e.g. due to the change of access point. Stateful services such as a video streaming service which keeps the frame position for each end user are one possible solution. In LASPD, we consider two mechanisms to handle the mobility issue: (i) if the expected interruption time is short, the original proxy may cache the requests for a limited time period, and once the service provider is attached to a new proxy, the original proxy is informed by the new proxy so that those cached requests can be forward to the service provider; (ii) if the expected interruption time is long, e.g. moving from one city to another, under the permission of the service provider, the service could be migrated to the original proxy and after a predefined time (or until there are no more requests for this service), the service is closed. Of course, for both caching and service migration, the challenges will be the privacy of the transferred data, deciding on when to do the operations, and who decides if the operation is required, i.e. machine or user.

3.3.3 Security and Privacy Protection

Security is always a concern in the public domain, especially when not all the entities are trusted. To provide secure service provision, the standard WS-Security[7] can

[7]http://docs.oasis-open.org/wss/v1.1/.

be adopted in LASPD. In addition, we allow the mobile service provider to explicitly choose his trusted proxy. For instance, the caregiver in Scenario 1 of Chapter 1 can select his home computer or an enterprise server as the proxy, for caching or migrating his service data. Aside from the security considerations, the location of a service provider can be revealed during service provision, while he may only want to be known in his surrounding or a specific area. To address this issue, we utilize the concept of service scope (Section 3.1.6). The service provider may choose his scope of service provision.

For private areas, the security function of the superpeer is extremely important. Our architecture and the network routing model could support tasks such as access control efficiently. The range-link indexing mechanism discussed in Section 3.2.3 could be disabled in an area so that all messages sent from the area are routed through the superpeer first. Also, in source sampling technique, the source peer address can be masked by the local superpeer so that the long-range link can only reach the superpeer but not any peer inside the area. In this way, all messages sent to the area should go through the superpeer. Other tasks such as service peer authentication can also be provisioned by a superpeer.

3.4 Performance Analysis

We have developed a simulation model of LASPD and conducted a detailed series of simulation to investigate its characteristics and performance in terms of service discovery. The simulation program models the architecture of LASPD, its components and processes; their details will be described together with simulation results in this section.

3.4.1 Simulation Modeling

- *Simulation Model.* Our simulation model considers service peers on the first two tiers of LASPD. The third tier is for mobile service providers, which rely on a

proxy (in the second tier) for their service provision and discovery. Therefore, when we simulate the process of query routing, these providers are not considered in the model. Rather, in Section 5.6.4 when we present the prototype implementation for SOLE, the tests on mobile service provision such as the latency of real-time service invocation will be demonstrated.

- *Topology Setup.* The simulated LASPD platform considers two types of area definitions: *a single area* and *a large area* composed by multiple sub-areas. The single area topology is constructed solely for the testing of query routing on the second tier, i.e. the local-area scenario; while the large area involves routing on both tiers (i.e. the cross-area scenario) and its detailed area definition is according to a predefined geographical tree. In the simulation, each area/sub-area is represented by a rectangle and is specified using $< x, y, width, height >$ format, where $< x, y >$ refers to the starting coordinate of the area, and $width \times height$ represents the size of the area. For each type of area definition, there are total N number of service peers generated for the area (N may vary from 2^{10} to 2^{14} in the actual setting), and each of them has up to $2 \cdot k$ number of long-range links (in most cases $k = 1$ unless otherwise specified), with k links assigned for local-area query routing and another k for cross-area routing.

- *Simulation Process.* The service peer joins LASPD by following the process as presented in Section 3.2.5. Note that a service peer will take the role of superpeer if there is no other peer in the same area. Each service peer provisions a local service identified with a unique name, and the service name is indexed through the process described in Section 3.1.6. The service discovery is then simulated by composing a query which contains the following information: the *source peer* where the query is originated, the *target area* where the query is routed to (i.e. indicated by $areaID$), and the *target service* which is selected from the set of the

services provisioned in the target area. Thereafter, the process of service discovery is triggered (Section 3.2.3), and the process is completed until the source peer receives the reply from the service peer which stores the key for the target service.

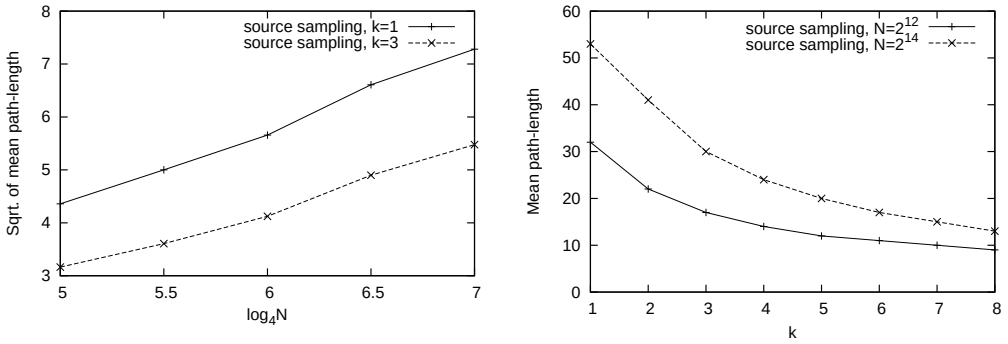
The following assumptions/simplifications are made in the simulation model:

- Service peers are uniformly distributed in the area, and each coordinate contains at most one service peer. This assumption is to ensure every service peer can be assigned with a unique ID which is used for query routing.
- Service queries are generated in a way that both the source peer and the *target peer* (i.e. the service peer which provisions the target service) are uniformly picked up from the service peer set. The $areaID$ of the target area is then encoded from that of the target peer. The process assumes every service peer can be the source peer for service discovery as well as the target peer for its service provisioned, and also every query is resolvable.
- The simulation model has not counted in parameters such as network transmission delay and message processing delay, as these parameters are largely dependent on the practical setups, e.g. networks and machines deployed. Instead, our simulation studies are by observing the path-length in terms of hop count to learn the query routing efficiency. The detailed performance metric will be discussed in the respective section.

3.4.2 Simulation Results

Network Navigability

This section aims to verify that the network model constructed by the proposed source sampling mechanism exhibits the navigability property; that is, with $O(1)$ long-range links allowed for each service peer, the service discovery can be completed in



(a) Network navigability test.

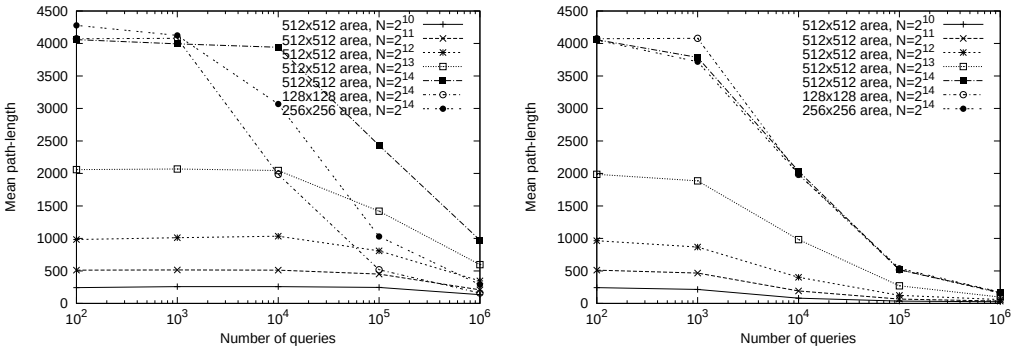
(b) The impacts of k values.

Figure 3.8: Experimental results for the mean path-length versus increased network size and k values.

$O(log^2 N)$ time. The simulations are carried out on a 512×512 single area. We use *path-length*, which is measured by the total number of hops (i.e. service peers) traversed by the query until the reception of the reply by the source peer, to evaluate the routing efficiency. Figure 3.8a plots the relationship between the number of peers (in terms of $log_b N$, where $b = 4$ as for the branching factor of the Hilbert tree) and the square root of *mean path-length* (derived by summing all the path-length for each service query and divided by the total number of queries sent) for query routing when the routing performance of source sampling converges to a stationary state. As illustrated by the figure, with $k = 1$ long-range link allowed for each peer, the mean routing path-length increases in $O(log^2 N)$ scale while the peer size N increases exponentially. The same pattern is also observed for the case when $k = 3$.

Figure 3.8b demonstrates the effects of changing the value of k for $N = 2^{12}$ and 2^{14} service peers in the same single area. The mean path-length is shortened as k increases; however, the performance gain becomes smaller when k reaches a certain value, e.g. $k = 3$. Indeed, with larger k value, we should expect that the cost from the operation for long-range link rewiring as well as that for link maintenance is increasing.[8]

[8]While this thesis focuses on the validation of network navigability for source sampling when $k = 1$, in practice the tuning of k's value is necessary to achieve optimal performance, i.e. by balancing network



(a) Convergence patterns without cov_{id}.

(b) Convergence patterns with cov_{id}.

Figure 3.9: Effect of parameter cov_{id} in determining convergence pattern.

Effects of Parameters cov_{id} and α

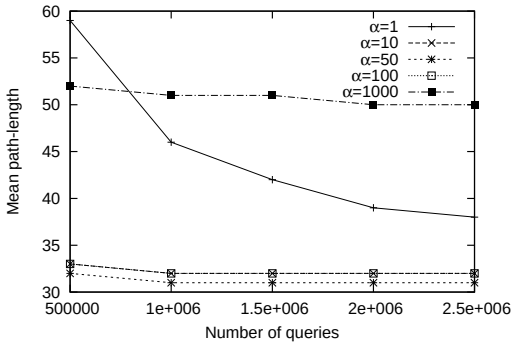
This section studies the two parameters (i.e. cov_{id} and α) we introduced in the default probability model from Kleinberg (Section 3.2.1). Equation 3.1 and Equation 3.2 are applied in the source sampling mechanism and the patterns of convergence (i.e. in terms of routing efficiency) are compared.

Figure 3.9a shows the patterns for the default probability model proposed by Kleinberg[9], in which the convergence of routing performance is dependent on both the area size (e.g. by comparing the case of 512×512 area, $N = 2^{14}$ and the case of 128×128 area, $N = 2^{14}$) and the network size (e.g. by comparing the case of 512×512 area, $N = 2^{10}$ and the case of 512×512 area, $N = 2^{14}$); while with the introduction of parameter cov_{id} (Figure 3.9b), the convergence patterns are only dependent on the network size. This justifies the usage of cov_{id}, which aims to remove the effect of empty cells in a local area (i.e. those cells without service peers allocated by the process of Topology Setup) and allow faster convergence speed for large-size area (Section 3.2.1).

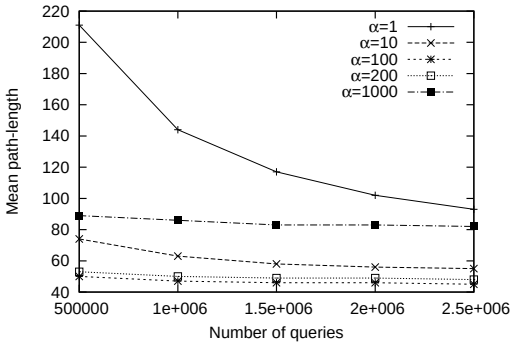
Figure 3.10 shows the effect of parameter α for network size $N = 2^{12}$ and $N =$

routing efficiency and maintenance cost. The network optimization problem has been left as part of our future work. In addition, for those networks with high churn rate, by setting larger k value the robustness in small world routing can be improved. Such an observation has already been demonstrated in [120].

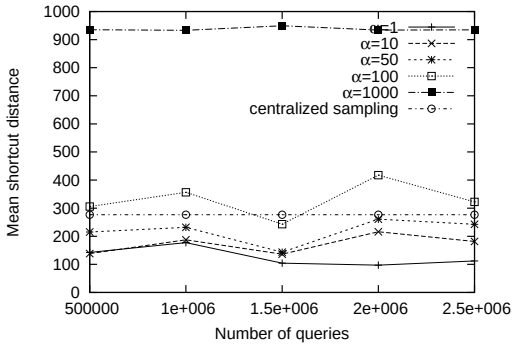
[9]Note that at the initial stage of the simulation, there are quite few number of long-range links augmented by our source sampling. As a result, the routing path-length can be extremely long.



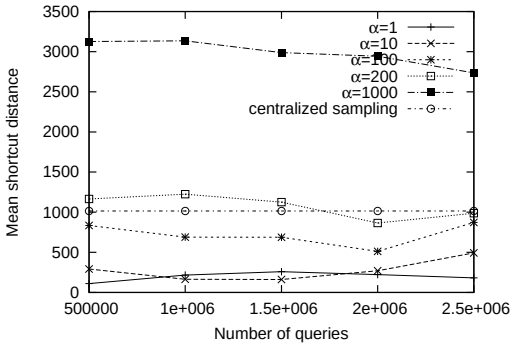
(a) Convergence patterns for $N = 2^{12}$.



(b) Convergence patterns for $N = 2^{14}$.



(c) Snapshots of shortcut distance for $N = 2^{12}$.



(d) Snapshots of shortcut distance for $N = 2^{14}$.

Figure 3.10: Effect of parameter α in determining routing performance.

2^{14}. From Figure 3.10a and 3.10b, we can observe that α plays an important role in determining the convergence pattern for source sampling. The convergence speed boosts up when the value of α is increasing. For instance, the routing performance almost reaches a stationary state after 10^6 queries in the case of $\alpha = 100$ for $N = 2^{12}$ service peers. However, the value of α cannot be set to be infinitely large. On one hand, it increases the cost for link-rewiring operations; on the other hand, it may severely impact on the underlying probability model (Equation 3.2). The latter problem is observed when α is set to 1000 for both $N = 2^{12}$ and $N = 2^{14}$ cases. Through further simulation studies, we have found that the value of α indeed affects the mean shortcut distance (i.e. the average link length in terms of hop count) for all the service peers in the network, as illustrated in Figure 3.10c and 3.10d. In both figures, we have plotted the respective

mean shortcut distance derived from the centralized sampling mechanism (the value is stable in both cases as there is no modification to the network topology after Topology Setup). As observed, when α achieves similar results for the value of mean shortcut distance as in centralized sampling (e.g. $\alpha = 50$ for $N = 2^{12}$ and $\alpha = 200$ for $N = 2^{14}$), its effect on source sampling (i.e. in terms of convergence speed and stationary routing performance) is almost the best (as illustrated in Figure 3.10a and 3.10b). In addition, the optimal value of α seems to be proportional to the network size N in our network model. One possible reason is that the value of cov_{id} in the probability model (Equation 3.2) is inversely proportional to the network size N (based on the definition of cov_{id} in Section 3.2.1). Therefore, to achieve the same sampling effect on each peer, α has to be increased along with the network size N. Nevertheless, further investigation on the optimal value of α is still required.[10] In later experiments, unless otherwise specified, we use $\alpha = 1$ for performance testing.

Routing with Different Sampling Mechanisms

This section studies the routing efficiency for the proposed source sampling and centralized sampling mechanisms (Section 3.2.1) in a 512×512 single area with $N = 2^{14}$ service peers. The performance in terms of routing path-length for both approaches has been plotted in Figure 3.11. The figure shows that by involving a centralized coordinator for the assignment of long-range links, the centralized sampling mechanism has already achieved a good routing performance, i.e. mean path-length is 57 with $k = 1$ for each service peer. While source sampling is still in its convergence process (the figure is plotted after 10^6 queries). However, as mentioned in Section 3.2.1, centralized sampling requires explicit maintenance of long-range links assigned for each service peer;

[10] Our empirical studies show that the hypothesis "when α achieves similar results for the value of mean shortcut distance as in centralized sampling, its effect on source sampling (i.e. in terms of convergence speed and stationary routing performance) is almost the best" is true in both ring (i.e. the initial setup of peer connectivity on a local Hilbert curve) and later grid topologies (Section 3.4.2).

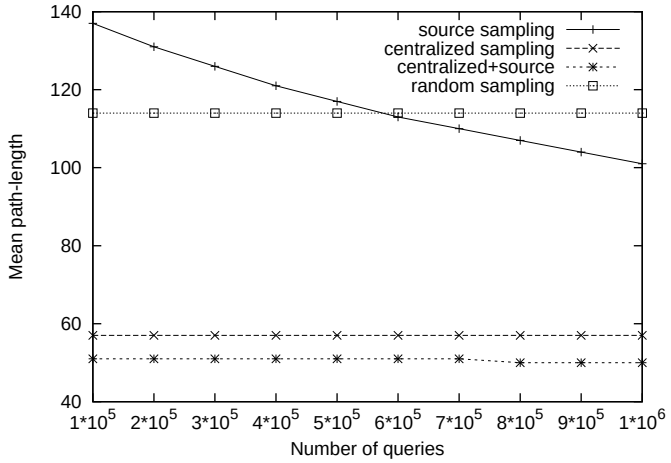


Figure 3.11: Performance comparison for local-area routing efficiency with different sampling mechanisms.

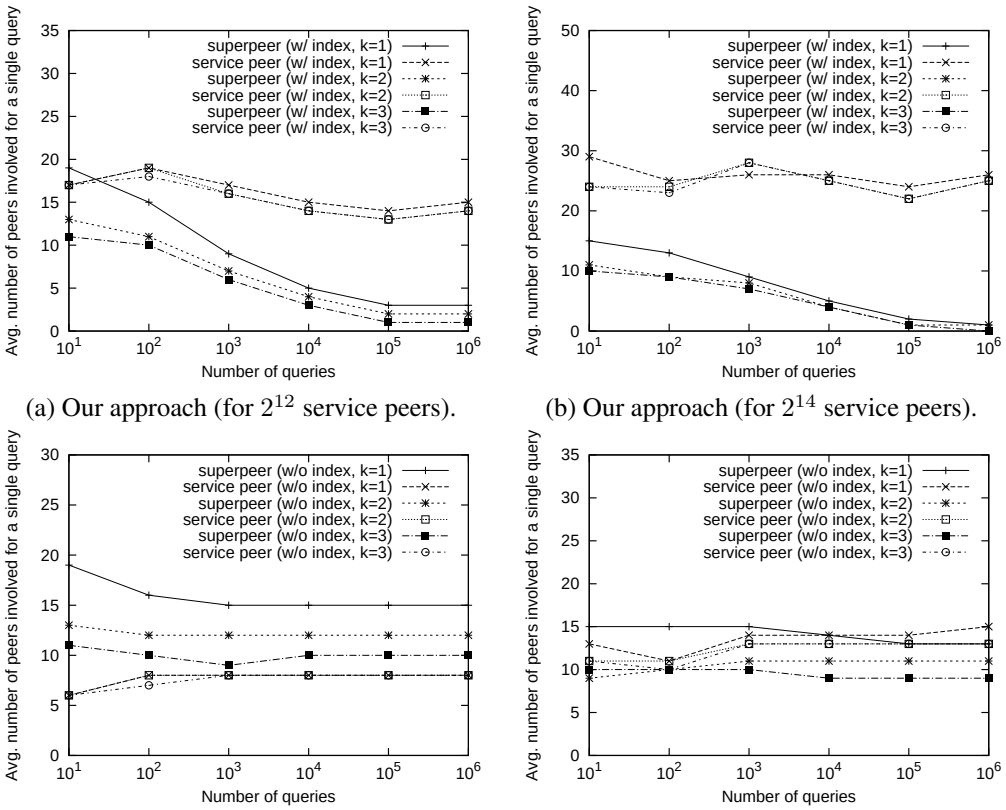
while source sampling adopts an autonomous approach to achieve so. Indeed, as shown in Figure 3.8a, source sampling may achieve 52 mean path-length for the same testing case when it reaches the steady state. As a result, a better approach in practice is to combine the two mechanisms as discussed in Section 3.2.1. The result of doing so has been plotted in Figure 3.11, which shows 50 mean path-length.

The centralized sampling mechanism may also deploy other types of probability models in its sampling process. To verify the effectiveness of the probability model we adopted (Equation 3.2), we have plotted the results for *Random Sampling*; that is, during the registration of a service peer, one of the peers in the network will be randomly selected and assigned as its remote peer. As shown in the figure, the mean path-length is 114 which is significantly worse than the performance of other mechanisms.

Effect of Long-Range Link Indexing

This section simulates the scenario for cross-area query routing. The simulation is carried out on a 512×512 large area, with its area hierarchy based on a geographical tree generated randomly. In a trail run of the simulation, the depth of the tree is 6 and

the number of tree nodes (i.e. local areas) is 364. The centralized sampling mechanism is adopted during the peer registration time: each service peer is assigned with k number of long-range links for local-area query routing, and each superpeer is assigned with additional k number of long-range links for cross-area routing, i.e. by targeting superpeers in different areas. Similar to the query generation process in 3.4.1, both the source peer and the target peer are selected randomly, except that they must be in different local areas. The *routing effort* imposed on superpeers (or service peers), defined as the average number of superpeers (or service peers) involved in routing of a single query, is studied. It reflects the required processing workload. Figure 3.12 compares the results for our current approach (with cross-area long-range link index) and the superpeer-based routing approach (without cross-area long-range link index). We observe that in our approach (Figure 3.12a and 3.12b), the routing effort required on superpeers decreases as the number of queries routed increases. This is because the more the number of queries, the more the long-range link indexes are created. It justifies that the role of the superpeer in LASPD can be performed by any non-dedicated server and the failure of a superpeer unlikely will affect cross-area routing severely, as will be shown in the next section. We also note that with larger k values (i.e. number of long-range links augmented for cross-area routing), a lesser number of queries is required to achieve the same effect in reducing the routing effort. The reason is simply due to more long-range links being created with the same number of queries routed. However, the performance gain becomes smaller when k increases. For the superpeer-based routing approach (Figure 3.12c and 3.12d), the routing effort required on superpeers does not change considerably at difference number of queries. Figure 3.12c shows that for the case of low-density distribution of service peers on the large area (with 2^{12} service peers), superpeers play a more critical role in cross-area routing, e.g. the routing effort imposed on superpeers is almost twice as that on service peers when $k = 1$.



(a) Our approach (for 2^{12} service peers).

(b) Our approach (for 2^{14} service peers).

(c) Superpeer-based approach (for 2^{12} service peers).

(d) Superpeer-based approach (for 2^{14} service peers).

Figure 3.12: Effect of long-range indexing mechanism in affecting the routing behavior of superpeers and service peers respectively.

Note that overall, our approach requires a slightly higher routing effort on service peers. For instance, when $k = 1$ we have an average of 26 service peers after 10^6 queries (Figure 3.12b) as compared to 15 in the superpeer-based approach (Figure 3.12d). It is also noticed that, the path-length for cross-area routing does not differ much, i.e. $1 + 26$ versus $13 + 15$ after 10^6 queries.

Fault Resilience

As mentioned in Section 2.1.3, one major problem of the superpeer-based clustering approach for service provision and discovery is that a lot of workload is imposed on superpeers, and their presence makes the network more vulnerable to the failure of

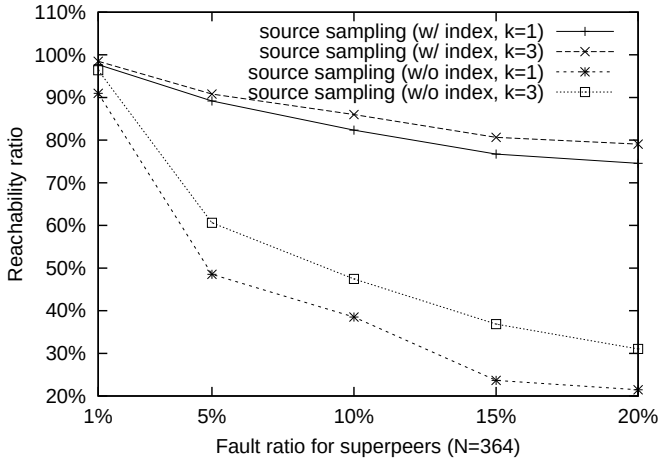


Figure 3.13: Fault resilience of LASPD in terms of routing reachability.

superpeers. In this section, we simulate the failures of superpeers and test our system fault resilience in terms of routing reachability. In the greedy routing algorithm (Definition 2), the destination peer is considered unreachable if the current peer cannot find any immediate neighbor or long-range contact, which is nearer to the target. The simulation considers the same setup as that in the previous section. The respective reachability for each case is recorded and plotted in Figure 3.13. As noted in the previous section, our approach doesn't depend on superpeers heavily, and thus, even if some of them fail, we can still provide the satisfied routing reachability, for example 75% reachability for $k = 1$ when 20% of superpeers fail. However, this is not the case for the superpeer-based approach: the reacheability falls to 38% for failure rate of 10% and further reduced to 21% when the failure rate is 20%.

Source Sampling versus Threshold Sampling and Destination Sampling

In Section 3.2.1, we have introduced two other evolutionary approaches to emerge small world network: the *Threshold Sampling (TS)* in [116] and the *Destination Sampling (DS)* in [118] (we follow the terminologies as defined in [118]). Their detailed algorithms can be described as follows: in TS, a random number T_{thresh} is picked up

from the range 1 to $dist(source, destination)$, i.e. the number of hops to the destination peer. If the destination peer cannot be reached in T_{thresh} hops, a shortcut is created from the source peer to the current peer with $dist = T_{thresh}$; while in DS, a probability p is predefined, e.g. $p = 0.1$. For the routing of query, every peer on the path has a chance p to create a shortcut to the destination peer. In fact, TS best resembles our *Source Sampling (SS)* by allowing dynamic creation of shortcuts from source peer to other peers on the routing path. As a result, in this section we focus on the comparison of SS and TS; while leaving the discussion of SS and DS to the end of this section.

We have carried out simulations to compare the ability of SS and TS in dealing with network topology changes, such as in mobile environments. To be fair, we apply the two-dimensional grid network topology from [115] (the topology is used by both TS and DS in their simulation studies). The grid contains 100×100 peers, and each peer is assigned with the label $(x, y) : x, y \in 1, 2, ..., 100$. The distance between any two peers is thus measured by lattice distance $dist(v, u) = |v.x - u.x| + |v.y - u.y|$. For the link setup, each peer always has a directed link to every other peer within distance 1. Besides, each peer is allowed to have a long-range link to the peer with distance larger than 1, i.e. $k = 1$. There are three phases during the simulation runtime:

1. *Initialization:* We start up with the Kleinberg's small world model [115]; that is, all the peers are already augmented with a long-range link (the probability to set up the link between v and u is proportional to $[dist(v, u)]^{-2}$).

