Coalition: A Platform for Context-Aware Mobile Application Development

Jian Zhu, PengHe Chen, HungKeng Pung, Mohammad Oliya, Shubhabrata Sen, and WaiChoong Wong
National University of Singapore, Singapore
{g0500652, g0901858, dcshk, oliya, g0701139, elewwcl}@nus.edu.sg

ABSTRACT
Context-awareness, as one of the key concept to achieve ubiquitous computing, has been widely applied in various mobile applications. The resulting adaptability, along with the usage of ubiquitous technologies, makes IT invisible and yet ever closer and seamless to people’s daily living. However, context-aware computing has not yet found large success in the commercial area so far, and existing context-aware mobile applications are mostly sensor-based. We argue that a context middleware is indeed necessary to help development and deployment of mobile applications as well as managing the heterogeneous context data. In this paper, we propose such a middleware named Coalition. More specifically, we demonstrate how Coalition (i) facilitates mobility issues during mobile application execution; (ii) manages large number of application services as utilized or developed by mobile applications. The two features are explained through a case study – Sharing Of Living Experience (SOLE). In the end of the paper, we show the experimental results obtained from the prototype testing and simulation study.

Keywords: context middleware, mobile application, context data management, mobile space, mobile service mangament, experience sharing.

1 INTRODUCTION

The rapid advances in the technologies of mobile computing and wireless communications have brought opportunities for various mobile applications running on handheld devices. Through enabling technologies (e.g. WiFi and 3G), mobile application users may access services (e.g. mobile payment) and share information at anywhere, anytime. Furthermore, along with the marked improvement in mobile device capabilities (e.g. CPU and memory), the provision of services by these devices is promising. For instance, interactive photography can be achieved by requesting the mobile device (with camera) owner to take a photo at real time. The resulted mobile hosted application services (mobile services for short) would potentially change the current mobile application development landscape, by including the role of mobile service provider in the provisioning of these mobile services.

To be more effective, applications running on mobile platforms should be aware of the user context and adapt to any change occurred, e.g. in the surrounding environment. For instance, consider a taxi booking application that filters search results based on the user location and returns only the relevant results. Fortunately, the recent success of ubiquitous technologies, such as RFID tagging, GPS and wireless sensors, enables the detection and monitoring of such contextual information. Context-awareness, as one of the key concept to achieve ubiquitous computing [1], would make IT invisible and yet seamlessly integrated with our daily living.

Our early work in this area include context modeling and reasoning techniques [2], while our recent effort focuses on the development of a context middleware known as Coalition to support context-aware (mobile) application development. Through its context data management layer, Coalition is capable of locating and extracting relevant context data from large number heterogeneous data sources distributed over many different operating environments [3]. Above the data management layer is its service management layer, which facilitates the organization and discovery of services [4] via a method known as Location Aware Service Organization and Discovery (LASOD). Applications access the functions of data management layer through its SQL-like context data query interfaces. Coalition supports the notion of physical spaces. Each space has various affiliated data sources (e.g. sensors and computing devices) whose contextual information, described by a context data model, could be accessed through a Physical Space Gateway (PSG). A physical space can either be static (such as a home) or mobile (such as a person wearing body sensors and a mobile phone). With the use of mobile phones as PSGs, we can effectively create a realistic large scale and yet low cost testbed for further research in the field of context-aware ubiquitous computing.

In this paper, we first investigate the design, integration and deployment issues of mobile
applications in the Coalition environment. More specifically, we demonstrate how Coalition can be used as a platform to: (1) help the development of context-aware mobile applications; (2) facilitate mobility issues during the application execution; (3) manage large number of (mobile) services as utilized or developed by applications. In the second part of the paper, we will use Sharing Of Living Experience (SOLE) as a case study, SOLE will have real mobile users playing the roles of Mobile Service Providers (MSPs) and Mobile Service Consumers (MSCs). The MSPs share experiences regarding real-world entities, while MSCs can access this information ubiquitously. The usage of Coalition for realization of context-awareness in SOLE is emphasized. In the end of the paper, the performance of Coalition in supporting mobile application development is learnt through simulation studies and real-time tests.

The rest of this paper is organized as follows: Section 2 presents the related work done in this area. Section 3 provides an overview of Coalition middleware. Section 4 describes the extended features of Coalition to support context-aware mobile application development. Section 5 discusses our SOLE and its relationship with Coalition. Section 6 shows the performance measurement of Coalition. Finally, Section 7 concludes the paper and highlights our future work plan.

2 RELATED WORK

According to a study commissioned by mobile application store operator – GetJar [5], the mobile application market will reach $17.5 billion by 2012. Besides the traditional mobile network operators and phone manufacturers, more and more independent and freelance developers are being involved in the development of mobile applications. Though there are many mobile platforms (e.g. Symbian [6], iOS [7] and Android [8]) in the mass market, context-aware mobile computing has not yet found large success in the industrial and commercial area. There are many reasons, and the two major ones include: (1) the lack of effective and agreed methods or platforms for seamless exchange and intelligent processing of data among the heterogeneous context information sources; (2) the under development of software engineering methodology for the management of ubiquitous applications and their related services.

Many context middleware systems have been proposed in the last two decades, but there is very little discussion on context modeling and managing for mobile entities. HiCon [9] is a hierarchical context middleware with good scalability, which creates PocketMon to model and manage the user context through their mobile devices. However, HiCon does not provide a general context modeling and managing solution for mobile entities. Gaia [10] is a distributed context middleware that coordinates software entities and network devices in a physical space. By defining active space, Gaia supplies services to manage context of entities inside the space. However, most context entities involved in active spaces are static. Though services related to a moving person have been developed, no clear and detailed description of data management for mobile entities has been given. CoBra [11] is an agent based middleware to support context-aware services in intelligent spaces that embeds intelligent systems into physical spaces to provide pervasive computing services to users. CoBra uses context broker to acquire and fuse context from heterogeneous sources into a coherent model which is shared with all context entities inside the smart space. However, CoBra does not consider the specialties of mobile entities for context data management and focuses on a single space without the discussion on relations among different spaces.

For mobile application development, the current landscape mostly prospects the mobile device as a gateway for the user to acquire outside world information. Nevertheless, existing mobile devices gain the capability to run business logics and provision them as services. With these mobile services, future devices will play a more active role by interacting with each other in the ubiquitous environments. This vision is also realized in Nokia’s “mobile web server” concept [12]. However, we anticipate the efficiency for discovering these mobile services will be the main concern of usage if these services are not deployed in a well coordinated fashion. The unreliable availability of mobile service providers, due to reasons such as mobility, would further complicate the discovery process. Moreover, issues such as security, privacy, and service administration should also be considered. Existing approaches for service provision and discovery mainly fall into two groups: centralized and decentralized. The centralized ones (e.g. UDDI [13] and Jini [14]) were popular in the early days due to the simplicity in their implementations and ease of administration; however, they do not scale well. In the decentralized approach, scalable Peer-to-Peer (P2P) techniques such as Chord [15] and CAN [16] are often used. Nevertheless, portable devices like smart phones can barely maintain P2P connections, due to their resource and communication constraints. Besides, data locality is often not maintained in most of these approaches, which makes them unsuitable for range queries, e.g. proximity-based search.

We are proposing a context middleware – Coalition in handling all the above issues in the development of context-aware mobile applications. More specifically, the two layers in Coalition: Context Data Management Layer and Service Management Layer handle the management of context information from heterogeneous sources (including mobile devices), as well as the provision
and discovery of mobile services. Besides, our newly proposed mobility support function would help application development in mobile environments in an easy and efficient manner. In the following section, we will briefly introduce Coalition and its key characteristics proposed in our previous work.