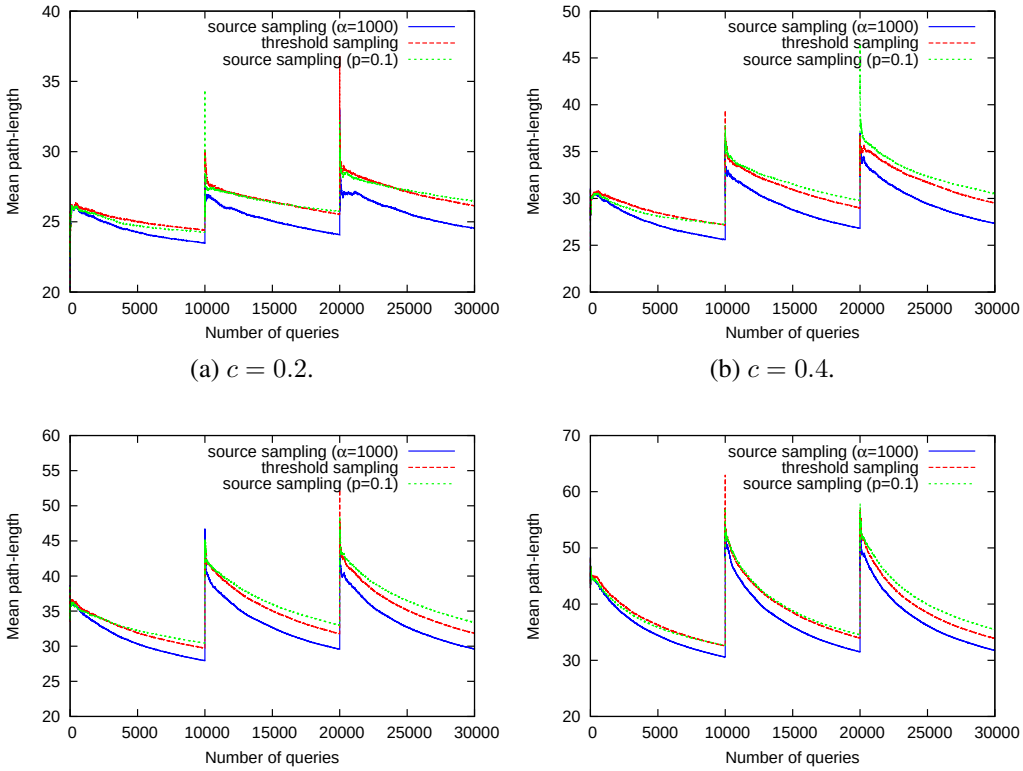
2. *Evolution:* For each round of query, two peers are randomly selected as the source and the destination. The greedy routing algorithm is then applied to do query routing. In SS, for each peer on the route, an independent sampling process is carried out to rewire a long-range link from the source peer to the current one. The probability p in SS is proportional to $[dist(v, u)]^{-2}$ and is normalized by the factor Z as defined in Equation 3.1. The constant parameter α is involved in the

calculation of p and is set to 1000 (we pick this value because it results in similar value of mean shortcut distance as in centralized sampling for the grid network). In TS, T_{thresh} is uniformly chosen from $[2, dist(v, u)]$, by assuming the distance from the source peer to the destination peer is known.

3. *Modification:* Starting from the first query, for every $10K$ queries (the number is chosen so that every peer has a chance to send a query before the next modification phase), the network topology is modified. The effect of the modification is determined by a constant parameter c; that is, every peer in the grid has a chance c to switch its position with another random peer. This is to simulate the peer movement in a real-life scenario. Once the two peers swapped their positions in the grid, whichever link targeting them is destroyed. However, the links coming out from themselves are still kept unless the remote peers targeted by these links also shift their positions.

Figure 3.14 plots the results for the cases $c = 0.2, 0.4, 0.6, 0.8$. There are a total of $30K$ queries generated. For each query, we ensure that the same set of queries are used for SS and TS respectively in a simulation run. The modification phase happens at $query\_num = 0, 10K, 20K$. To show the effect of network topology change, the mean path-length is reset for every $10K$ queries. We repeat the experiments 10 times with different sets of queries and topology modification behaviors. The averaged values are plotted in the figure. Note that to fully understand the characteristic of SS, we have also plotted the case for SS with uniform distribution, i.e. $p = 0.1$ for the sampling process (independent of the lattice distance on the grid network).

We have the following observations: as we start the network topology with Kleinberg's small world model, the topology modification occurring at $query\_num = 0$ have the same impact to both SS and TS; however, as time progresses, due to their different evolutionary behaviors, SS is more resilient to network change (e.g. peer movements)



(a) $c = 0.2$. (b) $c = 0.4$.

(c) $c = 0.6$. (d) $c = 0.8$.

Figure 3.14: Performance comparison of Source Sampling and Threshold Sampling in dealing with network topology changes.

for different c values. For the case of SS (uniform p), its performance is worse than both SS and TS. To have a further analysis, Table 3.1 and 3.2 have given more details about variations of number of long-range links and their mean shortcut distance for the three mechanisms (i.e. in the case $c = 0.6$). Further observations are provided as follows:

- As shown in Table 3.1, SS has more long-range links created in each evolution phase, and therefore it has more links preserved after each modification. This can be one of the reasons why it has better performance as compared to TS. In fact, a good estimation of the distance to the destination peer ($dist$) is critical for TS in practice: if $dist$ is too small (e.g. $dist = 5$), though more long-range links are allowed to be added in each evolution phase, its convergence speed is not improved

	$query\_num = 10K$	$20K$	$30K$
SS	1466/6733 (21.8%)	1406/6485 (21.7%)	1378/6449 (21.4%)
TS	1196/5506 (21.7%)	1129/5027 (22.5%)	1102/5009 (22%)
SS (uniform p)	1198/5369 (22.3%)	1054/4880 (21.6%)	992/4649 (21.3%)
TS ($dist = 5$)	1559/7120 (21.9%)	1545/6886 (22.4%)	1499/6881 (21.8%)
TS ($dist = 100$)	969/4359 (22,2%)	779/3651 (21.3%)	790/3600 (21.9%)

Table 3.1: Variation of number of long-range links (number after modification / number before modification) for the mechanisms shown in Figure 3.14c and 3.15.

	$query\_num = 0$	$10K$	$20K$	$30K$
SS	55/19	56/26	56/25	56/25
TS	55/19	60/45	60/44	60/44
SS (uniform p)	55/19	64/57	64/59	63/59

Table 3.2: Variation of mean shortcut distance of long-range links (distance after modification / distance before modification) for the mechanisms shown in Figure 3.14c.

due to high clustering effect; if $dist$ is too large (e.g. $dist = 100$), less long-range links are augmented, and the resulting convergence speed becomes slow. We have plotted the cases for TS ($dist = 5$) and TS ($dist = 100$) in Figure 3.15, and added their variations of number of long-range links in Table 3.1.

- As shown in Table 3.2, the network model constructed by SS best recovers the property of the original small world model in terms of mean shortcut distance, i.e. by comparing with the case of $query\_num = 0$. To further validate that SS outperforms TS in dealing with network topology changes, we have carried out the above simulations in a larger network (α for SS has been increased to 10000, as the same reason for $\alpha = 1000$). The results plotted in Figure 3.16 show that SS has better capabilities in recovering routing efficiency from topology changes.

With the same experimental setup, we have conducted simulations to compare SS and DS. As the authors in [118] have not specified any optimal value for p ($p = 0.1$ is used in their simulations) in DS, we have carried experiments with $p = 0.01$ and $p = 0.1$. Based on the value of p, the value of α in SS is calculated and set to 100 and

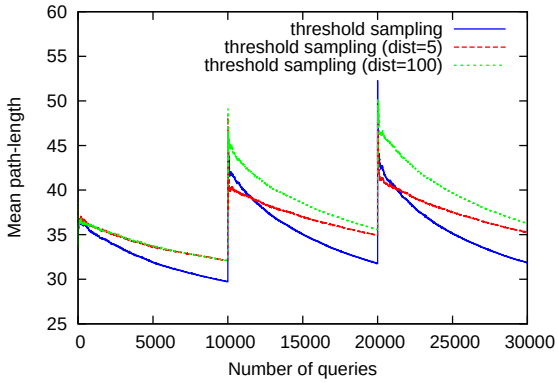


Figure 3.15: Performance comparison of Threshold Sampling with different $dist$ estimations in dealing with network topology changes ($c = 0.6$).

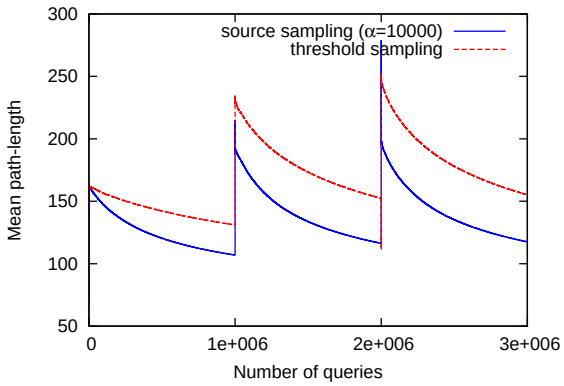
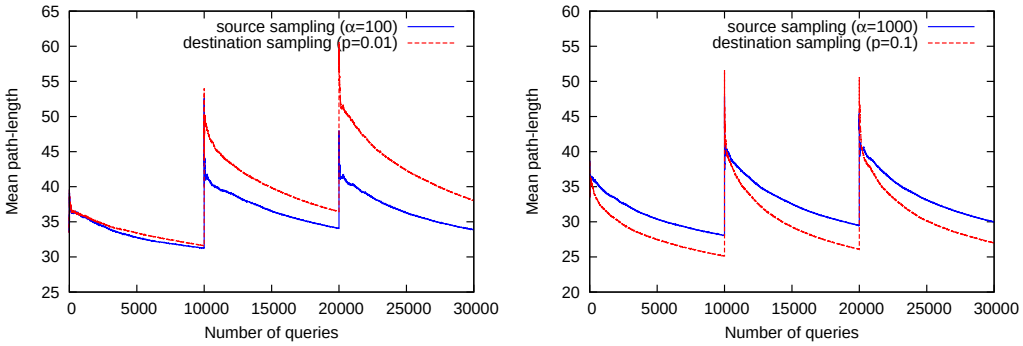


Figure 3.16: Performance comparison of Source Sampling and Threshold Sampling in 1000×1000 grid network ($c = 0.6$).

1000 respectively (we may think DS follows a uniform distribution for rewiring of long-range links). Figure 3.17 plots the results in 100×100 grid network. As observed, when p is relatively small as compared to the network diameter (i.e. the mean path-length), SS performs better than DS in coping with network topology changes. This is due to the fact that less long-range links are rewired in DS (average number of links rewired in each evolution phase is 3016). However, when p becomes larger, more long-range links are added (average number of links rewired in each evolution phase is 7561). As DS allows multiple long-range links to be rewired in routing of a single query (which is quite different from TS and SS), its performance convergence speed is faster. Nevertheless,



(a) $p = 0.01$ (DS). (b) $p = 0.1$ (DS).

Figure 3.17: Performance comparison of Source Sampling and Destination Sampling in dealing with network topology changes ($c = 0.6$).

the above results for DS are under the assumption that the destination peer is known in advance. Indeed, in our problem, because all the queries are routed based on the key hashed from the target service, the destination peer will not be known until it is discovered. Therefore, if DS must be applied, overhead to discover the destination peer has to be taken into account, such as by applying the reverse routing technique.

3.5 Summary

In this chapter, we described the design of a location-aware service provision and discovery platform (LASPD). The motivation is to have an effective and scalable platform to support large-scale service provision and discovery including those emerging in mobile ubiquitous computing. To achieve this, LASPD constructs a three-tier architecture for service provision: while the first tier provides classifications of areas of service administration, the second tier considers service providers which are not resource constrained in a structured peer-to-peer network. In both tiers, the Hilbert space filling curve is deployed to preserve the physical localities of service providers and to support area-based or distance-based service discovery. In the third tier, services hosted by mobile

devices are allowed to delegate to a proxy in the second tier. To minimize the network maintenance effort, LASPD leverages on the proposed Source Sampling mechanism to allow shortcut creation during the service discovery. Simulations have been carried out to study the routing behaviors of peer nodes in different tiers and the effect of long-range link indexing mechanism. The performance of Source Sampling is also compared with other similar approaches in dealing with network topology changes.

As demonstrated, by applying the proposed source sampling mechanism, it helps to achieve a navigable network in an autonomous manner. However, due to the nature of sampling, in practice an alternative approach — centralized sampling may be applied first to boost its convergence speed for routing performance. The effect of the long-range link indexing mechanism has been studied: less workload is imposed on superpeers and the system fault resilience is improved. As compared to threshold sampling, source sampling shows its resilience in dealing with network topology changes. While the current research focuses on the design and validation of a navigable and self-organized network in mobile environments, the various network optimization techniques (e.g. data replication, link caching) can be applied in practice. As our future work, we would like to further investigate on the network model of LASPD, such as the study on the effect of parameter α, the design of link replacement policy (i.e. when $k > 1$) and the development of load balancing strategy for long-range link indexes.

CHAPTER 4

ACE: A CONTEXT REALIZATION ENGINE FOR UBIQUITOUS APPLICATIONS WITH RUNTIME SUPPORT

This chapter presents a framework to facilitate the realization of context-related tasks and therein the context logic (defined as *application adaptation*, *constraint enforcement* and *context flow* in Section 1.5) is decoupled from the application layer, and its formulation is shifted to the application design time rather than during the implementation phase. This approach reduces the complexity of application implementation and further provides flexibility in handling requirement changes for evolving applications. The detailed designs of the *Application Context Model (ACM)* for capturing context logic specified by application developers and the *Application Context Engine (ACE)* for handling the

full life cycle — initialization, execution and termination — of each ACM instance are presented.

The rest of the chapter is organized as follows: Section 4.1 presents an application scenario to illustrate the design considerations and challenges for ubiquitous application development. Section 4.2 describes our proposed Application Context Model (ACM) that captures application contexts of a ubiquitous application. Section 4.3 presents the component design of the Application Context Engine (ACE) and its runtime support for context realization during the application execution. Section 4.4 shows a case study based on the illustrated application scenario to validate the proposed framework. Finally, Section 4.5 concludes the chapter.

4.1 Motivation: A ShoppingHelper Application

Alice loves to shop along the world famous shopping belts. On a recent business trip to Singapore, she is particularly interested in the goods and services provided on the Orchard Road. However, the tight schedule simply does not allow her to visit every shop for goodies; rather, she has to pick up those "most wanted". With the newly installed ShoppingHelper application on her Google Nexus, she does not need to waste time to search and review all the shops beforehand; instead, she could select and visit the interesting ones during her shopping time. The mobile application would smartly recommend those shops within her proximity and based on her shopping preferences. Shopping experiences shared by others are easily accessible even on the move to further influence her decision on the shops eventually to visit. Moreover, if any of her Singaporean friends is in the nearby shop, she will get notified with the name of the shop.

While shopping at the ION Center, Alice found that the goods and services provided by the mall are excellent and the stores inside are giving further discount to commemorate this opening occasion. She could not help but to share her impression of the newly opened center through her Nexus. Alice is given a couple of options to share her impression of the shops[1]*: (a) to store her data in an application server for ease of access by others or; (b) to store the data, if deemed private, in her smart phone for a full control over the data. She can share the information publicly or provide a friend list. All options could be configured with the touch of her finger tip.*

The above application illustrates the usage of Alice's contexts, e.g. her *location* (from the phone GPS) and *preferences* (from the data kept in the phone). These contexts are matched with the corresponding ones (e.g. *location* and *type*) of the shop services for the application to achieve context-awareness. Although the application scenario is trivial, it has typical features of ubiquitous applications so that we can use it to illustrate the following design considerations and challenges:

1. What are the contexts in the application? How could they be described in a machine-readable format? *(context modeling)*

2. How are the contexts gathered and retrieved for the application? *(context management and retrieval)*

3. How does the application leverage on the contexts and embed context-awareness into its application design? *(context realization)*

Available context modeling techniques vary from the simplest attribute-value approach to the most complicated ones such as ontology-based approach. For the detailed

[1]It is possible to distribute the data for storage in both the server and her mobile phone.

survey, interested readers may refer to [8]. As discussed in Section 1.2.4, ontology-based models are the most promising assets for context modeling in UbiComp. Nevertheless, existing ontology-based context models are mostly focusing on the definition of semantics for context data, including their types and sources. Therefore, the resulting model is mainly designed for the acquisition of context data. This enables the development of a common middleware (e.g. Solar, SOCAM and Coalition) for the management and retrieval of context data. But as mentioned in Section 1.3 and Section 2.2, these middlewares do not provide capabilities in dealing with higher level context-related tasks such as runtime constraint enforcement which are desirable from the application developers point of view. Consequently, developers have to explicitly implement the logic in the application code, for examples, calling the API to retrieve context data (e.g. shop's location and type), and enforcing constraints (e.g. nearby shops that match Alice's preferences) by using if/else statements. This results in a tight-binding model for application implementation. In case there are changes in the application requirements (e.g. in addition to the shop's location and type, its crowd level may also be considered in the shop recommendation process), application developers have to rewrite and redeploy the whole application. We argue that this issue is indeed caused by context modeling at a lower level: context logic embedded in the application such as context constraints and flows is not captured and reflected in the conventional context model of the middleware (Figure 4.1 left); thus, developers are not fully relieved from the considerations for the detailed process of context realization. We hereby propose a new context modeling approach considering the unsupported context logic and add a new conceptual layer, i.e. context realization layer (Figure 4.1 right), to help shift the formulation of context logic to the design time rather than during the implementation phase of the application. The design considerations for the illustrated framework architecture (Figure 4.1 right) include:

1. *What is the context logic in the application? How could it be formally represented?*

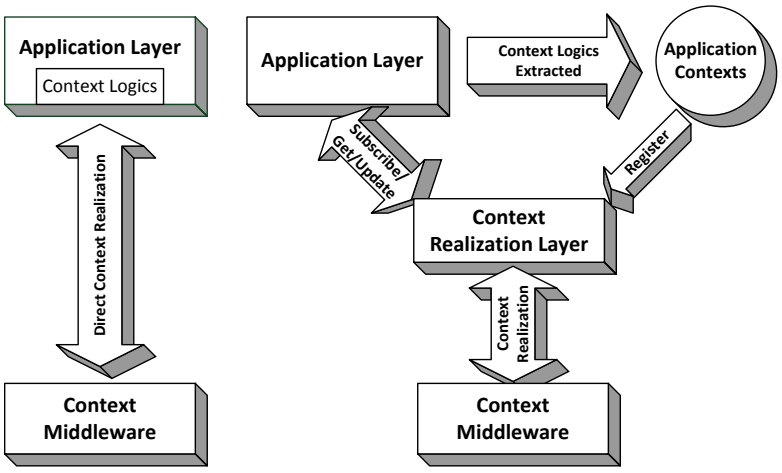


Figure 4.1: Comparison of context realization model for the conventional approach (left) and our approach (right).

An application context model is supposed to be developed to allow developers to specify their requirements over tasks such as for application triggering and for constraint checking, whereas some of these tasks may be carried out at application runtime. Such a model should be easy to learn and develop for application developers so that their design effort is minimized. In the taken approach, we have proposed an ontology-based application context model. The model follows the Entity-Relationship Modeling (ERM) style, which is intuitive and frequently adopted in the design of relational database. Its details will be presented in the next section (Section 4.2).

2. *How could the extracted application contexts be taken care of so that the tasks specified are completed as desired?* The context realization layer should be able to understand the application context model and help to fulfill the tasks required automatically. In the taken approach, we have developed an application context engine in the layer to cater the task of context realization. The design of the engine includes a set of functional components, such as for context model interpretation and for context data reasoning. Furthermore, it gives applications runtime support

for the context-related tasks as specified in the application context model. We will present its details in Section 4.3.

3. *How does the application layer interact with the context realization layer? Where should the application context engine be deployed?* To support context realization for ubiquitous applications, communications between the application layer and the context realization layer are necessary. For instance, after deriving the suitable shops for Alice, the application context engine should push the information of the shops to the ShoppingHelper application running on Alice's mobile device. In addition, some of the context data may be provisioned by the application itself, such as the experience providers' names for the shops interested by Alice. Therefore, APIs (i.e. register/subscribe/get/update as shown in Figure 4.1 right) have been defined to allow applications to successfully interact with the application context engine. (A more detailed implementation architecture for ubiquitous application development with ACE can be found in Figure 4.10). The deployment of the application context engine is yet another issue as it determines the scalability and efficiency of context realization. We will discuss this issue and present our ideas in Section 4.3.4.

4.2 ACM: Context Model for Application Contexts

We adopt a model-driven approach to develop our context model. The proposed context model, called *Application Context Model (ACM)*, is a conceptual graph model that defines a set of elements, including *ApplicationEntities*, *ContextEntities*, *ContextAttributes*, *ContextRelationships*, *StartingContexts*, and *EndingContexts*. The ApplicationEntities are entities involved in the application scenarios. They execute application logic such as message sending and task processing. The ContextEntities are abstractions

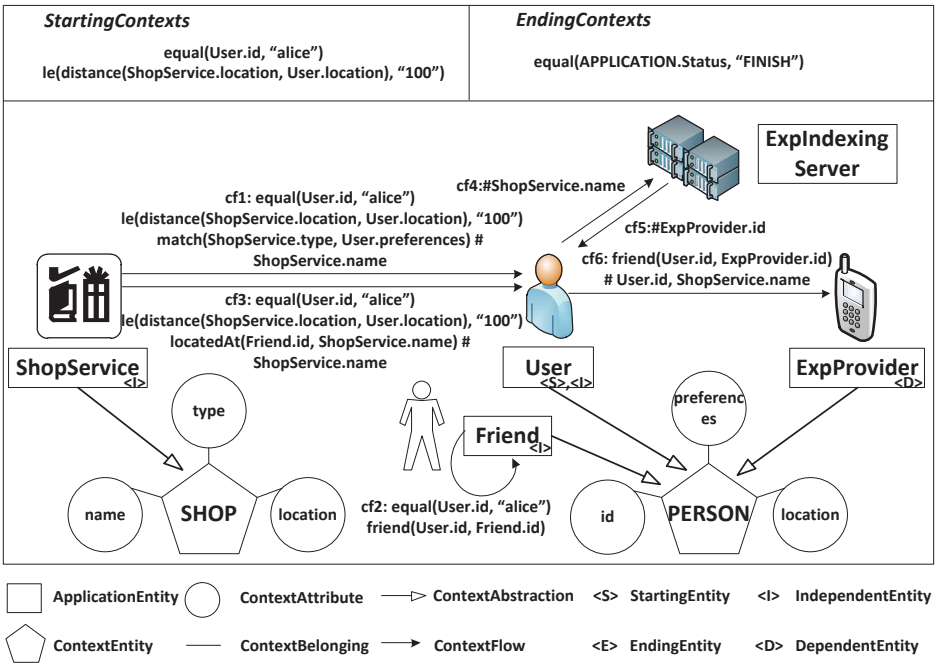


Figure 4.2: An example Application Context Model, constructed for the ShoppingHelper application in Section 4.1. Operator *le* means *less than or equal to*.

of the ApplicationEntities that act as virtual context sources of the application. We define them based on the ContextAttributes they possess, and a collection of ApplicationEntities having similar context attributes are abstracted to the same ContextEntity. The ContextRelationships define the types of context relations among the entities in the model. It is further divided into three types: *ContextBelongings* between ContextAttributes and ContextEntities; *ContextAbstractions* between ApplicationEntities and ContextEntities; and *ContextFlows* among ApplicationEntities. The last two elements StartingContexts and EndingContexts indicate the situations when the application should be triggered to get start or terminated to get stop. With regards to the processes of context realization, they refer to when the ACM should be instantiated or destroyed. They are specified by a set of constraints over ApplicationEntities, which are similar to those for ContextFlows.

The example ACM shown in Figure 4.2 illustrates the graphical notations used. This model has five ApplicationEntities represented by rectangles, namely “ShopService”,

“User”, “Friend”, “ExpProvider” and “ExpIndexing Server”, which are extracted from the ShoppingHelper application described in Section 4.1. Four of the ApplicationEntities are abstracted as ContextEntities shown in pentagons, namely “SHOP” and “PERSON”. “‘User”, “Friend” and “ExpProvider” are all abstracted to “PERSON” as they share the common ContextAttributes “id”, “location” and “preferences” (as shown in circles). The respective ContextRelationships are indicated using different types of edges. When describing the ContextAttribute of a particular ApplicationEntity, the operator “.” is used, e.g. “User.id”. To express constraints in ContextFlows, StartingContexts and EndingContexts, context rules are applied. In addition, high-level operators such as “distance()” and “match()” are supplied to help developers specify constraints in a more intuitive way. The detailed constraints could be either specified in the model directly or be referenced to a text file. In our current implementation, the ACM specification is ontology-based and marked up using W3C’s OWL language. The following illustrates part of the OWL specification for the illustrated ACM in Figure 4.2.

```
<!DOCTYPE rdf:RDF [
    <!ENTITY acm ‘http://www.comp.nus.edu.sg/acm/ApplicationContextModel#’>
]>

<rdf:RDF
    xmlns:rdf=“http://www.w3.org/1999/02/22-rdf-syntax-ns#”
    xmlns:acm=“http://www.comp.nus.edu.sg/acm/ApplicationContextModel#”
    xml:base=“http://www.comp.nus.edu.sg/acm/ApplicationContextModel#”>

    <acm:ACM rdf:ID=“ShoppingHelper”>
        <acm:hasApplicationEntity>
            <acm:ApplicationEntity rdf:ID=“ShopService”/>
        </acm:hasApplicationEntity>

        <acm:hasApplicationEntity>
            <acm:ApplicationEntity rdf:ID=“User”/>
        </acm:hasApplicationEntity>

        <acm:hasContextFlow>
```

```
            <acm:ContextFlow rdf:ID="cf1">
              <acm:fromApplicationEntity rdf:resource="#ShopService"/>
              <acm:toApplicationEntity rdf:resource="#User"/>
              <acm:hasConstraint>constraint1.txt</acm:hasConstraint>
              <acm:hasFlowData>ShopService.name</acm:hasFlowData>
            </acm:ContextFlow>
          </acm:hasContextFlow>
        </acm:ACM>
    </rdf:RDF>
```

The ACM reflects the context logic embedded in an application. For instance, the StartingContexts and EndingContexts are for application adaptation; constraints in ContextFlows are for constraint enforcement; and the edges between ApplicationEntities are for context flow. By separating them from the application logic, it has the flexibility in modification of any logic parameter without reimplementing and redeploying the application. For instance, in Figure 4.2 we could easily add another constraint (i.e. in "cf6") to the ACM for retrieving shopping experience from the ExpProvider; that is, only if the ExpProvider is a friend of Alice, then we retrieve the ExpProvider's data for Alice.

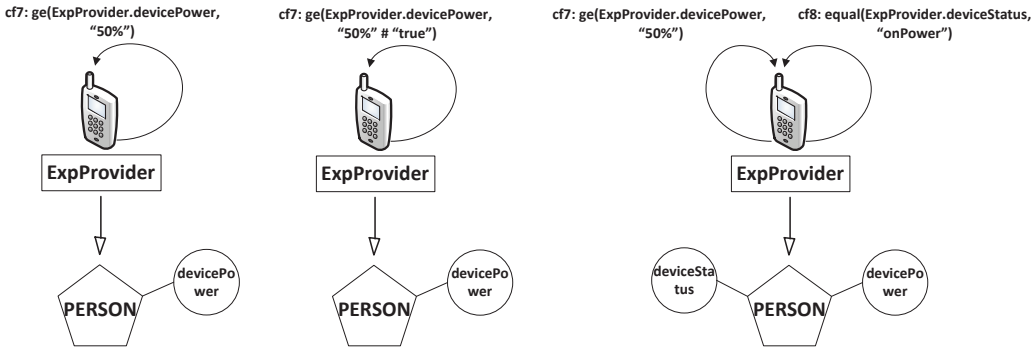
4.2.1 Context Flow Representation

Among the three types of ContextRelationships, the most important one is ContextFlows, as they determine what context data is relevant to which ApplicationEntity and under what situation the data can be derived from the relevant ApplicationEntity. They follow the *name:constraint#data* syntax as illustrated in Figure 4.2, where *name* is the ContextFlow name, *constraint* specifies the qualifications for participating ApplicationEntities or their ContextAttributes, *data* refers to the context data being passed in the form of either ContextAttributes or constant values. Both the constraint and data may be omitted, and for retrieving data, the ContextFlow name is used for reference. In the current framework design, there are two kinds of representations for ContextFlows: context

flow confined to a single entity and context flow between entities, to be discussed in the following sections.

Context Flow Confined to a Single Entity

The ContextFlow can be used to specify an ApplicationEntity. In its syntax, if *constraint* is specified but *data* is not, the ContextFlow is used for checking on the validity of the instances of the ApplicationEntity involved. Those unsatisfied instances are filtered. For instance, in Figure 4.3a, a ContextFlow is added on ExpProvider itself. It specifies the constraint that the contacted ExpProvider must have its "devicePower" greater than or equal to "50%". Note that to use the ContextAttribute "devicePower", it must be defined for the ContextEntity "PERSON". If both *constraint* and *data* are specified, the ContextFlow is for checking the status of ContextAttributes for the instances of the ApplicationEntity. For those satisfied instances, they are assigned with the constant value indicated in *data*, and this value is used by application logic to determine the next step of application execution. For instance, a requirement for context status checking is specified in Figure 4.3b. For those ExpProviders with "devicePower" greater than or equal to "50%", they are assigned with the constant value "true". The application logic running on the respective device can leverage on this value and decides whether it should send the experience information to requestors as for power saving consideration. The major difference between the above two representations is that the former representation filters those unsatisfied instances of the ApplicationEntity which are irrelevant to the application; while the latter only does the context status checking but without instance filtering. In addition, multiple ContextFlows are allowed on a single entity, we consider disjunctive relationships among them. For instance, the satisfied ExpProvider can be either with "devicePower" greater than or equal to "50%" or with "deviceStatus" being "onPower" as illustrated in Figure 4.3c.



(a) Context filtering. (b) Context checking. (c) Multiple ContextFlows.

Figure 4.3: Representation of the context flow on a single entity.

Context Flow between Entities

When ContextFlows are specified between two entities, *data* must be specified while *constraint* is optional. If *constraint* is specified, context filtering will be enforced on the instances of the related ApplicationEntities. The values of ContextAttributes of the satisfied instances (i.e. indicated in *data*) will then be available to the corresponding instances of the ApplicationEntity that is targeted by the ContextFlow. For instance, "cf1", "cf3" and "cf6" in Figure 4.2 illustrate such ContextFlows. Similar to the case of ContextFlow confined to a single ApplicationEntity, *data* can also be constant values so to be used by application logic, e.g. for context status checking. For instance, the ContextFlow "cf9" in Figure 4.4a checks whether there is any ShopService within $100m$ and matching Alice's preferences. If there are multiple ContextFlows specified between the two ApplicationEntities, the respective satisfied instances are merged by doing disjunction. Figure 4.4b illustrates the constraints ("cf1" and "cf3") that the ShopService should be nearby and either matching Alice's preferences or having Alice's friend visiting. Note that the *data* in these ContextFlows can also be different. Each *data* is identified and linked to the *name* of the specific ContextFlow, depending on the application requirements. For instance, in "cf3", the application may also want to pass "Friend.id" to "User".

(a) Context filtering. (b) Multiple ContextFlows.

Figure 4.4: Representation of the context flow between entities.

4.2.2 Context Constraint Specification

Each constraint in the ACM (i.e. *constraint* in ContextFlows) follows the rule syntax of Jena[2], which defines a rule by a list of body terms *(premises)* and a list of head terms *(conclusions)*. Only premises are required to be defined by developers, as the default conclusions are that the involved ApplicationEntities satisfy the constraint referred by *name* of the ContextFlow. Inside the premise, each term is represented either in a triple pattern *(node, node, node)* or with procedural primitive *builtin(node, ... node)*, where *node* may refer to: *uri-ref* (e.g. http://foo.com/eg), *prefix:localname* (e.g. rdf:type), <*uri-ref*> (e.g. <myscheme:myuri>), *?varname* (e.g. variable), *'a literal'* (e.g. a plain string literal), $'lex'\text{^^}typeURI$ (e.g. xsd:*), and *number* (e.g. 42). There are many built-in procedural primitives provided by Jena, such as those listed in Table 4.1, on which developers may leverage to do complex computation and reasoning tasks.

Due to its succinct syntax, Jena rules are considered as the easiest ones to read and write as compared to that in other rule-based inference engines [121]. However, the major limitation of Jena rule syntax is that it does not support nested functions. Consequently, temporary variables must be used to hold the returned values of functions, which may result in large-size premise specification and downgrade the rule readability. We therefore extend[3] the rule syntax to allow higher level constraint specifications:

[2]http://jena.sourceforge.net/inference/.

[3]The syntax extension we currently made is not on the source code of Jena directly; rather we have created a customized interpreter on top of Jena. The interpreter allows features such as nested functions and new operators (i.e. those based on Jena primitive operators) to be interpreted.

Builtin	Operations
isLiteral(?x) notLiteral(?x) isFunctor(?x) notFunctor(?x) isBNode(?x) notBNode(?x)	Test whether the single argument is or is not a literal, a functor-valued literal or a blank-node, respectively.
equal(?x,?y) notEqual(?x,?y)	Test if x=y (or x != y). The equality test is semantic equality so that, for example, the xsd:int 1 and the xsd:decimal 1 would test equal.
lessThan(?x, ?y), greaterThan(?x, ?y) le(?x, ?y), ge(?x, ?y)	Test if x is $<$, $>$, $<=$ or $>=$ y. Only passes if both x and y are numbers or time instants (can be integer or floating point or XSDDateTime).
sum(?a, ?b, ?c) addOne(?a, ?c) difference(?a, ?b, ?c) min(?a, ?b, ?c) max(?a, ?b, ?c) product(?a, ?b, ?c) quotient(?a, ?b, ?c)	Sets c to be (a+b), (a+1) (a-b), min(a,b), max(a,b), (a*b), (a/b). Note that these do not run backwards, if in sum a and c are bound and b is unbound then the test will fail rather than bind b to (c-a). This could be fixed.

Table 4.1: Part of the commonly used builtin operators in the Jena framework.

- Nested functions are allowed, as shown in "cf1" of Figure 4.2.
- Variable bindings are not necessary for ApplicationEntities and their ContextAttributes. For instance, "User" in "User.id" and "User.location" in "cf1" of Figure 4.2 are bound to the same variable by default.
- High-level operators can be customized and reused among different applications, such as "distance()" and "friend()".

To support the above features, additional rule parser is necessary for the ACM, which we will introduce it in Section 4.3. Alternatively, developers may follow the default Jena syntax in describing context constraints.

4.3 ACE: Application Independent Engine for Application Context Realization

To automate the process of context realization for ubiquitous applications, we have developed an application independent engine, named as *Application Context Engine*

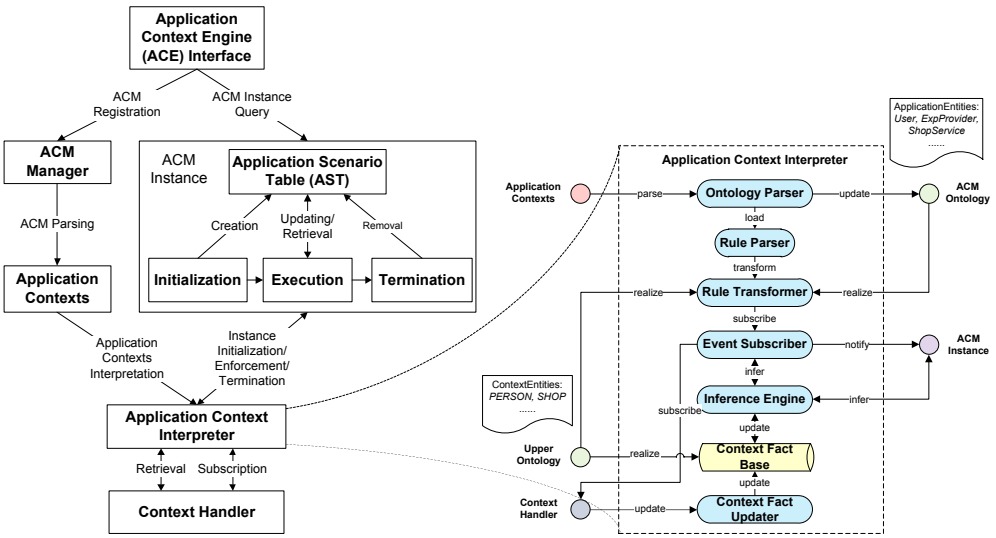


Figure 4.5: Functional components of the ACE and the Application Context Interpreter.

(ACE), in the context realization layer (Figure 4.1 right) to process their ACMs. More specifically, the engine is able to automatically trigger the application with its registered ACM and provides runtime support for each ACM instance. The functional components of ACE and their relationships are shown in Figure 4.5.

At first, the ACM specified by the application developer is registered to the ACE through the *ACE Interface*; thereafter, the *ACM Manager* extracts the *Application Contexts* (i.e. all the ACM elements) and passes them to the *Application Context Interpreter*, which is in charge of constructing the ACM specific ontology as well as parsing and transforming rules in the StartingContexts, EndingContexts and ContextFlows of the ACM. To enable automatic contextual events triggering and termination of each *ACM Instance*, the ContextAttributes involved in the StartingContexts and EndingContexts are subscribed through the *Context Handler* that in turn interacts with an underlying context framework to get the related context facts updated. When an ACM Instance is created, it is initialized with an *Application Scenario Table (AST)*, which is used to track all the relevant instances of ApplicationEntities involved in all possible application scenarios. The table is also updated along with the execution of the application instance.

4.3.1 Application Context Interpreter

The Application Context Interpreter with its functional components is shown in Figure 4.5. Once the *Application Contexts* are passed, its *Ontology Parser* constructs the specific *ACM Ontology*. For instance, the following OWL concepts about the ApplicationEntities in Figure 4.2 are updated.

```
<owl:Class rdf:ID="ShopService">
    <rdfs:subClassOf rdf:resource="&acm;ApplicationEntity"/>
</owl:Class>
<owl:Class rdf:ID="User">
    <rdfs:subClassOf rdf:resource="&acm;ApplicationEntity"/>
</owl:Class>
<owl:Class rdf:ID="Friend">
    <rdfs:subClassOf rdf:resource="&acm;ApplicationEntity"/>
</owl:Class>
<owl:Class rdf:ID="ExpProvider">
    <rdfs:subClassOf rdf:resource="&acm;ApplicationEntity"/>
</owl:Class>
```

As mentioned in Section 4.2.2, for the detailed constraint specification, developers may use the default rule syntax from Jena. Alternatively, they could use our extended version with high-level operators such as "distance()" and "friend()". The *Rule Parser* is responsible for converting the specification to the terms and those primitive operators supported by the *Inference Engine*, e.g. Jena.[4] The conversion is a recursive process that constructs the *parse tree* for a specific constraint, whereas each tree node is represented by an *operator* (e.g. "equal()") and a list of *arguments* (e.g. "User.id"). After the tree is constructed, the *Rule Parser* transverses it in a depth-first manner. Each tree node is interpreted in the way: if the operator is not built-in (i.e. non-primitive), the specification for the tree node is passed to the specific operator converter, and a binding variable is

[4]Note that in the current implementation of Application Context Interpreter, the high-level operators are composed from the primitive ones supported by the Inference Engine, e.g. Jena. Therefore, the types of queries supported by the ACE are restricted by the underlying Inference Engine.

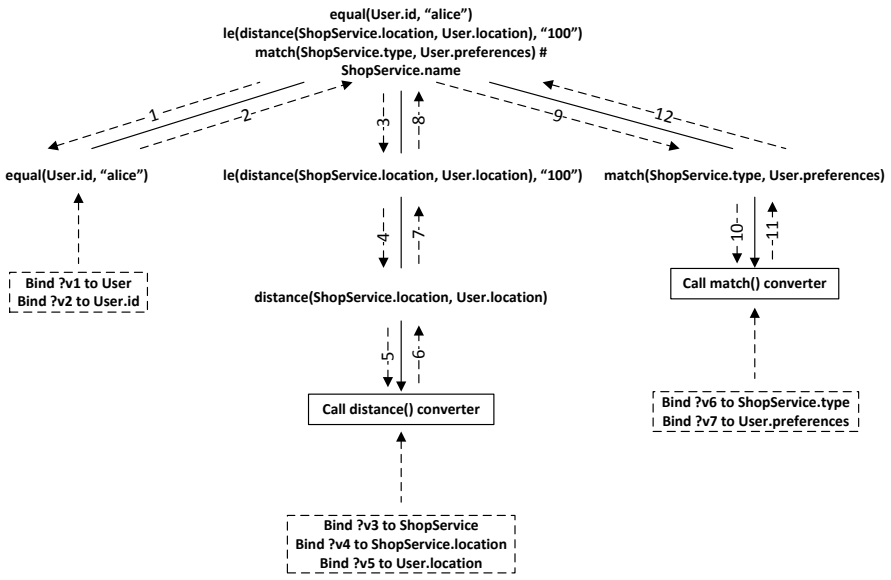


Figure 4.6: Construction of the parse tree for the constraint in “cf1” of Figure 4.2.

returned for the function if it is a nested function; if the ApplicationEntity or ContextAttribute is not bound, a variable is assigned to bind to it, and in the following transversal of the tree the occurrence of the ApplicationEntity or ContextAttribute will be replaced by the variable. Figure 4.6 illustrates the parse tree for constraint in “cf1” of Figure 4.2, and the following example shows the conversion for the constraint “friend(User.id, ExpProvider.id)” to the premises in Jena syntax.

friend(User.id, ExpProvider.id) → (?v1 rdf:type acm:User)
(?v2 rdf:type acm:User.Friend.ID)
(?v1 acm:hasContextAttribute ?v2)
(?v2 acm:hasContextValueS ?v3)
(?v4 rdf:type acm:ExpProvider)
(?v5 acm:type acm:ExpProvider.ID)
(?v4 acm:hasContextAttribute ?v5)
(?v5 acm:hasContextValueS ?v6)
equal(?v3 ?v6)

After the rule parsing, the *Rule Transformer* marks up the rules to satisfy different needs: if the rules are for constraints in ContextFlows, they are marked up by using the specific *ACM Ontology* (e.g. acm:User) and will be enforced by the *ACM Instance* at

application runtime (Section 4.3.3); otherwise, if the rules are for StartingContexts and EndingContexts, they are marked up by using the *Upper Ontology*[5] for ContextEntities (e.g. acm:PERSON) and subscribed to the *Event Subscriber*. The following illustrates the ontology about ContextEntity "PERSON".

```
<owl:Class rdf:ID="PERSON">
    <rdfs:subClassOf rdf:resource="&acm;ContextEntity"/>
</owl:Class>
<owl:Class rdf:ID="PERSON.Name">
    <rdfs:subClassOf rdf:resource="&acm;ContextAttribute"/>
</owl:Class>
<owl:Class rdf:ID="PERSON.Preferences">
    <rdfs:subClassOf rdf:resource="&acm;ContextAttribute"/>
</owl:Class>
```

The *Event Subscriber* fulfills two tasks: (i) it subscribes via the *Context Handler* the ContextEntities required to be updated, e.g. PERSON, SHOP;(ii) it interacts with the *Inference Engine* to check the occurrence of an event. Depending on the event type, if the StartingContexts for an ACM are satisfied, the corresponding *ACM Instance* is initialized, and meanwhile, the notifications are sent to those ApplicationEntities labeled with *StartingEntity* as illustrated in Figure 4.2; otherwise, if the EndingContexts are satisfied, then the corresponding ACM Instance is terminated and ApplicationEntities labeled with *EndingEntity* are notified. These events are derived by the *Inference Engine* that performs the computation and reasoning tasks as requested by both the *Event Subscriber* and the *ACM Instance*.

At this stage, there are two *Inference Engines* deployed in the Application Context Interpreter, i.e. Jena and Jess[6]. Both of them are well-known engines for rule-based reasoning. Jena, as mentioned before, it has succinct syntax for rule specification and

[5]The upper ontology (top-level ontology, or foundation ontology) is an ontology which describes very general concepts that are the same across all knowledge domains. In the context of the ACE framework, it describes ContextEntities that are shared across different ubiquitous applications.

[6]http://www.jessrules.com/.

provides native support for OWL ontology language; Jess, on the other hand, has no direct ontology support, but it is small, light and one of the fastest rule engines available. It adopts an enhanced version of the Rete algorithm [122] to process rules, and the intermediate reasoning results are cached to improve reasoning efficiency. We will discuss and compare both deployments in the later case study (Section 4.4).

At the bottom of the Application Context Interpreter is the *Context Fact Base* together with the *Context Fact Updater*. The *Context Fact Base* consists of a set of context facts (i.e. information about a ContextEntity and its ContextAttributes), wherein each of them is marked up by the *Upper Ontology* (i.e. for ContextEntities such as "PERSON" and "SHOP"). The *Context Updater* is in charge of feeding fact data to the *Inference Engine* through the *Context Fact Base*. Depending on the type of the engine deployed, it updates the data in different formats, e.g. triple for Jena and template record for Jess. The Context Updater is invoked by the *Context Handler* that gets the actual data by either subscribing with or retrieving directly from an underlying context acquisition framework, such as a middleware like Coalition. The following statements illustrate the context facts related to Alice's name and preferences in the *Context Fact Base*:

(acm:alice rdf:type acm:PERSON)
(acm:alice.PERSON.Name rdf:type acm:PERSON.Name)
(acm:alice.PERSON.Name acm:hasContextValueS "$Alice$"^^xsd:string)
(acm:alice.PERSON.Preferences rdf:type acm:PERSON.Preferences)
(acm:alice.PERSON.Preferences acm:hasContextValueS "$clothing, shoes, jewellery$"^^xsd:string)
(acm:alice acm:hasContextAttribute acm:alice.PERSON.Name)
(acm:alice acm:hasContextAttribute acm:alice.PERSON.Preferences)

4.3.2 Context Fact Management

As the management of context data is extremely important for efficient data processing as well as context reasoning, the following two schemes are applied to help improve memory usage and to reduce the size of the data for inference:

- *Selective Context Attribute Update:* In reality, there are numerous context facts to be updated in the Context Fact Base. However, not all the information (i.e. ContextAttributes for a particular ApplicationEntity) is useful. For instance, to infer those shops within a certain proximity, only the context fact regarding the location of the shop should be updated. As mentioned in the previous section, the Event Subscriber in the Application Context Interpreter is able to subscribe to the Context Handler only for those attributes required by the ACM. Therefore, the context facts will be selectively updated.

- *Two-Level Application Context Management:* In addition to the Context Fact Base in Figure 4.5, each ACM Instance has its own *Application Context Base* that stores the context facts relevant to its application usage (as shown in Figure 4.7). The Application Context Base is marked up by the specific ACM Ontology (i.e. for ApplicationEntities such as "User" and "ExpProvider"). To utilize fact data from the Context Fact Base, an ontology update is required for each Application Context Base; that is, to associate the ontology of a specific ApplicationEntity (e.g. "User") with that of a ContextEntity (e.g. "PERSON"). For instance, to allow Alice's context facts in the Context Fact Base (e.g. her name and preferences as illustrated in the previous example of statements) to be used by the ShoppingHelper ACM Instance, a triple statement *(acm:alice rdf:type acm:User)* has to be inserted to the Application Context Base, so that Alice performs the role of "User" in the ShoppingHelper application and her data is available for reasoning on the constraints related to "User". With such a scheme, it allows the isolation of application context management, and the same context facts in the Context Fact Base can be reused among all the ACM Instances when they are needed.

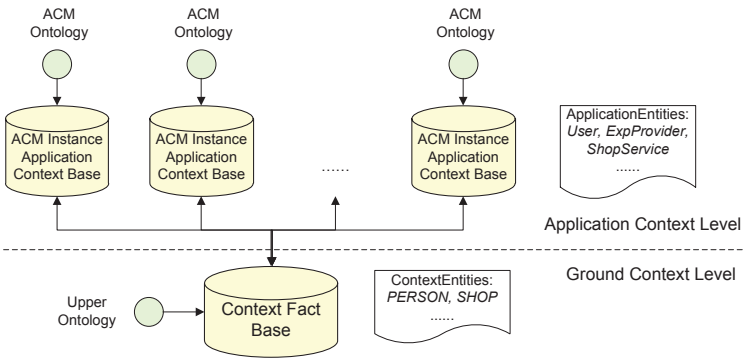


Figure 4.7: Context fact management model in the ACE.

4.3.3 Application Context Runtime Support

ACM Instance Initialization

Once an ACM Instance is created, the *Application Scenario Table (AST)* is initialized. The AST is used to keep track of all the possible ApplicationEntities and their context facts at application runtime. At first, only ApplicationEntities labeled with *IndependentEntity* (as illustrated in Figure 4.2) are instantiated to fill the AST. ApplicationEntities labeled with *DependentEntity* are not instantiated right now, as their attributes can be instantiated during application execution and certain attributes may be derived from the application logic, e.g. ExpProvider.id from the matching process by ExpIndexingServer. We differentiate these two types of ApplicationEntities so to improve the speed of context realization and the efficiency of memory usage. They are automatically differentiated by checking whether there is any ContextAttribute of the ApplicationEntity that is not derived from the ApplicationEntity itself but is involved in any ContextFlow. For instance, in Figure 4.2, ShopService, User and Friend are IndependentEntities; ExpIndexingServer is not of either type since it is not abstracted to any ContextEntity; ExpProvider is a DependentEntity as its ContextAttribute — ExpProvider.id is not derived by itself. Once all the possible ApplicationEntities are instantiated and the corresponding context facts are stored in the Application Context Base, the available rules from the ACM Ontol-

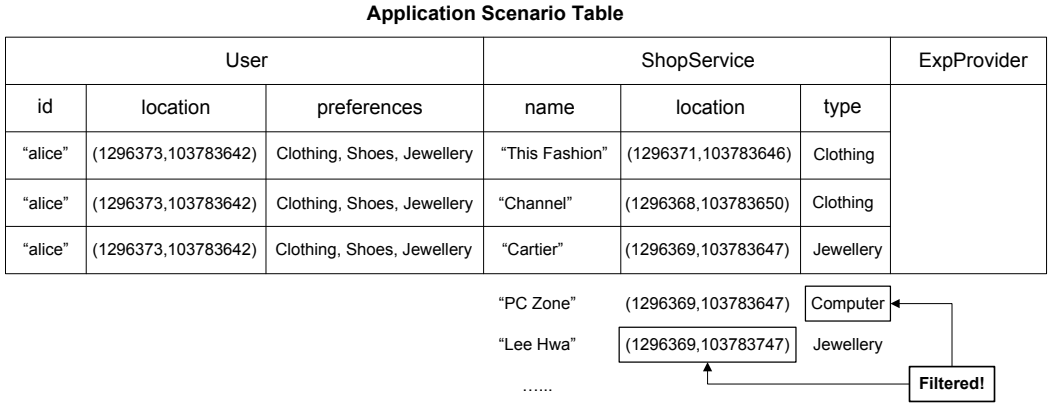


Figure 4.8: Initialization of the AST due to ContextFlow "cf1" in Figure 4.2 (the values shown in the AST are represented without ontology markups for better clarity).

ogy (i.e. transformed by the Rule Transformer in Figure 4.5) are applied on these facts. Only those satisfied application scenarios are filled in the AST. As illustrated in Figure 4.8, the constraint that the shops are within $100m$ from Alice and matching her preferences is enforced during the initialization of the AST. Note that the ContextAttributes of ExpProvider are not instantiated at this moment as ExpProvider is a DependentEntity.