3 COALITION OVERVIEW

Coalition, previously known as CAMPH [3], is a context middleware designed for pervasive homecare. The design goal of Coalition is not just to be part of a particular homecare application, but rather to be an open, programmable and reusable infrastructure. By considering this, it is designed as a service-oriented architecture: the various system functionalities such as context data acquisition and reasoning, service registration and discovery, are all designed and deployed as system services for developers and end-users to access. The middleware architecture consists of four logical layers as shown in Figure 1.

- **Physical Space Layer.** This layer consists of real-world physical spaces that represent the various context data sources. A physical space is an operating environment (e.g. people’s homes and offices) that provides context data from its attached entities such as sensors, actuators and computing devices. It mandates all the 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). We use a simple and pragmatic attribute-value approach to context modeling at the physical spaces and define a context attribute as a specific kind of context data, e.g. location.

- **Context Data Management Layer.** To efficiently manage physical spaces and support context data lookup, we further define the concept of context space in this layer. A context space can be thought of an abstraction of a collection of physical spaces having similar attributes. Examples of context spaces are OFFICE, SHOP and PERSON, as depicted in Figure 1. The physical spaces in a context space are organized as peers in several semantic clusters (each cluster represents a particular context attribute), over which the context queries for data acquisition are processed [17]. On top of this layer, a declarative, SQL-based context query interface is implemented for services and applications to acquire context data or subscribe to event notifications.

- **Service Management Layer.** This layer considers the organization and discovery of services, including both system services and third-party services. In many of real-world application scenarios, the geographical location of a requester with respect to that of the service provider can greatly affect the relevance of the discovered services. This observation has motivated us to take a Location-Aware approach for Service Organization and Discovery (LASOD). The location information of service peers is preserved using the locality preserving Hilbert Space Filling Curve (SFC) [18]. Besides, the Small World Model [19] is applied in our novel Source Sampling method to make the resulting network adaptable and navigable. To further enhance the context-aware capability during service organization and discovery, the context data may be queried and utilized from the context data management layer.

- **Application Layer.** Context-aware applications lie on top of the service management layer. They can interact with middleware services to retrieve contextual information, use third-party services, or orchestrate services to fulfill tasks requiring the collaboration of multiple services. The query interface for service discovery is both keyword-based and location-based. In the latter case, the applications can limit the search scope to an area or to a geographical range. We will use Sharing Of Living Experience (SOLE) as a case study in the second part of this paper.

In the following, we highlight the mechanisms used for context data and service management. Section 4 will discuss the extra features developed for context-aware mobile application development. For the other mechanisms of Coalition such as context space schema matching, activity recognition, interested readers may refer to [3] for details.
3.1 Context Data Management

As mentioned before, we apply the concept of physical space to manage context data from heterogeneous sources, as this approach provides scalability and flexibility while reducing complexity for applications to utilize. An application (e.g., Healthcare) may rely on one or more physical spaces (e.g., the person, his home and the hospital) to finish its desired tasks such as remote monitoring and these physical spaces may again be utilized by other applications to achieve reusability. Each physical space defines a set of context attributes, where some of them are directly derived from the sensors of that type and some are based on context reasoning over the data collected from multiple sensors. To allow applications or other spaces to access and acquire context data from local sensors or other computing devices, each physical space is associated with a Physical Space Gateway (PSG). The PSG is a logical software module that can be deployed at any computer of choice in the physical space. For example, the PSG of a person can be the person’s PDA or smart phone, and a home PSG can be a dedicated PC at the person’s home. Figure 2 shows the functional components of a PSG.

To efficiently locate and acquire context data from the desired physical space, the context management layer of Coalition manages all physical spaces (PSGs) using semantic P2P overlays. More specifically, all physical spaces are first mapped to their conceptual context spaces by using the respective context schema that specifies all the context attributes within the space. In Coalition, each context space has a space schema that describes the characteristics of the physical spaces it prototyes. As an example, the space schema in PERSON embodies the properties of all physical spaces representing “persons” in the real world, and it has attributes such as name, age and location etc. The mapping is thus done between the context schema of the physical space and the space schema of the target context space. Of course, in reality different physical spaces may have different properties or different names for the same space schema attribute. Coalition allows such heterogeneity by providing the matching between context schema with the space schema in the middleware, and if there are additional attributes that need to be defined, the space schema will also be dynamically updated so to provide a unified view for all physical spaces within the same context space.

The detailed schema matching mechanism is discussed in [20]. Once the physical space is mapped to the corresponding context space, the associated PSG will join the semantic overlay created by Coalition. Each context space is mapped to a logical, one-dimensional ring network of semantic clusters, where each semantic cluster corresponds to a context attribute in the space and is implemented as a P2P network by PSGs. As a physical space may not provide all current attributes in the corresponding space schema, its PSG will only join those clusters for the attributes it has. The rings for all context spaces are created and maintained by the context space managers at the system servers. A Context Space Gateway (CSG) is created as a special cluster in each ring to serve as the entrance for routing of context queries (see Figure 1). The CSG connects to the ring as any other cluster while it is actually a subcomponent of the physical space manager rather than a P2P network. It maintains the ring topology and creates new semantic clusters when the context space evolves during the process of schema matching. A PSG can leave the semantic clusters it joins at any time. The leaving is automatically detected by the P2P protocol at the neighboring peers. The PSG must send a new registration request in order to rejoin the infrastructure.

Through the context management layer, Coalition provides a declarative, SQL-based query language interface for applications to acquire context data from physical spaces. Two classes of context queries are supported: data collection queries and event subscription queries. The following two queries exemplify the two query classes.

**QUERY 1:**
```
SELECT temperature, heartbeat
FROM PERSON
WHERE name = “Jennifer”
```

**QUERY 2:**
```
SUBSCRIBE isVacant
FROM HOME
WHERE location = “ION Orchard”
```

We have also extended the default SQL query language by supporting new keywords. For instance, a CONT keyword can be used to specify a data acquisition query that is continuous and push-based; the SAMPLE INTERVAL and LIFETIME clauses are designed to indicate the sampling interval and duration of the continuous data acquisition. In addition, Coalition has proposed a hierarchical reasoning scheme from the low-level sensor data reasoning at PSG to high-level physical space data reasoning at the context space. Applications may also have their own reasoning scheme designed in the application level. The interested readers may refer to [3] for details.

![Figure 2. Functional components of a PSG.](image-url)
3.2 Service Management

Above the context data management layer is the service management layer, which facilitates service organization and discovery for both system services and third-party services. Applications may utilize any service in this layer during their development or execution time. The detailed service management is via a scheme known as Location-Aware Service Organization and Discovery (LASOD) [4]. LASOD organizes areas and the service providers within each area in two distinct tiers for local administration and scalability purposes.

- **1st Tier (Area Organization).** In this tier, administrative areas are organized based on a predefined geographical tree. It specifies the semantic name and the geographical boundary of each area. The tree also specifies if an area is contained by another one, i.e. area hierarchy. All the service providers are assumed to reside in one of the specified areas, as illustrated in Figure 3. The complete specification of each area is maintained in a special peer node known as a superpeer. It represents the geographical center of the area, although it may be physically located anywhere. Each superpeer performs some fundamental operations: it helps new service providers to join the area and facilitates cross-area queries by maintaining links with superpeers of neighboring areas. The role of a superpeer can be played by a dedicated server or by any of the service providers’ servers within the area. While the former may have full administrative and management controls, the latter may only perform the mentioned fundamental operations.

- **2nd Tier (Local Area Service Organization).** After dealing with the area organization, the actual service providers within a specific area are managed in the second tier. In this tier, P2P concepts are applied to achieve scalability and to mitigate the negative effects of joining and leaving of (mobile) 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.