ACM Instance Execution

Once ApplicationEntities labeled with *StartingEntity* are notified by the ACE, the application logic start execution. The latter caters to application-level processes such as data storage and communication, and may be distributed to several ApplicationEntities. With the ACE, application logic no longer need to include context logic such as context constraints into their consideration. For instance, when Alice's smart phone is notified, it could simply call "retrieve('ShopService.name')" from the ACE Interface (which can be deployed on a remote server) to retrieve the names of the nearby shops that match her interests (Note that the type of data to be retrieved is restricted by the *data* indicated in the ContextFlow that is incoming to the ApplicationEntity). The semantics of "nearby shops that match interests" are realized by the ACE but not embedded in the

Scenario Update Table

ShopService	ExpProvider
name	id
"This Fashion"	"John Smith"
"This Fashion"	"Kate Hudson"
"Channel"	"John Smith"
"Channel"	"John Smith"

⋈ **ShopService .name**

Application Scenario Table

User			ShopService		
id	location	preferences	name	location	type
"alice"	…...	…...	"This Fashion"	…...	…...
"alice"	…...	…...	"Channel"	…...	…...
"alice"	…...	…...	"Cartier"	…...	…...

Figure 4.9: Update of the AST due to ContextFlow "cf6" in Figure 4.2 (the left table is constructed after filtering ExpProviders that are not friends of Alice).

application logic. The same thing occurs when Alice wishes to get experience data from ExpProviders through her smart phone, which calls "retrieve('ExpProvider.id')" from the ACE Interface, with the constraint that the provider must be a friend of Alice is automatically enforced in the returning results. To achieve this, (i) each AST is identified with a unique ID that is assigned during the initialization of the corresponding ACM Instance. The application logic should know which AST to interact with during their executions; hence, the ID of the AST is provided as the additional parameter in all the interactions with the ACE Interface. (ii) The AST should be updated along with the execution of application logic so that other ApplicationEntities could work with the latest application scenarios. For instance, once ExpIndexingServer gets ShopService.name from User, it updates the AST by invoking "update(*shop_service*, *exp_provider*)" to the ACE Interface, where *shop_service* may refer to "Channel" and *exp_provider* refers to the person who has experience on the shop service, e.g. "John Smith". However, due to the constraint (i.e. ExpProvider must be a friend of Alice) indicated in the ACM, the ACE will filter those irrelevant ExpProviders. The IDs of the remaining ExpProviders are then filled in the AST by doing the JOIN operation based on the matched shops (Figure 4.9). By doing so, the ShopService.name is provided as the right parameter to the right ExpProvider for each application scenario in the AST.

At application runtime, the application contexts may change frequently. For instance,

when Alice moves around, her location is changing continuously. A good design of the ShoppingHelper application should consider this factor and always reflects the most up-to-date information (e.g. shops nearby) to Alice. At this moment, the support of "continually changing context" in ACE is done at two levels: at the inference engine level and at the ACM instance level. As mentioned before, the Context Handler in the Application Context Interpreter (Figure 4.5) will always get the latest context data from the underlying context acquisition framework. By default, the newly updated data will be immediately inserted into the inference engines of all the registered ACM instances (referring to the Selective Context Attribute Update mechanism as discussed in Section 4.3.2). Depending on whether the inference engine supports incremental reasoning[7], the time to re-evaluate the rules and update the respective AST can be quite different. We select Jena and Jess in our prototype implementation as both of them support incremental reasoning on additions (Jess also supports on deletions if "logical" keyword is used). To further reduce the overhead in handling continually changing context at the inference engine level, we have also considered the "updating frequency of context data" at the ACM instance level. The motivation is simple: in practice the demand of updating context data varies for each application. For instance, to achieve the best outcome in terms of user experience and reasoning efficiency, the ShoppingHelper may be suggested to have $1min$ updating frequency of Alice's location to its ACM instance. The ACM allows this property to be defined for each ContextAttribute, i.e. "updatingFrequency". With this attribute defined, e.g. for Alice's location, the inference engine in the ACM instance will only be updated by the Context Updater with the specified frequency. This approach effectively lessens the workload on the inference engine in handling continually changing context, and it also offers flexibility for developers in designing their applications.

[7]Incremental reasoning means the ability of the inference engine to process updates (additions or deletions) applied to an ontology without having to perform all the reasoning steps from scratch.

ACM Instance Termination

The termination of an ACM Instance can be explicitly invoked by the application, so that the "APPLICATION.Status" becomes "FINISH". Alternatively, it can be terminated by indicating the EndingContexts in the ACM. Similar to the StartingContexts, they are subscribed by the Event Subscriber in the Application Context Interpreter. Once the conditions are satisfied, the EndingEntities are notified, and the ACM Instance together with its ACM Ontology, Application Context Base and AST are removed from the ACE.

4.3.4 ACE Deployment

As mentioned before, the developed ubiquitous application is executed by different ApplicationEntities, including devices such as Alice and ExpProvider's smart phones. Nevertheless, the proposed ACE is conceptually designed as a layer between ubiquitous application and the underlying context management framework; hence, it can be deployed in a flexible manner: if there is a management system (e.g. a middleware such as SOCAM and Coalition) managing large-scale context sources, the ACE can be deployed on top of it and be provisioned as part of the middleware service for different applications to interact with. Alternatively, for small-scale applications which usually have their own context management schemes (e.g. a relational database hosted by one of the ApplicationEntities), the ACE can be co-hosted in the computer system hosting the corresponding ApplicationEntity[8]. For instance, the host of the ExpIndexingServer is such a suitable place to co-host the ACE in the illustrated ShoppingHelper application. Nevertheless, in both cases, the underlying context management framework have to support the retrieval and subscription tasks of the Context Handler in the Application Context Interpreter. More specifically, if the management framework already has an ontology-based

[8]Mobile portable devices are not recommended for co-hosting the ACE due to their memory and battery constraints.

context model for context data, the Upper Ontology may directly copy the concepts and relationships defined in the model, so there is no need to do ontology mapping in the retrieval and subscription processes; otherwise, a mapping is required, e.g. by mapping of ContextAttributes of ContextEntities in the Upper Ontology to column attributes in the database.

However, the current deployment of ACE for context reasoning is still centralized: a single machine is supposed to carry out reasoning tasks for all the applications, with multiple instances of the reasoning engine (e.g. Jena and Jess) created. This approach would potentially result in scalability issue when there is a large number of applications to support or there is a lot of context data to process. For instance, if shops within $1km$ are considered as "nearby" in the ShoppingHelper application, then there could be thousands of shops to filter for Alice. The problem would become more obvious when the application supports recommendations for multiple users[9]. In order to overcome this limitation, it is possible to distribute the application-dependent reasoning tasks to different machines so to remove the hardware constraints, e.g. CPU and RAM. Nevertheless, we believe a more scalable way is to further divide the reasoning task into smaller sub-tasks and allow each sub-task to be handled by a separate ACE. The decentralized deployment approach for the ACE will be further discussed in Section 6.2 as our future work.

4.4 A Case Study

As a proof of concept, we present the use of our context realization framework for the development of the exemplar ShoppingHelper application. The middleware — Coalition [14] is leveraged on to support the retrieval of context facts and subscription of con-

[9]With the specific user constraints such as "equal(User.id, "alice")" removed from the ACM (Figure 4.2), a single ACM instance can be used to support shop recommendations for multiple users. In Section 4.4, this scenario will be further discussed and used as one of the test cases to validate the inference support from ACE.

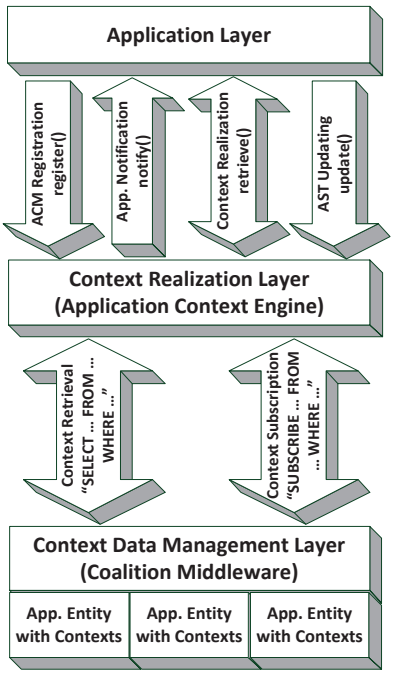


Figure 4.10: Software implementation architecture for ubiquitous application development with ACE.

textual events.[10] Figure 4.10 presents the detailed software implementation architecture (as compared to Figure 4.1). The application layer considers application logic such as those deployed on application servers or mobile devices. It interacts with the context realization layer through a set of APIs provided by the ACE to get its required context logic realized. ApplicationEntities with contexts are managed by the context middleware Coalition, whose functions are invoked by the Context Handler in the ACE to get ApplicationEntities' context data updated.

Figure 4.11 shows the sequence diagram of interactions between system components and users in the ShoppingHelper application which was implemented without the use of ACE. Figure 4.12 presents a similar sequence diagram of interactions for the same application implemented and deployed with the use of ACE with the support of Coalition

[10]The current ACE framework is built on top of Coalition: the *Context Handler* component in the Application Context Interpreter (Figure 4.5) communicates with the middleware services for the acquisition and preliminary processing of context data. A detailed architecture design of Coalition and its middleware services will be introduced in Section 5.5.

middleware (the interactions between ACE and Coalition are omitted for better clarity of the diagram). Note that in both diagrams, User and ExpProvider components are deployed on mobile devices; while the rest components are running on desktop computers. By referring the two figures, we can summarize the following advantages for developing ubiquitous application with our ACE context realization framework:

- The triggering of the application is no longer initialized by the application itself, i.e. by constantly checking the condition (the Loop structure in Figure 4.11); rather it can be notified by the ACE by registering its ACM. This approach is useful in mobile computing where power consumption is a major concern for applications running on mobile devices;

- The context logic embedded in the application is taken care of by the ACE but not the application logic running on different ApplicationEntities. This improves processing efficiency and saves power consumption;

- Once the context logic is extracted from the application layer, it is flexible to modify them. For instance, in addition to the proximity and type constraints, the ACM can be easily modified to add in the crowd level consideration (i.e. if the shop is too crowd, then it would not be appropriate for her tight schedule). The new ACM can be uploaded to the ACE and the modification immediately takes effect for the next instance initialization; while the application running on Alice's mobile device is not affected, and the whole updating process is transparent to the ubiquitous application users.

Indeed, as mentioned in Section 4.2, our ACE framework follows the Model-Driven Approach (MDA) for the realization of context logic required by the application. The MDA has recently become popular in both academia and industry as a way to handle the increasing complexity of modern software and it offers advantages in terms of chiefly,

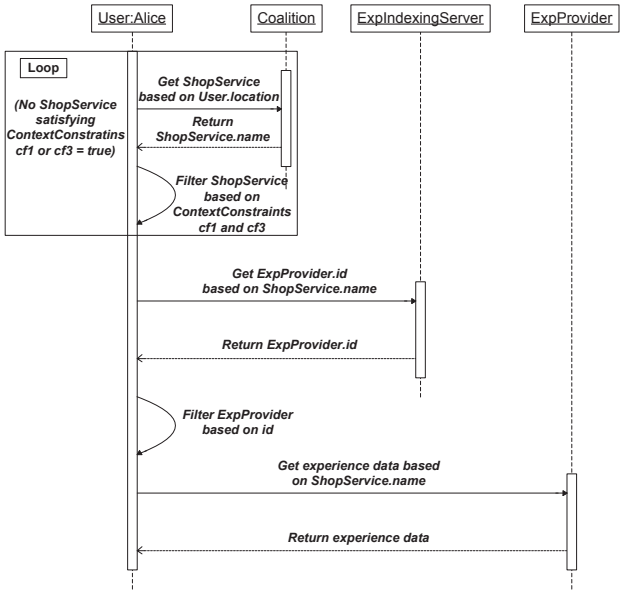


Figure 4.11: Sequence diagram for ShoppingHelper application without ACE. The context logic such as context retrieval and filtering is all realized in the application logic running on Alice's smart phone.

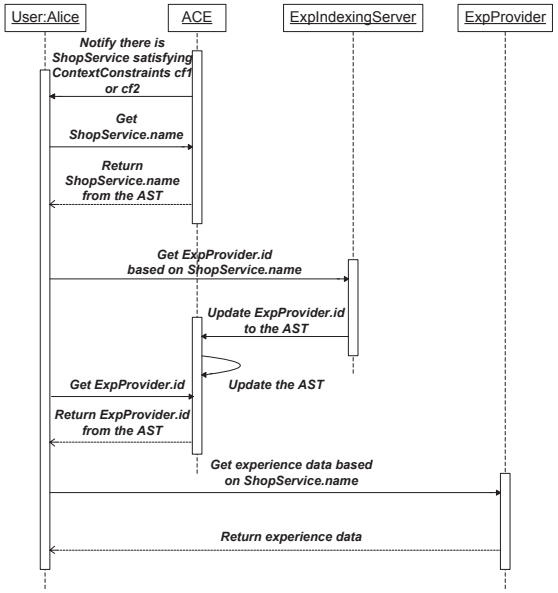


Figure 4.12: Sequence diagram for ShoppingHelper application with ACE. The context logic is separated from the application logic running on Alice's smart phone.

gains in productivity, portability, maintainability and interoperability [123]. For instance, during the design phase of ShoppingHelper, the various entities and relationships can be abstracted and modeled. More specifically, the context tasks are specified in an intuitive manner, e.g. under what situations there will be context data passing between the two entities. This results in a much cleaner semantics for application design, especially on context-related tasks. In the actual implementation of ShoppingHelper, the defined tasks are catered by ACE, and therefore the programming effort on context-related tasks can be minimized: after utilizing ACE for ContextFlow "cf1" in Figure 4.2, it results in 26 line code reduction in the source file (Note that the code reduction is task dependent). Lastly, in the future, if there is any updates or modifications on the design of context logic (e.g. adding the crowd level consideration), we can easily fulfill the requirement at the model level, but without changing the implementation code, which saves a lot of effort and time. However, there are still issues related to MDA which require further investigation. For instance, the feature of automatic code generation in the MDA helps to improve productivity. But the extra effort required to develop the model, along with the possible need to make manual modifications, would appear to have a negative effect on productivity. Similarly, in the ACE framework, the learning and construction of the ACM require efforts from application developers. They have to understand the syntax of ACM such as for specification of rules. To minimize such efforts, We are currently in the process of developing an *ACM modeler* that offers a Graphical User Interface (GUI)to allow application developers to specify the ACM by using graphical notations, and the modeler is capable to transform them to ontology languages automatically.

Based on the case study, we have also measured the overhead occurred during the ACM registration, which mainly involves ontology and rule parsing as mentioned for the Application Context Interpreter. Figure 4.13 plots the processing time over the ACM specifications with different size. Note that the specification for the case scenario as

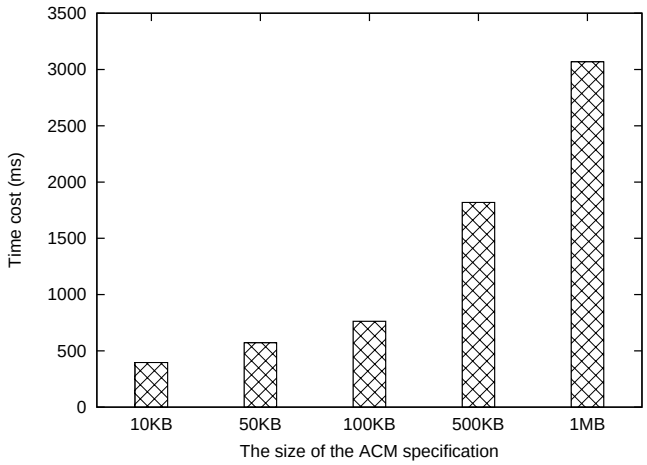


Figure 4.13: Processing overhead for ACM ontology parsing.

shown in Figure 4.2 costs around $10KB$ file size. After the creation of the ACM Ontology, the next step is to do rule parsing. With high-level operators (e.g. "distance()") required for conversion, the processing overhead is around $50ms$ for the illustrated rules in Figure 4.2.

In addition, we have carried out experiments to evaluate the performance of the ACE prototype in terms of the updating overheads of its AST operation and the reasoning delay of the inference support. The prototype is implemented in Java on a 32-bit Intel Core2 Duo @3.0GHz PC with 4GB of RAM. The AST updating involves the Scenario Update Table and the Application Scenario Table as illustrated in Figure 4.9. Upon receiving all the updating context facts from the ApplicationEntity (e.g. ExpIndexingServer), the Scenario Update Table is constructed. It is merged with the AST to derive the application scenarios that involve all the relevant instances of ApplicationEntities for further execution of the application. In all of our tests, we found the JOIN operation on the two tables has taken less than one second.

For the inference support, the reasoning efficiency over the fact data is measured. There are two kinds of detailed scenarios: *offline* and *online*. In the offline scenario, all the fact data is assumed to be in the working memory before the inference engine starts

reasoning. It is applied to enforce context constraints and is used for the construction of the AST. While in the online scenario, context facts are dynamically updated and reasoned with the inference engine. It is used for runtime application adaptation, and those StartingEntities defined in the ACM will be notified by ACE. The tests are carried out for constraint in "cf1" of Figure 4.2. Two test patterns are applied: "N-1" and "N-N". The first pattern represents Many ShopServices and One User; that is, in the ACE there are context facts regarding many instances of ShopService but only one instance of User (i.e. Alice). While the second pattern represents Many ShopServices and Many Users, i.e. by excluding the constraint "User.id="alice"". Up to 10K instances of User are emulated in the "N-N" pattern. In the offline scenario, the reasoning delay, defined as the time spent for the engine to get the potential User and his relevant ShopServices, is measured. Table 4.2 presents the results by using Jena and Jess, whereas forward-chaining inference mode is applied for both of them. As observed, Jena is not a good choice for real-time reasoning despite of its better features such as direct support for RDF/OWL. Jess, on the other hand, has no semantic support and requires preprocessings such as RDF triple transformation (i.e. to its template record format). Nevertheless, its reasoning speed is much faster than Jena. In the tests, we have also tried both engines in backward-chaining inference mode. Generally speaking, the reasoning speed of Jena is boosted up as compared to that in forward-chaining mode. However, the current implementation of Jena in backward-chaining mode does not support complex rules that involve more than 15 variables. In fact, as nested functions are not supported in Jena, the variable number limitation will be quickly exhausted by using temporary variables to hold the returned values of functions, e.g. for the computation of "distance()".

Due to its faster reasoning performance, Jess is further evaluated in ACE for the online scenario. Similarly, the StartingContexts defined in Figure 4.2 is modified by excluding the constraint "User.id="alice"". The "N-N" pattern is applied, and we assume

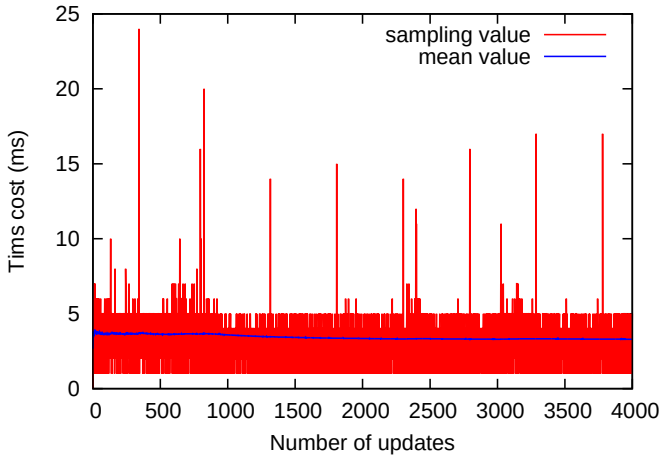
Pattern	N-1		
Test Case	100-1	1K-1	10K-1
Jena	$2832ms$	$\approx 45min$	$> 5hour$
Jess	$10ms$	$21ms$	$131ms$
Pattern	N-N		
Test Case	100-100	1K-1K	10K-10K
Jena	$\approx 8min$	OOM	OOM
Jess	$45ms$	$2652ms$	$\approx 6min$

Table 4.2: Reasoning delay for inference engines Jena and Jess in the *offline* scenario. OOM stands for out of memory when the maximum Java heap size set to 1GB.

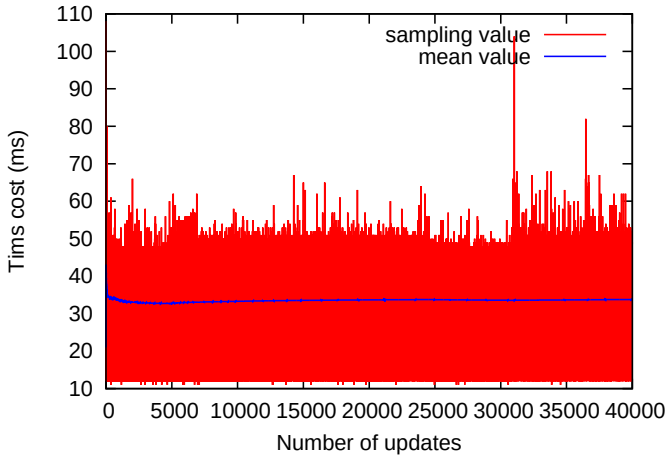
in the working memory of the inference engine there are already "N" ShopServices. The context facts of "N" different Users are updated sequentially, and each User has four context facts, i.e. one for User.id, two for User.location, and one for User.preferences. The results for the reasoning delay to get the potential Users to send notifications are plotted in Figure 4.14. Figure 4.14a shows that given "1K" ShopServices in the working memory, it takes average $4ms$ for each User to derive the fact that whether ACE should send the notification. While Figure 4.14b shows the case for "10K" ShopServices, and it takes average $35ms$ for each User.

4.5 Summary

In this chapter, we presented a framework of context realization for ubiquitous applications. The fundamental goal is to ease the task for application developers in embedding context logic (i.e. application adaptation, constraint enforcement and context flow) in their application design. To achieve this, we have proposed the Application Context Model (ACM) which considers context logic required by the application and allows the developer to specify them easily at design time. The defined ACM is then managed by the Application Context Engine (ACE) at runtime. The detailed context realization for each ubiquitous application instance is carried out on the corresponding ACM instance



(a) 1K-1K ACE online scenario.



(b) 10K-10K ACE online scenario.

Figure 4.14: Inference support of Jess for the *online* scenario in ACE.

automatically. With context logic extracted from the application layer, the whole framework provides flexibility such as in modifications to context constraints. We have done a case study to demonstrate the use of the framework. Experiments have also been carried out to validate the prototype.

As observed, the reasoning delay from the underlying inference engine raises a big issue for the effectiveness of the framework. Indeed, context reasoning is a common problem in UbiComp. The issue is how to handle reasoning over large-scale context data that may frequently change. In spite of optimizing the inference engine itself, a more practical way is to distribute the data over multiple machines and carry out distributed reasoning. We are currently working towards that direction, and we believe the two-level application context management scheme and the flexible ACE deployment plan as mentioned in Section 4.3.2 and 4.3.4 serve as a good starting point. In addition, a more rigorous evaluation of the framework system such as through a field trial involving real users will be carried out in the near future.

CHAPTER 5

SOLE: A CONTEXT-AWARE EXPERIENCE SHARING APPLICATION BASED ON LASPD AND ACE

As described in Section 2.3, one major limitation of existing information sharing applications is that their context-awareness is restricted to attributes such as user's location and social relationship. It is hard for them to support complex context-aware tasks due to lack of context infrastructure support. For instance, to enable intelligent shop recommendation, besides the user's location, other contexts such as his preferences, the shop's location and crowd level should also be considered in the application design. Without any common context infrastructure (a middleware such as Coalition [14]), each application has to define and implement its own way to support these tasks, e.g. retrieval of

shop's crowd level. Another issue is about people's privacy, especially for those Web 2.0 applications. The user profile, his preferences and his generated content are all kept by the server in the application. There is no guarantee that how the application developer is going to utilize the data, e.g. selling to a third-party. As a result, there is a need of a new application for context-aware information sharing in mobile ubiquitous computing.

This chapter presents the design and development of a prototype of *Sharing of Living Experience (SOLE)*, which enables mobile users in addition to those of desktop to share their experience of any daily activity such as shopping, entertainment, traveling and learning. More specifically, SOLE makes use of LASPD — the mobile service concept presented in Chapter 3 — to register and deploy services offered by service providers (mobile or static); it utilizes ACE — the context realization layer — presented in Chapter 4 and the Coalition middleware, for the development of context-aware SOLE. Users of popular services like Facebook and Twitter usually play the role of content provider and *Application Service Consumer (ASC)*. However in SOLE, they can play an additional role of *Application Service Provider (ASP)*, whereas as ASP, they each creates and shares his information (i.e. as content provider) of living experiences, and may offer them directly to ASC in a peer-to-peer fashion (i.e. as service provider).

The rest of the chapter is organized as follows: Section 5.1 gives discussion on the respective design consideration of SOLE. Section 5.2 presents the data schema and storage schemes adopted by SOLE. Section 5.3 describes the detailed design of constituent components of SOLE. Section 5.4 discusses the deployment of SOLE application services in LASPD. Section 5.5 considers the deployment of ACE and Coalition frameworks to achieve context-aware SOLE. Section 5.6 shows a prototype implementation of SOLE with LASPD and ACE. Experimental results for the validation and assessment of the prototype are also presented. Finally, Section 5.7 concludes the chapter.