To enable location-aware service discovery, we derive the identifier for each service based on its geographical location. The identifier is represented in a binary form and composed of two parts: areaID and peerID. As an example, the identifier for service peer c in Figure 3 is 1110 1001. The areaID is determined by the area position in the geographical tree. Figure 4 illustrates the allocation of areaID for the areas in the first tier of Figure 3. Once the areaID is determined, each superpeer is connected to the first superpeer whose areaID is greater than that of its own in terms of decimal value. While the areaID reflects the coarse-grained location information of a service peer, the peerID is supposed to contain the fine-grained location information. For this purpose, we deploy the Hilbert Space Filling Curve (Hilbert SFC) [21] for each local area. For instance, consider Figure 5 (top) for an area consisting of seven service peers. Initially, the curve consists of lines which lay over 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. The set of cell IDs generated in different iterations can be represented in a hierarchy which we call Hilbert construction tree (Figure 5 bottom). In practice, we target a predefined cell size of 1m² 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. After assigning identifiers, if two service peers follow each other on the curve, they are supposed to maintain a connection to each other.

![Figure 3](image-url) The two-tier service organization architecture.

![Figure 4](image-url) Labeling areaIDs for a geographical tree.

![Figure 5](image-url) Using Hilbert SFC for assigning identifiers to service peers (top). The IDs generated in different iterations form Hilbert construction tree (bottom).
To make the network adaptable and navigable, LASOD adapts the Small World Model [19] in our novel Source Sampling method. The method generates long-range links (i.e. shortcuts) between peers in a probabilistic manner along with the process of query routing. To do so, each service peer is augmented with $k$ number of long-range links for greedy routing. During a query routing, every service peer that receives the query will perform a sampling process for the creation of long-range link, i.e. from the service peer where the query originated to itself. The sampling is based on our modified Kleinberg’s hierarchical model and the probability is derived from the identifiers of the two service peers. Using the method, the routing efficiency is improved, when compared with early approaches. Besides, we have also made use of long-range link indexing [4] to improve the fault resilience of the framework. The indexing scheme takes advantage of the long-range links created during the source sampling process and is used for cross-area routing, which helps to relieve the routing efforts imposed on superpeers. In Section 6, we show through experiments the benefits of using the scheme. As the network model and routing mechanism are not the main focus of this paper, the interested readers may refer to [4] for details.

4 COALITION FOR MOBILE APPLICATIONS

Two main issues arise when dealing with applications on mobile devices:

- **Mobility.** When using mobile devices, the connection or the session required by the application will be inevitably lost during the user movement. The naïve way to work around this problem is to let the user or the application re-initiate the connection or session. A much preferred solution deals with the issue transparently.

- **Resource Constraints.** When services are hosted by personal mobile devices, we need to distinguish the limitations of mobile devices as compared to the conventional desktop, as mobile devices have limited memory, processing power and battery life, and their availability may be intermittent.

We have devised features to make Coalition an application-independent platform to support mobility and service provision on low-end devices. Mobile applications are only required to utilize the APIs provided by Coalition to handle tasks such as mobility callback, mobile service registration and discovery. In the following, we will discuss the two features in detail.

4.1 Mobility Support

4.1.1 Mobile Space

To support context-aware applications for mobile entities, we define mobile spaces to model and manage context data pertaining to mobile devices. More specifically, for any mobile physical space (e.g. a person), if its PSG (e.g. the smart phone carried by him) is mobile, we call this physical space mobile physical space, the corresponding context space mobile context space, and the PSG mobile PSG (M-PSG).

4.1.2 Availability Management for Mobile Spaces

Because a physical space may lose its network connectivity due to mobility or poor network coverage, availability must be properly managed to support context data management and context aware applications. We define the availability of a physical space (including M-PSG) as the ability of reaching-out to other PSGs through communication networks such as the Internet. Additionally, we define the concept of callback as a service that provides a notification for any change about availability status of a particular MPSG. Two new system services are introduced to Coalition to handle availability management for a mobile space: Availability Updating Service (AUS) and Application Callback Service (ACS). AUS handles the reachability of a mobile space for context data acquisition via networks. Each PSG creates a session with Coalition during registration defined as Middleware Session ID (MSID) which is recorded by a MSID Table with availability information. As a result, any variation about availability status of a specific MPSG can be updated automatically and benefit other services accordingly. ACS handles the disruption and resumption of any application due to the availability issue of MPSGs. Each application launch is given an Application Session ID (ASID) by its respective host (i.e. MPSG). By combining MSID and ASID, a unique Session ID (SID) is generated to identify a particular application running in a specific MPSG. By using this ACS, an application in a MPSG can register a callback upon the availability change of one particular MPSG. Whenever a MPSG issues updating through AUS, all callbacks registered upon that particular MPSG are retrieved as a list. With the availability information stored in middleware, application callback notifications are sent to all MPSGs in the list. In the following we will discuss the two services in details.