5.1 Overview of SOLE

SOLE is considered as an extension to the ShoppingHelper application (Section 4.1). Instead of restricted to the shopping experience, SOLE extends the sharing of experience to generally any information domains. Mobile users may turn on the "push" feature of SOLE so that it could automatically update experience information of around proximity in response to the changes of user preference of types of experience. Alternatively, users could query (pull) "information" by searching certain keywords or specifying an area on the map for browsing the available information. Nevertheless, in order to support the various application scenarios in SOLE, the following design considerations have to be bear in mind for the development of SOLE:

1. *What are the application participants of SOLE, i.e. ApplicationEntities as defined in Section 4.2? How could they interact with each other?* There are three types of participants in SOLE to facilitate sharing of experiences, namely *SOLE Application Server (SOLE-AS)*, *SOLE Experience Provider (SOLE-EP)* and *SOLE Experience Consumer (SOLE-EC)*[1]. The main task of the SOLE-AS is to maintain an index to the shared data of experiences. The index key represents the entity of interest such as a location or a shop service on the map, and the value describes certain qualifications of the experience, e.g. who shared it, when and where it was shared, and who can access the data. Users play the role of SOLE-EP/EC to share or retrieve experience information. As SOLE allows the actual experience to be stored on the mobile device, and when retrieving data from the device, the respective SOLE-EP is known as a mobile service provider. Due to the fact that the functional components of these participants are implemented in different devices, cares must be taken to ensure they can interact. We leverage on the open standard

[1]In our previous paper [124], we have used terminologies SOLE-ASP and SOLE-ASC to refer to SOLE-EP and SOLE-EC respectively. However, in this thesis, SOLE-EP and SOLE-EC are adopted instead since they are more intuitive and easy to understand.

of Web services to ensure interoperability. That is, the functions of the SOLE-AS and SOLE-EP are exposed as Web services. Retrieval of experience data is taken place when the SOLE-EC invokes the service provisioned by the SOLE-AS. Subsequently, if the required information is stored in the device of the experience information provider, the SOLE-EC invokes the service hosted by the SOLE-EP. The detailed component design of the three participants will be introduced in Section 5.3.

2. *How is the application data of SOLE (e.g. experience information) managed and shared?* As surveyed in Section 2.3, privacy is the major concern for the use of current information sharing applications such as Facebook and Twitter. One of the main reasons is that the user data is kept by their Web servers and therefore, the user has no absolute control over the usage of his personal data. SOLE facilitates this issue by considering data to be stored on the user's mobile device. These devices such as smart phones are usually carried by people in their daily life; thus, they are such suitable places for putting people's private data, and it is also convenient for people to access the data when it is needed. As a result, three storage schemes have been defined in SOLE, with the details to be presented in Section 5.2. To allow data (e.g. the user profile, preferences and experience data) to be provisioned from mobile devices, the concept of mobile service (presented in Chapter 3) is applied. In summary, there are two types of application services for the provision of experience data in SOLE: one provisioned by SOLE-AS and one provisioned by SOLE-EP, which correspond to the two types of service provisioning models in LASPD as will be discussed in Section 5.4.

3. *How can SOLE achieve context-awareness in the process of experience sharing and retrieving?* To achieve context-awareness, as mentioned in Section 4.1, the application has to cater all the context-related tasks including context modeling,

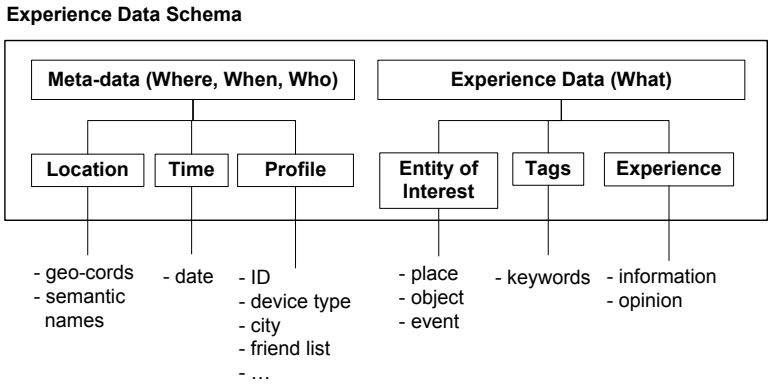


Figure 5.1: Experience data schema defined in SOLE.

context management and context realization. Fortunately, with our proposed ACE framework and Coalition middleware, most of these tasks can be automatically taken care of. Application developers may focus on the design (i.e. in ACM) rather than the implementation of context logic. We will briefly introduce the Coalition middleware and further illustrate its context-aware support together with ACE in Section 5.5. In addition, context-awareness of SOLE can also be achieved by utilizing the context information that is captured during the process of experience sharing, such as the time and the location. As a result, such information has been incorporated in the design of the experience data schema (Section 5.2) and will be used in the process of experience discovery (Section 5.5.3).

5.2 Experience Representation and Storage Schemes

In SOLE, each experience data issued by the SOLE-EP is represented by the data schema defined in Figure 5.1. The experience is expressed in a storytelling structure and the contexts of the user when creating the experience is also captured to enable context-aware experience sharing, as will be discussed in Section 5.5. In the current prototype, the data schema is implemented using XML format for its better extensibility.

The first segment is the *Meta-data* which captures three contextual elements: *Where*, *When* and *Who*. "Where" specifies the geographical location where the experience information is meant. It can be the user's smart phone GPS coordinates or a meaningful semantic name (e.g. Orchard Road) for the location. The geo-coordinates can be mapped into semantic names by our system with the use of an annotated digital site map. "When" indicates the time of creation of the experience data which could further be used for aggregation or filtering of experience data by the applications, for instance, to retrieve all the experiences for the last month. "Who" specifies the profile of the person sharing the experience. The most important attribute of the profile is the identity of the user (ID), which must be unique. We currently use the email address of the user to determine his identity. Providing the rest of the profile information, though is not mandatory, may result in an improved discovery of experience data. Note that all the meta-data is either filled once (e.g. ask the user explicitly) or captured implicitly (e.g. location and time).

The second segment contains the actual *Experience Data* which corresponds to *What* the user is sharing. It is further divided into three aspects: *Entity of Interest*, *Tags* and *Data*. "Entity of Interest" specifies the subject of interest, which could be an object, a place, or an event. We use the entity's name together with the location information associated with the experience to identify it. Moreover, if the entity has a corresponding service registered with LASPD, the service can be searched or browsed on the map and be selected for sharing the experience or retrieving the experience attached to it (assuming SOLE is integrated with LASPD, a service can be discovered or browsed by sending service discovery queries to LASPD). The "Tags" part consists of the keywords that best reflect the experience shared by the user. It is a kind of abstraction to allow others to grasp the content of the experience quickly. Besides, it can further be exploit for high level knowledge representations such as *tag clouds*[2] to facilitate aggregation of

[2]http://en.wikipedia.org/wiki/Tag_cloud.

several experiences. The last part of the schema is the details of the "Experience" to be shared. We consider two types of experience: descriptive information and personal opinions. The descriptive 'information' is to further elaborate the tagged entity, e.g. the 'history' of a picture in a museum. The personal 'opinion' is the subjective comments of user's to the tagged entity. For both types of experience, the representation format could be texts, images, or videos.

The decision for the storage of the experience data must take security, privacy, and performance into consideration. In SOLE, we offer three storage schemes for the experience data: *Public*, *Restricted* and *Private*. The user (i.e. SOLE-EP) is asked to choose one of the schemes during the process of experience creation. "Public" allows full data visibility to everyone. All the experience data is encouraged to be stored on the SOLE-AS to maximize the efficiency of retrieval, filtering, and aggregation of experience data. This will also prevent excessive load on the SOLE-EP's device, if the shared content becomes popular. Therefore, this scheme is set to default. "Restricted" is to limit the sharing experience by providing a list of friend IDs (as a part of the data schema). Note that the actual experience data can be stored on the responsible SOLE-AS or at the SOLE-EP's device. "Private" is for personal reference and helps the SOLE-EP in organizing his private experience data; all the data is preferred to be stored on his mobile device for restricted access. Nevertheless, through authentication mechanisms, external access to the data is also allowed by the SOLE-EP. In Section 5.6, we will demonstrate an example of "restricted" scheme.

5.3 Functions of SOLE Participants

The detailed design of components for the SOLE-AS and SOLE-EP/EC are presented in this section.

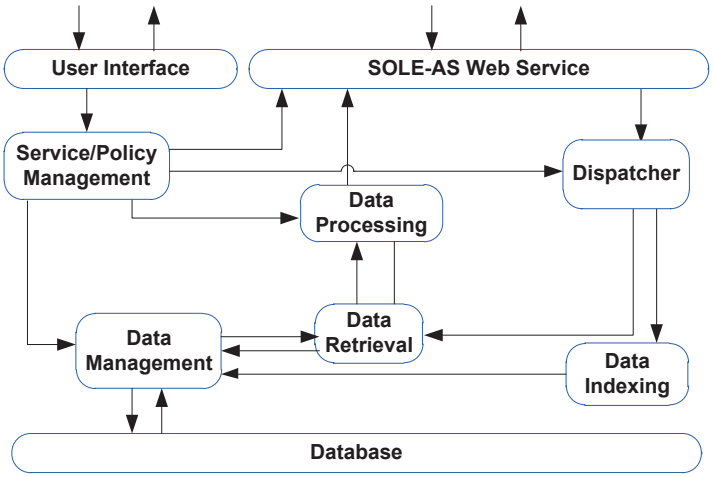


Figure 5.2: Functional components of SOLE-AS.

5.3.1 SOLE Application Server

Figure 5.2 highlights the major functional components for the SOLE-AS with data stored for public access. The SOLE-AS provisions its service to the SOLE-EP/EC through its *SOLE-AS Web Service*. The administrator uses the *User Interface* to enable/disable the service, manage data processing policies (e.g. returning friends' experience first), or organize the local experience database. The *Dispatcher* authenticates the arriving requests and delivers them to the appropriate component. Requests by the SOLE-EP are dispatched to *Data Indexing* component, while those issued by the SOLE-EC are sent to *Data Retrieval*. The data provided by the SOLE-EP is stored into the *Experience Database* through *Data Management* module.

The *Data Retrieval* component responds to the SOLE-EC by asking the *Data Management* component to search through the *Experience Database*. Once the available records are identified, they are ranked by *Data Processing* based on the requester's context (e.g. location). After processing, all the publicly available experience and those which the requester has access to are returned. The component can also aggregate all the matching records into a unified representation. Nevertheless, if the actual experience data is not stored on the SOLE-AS, the SOLE-EP's ID where the data is stored will be

returned instead, so that the SOLE-EC can retrieve the data directly from the SOLE-EP. Details of this will be described in Section 5.4.

5.3.2 SOLE Experience Provider/Consumer

The components for the SOLE-EP/EC are shown in Figure 5.3. They allow a user to create new experiences or retrieve those shared by others. A new experience is created using *Experience Creation* component. It first asks the *Data Management* component to store the data on the local *Experience Database*. Subsequently, the SOLE-AS is notified of the new experience (if is not private) through the *SOLE-AS/EP WS Invocation* component. Note that it is possible to have multiple application servers in SOLE to reduce the workload on a single machine and to cater practical concerns such as administration control of SOLE-AS. In the case of multiple SOLE-ASs, their deployment leverages on LASPD and the *SOLE-AS/EP Discovery* component finds the appropriate SOLE-AS (e.g. based on its administrative area). Similarly, to retrieve an experience, the SOLE-AS hosting the relevant index information should first be discovered, and then the *SOLE-AS/EP WS Invocation* component asks the SOLE-AS for experience data. The result of the request is processed and shown to the user. If the data is hosted in another SOLE-EP, the discovery and invocation process is repeated to the SOLE-EP.

On the right side of Figure 5.3, we have the *SOLE-EP Web service* which can be discovered and invoked by a SOLE-EC. Upon service invocation, the *Request Handler* authorizes the request (if required) and asks the *Data Management* component for the requested experience. The *Experience Database* is then searched and the matched experience is returned. Using the *Service/Policy Management* component, the user can set access rules for the *Request Handler*, for instance, whether a group of people such as friends may access a particular experience record. Additionally, the component can be used to organize the available data in the local *Experience Database*.

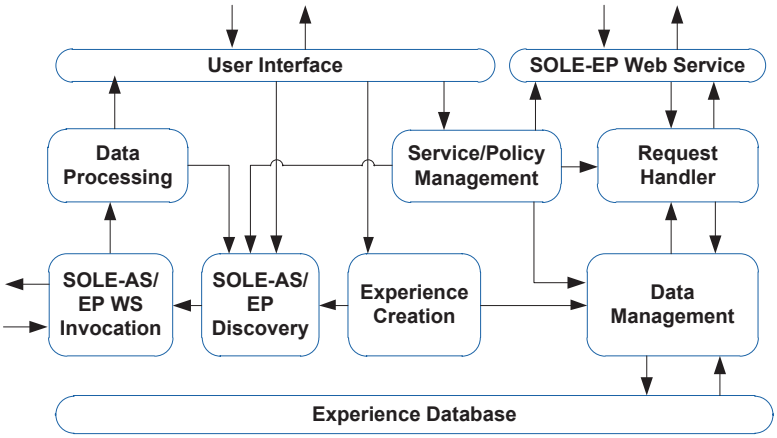


Figure 5.3: Functional components of SOLE-EP/EC.

5.4 SOLE with LASPD

Any application service can be deployed and registered in LASPD. Figure 5.4 shows the software architecture of application services. The architecture has three logical functional layers: (i) service management specific which interacts with the underlying service management functions of LASPD to complete tasks such as service registration and query routing for the application service (e.g. the functions of service mediating); (ii) application specific which performs specific business logic plus session management (e.g. the functions of SOLE-AS); and (iii) user interface specific which interacts with the user to allow functions of the application service to be invoked (e.g. the functions of SOLE-EC). The functions of these layers can be implemented and deployed completely on any service peer, or be deployed onto several computing including mobile devices.

For the example of SOLE, as mentioned before, there are two types of service providers: SOLE-AS — for sharing and retrieving experience by SOLE-EP/EC, and SOLE-EP — for retrieving experience by SOLE-EC. For the deployment of their application services in LASPD, the service peer is the most logical place to co-host the service mediating layer, which refers to the SM module in Figure 3.4 of Chapter 3. For the application specific layer, the functions of SOLE-AS can either be resided on a service peer, or on

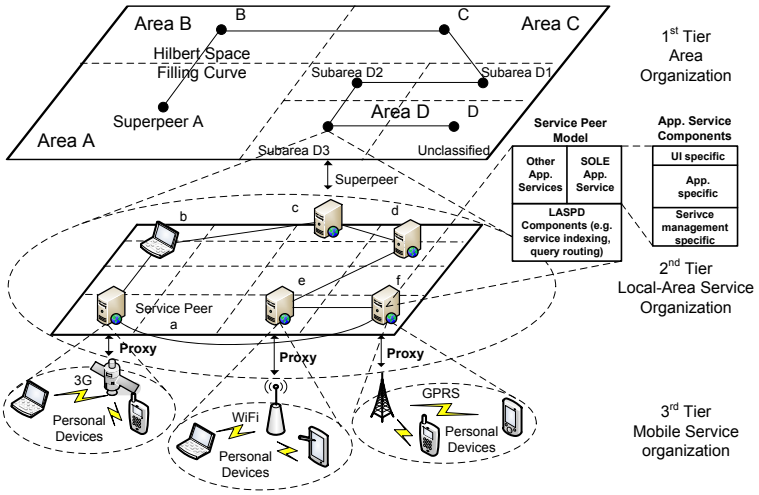


Figure 5.4: Illustration of service deployment in LASPD. Multiple services can be hosted on the same service peer.

a designated server if additional processing and storage capacities are needed; while the functions of SOLE-EP must be deployed on user devices such as mobile phones (or laptop or desktop). Depending on the application service to be invoked, i.e. whether it is for experience sharing or retrieving, the user interface specific layer is implemented on either the device of SOLE-EP or the device of SOLE-EC. Indeed, the application services provisioned on the SOLE-AS and the SOLE-EP represent two different types of service provisioning models. In the former case, it describes a "client-server" model as in the conventional Web applications such as Facebook and Twitter; while the latter case represents a "peer-peer" model among user devices. Figure 5.5 illustrates the two service provisioning models for SOLE in LASPD: in the "client-server" model, the SOLE-AS is resided on a service peer and its application services are registered with LASPD through the mediator running on the peer. SOLE-EP/EC may discover and invoke its services for creating and retrieving experience. In the "peer-peer" model, the SOLE-EP registers its service through a proxy (i.e. a service peer) in LASPD. For its "restricted experience", the SOLE-EC can retrieve the reference of the SOLE-EP (i.e. its ID) from the SOLE-AS, and then discover the SOLE-EP's service in LASPD. Once the service is discovered, a

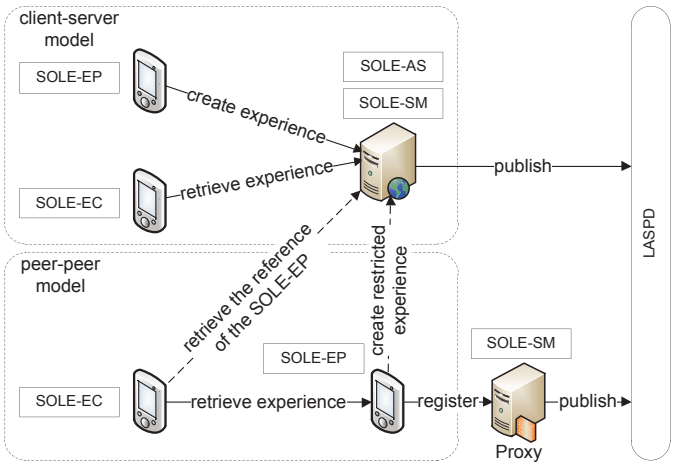


Figure 5.5: Two types of service provisioning models for SOLE in LASPD. "SOLE-SM" stands for the service mediator for SOLE application services.

request can be sent from the SOLE-EC to ask for the restricted data. In practice, as the SOLE-EP may frequently move from one place to another, we can incorporate the context retrieval function of Coalition to efficiently locate it (as will be discussed in Section 5.5.2). In fact, in the latter "peer-peer" model, the SOLE-EC may also directly invoke the service on SOLE-EP without the presence of SOLE-AS. For instance, in a local party, friends may share their experience to each other through WiFi network.

As mentioned before, when SOLE is integrated with LASPD, it can use the service browsing and discovery features of LASPD. If an entity of interest has a corresponding service registered with LASPD, the service is visible on the map and users can associate their experience with the representative service. Similarly, to retrieve the experience information, the users can select a service to check whether the corresponding entity has any experience information associated with it or not. SOLE can easily support proximity accessing (or sharing) of experiences or a range-based access by SOLE-ECs due to the location-awareness of LASPD. Alternatively, the users may also search for the relevant experience using keywords.

In the current prototype, we have deployed only one SOLE-AS for experience shar-

ing and retrieving. But it is possible to have multiple SOLE-ASs deployed, whereas each of them is in charge of a particular area of administration, e.g. a city or a country. The management of these SOLE-ASs can be handled over to LASPD, so that the experience data could be correctly updated to or retrieved from the right server, i.e. by sending service queries to LASPD based on the location of the entity of interest.

5.5 SOLE with Coalition and ACE

Coalition [14] is a context middleware designed for pervasive homecare. Its predecessor CAMPH [96] was proposed to support the development and deployment of various homecare services for the elderly such as patient monitoring, location-based emergency response, data and social networking. The middleware offers several key-enabling system functionalities that consist of P2P-based context query processing and context reasoning (e.g. for activity recognition). To be an open, programmable and reusable infrastructure, Coalition is designed as a service-oriented architecture; that is, the various system functionalities such as context data acquisition are all designed and deployed as system services for developers and end-users to access. More specifically, the middleware architecture consists of two main logical layers for context-related tasks.

- *Physical Space Layer.* This layer consists of real-world physical spaces that represent the various sources of context data. A physical space is an operating environment (e.g. people's homes, offices) that provides context data from its attached entities such as sensors, actuators and computing devices. It mandates all interactions of its entities with the "outside world" through a designated gateway known as *Physical Space Gateway (PSG)*. Moreover, a PSG can be static (e.g. at home) or mobile (e.g. worn by a person).

- *Context Data Management Layer.* To efficiently manage physical spaces and sup-

port context data lookup, the concept of *context space* is defined in this layer. A context space can be thought of an abstraction of a collection of physical spaces having some common attributes. Examples of context spaces in pervasive home-care are HOSPITAL, PERSON and HOME. The physical spaces in a context space are organized as peers in several clusters (i.e. context attributes) of a semantic overlay network, over which the context queries for data acquisition are processed. On top of this layer, a declarative, SQL-based query interface is implemented for services or applications to acquire context data or subscribe to event notifications.

Figure 5.6 illustrates the overall layering architecture of Coalition. To achieve *context-aware* SOLE, the software implementation model shown in Figure 4.10 for UbiComp applications is adopted. Coalition is used to support low-level context-related tasks such as the retrieval of context facts and subscription of contextual events; while the ACE framework discussed in Chapter 4 is used to simplify the development process for high-level tasks. In the illustrated application scenarios for SOLE in Section 5.1, we can consider the smart phone carried by Alice as the PSG of her PERSON space, which stores and supports the retrieval of Alice's personal information such as her ID, location and preferences. With the support of Coalition middleware, the context-aware applications to be described in Section 5.5.1 to Section 5.5.3 can be easily realized.

5.5.1 Information "Pushing" to the SOLE-EC

SOLE for shopping as described in Section 4.1 can be more proactive by pushing relevant information to the user (i.e. SOLE-EC) at the right time and manner. For example, the push can be triggered by the location of Alice, the matching between sales offer and her preferences, or her coming soon birthday. Each SOLE-AS can use the capability of ACE for the realization of such context-aware customizing of application for its administrative area. More specifically, the user's interest and preferences, as well as

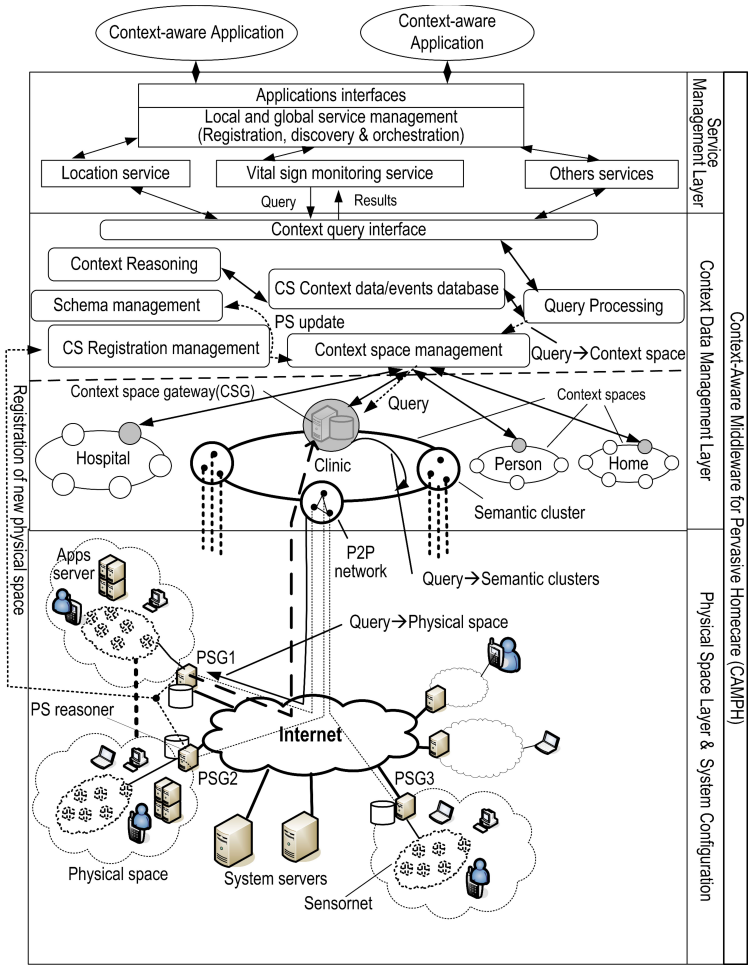


Figure 5.6: Overall layering architecture of Coalition [96].

discount margin of shops and crowd level of restaurants should be included as context constraints to the ACM model to further enhance the shopping and dinning experiences. With all these specifications, the ACE will subscribe to the Coalition middleware for the desirable user contexts and events. For instance, if the SOLE-AS is designated for information pushing to appropriate shoppers whose presences to Orchard shopping mall are detected, the ACE will fire the corresponding query "*SUBSCRIBE id, preferences FROM PERSON WHEN LocationChange WHERE new_location = 'Orchard shopping mall'*" to Coalition. Later when the *LocationChange* event is detected for Alice and her presence to the shopping mall is also detected by the middleware through the shop's PSG, Coali-

tion will notify SOLE-AS with Alice's ID and her Preferences as stored in her PERSON PSG (i.e. her mobile phone). The contexts of shops and services such as their types and crowd levels can also be retrieved in a similar manner through their respective PSGs by the middleware upon the requests from the ACE. Once all the relevant context facts are retrieved, the realization of the context logic (i.e. pushing of recommendations) can be achieved by the ACE automatically.

5.5.2 Mobility of the SOLE-EP

If the experience data is stored on a mobile device (instead of a static server), it would be a challenge for the SOLE-EC to locate the SOLE-EP without assistance. Fortunately in Coalition, such a problem can be handled in two phases: (i) retrieving the current location of the mobile PSG by issuing to the middleware the query "*SELECT location FROM PERSON WHERE id = ep_id*", where ep_id is the SOLE-EP's ID; and (ii) sending a service discovery query to LASPD for services by limiting the search scope to the proximity of SOLE-EP's location. Once the reference of the SOLE-EP is retrieved, its service can be invoked. However, if the reference is corresponding to a proxy for the mobile device, the proxy will instead take care of the request by directing it to the right SOLE-EP. In this case, the involvement of the proxy is transparent to the SOLE-EC. In addition, to enable seamless communication of delivery of multimedia content (e.g. streaming a video) during an experience sharing process, we are currently optimizing a mobility support module in Coalition for both the SOLE-EP and SOLE-EC.