4.1.2.1 Availability Updating Service (AUS)

Each MSID is a named pair identifier: context domain indicator showing the context domain that the MSG belongs to, and counter using a numerical value to uniquely identify each MSG. The entire AUS can be divided into two parts: Middleware Session Processor inside Coalition server and Availability Updating Processor inside each MSG (Figure 6).

- **Middleware Session Processor.** It generates the
MSID information for each MPSG during registration, and manages it through the life cycle of the MPSG. This component contains two sub-components: MSID Table and Query Handler. MSID Table records availability information of each MPSG with respect to MSID. It contains two information fields: MSID and Availability Information which contains the IP address and port number of a MPSG. To increase the scalability, many MSID Tables are created and distributed into different CSGs, and each CSG maintains its own MSID table for the registered MPSGs. Query Handler handles different kinds of MSID related queries from MPSGs including following operations: insert, update, retrieve and delete. MSID can be utilized to identify its CSG information to route its related queries to the correct place.

- **Availability Updater**. It detects any availability change of a MPSG, and updates it with Middleware Session Processor. This component contains two sub-components: Network Monitor and Availability Updater. Network Monitor collects the availability information of a MPSG persistently. If a change is detected, an availability updating alert is generated and forwarded to Availability Updater which contains the MSID information of this MPSG. Subsequently, Availability Updater communicates with middleware to update the availability information.

4.1.2.2 **Application Callback Service (ACS)**

ACS resumes halted application sessions caused by any unforeseen availability problem of related MPSGs. Based on the previous definition of callback, we further define the MPSG requesting the callback as a Callback Caller and the MPSG for which the callback is issued upon as a Callback Callee. The entire ACS can be divided into two parts: Application Callback Processor inside Coalition and Application Callback Handler inside each MPSG or application server (Figure 7).

- **Application Callback Processor**. It manages application callbacks, handles callback queries and distributes application callback notifications. This component contains three sub-components: Callback Table, Query Handler and Callback Notifier. Callback Table records all application callbacks issued by applications residing in different MPSGs. A callback table contains many named pairs: Callback Callee and Callback Caller. Similar with MSID Tables, many Callback Tables are distributed to different CSGs to achieve scalability. Query Handler processes callback queries from different MPSGs such as callback registration and deletion. Fired by AUS, Callback Notifier receives a callee MSID and searches callback callers with respect to this callee from corresponding Callback Table. After obtaining the caller list, it retrieves availability information of all callers from middleware, and then generates and distributes application callback notifications to all caller MPSGs.

- **Application Callback Handler**. It issues callback queries to Application Callback Processor as well as processing received application callback notifications to resume previously halted applications. This component contains three sub-components: Query Controller, Notification Controller and Network Controller. Query Controller receives a message from an arbitrary application, generates and issues one callback query to Application Callback Processor. Notification Controller receives and analyzes each application callback notification on behalf of the halted applications. Whenever receiving an application callback notification, it parses the ASID, MSID and availability information of the callee, and then interprets the ASID and passes the MSID and availability information to the corresponding application to resume previously affected session. Network Controller delivers each query to as well as receiving application callback notifications from Application Callback Processor.

4.1.3 **APIs for Mobility Support**

To give a more concrete idea for application developers to utilize the mobility support services of Coalition, Figure 8 shows the APIs for mobility support and their usage flow.