5.5.3 Other Features of Context-Awareness

The location-awareness capability of SOLE has been enabled by LASPD. Consequently, users of SOLE-EC/EP may browse for service entities within the local range or issue keyword-based queries to select a service entity of interest. Other types of

context-awareness can be realized by exploiting the experience meta-data (Figure 5.1). For instance, “Time” can be used for filtering old experience data; similarly “Friend List” can be used for filtering irrelevant SOLE-ECs not in the list. Users (i.e SOLE-EP) may choose to keep data of selected attributes private for better privacy and keep the rest opened to the public (three data storage schemes as mentioned in Section 5.2).

5.6 Prototype Implementation

We have developed a prototype of SOLE in our laboratory. The prototype involves implementations of SOLE-AS on desktops and SOLE-EP/EC on user mobile devices. Moreover, the prototype of LASPD for supporting SOLE application services (as discussed in Section 5.4) and the prototype of ACE for supporting SOLE context-awareness (as discussed in Section 5.5) will be presented in this section. Lastly, prototype tests have been carried out with observations and measurements to demonstrate the feasibility of SOLE.

5.6.1 Prototype of SOLE

In the SOLE prototype, it consists of a SOLE-AS which relies on MySQL for the data management. Its functions of experience indexing and retrieval are exposed using Web services, as shown Table 5.1. The SOLE-EP/EC is implemented using HTC Hero Android handphone. Different from the SOLE-AS, the private data management in the mobile phone is implemented using the SQLite library. Figure 5.7 presents the screenshots for the user interface of SOLE-EP/EC: Figure 5.7a demonstrates that by leveraging on LASPD, the user can search/browse service entities (stars) for sharing or retrieving experience; Figure 5.7b shows that upon selecting an entity on the map, the user can share experience about the entity or retrieve experience associated with it; Figure 5.7c il-

Public APIs	Operations
registerAccount()	Create account for SOLE-EP/EC. The email address is registered as account ID.
login()	Log in to SOLE with account ID and password.
getAccountInfo()	Get the detailed information about an account, including its nick name and avatar.
retrievePassword()	Retrieve the password of an account in case the owner forgets it.
updateAccountAvatar()	Update the avatar image of an account.
discoverExperiences()	Discover entities that are associated with people's experiences. The discovery can be keyword-based and within a predefined scope (e.g. ≤ 200 meters).
getEntityTags()	Retrieve experience tags associated with a particular entity of interest.
retrieveExperiences()	Retrieve experience details about an entity of interest or based on the tags selected by SOLE-EC.
createExperience()	Create an experience about an entity of interest by SOLE-EP.
deleteExperience()	Delete an experience by SOLE-EP. The experience must be created by the same SOLE-EP.
updateExperience()	Update an experience by SOLE-EP. The experience must be created by the same SOLE-EP.
retrieveAccountExperiences()	Retrieve all the experiences created by the SOLE-EP (identified by the account ID).

Table 5.1: SOLE-AS functions which are exposed as Web services.

lustrates that for sharing experience, besides filling the required information in the form, the user may check/edit previously shared experience list; Figure 5.7d demonstrates that for retrieving experience, the user can view the list of experiences shared by others and he may add in his experience directly; Figure 5.7e illustrates a sample of an experience list associated with the entity; Figure 5.7f presents the details for the user selected experience.

Table 5.2 illustrates a scheme for the storage of "restricted" experience data. The user relies on the SOLE-AS for access control, i.e. by uploading friend list. He may also keep the whole experience data (especially those large-size data such as images and videos) on his own device for future reference or retrieval. Note that the user can always change the scheme by manipulating with the Service/Policy Manager in the SOLE-EP

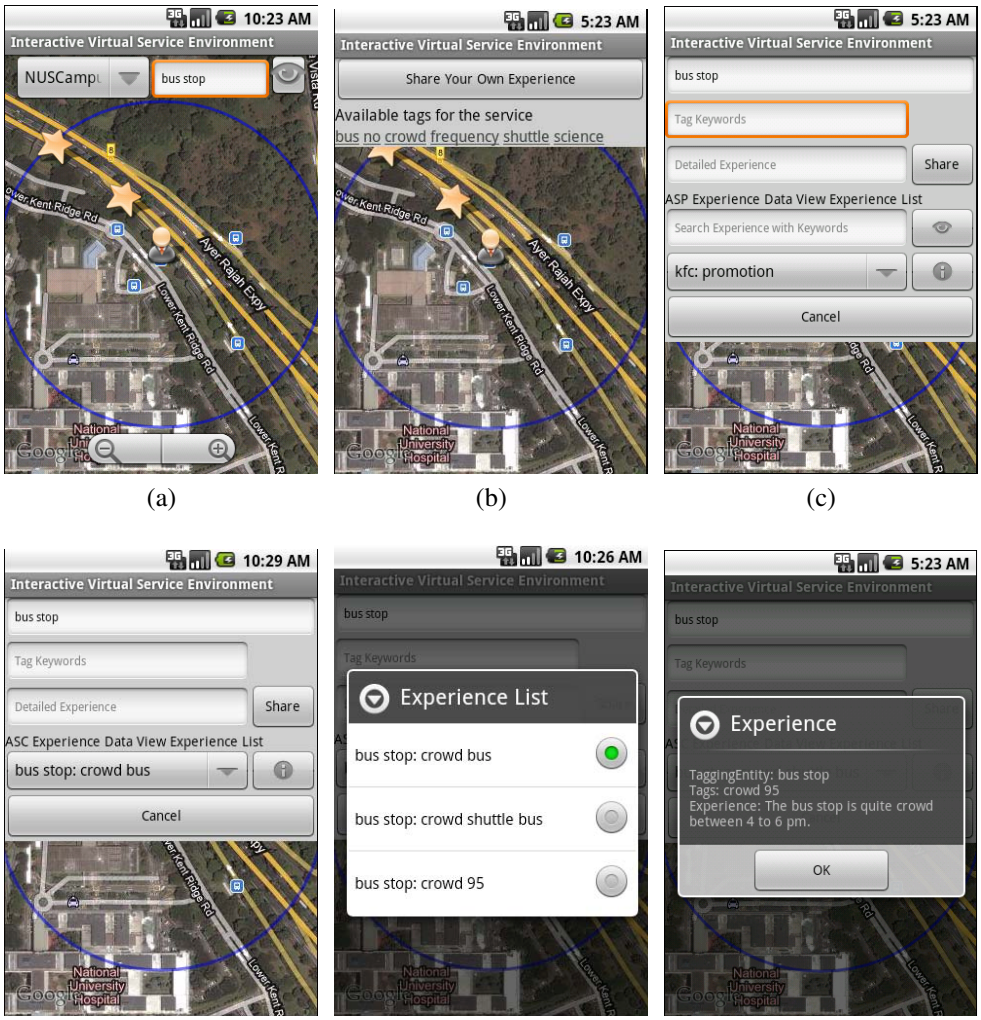


Figure 5.7: UI screenshots for SOLE-EP/EC.

(Figure 5.3) through its User Interface. For instance, he may remove all the ticks in the column which represents storage on the SOLE-EP device in case his smart phone is low on memory.

5.6.2 Prototype of LASPD

The whole package of LASPD consists of three parts: implementation of service peers, implementation of LASPD clients and implementation of a public Web server.

		SOLE-AS	SOLE-EP Device
Location	Geo-Coords	√	√
Time	Date	√	√
Personal Profile	ID	√	√
	Home Address		√
	Friend List	√	√
	Push Preference		√
Entity of Interest	Place, Object, Event	√	√
Tags	Keywords	√	√
Experience Data	Text	√	√
	Image		√
	Video		√

Table 5.2: A scheme for "restricted" storage of experience. The ticks show the support or need to store the schema attributes on SOLE-AS or SOLE-EP's device.

For service peers, the implementation focuses on the connection setup, query routing and creation of long-range links as all those have been discussed in Chapter 3. In the implementation of LASPD clients (i.e. service providers and consumers), all the APIs for interacting with LASPD are wrapped into a Java class, so the clients do not have to worry about the details, such as the message format used in LASPD. The public Web server is used to monitor service peers in LASPD. In addition, it helps mobile service providers to discover their nearby proxies (Section 3.1.5) and helps service peers to discover their local superpeers (Section 3.2.5).

For applications such as SOLE which consists of several application services (e.g. provisioned by SOLE-AS and SOLE-EP), they may leverage on the client's side implementation (i.e. the wrapper class) to interact with LASPD. The following statements demonstrate the methods to get their services published:

```
LASPDClientAPI api = new LASPDClientAPI();
ServiceData service = new ServiceData(name, location, type, description, reference);
SM mediator = api.registerService(service);
mediator.publishService();
```

In the above statements, the reference to a service peer is retrieved by instantiating the wrapper class LASPDClientAPI. By default, it queries our Web server to get the nearest

service peer information; alternatively, the client may explicitly state the reference of a service peer (i.e. IP and port) in the constructor of LASPDClientAPI. The service peer will then act as a proxy and its functions will be invocable through the wrapper instance *api*. The ServiceData *service* is used to describe the registered service, including its *name*, *location*, *type*, *description* and *reference*. "type" indicates the category of the service so that later application-specific query received by the service peer can be directed to the correct service. "description" of the service is described so to let others discover it. By default, we use the name together with the type as the keywords. Nevertheless, application developer may use richer descriptions such as those parsed by WSDL file to support Web service function discovery. "reference" refers to the mean to trigger the service in case it is a software service. It can be a memory function reference, a Web service URL or a TCP/IP socket. Once "registerService()" is called, the reference to the newly created SM object on the service peer is returned back to the application, so application can let the mediator to publish its associated service. The service indexing function of the service peer will then be triggered to index the service in LASPD. Note that once the application service is bound to a particular SM instance, it can utilize the functions provided by the SM instance to communicate with LASPD and to use its resources or functions. Figure 5.8 illustrates the relationships and interactions among the software components *Application*, *ApplicationService*, *ServicePeer* and *ServiceMediator*.

In the package, we have also provided a set of User Interfaces (UIs) to help users of LASPD. The screenshots for the registration of SOLE-AS service are presented in Figure 5.9, wherein Figure 5.9a illustrates the UI on the Web server to monitor the deployment of the distributed service peers in a university campus; Figure 5.9b demonstrates the UI for the client to register the SOLE-AS service; Figure 5.9c shows the topology for all the service registered. Note that each service is bound to a service peer (i.e. proxy) shown in Figure 5.9a.

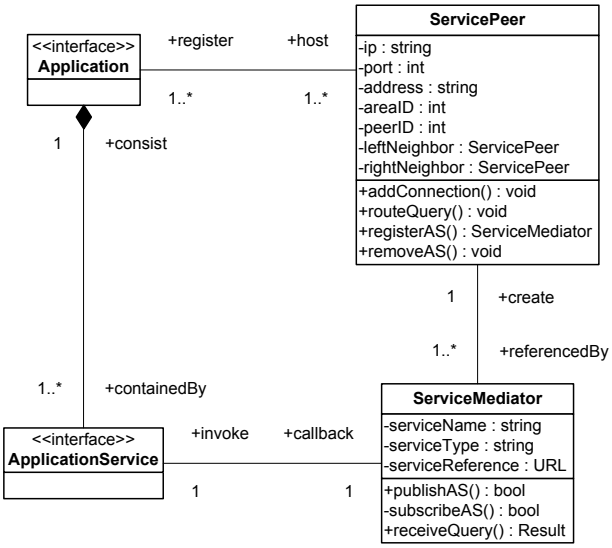
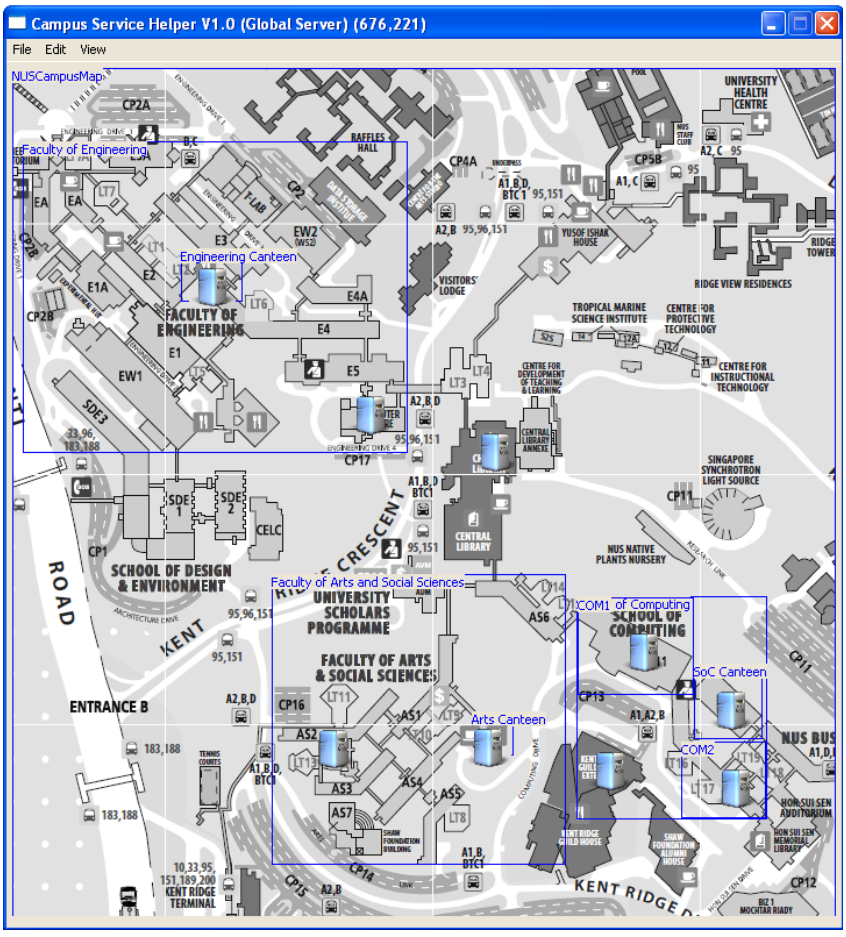
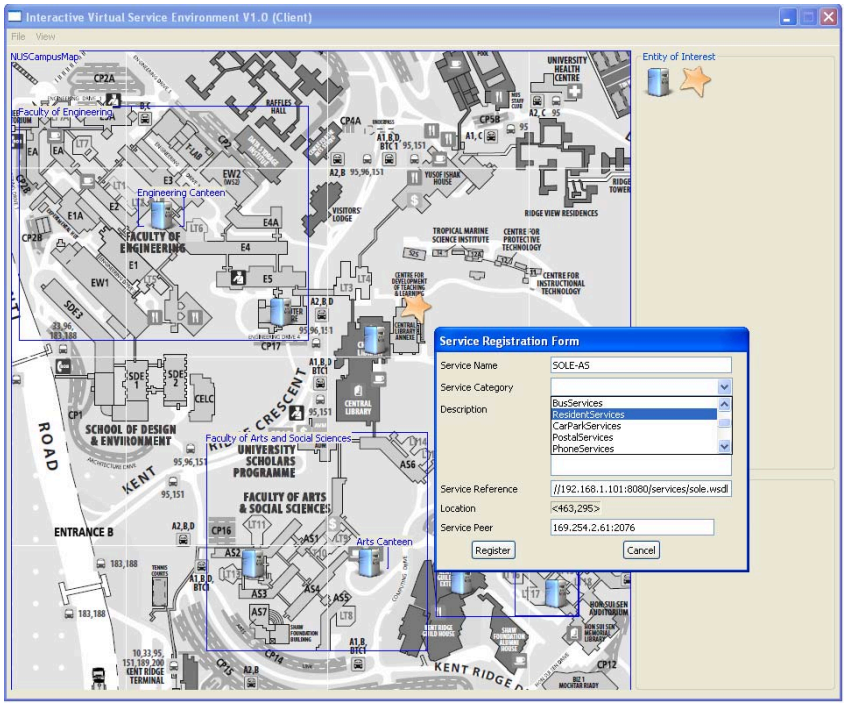


Figure 5.8: Software component interactions between application and service peer.

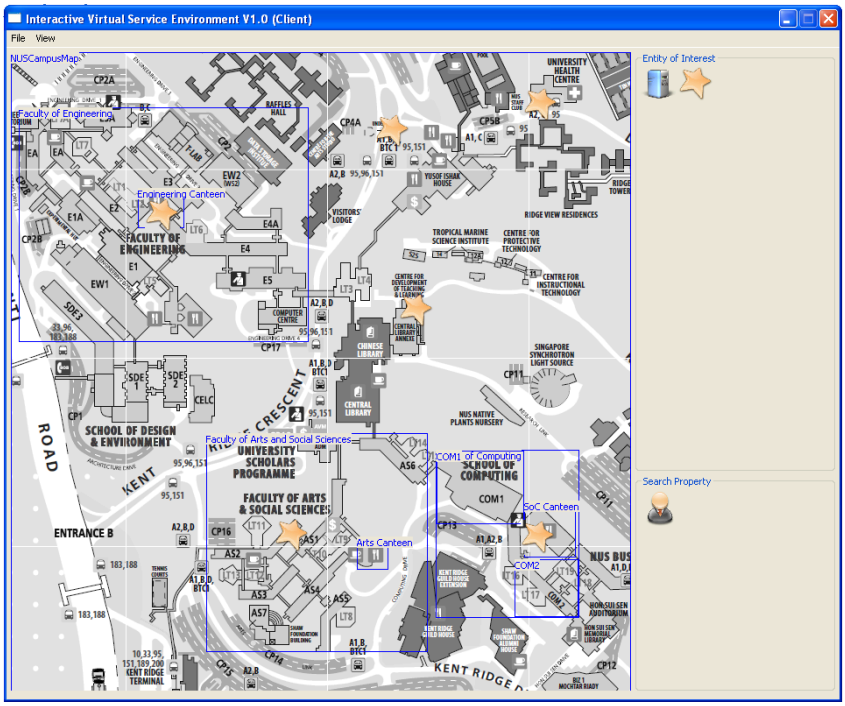


(a) Monitoring UI on Web server side.

Figure 5.9: UI screenshots of LASPD for supporting SOLE-AS service (I).

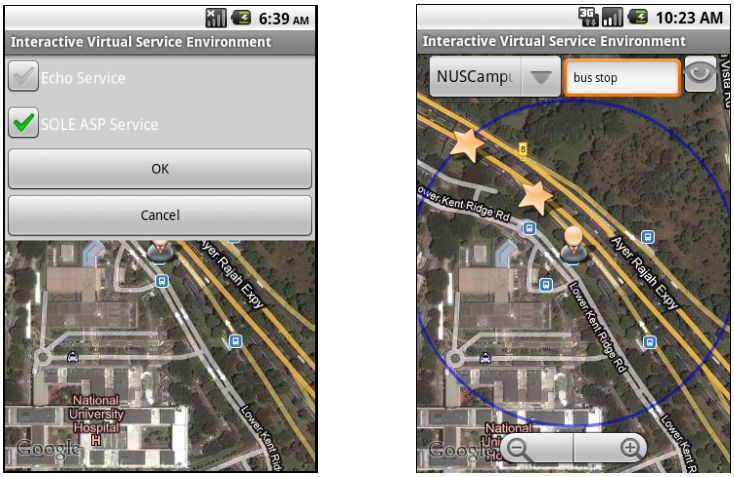


(b) Registration UI on client side.



(c) Overall service topology.

Figure 5.9: UI screenshots of LASPD for supporting SOLE-AS service (II).



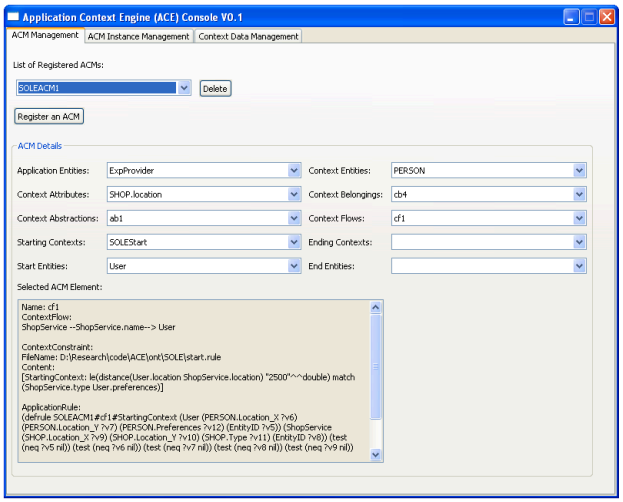
(a) Mobile service provision. (b) Service discovery.

Figure 5.10: UI screenshots for mobile service provision and discovery.

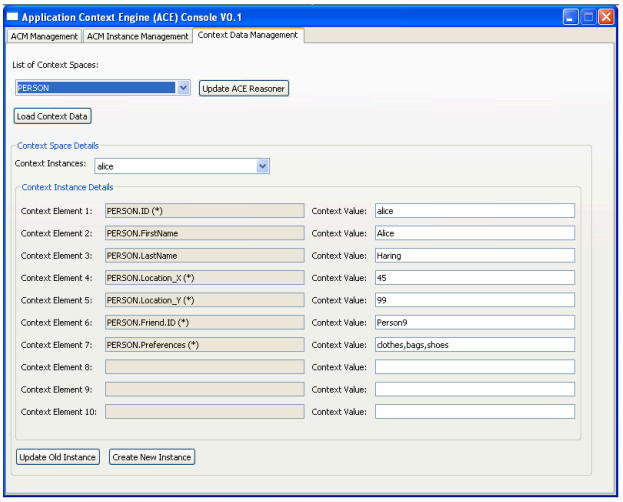
The client's side implementation has also been integrated with the mobile platform, i.e. Android OS. In Figure 5.10a, the mobile user decides which of his available service should be registered with LASPD; for example, the SOLE-EP's service. Figure 5.10b shows the result of a range search issued by a mobile user. The available services (e.g. "bus stop") are annotated by stars. Upon a click on a service (star), the user is prompted for the inputs as required by that service.

5.6.3 Prototype of ACE

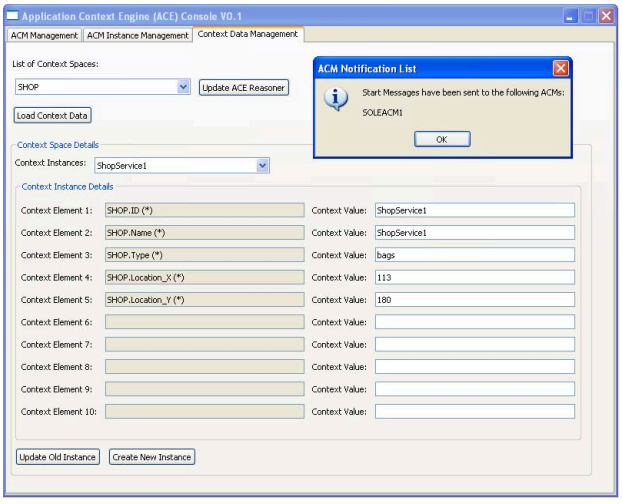
The prototype of ACE follows the architecture design as shown in Figure 4.5. We have also implemented a console for the system administrator to monitor: (i) all the ACMs registered; (ii) all the context facts in the Context Fact Base; (iii) all the ASTs of running ACM Instances. Figure 5.11a illustrates the UI of ACM Management. The administrator may explicitly register an ACM from the local repository or wait for requests from remote sites (i.e. handled by the ACE Interface in Figure 4.5). Once the ACM specification is retrieved, the various ACM elements are parsed and reflected in the console. The administrator may browse each element and see the detail. For a com-



(a) UI of ACM Management.

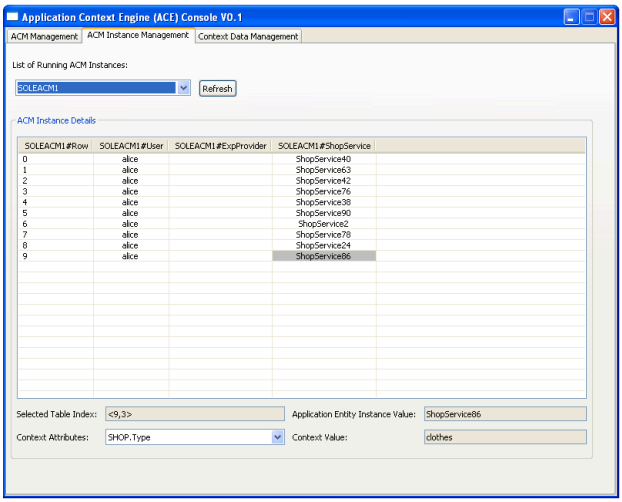


(b) UI of Context Data Management.

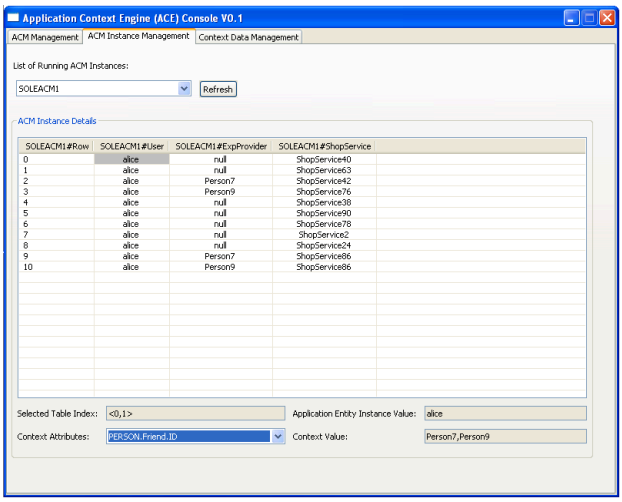


(c) Creation of a SOLE ACM Instance.

Figure 5.11: UI screenshots of ACE for supporting SOLE context-awareness (I).



(d) Initialization of a SOLE ACM Instance.



(e) Execution of a SOLE ACM Instance.

Figure 5.11: UI screenshots of ACE for supporting SOLE context-awareness (II).

plete ACM specification of SOLE, it can be found in the Appendix. Figure 5.11b shows the UI of Context Data Management. Similarly, the administrator may load context facts from the local management system (e.g. DBMS) or subscribe with our Coalition middleware (Section 5.5.1). As context facts are selectively updated according to the registered ACMs (Section 4.3.2), only the relevant context facts are retrieved. For SOLE, we implement the application intelligence as illustrated in the ShoppingHelper application (Section 4.1); therefore, context facts of PERSON and SHOP are required. The fact data

is updated to the Context Fact Base in ACE, which is utilized by the Inference Engine to do the reasoning tasks, i.e. over the StartingContexts subscribed by each registered ACM. If the reasoning is successful, the corresponding ACM is instantiated and notifications are sent to the StartingEntities as shown in Figure 5.11c. Figure 5.11d illustrates the list of the running ACM Instances. For each instance, its AST is shown. As the ACM Instance of SOLE is just initialized, only IndependentEntities are instantiated and filled in the table, i.e. User and ShopService. Alice's smart phone may retrieve the satisfied ShopServices from ACE and send their names to the SOLE-AS together with the ID of the ACM Instance, i.e. "SOLEACM1". The SOLE-AS matches the ShopServices to their corresponding experience data. However, if the data is stored on people's devices, the SOLE-AS updates ACE with the pairs of ShopService's name and ExpProvider's ID (i.e. the ID of the SOLE-EP), which in turn updates the Application Context Base of the ACM Instance. The constraint that the provider must be a friend of Alice is enforced by ACE, with the provider's ID filled in the AST (as shown in Figure 5.11e). With the AST updated, Alice's could then retrieve experience data from her friends' devices.

5.6.4 Prototype Validation of SOLE

For a more rigorous study and evaluation of SOLE performance, we have run the distributed SOLE-AS on desktop PC (Intel Dual-Core E8400) and the SOAP-EP on mobile device (HTC Hero) to measure the overhead occurred during the application execution. Several metrics based on real-time evaluation are compared and shown in Table 5.3. In the table, "SOAP Deserialization/Serialization" is referring to the time spent for parsing/composing SOAP messages when the application services are invoked on SOLE-AS and SOLE-EP. As we are using Web services for communications, this overhead must be studied to test the application feasibility, especially for mobile devices. "Application Execution" is referring to the overhead occurred for application logic of SOLE. We further

Metric	Time (Milliseconds)	
	Desktop	Mobile Device
SOAP Deserialization	12	132
SOAP Serialization	5	45
Application Execution		
Experience Insertion	17	62
Experience Deletion	2	12
Experience Search	4	14

Table 5.3: Performance comparisons for application services running on desktop (i.e. SOLE-AS) and mobile device (i.e. SOLE-EP) in real-time.

divided it into three categories of operations: experience insertion, deletion and search. For all the three operations, the network latency has not been included for assessment in the table. Though compared to the conventional desktops, applications running on mobile devices incur more overhead due to their slower CPU and smaller memory, the performance is still acceptable. Indeed, with rapid advancement in mobile technologies, we can foresee mobile devices as service providers will be popular in the near future.

To further investigate SOLE's performance in real-life scenarios, we have set up its prototype in the university campus. A public Web server is deployed for the functions of SOLE-AS. Mobile clients (i.e. SOLE-EP/EC) may access its Web services through wireless networks. In addition, the machine running the Web server also implements the functions of a service peer which acts as the proxy for both SOLE-EP and SOLE-EC. As a result, the SOLE-EC may directly retrieve experience data from the SOLE-EP (if the SOLE-EP's ID is known). The tests have been carried out in the campus environment and 3G network is used for communication by mobile devices. For SOLE-EP, the latency for the process of experience sharing is measured; while for SOLE-EC, the latencies of experience discovery and retrieval are studied. Table 5.4 shows the results for each category of measurement.

For SOLE-EP, the sharing of experience to SOLE-AS is divided into two types: with text only, with both text and image. The sample text contains 50 words and the size

Metric	Time (Milliseconds)
Experience Sharing to SOLE-AS	
Text	1753
Text + One Image	2540
Text + Two Images	3577
Experience Discovery and Retrieval from SOLE-AS	
Experience Discovery	1482
Experience Retrieval	1778
Image Downloading	526
Experience Retrieval from SOLE-EP	
Text	1679
One Image	2902
Two Images	4287

Table 5.4: Performance measurements of SOLE application in real-life scenarios.

of each sample image is around $25KB$. The increment in the latency of the sharing process is caused by the process of image uploading, which is sequential in the current implementation. The experience discovery process concerns about retrieval of the relevant information from SOLE-AS that matches the SOLE-EC's request, such as the keywords specified and the distance indicated. The details about each discovered experience such as text content and images are not retrieved in the process; instead, only the name of the entity and its location are retrieved and displayed on the map. The number of the discovered experiences in the tests varies from 1 to 40, and the mean value for the latency is shown in the table. Once the SOLE-EC wants to know the details about a particular experience (e.g. by long-pressing the star icon in Figure 5.7a), the experience retrieval process is triggered. The process retrieves the text content and image references (i.e. Web URL) about the experience from SOLE-AS. After that, images are downloaded concurrently in the background processes. However, if the experience data is restricted, i.e. the text content and images are stored on the mobile device of SOLE-EP, the SOLE-EC may ask for the SOLE-EP's permission to retrieve them. The retrieval process is completed with the help of the proxy (as described in Section 3.1.5 of Chapter 3). Overall, from Table 5.4 we demonstrate that SOLE is feasible for practical usage.

5.7 Summary

In this chapter, we proposed a generic Sharing Of Living Experience (SOLE) application, which fits in the vision of ubiquitous computing to let people communicate and share information at anywhere, anytime. SOLE has been used as a test case for application development on our context middleware (Coalition), service management platform (LASPD) and context realization framework (ACE). More specifically, SOLE may deploy its application services (i.e. the two types of services provided by SOLE-AS and SOLE-EP) and utilize the resources (e.g. registered services) and mechanisms (e.g. location-aware service discovery) in LASPD. It could enhance the quality of service by incorporating context-awareness in the process of experience sharing and retrieving, with the support of Coalition and ACE.

As a proof of concept, we have implemented a prototype of SOLE including the introduced components (i.e. SOLE-AS, SOLE-EP and SOLE-EC) as well as the three data storage schemes for experience data. The use of the underlying platform/framework (i.e. LASPD and ACE) to support SOLE is also demonstrated. Tests on the SOLE prototype show that it is feasible for practical deployment. In the near future, we would like to carry out a field trial of SOLE with more users involved, such as by involving the students in the university campus.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This chapter concludes the whole thesis. Section 6.1 summarizes the research work we have done to resolve the problems as raised in Section 1.4; the respective results and contributions are highlighted. Section 6.2 indicates potential extension of research and directions for future work.

6.1 Conclusion

Many researchers have envisaged that Ubiquitous Computing (UbiComp) would lead to the development of smart ambient that would seamlessly assist and improve the quality of our daily living to next level that close to what being described in science-fiction. Context-awareness, as a key enabling technology of UbiComp, will help to achieve this vision by adapting applications and services to user contexts and environment settings. However, in early days, due to technology restrictions such as high-cost sensor chips and low-speed wireless communication, the vision of UbiComp could only demonstrated

partially in laboratories. With the recent advancement in hardware and communication technologies, it is becoming feasible to develop the concept of UbiComp for real-life applications in larger scale. In particularly, portable devices such as smart phones and tablets which are becoming pervasive, will play a major role in the transition to real life commercial UbiComp from the lab-based research prototypes. On one hand, these mobile devices could act as the gateways to collect user contexts (e.g. user's location) through embedded sensors; on the other hand, they possess computing and networking capabilities to allow services to be hosted. This will open up a new era of true *mobile services*, marking a major paradigm shift of providing services from the fixed server-centric service model to the new mobile people-centric service model.

Indeed, realizing UbiComp for real world applications has posed many research challenges in computer science. These include resource and service discovery, context-aware system design, and ubiquitous application development. Most existing approaches for UbiComp systems are domain and application-centric, with simple use of sensors, involving fewer context entities in the application. Due to the complexity in its design and implementation, UbiComp has yet to reach the stage that involves the exchange of information from large-scale distributed context sensing sources. Our early work in this area includes context data modeling, retrieval and reasoning techniques in a service-oriented middleware known as Coalition. In fact, Coalition provides the essential middleware infrastructure for large-scale context-aware operations, and has been used for the development and testing of the design concepts proposed in this thesis.

The main deliverable and achievements of this thesis can be summarized as follow: (i) the proposal of an advanced framework for the provision, discovery and development of ubiquitous services and applications; (ii) the proof of the design concept of the proposed framework through simulation study and prototyping; and (iii) the integration of the prototype to an existing middleware with an extended architecture and the demon-

stration of its enhanced capability through the design and development of a ubiquitous application. The technical contributions of this thesis are elaborated as follows:

- We have proposed the LASPD platform (Chapter 3) which has a three-tier architecture for the provision and discovery of services, including the support for the emerging mobile services. Peer-to-Peer (P2P) concepts have been deployed in the first two tiers to achieve scalability. Meanwhile, the design of the P2P overlay structure has facilitated the management of services and a location-based range search of services. The deployment of the Hilbert space filling curve preserves the geographical locations of service providers, and contributes to the flexibility of managing service providers and service administrative areas. The proposed Source Sampling mechanism defines a probabilistic process of augmenting long-range links in the P2P network which helps to achieve the "small world effect" autonomously. As demonstrated in simulation studies, our system can achieve network navigability by using source sampling. The modified probability model for source sampling also exhibits its effectiveness in adapting to real-world settings, where parameter $cov_{id}(u)$ is used to remove the assumption that every cell in the area must be occupied by a peer and parameter α is to boost up the performance convergence speed. The performance of the source sampling mechanism has been compared with that of other similar approaches (e.g. Random Sampling, superpeer-based approach) in local-area and cross-area routings. The simulation results demonstrated that our scheme has better routing efficiency and is more resilience to peer node (i.e. superpeer) failure. Furthermore, compared with Threshold Sampling in mobile environments, our source sampling has better capabilities in recovering routing efficiency in the presence of network topology changes. However, as observed, source sampling requires a longer time to converge to its peak routing performance. In view of this, a way to overcome this slow start is

to apply a Centralized Sampling during the initial bootstrapping phase of the peer, and then reply on source sampling to maintain its long-range links.

- We have proposed the ACE framework (Chapter 4) which aims to fast-track the implementation of context-related tasks in a ubiquitous application. Three types of context-related tasks (i.e. context logic) respectively for application adaptation, context constraint enforcement and context flow are considered in this thesis. They may all be specified and modeled in the Application Context Model (ACM). The respective specification is then registered with the Application Context Engine (ACE) which is capable of interpreting and realizing the tasks at application runtime. The case study on a ubiquitous application (i.e ShoppingHelper) demonstrates that the ACE framework simplifies implementation process of application as well as enhancing its maintainability when dealing with changing context requirements. With the use of ACM and ACE, the formulation of context-related tasks is shifted to the design time from the implementation phase under normal circumstances. An important implication of this 'shift' is the de-coupling of 'the design and implementation of context-related tasks' from that of the application itself. Hence any changes of the underlying context data acquisition or processing affect only the context-related tasks instead of the whole application. In addition, ACE also helps to shift the context-related computation tasks away from resource-constrained devices (e.g. smart phones) where power consumption is a major concern. Based on the case study, we have conducted experiments and validated the prototype performance on ACM parsing, AST updating, and offline/online context reasoning. However, during the experimental tests, we observed that the rule-based reasoning engine (e.g. Jena and Jess) used by the ACE has incurred substantial delay. The delay incurred would increase almost exponentially with the increase of data size of context facts. We therefore conclude that the state of the art reasoning

technology is not suitable for reasoning over large-scale context data, and clearly this is an important area for future research.

- We have proposed the SOLE application (Chapter 5) which fits in the vision of UbiComp to let people communicate and share experience at anywhere, anytime. It leverages on LASPD to manage its application services (i.e. services provisioned by SOLE-AS and SOLE-EP), and it incorporates ACE (with the help of Coalition) to achieve context-aware experience sharing and retrieving. As a result, the successful development of SOLE is a clear demonstration of the feasibility of the design concepts of the integrated framework consisting LASPD, ACE and Coalition middleware. Compared with other information sharing applications, SOLE is more flexible as it allows the user to store his context data (e.g. location and profile) and application data (e.g. experience) on his mobile device. The data is then provided through the mobile service hosted on the device and accessible with user permission. This approach allows the user to have a better control over his personal data instead of relying on a third-party server which may cause privacy issue. Moreover, the context-awareness of SOLE is not restricted to simple attributes such as user's location. Complex context-aware tasks can also be supported with the help of the context infrastructure — Coalition and the context realization framework — ACE. Our current prototype of SOLE is performing satisfactorily. However, we still need more rigorous study of its performance with more users involved, such as for a field trial in the university campus.

6.2 Future Work

Driven by the advancement in technologies such as microelectronics and information computing, it is clear that UbiComp will increasingly influence everyday life. Neverthe-

less, as mentioned before, much remains to be done and learned in this area. Despite of the work proposed in this thesis, the following problems have been identified:

- *Improvement of Reasoning Efficiency:* Reasoning plays an important role in UbiComp, and due to the characteristics of context data (e.g. uncertainty and frequent changes), it is crucial to improve the reasoning efficiency to make the developed context frameworks and ubiquitous applications effective in practice. As demonstrated in Chapter 4, existing reasoners (e.g. Jena) are falling short in supporting context reasoning over dynamic data at real-time. This is simply because their engines are not designed for context data, and the update (involving both deletion and addition) of data may result in the re-evaluation and execution of the whole reasoning process over all the data and rules. It is therefore necessary to develop an efficient algorithm to effectively support reasoning over dynamically changing data [125]. To further enhance the reasoning efficiency, the reasoning process involved in a particular task such as the recommendation of shops based on their distances and types may be distributed to multiple nodes. Each node is supposed to handle a sub-task which may refer to an operator function such as checking the shop type using "equal()" or refer to a computation task such as computing distance between the shop and the user. Indeed, these sub-tasks can be further divided and eventually form a tree structure for context reasoning. Each node of the tree can be considered as an *aggregation point*, which can consume the data from low-level points and provide the results to high-level points. Moreover, the aggregation point can be reused by any other application that has the same sub-task. We believe by distributing the reasoning process, the size of the context data for reasoning can be reduced and the efficiency of reasoning on each sub-task can be improved. Of course, the challenges of this approach will be how to divide a given task, namely the plan to construction the tree, and how to distribute the aggrega-

tion point. For the latter issue, one possible solution is to map each aggregation point to a Physical Space Gateway (PSG) in Coalition. The mapping criteria can be relevancy-based, for instance, if the task is concerning about whether there is anyone in a specific room, then the aggregation point assigned with the task can be mapped to the room's PSG, and a separate ACE framework is deployed there to fulfill the task.

- *Personalization of Application Adaptation:* The current ACM designed in the ACE framework requires that the conditions for application adaptation (i.e. StartingContexts and EndingContexts) must be explicitly predefined by application developers. Although personalization can be achieved by imposing different specifications for different users, it can be a quite tedious work in practice. Moreover, there can be mismatching between the scheme designed by developers and that desired by users. As a result, intelligence should be incorporated into the design for personalization of application adaptation, and the whole process should be made automatic and possibly transparent to the user. Techniques such as machine learning can be applied so that useful data is mined and used for future predication. The range of context data to be mined must be restricted for the considerations of efficiency and effectiveness. For instance, ContextAttributes of the user that are relevant to the application adaptation should be predefined and this kind of data is collected whenever the user uses the application. Artificial intelligence techniques are then applied on the historical data over certain period. Upon the next occurrence of the similar situation, the application can be adapted automatically. Of course, this approach for personalization should not be intrusive, especially in the early stage of machine learning when there is less data to mine.

- *Practical Deployment of LASPD and SOLE:* As mentioned in Chapter 5, we have implemented the prototypes for LASPD and SOLE in our laboratory. However,

they are currently deployed on a small scale and mainly for demonstration purpose. We would like to put more development effort to make them suitable for a field trial. To achieve so, we are currently working on a project — *Dig'it!*[1] which combines the features of LASPD and SOLE. The project platform supports the sharing, provision and discovery of information and services. As in LASPD, the service can be provisioned by mobile devices, and it can be invoked directly through the developed platform. Other features such as information/service push, user activity tracing, local data storage are also provided. The goals of the project are to create an interactive environment for information/service providers and consumers, and to improve user experiences with our developed frameworks and techniques for mobile ubiquitous computing.

[1] http://137.132.145.205/DigitWeb.

BIBLIOGRAPHY

[1] Anind K. Dey. Understanding and Using Context. *Personal Ubiquitous Comput.*, 5:4–7, January 2001.

[2] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.

[3] Norbert Streitz and Paddy Nixon. The Disappearing Computer. *Commun. ACM*, 48(3):32–35, March 2005.

[4] Adwait Gupte and Phillip Jones. Towards hardware support for common sensor processing tasks. In *Proceedings of the 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '09, pages 85–90, Washington, DC, USA, 2009. IEEE Computer Society.

[5] Xiaohang Wang, Jinsong Dong, Chung Yau Chin, SankaRavipriya Hettiarachchi, and Daqing Zhang. Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, 3:32–39, 2004.

[6] Sailesh Sathish and Cristiano di Flora. Supporting smart space infrastructures: a dynamic context-model composition framework. In *Proceedings of the 3rd international conference on Mobile multimedia communications*, MobiMedia '07, pages 67:1–67:6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[7] Manal Al-Bahlal and Jalal Al-Muhtadi. A middleware for personal smart spaces. In *Proceedings of the 34th Computer Software and Applications Conference Workshops*, pages 299–304, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[8] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6:161–180, April 2010.

[9] Thomas Strang and Claudia L. Popien. A Context Modeling Survey. In *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, September 2004.

[10] R. Krummenacher and Thomas Strang. Ontology-Based Context Modeling. In *Workshop on Context-Aware Proactive Systems*, 2007.

[11] Harry Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, 2004.

[12] Patrick Fahy and Siobhan Clarke. Cass — a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*, 2004.

[13] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.*, 28:1–18, January 2005.

[14] Jian Zhu, Penghe Chen, Hung Keng Pung, Mohammad Oliya, Shubhabrata Sen, and Wai Choong Wong. Coalition: A platform for context-aware mobile application development. *Ubiquitous Computing and Communication Journal*, 6, January 2011.

[15] A. van Halteren and P. Pawar. Mobile service platform: A middleware for nomadic mobile service provisioning. In *Proceedings of the 2006 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 292–299, Washington, DC, USA, 2006. IEEE Computer Society.

[16] Satish Narayana Srirama, Vladimir Sor, Eero Vainikko, and Matthias Jarke. Supporting mobile web service provisioning with cloud computing. *Int. J. on Advances in Internet Technology*, 3:261–273, 2010.

[17] Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. iCAP: Interactive Prototyping of Context-Aware Applications. In *Proceedings of the Pervasive Computing*, pages 254–271, 2006.

[18] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive Mob. Comput.*, 2:37–64, February 2006.

[19] Thorsten Caus, Stefan Christmann, and Svenja Hagenhoff. Hydra — An application framework for the development of context-aware mobile services. *Business Information Systems*, 7(14):471–481, 2008.

[20] Bin Guo, Daqing Zhang, and Michita Imai. Toward a cooperative programming framework for context-aware applications. *Personal Ubiquitous Comput.*, 15:221–233, March 2011.

[21] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, pages 17–24, New York, NY, USA, 2000. ACM.

[22] P. J. Brown and G. J. F. Jones. Context-aware retrieval: Exploring a new environment for information retrieval and information filtering. *Personal Ubiquitous Comput.*, 5:253–263, January 2001.

[23] Guanling Chen and David Kotz. Solar: A pervasive-computing infrastructure for context-aware mobile applications. Technical Report TR2002-421, Dartmouth College, Computer Science, Hanover, NH, February 2002.

[24] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[25] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM.

[26] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[27] David Hilbert. Ueber die stetige Abbildung einer Line auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.

[28] Jon Kleinberg. Small-World Phenomena and the Dynamics of Information. *Advances in Neural Information Processing Systems*, 14:431–438, 2001.

[29] R. Marin-Perianu, P. Hartel, and H. Scholten. A Classification of Service Discovery Protocols. Technical report, Dept. Electrical Eng., Mathematics, and Computer Science, Univ. of Twente, Netherlands, 2005.

[30] Fen Zhu, Matt W. Mutka, and Lionel M. Ni. Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, 4:81–90, October 2005.

[31] S.A. Hosseini-Seno, R. Budiarto, and T.C. Wan. Survey and New Approach in Service Discovery and Advertisement for Mobile Ad Hoc Networks. *Computer Science and Network Security*, 7:275–284, 2007.

[32] Christopher N. Ververidis and George C. Polyzos. Service Discovery for Mobile Ad Hoc Networks: A Survey of Issues and Techniques. *IEEE Communications Surveys and Tutorials*, 10:30–45, 2008.

[33] Elena Meshkova, Janne Riihijärvi, Marina Petrova, and Petri Mähönen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Comput. Netw.*, 52:2097–2128, August 2008.

[34] Stephan Hagemann, Carolin Letz, and Gottfried Vossen. Web service discovery - reality check 2.0. In *Proceedings of the Third International Conference on Next Generation Web Services Practices*, NWESP '07, pages 113–118, Washington, DC, USA, 2007. IEEE Computer Society.

[35] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of*

the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Middleware '01, pages 329–350, London, UK, 2001. Springer-Verlag.

[36] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California at Berkeley, Berkeley, CA, USA, 2001.

[37] W. Acosta and S. Chandra. Unstructured peer-to-peer networks — next generation of performance and reliability. In *Proceedings of the 24th Annual IEEE International Conference on Computer Communications*, 2005.

[38] Farnoush Banaei-Kashani, Ching-Chien Chen, and Cyrus Shahabi. WSPDS: Web services peer-to-peer discovery service. In *Proceedings of the International Conference on Internet Computing*, pages 733–743, 2004.

[39] Farnoush Banaei-Kashani and Cyrus Shahabi. Searchable Querical Data Networks. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in conjunction with VLDB*, 2003.

[40] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 333–347, London, UK, UK, 2002. Springer-Verlag.

[41] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. Deapspace: transient ad-hoc networking of pervasive devices. In *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '00, pages 133–134, Piscataway, NJ, USA, 2000. IEEE Press.

[42] Matthew Denny, Michael J. Franklin, Paul Castro, and Apratim Purakayastha. Mobiscope: A scalable spatial discovery service for mobile network resources. In *Proceedings of the 4th International Conference on Mobile Data Management*, MDM '03, pages 307–324, London, UK, UK, 2003. Springer-Verlag.

[43] Celeste Campo, Mario Munoz, Jose Carlos Perea, Andres Marin, and Carlos Garcia-Rubio. Pdp and gsdl: A new service discovery middleware to support spontaneous interactions in pervasive systems. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, PERCOMW '05, pages 178–182, Washington, DC, USA, 2005. IEEE Computer Society.

[44] Ouri Wolfson, Bo Xu, Huabei Yin, and Hu Cao. Search-and-discover in mobile p2p network databases. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, ICDCS '06, pages 65–, Washington, DC, USA, 2006. IEEE Computer Society.

[45] Hossam Hassanein, Yu Yang, and Afzal Mawji. A new approach to service discovery in wireless mobile ad hoc networks. *Int. J. Sen. Netw.*, 2:135–145, April 2007.

[46] J. Antonio Garcia-Macias and Dante Arias Torres. Service discovery in mobile ad hoc networks: Better at the network layer? In *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, ICPPW '05, pages 452–457, Washington, DC, USA, 2005. IEEE Computer Society.

[47] Christopher N. Ververidis and George C. Polyzos. Routing layer support for service discovery in mobile ad hoc networks. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*,

PERCOMW '05, pages 258–262, Washington, DC, USA, 2005. IEEE Computer Society.

[48] A. Varshavsky, B. Reid, and Eyal de Lara. The Need for Cross-Layer Service Discovery in MANETs. Technical Report CSRG-492, University of Toronto, 2004.

[49] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: a scalable overlay network with practical locality properties. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 9–9, Berkeley, CA, USA, 2003. USENIX Association.

[50] H. Unger and M. Wulff. Cluster-building in P2P-Community Networks. In *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 680–685, 2002.

[51] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In *Proceedings of the 1st International Conference on Pervasive Computing*, Pervasive '02, pages 195–210, London, UK, 2002. Springer-Verlag.

[52] Michael Klein, Birgitta König-Ries, and Philipp Obreiter. Lanes - A lightweight overlay for service discovery in mobile ad hoc networks. In *Proceedings of the 3rd Workshop on Applications and Services in Wireless Networks*, 2003.

[53] Eunyoung Kang, Moon Jeong Kim, Eunju Lee, and Ungmo Kim. Dht-based mobile service discovery protocol for mobile ad hoc networks. In *Proceedings of the 4th international conference on Intelligent Computing: Advanced Intelligent Computing Theories and Applications - with Aspects of Theoretical and Method-*

ological Issues, ICIC '08, pages 610–619, Berlin, Heidelberg, 2008. Springer-Verlag.

[54] NOMAD Deliverable 3.5. Service Discovery Middleware. Technical report, European project NOMAD (Integrated Networks for Seamless and Transparent Service Discovery), Brussels, February 2004.

[55] Z. Du, J. Huai, and Y Liu. Ad-UDDI: An active and distributed service registry. In *Proceedings of the 6th VLDB Intl. Wsp. on Technologies for E-Services*, pages 58–71, 2006.

[56] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 24–35, New York, NY, USA, 1999. ACM.

[57] Tim Hsin-Ting Hu and Aruna Seneviratne. Autonomic peer-to-peer service directory. *IEICE - Trans. Inf. Syst.*, E88-D:2630–2639, December 2005.

[58] O.D. Sahin, C.E. Gerede, D. Agrawal, A.E. Abbadi, O.H. Ibarra, and J.W. Su. SPiDeR: P2P-Based Web Service Discovery. In *Proceedings of the 3rd International Conference on Service Oriented Computing*, pages 157–170, 2005.

[59] J.B. Tchakarov and N.H. Vaidya. Efficient Content Location in Wireless Ad Hoc Networks. In *Proceedings of the IEEE International Conference on Mobile Data Management*, pages 74–85, 2004.

[60] H.W. Tsai, T.S. Chen, and C.P. Chu. Service Discovery in Mobile Ad Hoc Networks Based on Grid. *IEEE Transactions on Vehicular Technology*, 58:1528–1545, 2009.

[61] Michael Klein, Birgitta König-Ries, and Philipp Obreiter. Service rings - a semantic overlay for service discovery in ad hoc networks. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, DEXA '03, pages 180–, Washington, DC, USA, 2003. IEEE Computer Society.

[62] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Inf. Technol. and Management*, 6:17–39, January 2005.

[63] Zakaria Maamar, Hamdi Yahyaoui, and Qusay H. Mahmoud. Dynamic management of uddi registries in a wireless environment of web services: Concepts, architecture, operation, and deployment. *J. Intell. Inf. Syst.*, 28:105–131, April 2007.

[64] Cristina Schmidt and Manish Parashar. A peer-to-peer approach to web service discovery. *World Wide Web*, 7:211–229, June 2004.

[65] Giuseppe Pirrò, Paolo Trunfio, Domenico Talia, Paolo Missier, and Carole Goble. Ergot: A semantic-based system for service discovery in distributed infrastructures. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 263–272, Washington, DC, USA, 2010. IEEE Computer Society.

[66] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. psearch: information retrieval in structured overlays. *SIGCOMM Comput. Commun. Rev.*, 33:89–94, January 2003.

[67] Y. Zhu and Y. Hu. Semantic search in peer-to-peer systems. In Jie Wu, editor, *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-*

Peer Networks, pages 643–664. Auerbach Publications, Taylor and Francis Group, USA, 2006.

[68] K. Arabshian and H. Schulzrinne. GloServ: global service discovery architecture. In *Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 319–329, 2004.

[69] Lican Huang. A p2p service discovery strategy based on content catalogues. In *Proceedings of the 20th International CODATA Conference*, pages 492–499, 2006.

[70] Linh Pham and Guido Gehlen. Realization and performance analysis of a SOAP server for mobile devices. In *Proceedings of the 11th European Wireless Conference*, pages 791–797, 2005.

[71] Daniel Schall, Marco Aiello, and Schahram Dustdar. Web services on embedded devices. *WEB INFOR. SYST.*, 2:45–50, 2006.

[72] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile host: a feasibility analysis of mobile web Service provisioning. In *Proceedings of the 4th International Workshop on Ubiquitous Mobile Information and Collaboration Systems, a CAiSE'06 Workshop*, pages 942–953, 2006.

[73] Guido Gehlen and Linh Pham. Mobile web services for peer-to-peer applications. In *Proceedings of the 2nd Consumer Communications and Networking Conference*, pages 427–433, 2005.

[74] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile web service provisioning. In *Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*,

AICT-ICIW '06, pages 120–125, Washington, DC, USA, 2006. IEEE Computer Society.

[75] Yeon-Seok Kim and Kyong-Ho Lee. A lightweight framework for mobile web services. *Computer Science - Research and Development*, 24:199–209, 2009.

[76] Sonja Zaplata, Viktor Dreiling, and Winfried Lamersdorf. Realizing mobile web services for dynamic applications. In *Proceedings of IFIP Advances in Information and Communication Technology*, pages 240–254, 2009.

[77] Pravin Pawar, Bert-Jan van Beijnum, Hailiang Mei, and Hermie Hermens. Towards proactive context-aware service selection in the geographically distributed remote patient monitoring system. In *Proceedings of the 4th international conference on Wireless pervasive computing*, ISWPC'09, pages 318–325, Piscataway, NJ, USA, 2009. IEEE Press.

[78] Andrew Meads, Adam Roughton, Ian Warren, and Thiranjith Weerasinghe. Mobile service provisioning middleware for multihomed devices. In *Proceedings of the 2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, WIMOB '09, pages 67–72, Washington, DC, USA, 2009. IEEE Computer Society.

[79] Andrew John Dennis Meads. A holistic approach to mobile service provisioning. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 698–702, Washington, DC, USA, 2009. IEEE Computer Society.

[80] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. A mediation framework for mobile web service provisioning. In *Proceedings of the 10th IEEE*

on International Enterprise Distributed Object Computing Conference Workshops, EDOCW '06, pages 14–17, Washington, DC, USA, 2006. IEEE Computer Society.

[81] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mwsmf: a mediation framework realizing scalable mobile web service provisioning. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, MOBILWARE '08, pages 43:1–43:7, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[82] Satish Narayana Srirama and Matthias Jarke. Mobile enterprise a case study of enterprise service integration. In *Proceedings of the 2009 Third International Conference on Next Generation Mobile Applications, Services and Technologies*, NGMAST '09, pages 101–107, Washington, DC, USA, 2009. IEEE Computer Society.

[83] Satish Narayana Srirama and Matthias Jarke. Mobile hosts in enterprise service integration. *Int. J. Web Eng. Technol.*, 5:187–213, September 2009.

[84] Satish Narayana Srirama, Matthias Jarke, Hongyan Zhu, and Wolfgang Prinz. Scalable mobile web service discovery in peer to peer networks. In *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 668–674, Washington, DC, USA, 2008. IEEE Computer Society.

[85] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The enterprise service bus: making service-oriented architecture real. *IBM Syst. J.*, 44:781–797, October 2005.

[86] Satish Narayana Srirama. Publishing and Discovery of Mobile Web Services in Peer to Peer Networks. *ArXiv abs/1007.2980v1*, 2010.

[87] Johan Wikman and Ferenc Dosa. Providing HTTP Access to Web Servers Running on Mobile Phones. Technical Report NRC-TR-2006-005, Nokia Research Center Helsinki, 2006.

[88] Feng Zhu, Matt Mutka, Anish Bivalkar, Abdullah Demir, Yue Lu, and Chockalingam Chidambarm. Toward secure and private service discovery anywhere anytime. *Front. Comput. Sci China*, 4:311–323, September 2010.

[89] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16:97–166, December 2001.

[90] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 434–441, New York, NY, USA, 1999. ACM.

[91] Norman H. Cohen, Hui Lei, Paul Castro, John S. Davis II, and Apratim Purakayastha. Composing pervasive data using iql. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, WMCSA '02, pages 94–, Washington, DC, USA, 2002. IEEE Computer Society.

[92] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. a cappella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 33–40, New York, NY, USA, 2004. ACM.

[93] Harry Chen, Filip Perich, Dipanjan Chakraborty, Tim Finin, and Anupam Joshi. Intelligent agents meet semantic web in a smart meeting room. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '04, pages 854–861, Washington, DC, USA, 2004. IEEE Computer Society.

[94] Tao Gu, Hung Keng Pung, and Da Qing Zhang. Toward an osgi-based infrastructure for context-aware applications. *IEEE Pervasive Computing*, 3:66–74, October 2004.

[95] Torben Weis, Mirko Knoll, Andreas Ulbrich, Gero Muhl, and Alexander Brandle. Rapid prototyping for pervasive applications. *IEEE Pervasive Computing*, 6:76–84, April 2007.

[96] Hung Keng Pung, Tao Gu, Wenwei Xue, Paulito P. Palmes, Jian Zhu, Wen Long Ng, Chee Weng Tang, and Nguyen Hoang Chung. Context-aware middleware for pervasive elderly homecare. *IEEE J.Sel. A. Commun.*, 27:510–524, May 2009.

[97] Yang Li, Jason I. Hong, and James A. Landay. Topiary: a tool for prototyping location-enhanced applications. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, pages 217–226, New York, NY, USA, 2004. ACM.

[98] David Bannach, Oliver Amft, and Paul Lukowicz. Rapid prototyping of activity recognition applications. *IEEE Pervasive Computing*, 7:22–31, April 2008.

[99] Yang Li and James A. Landay. Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1303–1312, New York, NY, USA, 2008. ACM.

[100] Zhiwen Yu, Xingshe Zhou, Zhiyong Yu, Daqing Zhang, and Chung-Yau Chin. An osgi-based infrastructure for context-aware multimedia services. *IEEE Communications Magazine*, 44:136–142, October 2006.

[101] Diego López-de Ipiña, Juan Ignacio Vazquez, and Joseba Abaitua. A web 2.0 platform to enable context-aware mobile mash-ups. In *Proceedings of the 2007 European conference on Ambient intelligence*, AmI'07, pages 266–286, Berlin, Heidelberg, 2007. Springer-Verlag.

[102] Adrien Joly, Pierre Maret, and Johann Daigremont. Context-awareness, the missing block of social networking. *International Journal of Computer Science and Applications*, 6:50–65, 2009.

[103] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: a mobile context-aware tour guide. *Wirel. Netw.*, 3:421–433, October 1997.

[104] Yasuyuki Sumi, Tameyuki Etani, Sidney Fels, Nicolas Simonet, Kaoru Kobayashi, and Kenji Mase. C-map: Building a context-aware mobile assistant for exhibition tours. In *Community Computing and Support Systems, Social Interaction in Networked Communities [the book is based on the Kyoto Meeting on Social Interaction and Communityware, held in Kyoto, Japan, in June 1998]*, pages 137–154, London, UK, 1998. Springer-Verlag.

[105] Youn-ah Kang, John Stasko, Kurt Luther, Avinash Ravi, and Yan Xu. RevisiTour: enriching the trourism experience with user-generated content. In *Proceedings of the International Conference on Information and Communication Technologies in Tourism*, pages 59–69, 2008.

[106] Sherry Hsi and Holly Fait. Rfid enhances visitors' museum experience at the exploratorium. *Commun. ACM*, 48:60–65, September 2005.

[107] Hiroaki Ogata, Yoshiki Matsuka, Moushir M. El-Bishouty, and Yoneo Yano. Lorams: linking physical objects and videos for capturing and sharing learning experiences towards ubiquitous learning. *Int. J. Mob. Learn. Organ.*, 3:337–350, July 2009.

[108] Felix von Reischach, Dominique Guinard, Florian Michahelles, and Elgar Fleisch. A mobile product recommendation system interacting with tagged products. In *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[109] Pravin Pawar, Satish Srirama, Bert-Jan Beijnum van, and Aart Halteren van. A comparative study of nomadic mobile service provisioning approaches. In *Proceedings of the 2007 International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 277–286, Cardiff, 2007.

[110] M. Knoll and T. Weis. Optimizing locality for self-organizing context-based systems. In *Proceedings of the 1st International Workshop on Self-Organizing Systems*, pages 62–73, 2006.

[111] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[112] Jian Zhu and Hung Keng Pung. Process Matching: A Structure Behavioral Approach for Business Process Search. In *Proceedings of the 1st International Conferences on Pervasive Patterns and Applications*, pages 227–232, 2009.

[113] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[114] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[115] Jon Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 163–170, New York, NY, USA, 2000. ACM.

[116] A. Clauset and C. Moore. How do networks become navigable? *ArXiv cond-mat/0309415*, 2003.

[117] Augustin Chaintreau, Pierre Fraigniaud, and Emmanuelle Lebhar. Networks Become Navigable as Nodes Move and Forget. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pages 133–144, Berlin, Heidelberg, 2008. Springer-Verlag.

[118] Oskar Sandberg and Ian Clarke. The evolution of navigable small-world networks. *ArXiv cs/0607025*, 2006.

[119] Olof Mogren, Oskar Sandberg, Vilhelm Verendel, and Devdatt Dubhashi. Adaptive Dynamics of Realistic Small-World Networks. *ArXiv cs/0804.1115v1*, 2008.

[120] Felix Halim, Yongzheng Wu, and Roland H. C. Yap. Small world networks as (semi)-structured overlay networks. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, SASOW '08, pages 214–218, Washington, DC, USA, 2008. IEEE Computer Society.

[121] Jakub Moskal and Christopher J. Matheus. Detection of suspicious activity using different rule engines – comparison of basevisor, jena and jess rule engines. In *Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web*, RuleML '08, pages 73–80, Berlin, Heidelberg, 2008. Springer-Verlag.

[122] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[123] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 633–642, New York, NY, USA, 2011. ACM.

[124] Jian Zhu, Mohammad Oliya, Hung Keng Pung, and Wai Choong Wong. SOLE: Context-aware sharing of living experience in mobile environments. In *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia*, pages 366–369, 2010.

[125] Mohammad Oliya, Jian Zhu, Hung Keng Pung, and Antoine Veillard. Incremental query answering over dynamic contextual information. In *Proceedings of the 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, ICTAI '11, pages 452–455, Washington, DC, USA, 2011. IEEE Computer Society.

Appendix

- ACM_Ont.owl: the ontology specification for the Application Context Model.

```
<!DOCTYPE rdf:RDF [
   <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
]>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:acm="http://www.comp.nus.edu.sg/acm/ApplicationContextModel#"
   xml:base="http://www.comp.nus.edu.sg/acm/ApplicationContextModel#">

 <owl:Class rdf:ID="ACM"/>

 <owl:Class rdf:ID="ApplicationEntity"/>

 <owl:Class rdf:ID="ContextEntity"/>

 <owl:Class rdf:ID="ContextAttribute"/>

 <owl:Class rdf:ID="ContextBelonging"/>
 <owl:Class rdf:ID="ContextAbstraction"/>
 <owl:Class rdf:ID="ContextFlow"/>
 <owl:Class rdf:ID="ContextRelationship">
     <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#ContextBelonging"/>
      <owl:Class rdf:about="ContextAbstraction"/>
              <owl:Class rdf:about="#ContextFlow"/>
  </owl:unionOf>
 </owl:Class>

 <owl:Class rdf:ID="StartingContext"/>

 <owl:Class rdf:ID="EndingContext"/>

 <owl:Class rdf:ID="ACMComponent">
   <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#ApplicationEntity"/>
     <owl:Class rdf:about="#ContextEntity"/>
              <owl:Class rdf:about="#ContextAttribute"/>
              <owl:Class rdf:about="#ContextRelationship"/>
              <owl:Class rdf:about="#StartingContext"/>
              <owl:Class rdf:about="#EndingContext"/>
   </owl:unionOf>
 </owl:Class>

 <owl:Class rdf:ID="ContextConstraint">
   <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#ContextFlow"/>
     <owl:Class rdf:about="#StartingContext"/>
              <owl:Class rdf:about="#EndingContext"/>
   </owl:unionOf>
 </owl:Class>

 <owl:ObjectProperty rdf:ID="hasComponent">
   <rdfs:domain rdf:resource="#ACM"/>
```

```
  <rdfs:range rdf:resource="#ACMComponent"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasPrimaryEntity">
  <rdfs:domain rdf:resource="#ACM"/>
  <rdfs:range rdf:resource="#ApplicationEntity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="definesContextAttribute">
  <rdfs:domain rdf:resource="#ContextBelonging"/>
  <rdfs:range rdf:resource="#ContextAttribute"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="belongsTo">
  <rdfs:domain rdf:resource="#ContextBelonging"/>
  <rdfs:range rdf:resource="#ContextEntity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="abstractsFrom">
  <rdfs:domain rdf:resource="#ContextAbstraction"/>
  <rdfs:range rdf:resource="#ApplicationEntity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="abstractsTo">
  <rdfs:domain rdf:resource="#ContextAbstraction"/>
  <rdfs:range rdf:resource="#ContextEntity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="fromApplicationEntity">
  <rdfs:domain rdf:resource="#ContextFlow"/>
  <rdfs:range rdf:resource="#ApplicationEntity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="toApplicationEntity">
  <rdfs:domain rdf:resource="#ContextFlow"/>
  <rdfs:range rdf:resource="#ApplicationEntity"/>
</owl:ObjectProperty>

<owl:DataProperty rdf:ID="hasFlowData">
  <rdfs:domain rdf:resource="#ContextFlow"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DataProperty>

<owl:DataProperty rdf:ID="hasConstraint">
  <rdfs:domain rdf:resource="#ContextConstraint"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DataProperty>

<owl:DataProperty rdf:ID="hasProtocol">
  <rdfs:domain rdf:resource="#ApplicationEntity"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DataProperty>

<owl:DataProperty rdf:ID="hasStartEntity">
  <rdfs:domain rdf:resource="#ACM"/>
  <rdfs:range rdf:resource="#ApplicationEntity"/>
</owl:DataProperty>

<owl:DataProperty rdf:ID="hasEndEntity">
  <rdfs:domain rdf:resource="#ACM"/>
```

```
    <rdfs:range rdf:resource="#ApplicationEntity"/>
  </owl:DataProperty>


  <!-- The followings are for Jena-based interpreter -->
  <owl:DataProperty rdf:ID="hasContextAttribute">
    <rdfs:domain rdf:resource="#ContextEntity"/>
    <rdfs:range rdf:resource="#ContextAttribute"/>
  </owl:DataProperty>

  <owl:DataProperty rdf:ID="hasContextValueS">
    <rdfs:domain rdf:resource="#ContextAttribute"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DataProperty>

  <owl:DataProperty rdf:ID="hasContextValueD">
    <rdfs:domain rdf:resource="#ContextAttribute"/>
    <rdfs:range rdf:resource="&xsd;double"/>
  </owl:DataProperty>

  <owl:DataProperty rdf:ID="hasContextValueI">
    <rdfs:domain rdf:resource="#ContextAttribute"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
  </owl:DataProperty>

  <owl:ObjectProperty rdf:ID="satisfyContextConstraint">
    <rdfs:domain rdf:resource="#ApplicationEntity"/>
    <rdfs:range rdf:resource="#ContextConstraint"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="satisfyConstraintWith">
    <rdfs:domain rdf:resource="#ApplicationEntity"/>
    <rdfs:range rdf:resource="#ApplicationEntity"/>
  </owl:ObjectProperty>

  <owl:DataProperty rdf:ID="hasACMStatus">
    <rdfs:domain rdf:resource="#ACM"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DataProperty>

</rdf:RDF>
```

- SOLE_Ont.owl: the ontology specification for the Sharing of Living Experience application (to be registered to the Application Context Engine).

```
<!DOCTYPE rdf:RDF [
   <!ENTITY acm 'http://www.comp.nus.edu.sg/acm/ApplicationContextModel#'>
]>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:acm="http://www.comp.nus.edu.sg/acm/ApplicationContextModel#"
   xml:base="http://www.comp.nus.edu.sg/acm/ApplicationContextModel#">

 <acm:ACM rdf:ID="SOLEACM">
    <acm:hasComponent>
            <acm:ApplicationEntity rdf:ID="User">
                    <acm:hasProtocol>192.168.1.100:8080</acm:hasProtocol>
            </acm:ApplicationEntity>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ApplicationEntity rdf:ID="Friend"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ApplicationEntity rdf:ID="ShopService"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ApplicationEntity rdf:ID="ExpIndexingServer"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ApplicationEntity rdf:ID="ExpProvider"/>
    </acm:hasComponent>

    <acm:hasComponent>
            <acm:ContextEntity rdf:ID="PERSON"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ContextEntity rdf:ID="SHOP"/>
    </acm:hasComponent>

    <acm:hasComponent>
            <acm:ContextAttribute rdf:ID="PERSON.id"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ContextAttribute rdf:ID="PERSON.location"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ContextAttribute rdf:ID="PERSON.preferences"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ContextAttribute rdf:ID="SHOP.name"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ContextAttribute rdf:ID="SHOP.location"/>
    </acm:hasComponent>
    <acm:hasComponent>
            <acm:ContextAttribute rdf:ID="SHOP.type"/>
    </acm:hasComponent>

    <acm:hasComponent>
```

```
		<acm:ContextBelonging rdf:ID="cb1">
			<acm:belongsTo rdf:resource="#PERSON"/>
			<acm:definesContextAttribute rdf:resource="#PERSON.id"/>
		</acm:ContextBelonging>
	</acm:hasComponent>
	<acm:hasComponent>
		<acm:ContextBelonging rdf:ID="cb2">
			<acm:belongsTo rdf:resource="#PERSON"/>
			<acm:definesContextAttribute rdf:resource="#PERSON.location"/>
		</acm:ContextBelonging>
	</acm:hasComponent>
	<acm:hasComponent>
		<acm:ContextBelonging rdf:ID="cb3">
			<acm:belongsTo rdf:resource="#PERSON"/>
			<acm:definesContextAttribute rdf:resource="#PERSON.preferences"/>
		</acm:ContextBelonging>
	</acm:hasComponent>
	<acm:hasComponent>
		<acm:ContextBelonging rdf:ID="cb4">
			<acm:belongsTo rdf:resource="#SHOP"/>
			<acm:definesContextAttribute rdf:resource="#SHOP.name"/>
		</acm:ContextBelonging>
	</acm:hasComponent>
	<acm:hasComponent>
		<acm:ContextBelonging rdf:ID="cb5">
			<acm:belongsTo rdf:resource="#SHOP"/>
			<acm:definesContextAttribute rdf:resource="#SHOP.location"/>
		</acm:ContextBelonging>
	</acm:hasComponent>
	<acm:hasComponent>
		<acm:ContextBelonging rdf:ID="cb6">
			<acm:belongsTo rdf:resource="#SHOP"/>
			<acm:definesContextAttribute rdf:resource="#SHOP.type"/>
		</acm:ContextBelonging>
	</acm:hasComponent>

	<acm:hasComponent>
		<acm:ContextAbstraction rdf:ID="ab1">
			<acm:AbstractsFrom rdf:resource="#ShopService"/>
			<acm:AbstractsTo rdf:resource="#SHOP"/>
		</acm:ContextAbstraction>
	</acm:hasComponent>
	<acm:hasComponent>
		<acm:ContextAbstraction rdf:ID="ab2">
			<acm:AbstractsFrom rdf:resource="#User"/>
			 <acm:AbstractsTo rdf:resource="#PERSON"/>
		</acm:ContextAbstraction>
	</acm:hasComponent>
	<acm:hasComponent>
		<acm:ContextAbstraction rdf:ID="ab3">
			<acm:AbstractsFrom rdf:resource="#Friend"/>
			 <acm:AbstractsTo rdf:resource="#PERSON"/>
		</acm:ContextAbstraction>
	</acm:hasComponent>

	<acm:hasComponent>
		<acm:ContextAbstraction rdf:ID="ab4">
			 <acm:AbstractsFrom rdf:resource="#ExpProvider"/>
			 <acm:AbstractsTo rdf:resource="#PERSON"/>
		</acm:ContextAbstraction>
```

```
    </acm:hasComponent>

    <acm:hasComponent>
        <acm:ContextFlow rdf:ID="cf1">
            <acm:fromApplicationEntity rdf:resource="#ShopService"/>
            <acm:toApplicationEntity rdf:resource="#User"/>
            <acm:hasConstraint>cf1.rule</acm:hasConstraint>
            <acm:hasFlowData>ShopService.name</acm:hasFlowData>
        </acm:ContextFlow>
    </acm:hasComponent>
    <acm:hasComponent>
        <acm:ContextFlow rdf:ID="cf2">
            <acm:fromApplicationEntity rdf:resource="#Friend"/>
            <acm:toApplicationEntity rdf:resource="#Friend"/>
            <acm:hasConstraint>cf2.rule</acm:hasConstraint>
        </acm:ContextFlow>
    </acm:hasComponent>
    <acm:hasComponent>
        <acm:ContextFlow rdf:ID="cf3">
            <acm:fromApplicationEntity rdf:resource="# ShopService "/>
            <acm:toApplicationEntity rdf:resource="#User"/>
            <acm:hasConstraint>cf3.rule</acm:hasConstraint>
            <acm:hasFlowData>ShopService.name</acm:hasFlowData>
        </acm:ContextFlow>
    </acm:hasComponent>
    <acm:hasComponent>
        <acm:ContextFlow rdf:ID="cf4">
            <acm:fromApplicationEntity rdf:resource="#User"/>
            <acm:toApplicationEntity rdf:resource="# ExpIndexingServer "/>
            <acm:hasFlowData>ShopService.name</acm:hasFlowData>
        </acm:ContextFlow>
    </acm:hasComponent>
    <acm:hasComponent>
        <acm:ContextFlow rdf:ID="cf5">
            <acm:fromApplicationEntity rdf:resource="#ExpIndexingServer"/>
            <acm:toApplicationEntity rdf:resource="#User"/>
            <acm:hasFlowData>ExpProvider.id</acm:hasFlowData>
        </acm:ContextFlow>
    </acm:hasComponent>
    <acm:hasComponent>
        <acm:ContextFlow rdf:ID="cf6">
            <acm:fromApplicationEntity rdf:resource="#User"/>
            <acm:toApplicationEntity rdf:resource="#ExpProvider"/>
            <acm:hasConstraint>cf6.rule</acm:hasConstraint>
            <acm:hasFlowData>User.id</acm:hasFlowData>
            <acm:hasFlowData>ShopService.name</acm:hasFlowData>
        </acm:ContextFlow>
    </acm:hasComponent>


    <acm:hasComponent>
        <acm:StartingContext rdf:ID="SOLEStart">
            <acm:hasConstraint>start.rule</acm:hasConstraint>
        </acm:StartingContext>
    </acm:hasComponent>

    <acm:hasStartEntity rdf:resource="#User"/>
  </acm:ACM>
</rdf:RDF>
```

- start.rule: the starting contexts specification for the SOLE application. Note that distance() is a customized operator provided by the interpreter in the ACE.

  ```
  [StartingContexts: le(distance(User.location ShopService.location) "100"^^double)]
  ```

- cf1.rule: the constraint specification for ContextFlow cf1. Note that match() is a customized operator provided by the interpreter in ACE.

  ```
  [cf1: le(distance(User.location ShopService.location) "100"^^double)
        match(ShopService.type User.preferences)]
  ```

- cf2.rule: the constraint specification for ContextFlow cf2. Note that friend() is a customized operator provided by the interpreter in ACE.

  ```
  [cf2: friend(User.id Friend.id)]
  ```

- cf3.rule: the constraint specification for ContextFlow cf3. Note that locatedAt() is a customized operator provided by the interpreter in ACE.

  ```
  [cf3: le(distance(User.location ShopService.location) "100"^^double)
        locatedAt(Friend.id ShopService.name)]
  ```

- cf6.rule: the constraint specification for ContextFlow cf6.

  ```
  [cf6: friend(User.id ExpProvider.id)]
  ```