During the normal operating of a MPSG (Figure...
receive each application callback notification from middleware, and then leverages on Notification Controller to serve applications.

4.2 Mobile Service Provision

4.2.1 Three-tier Service Provision Architecture

LASOD is designed as a platform for any application to leverage on so to publish and discover services. To utilize the functions of LASOD, three logical software components are defined for the application (as represented in the Application Components of Figure 9), which includes (1) user interface specific which interacts with the user to allow functions of the application to be invoked; (2) application specific which performs application-specific logic; and (3) service management specific which interacts with the underlying service management functions of LASOD to complete tasks such as service registration and query routing. The first two components should be designed by the application itself; while the last component is created by LASOD, which we name as Service Mediator (SM). More specifically, it mediates the application with the LASOD functions: on one hand, SM allows applications to utilize the indexing and routing mechanisms of LASOD to publish their application services or discover available ones (e.g. for dynamic service composition); on the other hand, SM keeps the service information of each application so that it helps LASOD to direct any application-specific query to the responsible application service. Indeed, the above three components defined may reside in one or be distributed to more than one computing devices. This allows applications to utilize the functions of LASOD without actually deploying LASOD components (e.g. indexing and routing). A typical example is for mobile service provision, where it may not be feasible for mobile devices to perform the peer operations as expected in LASOD, such as facilitating P2P connectivity and routing of queries. By distributing the SM component to a more capable machine (e.g. a desktop), this machine can handle the communications between the mobile service and LASOD. Besides, with the help of SM, it maximizes the flexibility of LASOD, as the service
registration does not need to differentiate between local service and mobile service provisioning.

To cater for resource constraints of Mobile Service Providers (MSPs), the service management layer of Coalition organizes the service providers within each area according to their capabilities. More specifically, relatively powerful service providers can take the role of a proxy to host services delegated by the resource-constrained mobile devices, i.e. by running the respective SM component. This results in a three-tier architecture for service organization in Figure 9, as compared to the two-tier architecture in Figure 3. The third tier is for MSPs via less capable mobile devices. We assume they can connect to the Internet in one or more ways, such as via GPRS, WiFi, or 3G. Each device in this tier shares its service through a proxy in the second tier, which is a service peer with supposedly higher availability and resources. For instance, if a doctor wishes to provide an emergency care service anytime anywhere, he can run a mobile care application in his mobile phone which announces his presence. When entering a specific area, the mobile care application will look for a proxy that will in turn register the mobile service in LASOD for discovery and invocation.

4.2.2 Service Peer Modeling

Figure 10 shows the new functional model of a service peer in LASOD. Each service peer can perform two basic tasks: service provision and service discovery. The service provision relates to publication and indexing of services; while the service discovery deals with lookup and invocation of the desired services. As a service peer may host multiple services, local service management is essential: it controls the start/termination of a service and may support service migration. The service registration component handles service registration and publication. In the case there are services delegated by MSPs, the service peer acts as a proxy. To enable efficient service discovery, registered service information such as name and description are indexed using the DHT technique over the respective P2P network for each local area. The range link indexing component is to relieve the workload of a superpeer and also improve the network resilience.

The peer link maintenance component is for link monitoring among service peers. In case there is a peer link failure, the corresponding link repair function will be triggered. The rest of components perform tasks related to query generation, routing and processing which are necessary for service discovery and they were discussed in [4]. Once the desired service is looked up, the requester may invoke the service directly through LASOD.

The software architecture of a service peer has been implemented in the Coalition toolkit. Local applications can simply rely on the following statements to get their services published; assuming the local service peer $sp$ (on the same machine) has been started.

ServiceData service = new ServiceData(name, address,type,description,reference);
SM mediator = sp.registerService(service);
mediator.publishService();

In the above statements, the ServiceData service is used to describe the registered service, including its name, address, type, description and reference. The type indicates the category of the service so the application-specific query received by the $sp$ can be directed to the correct service. The description of the service is utilized for service matching. Application developers may use richer descriptions such as those parsed by WSDL file to support Web service function discovery. The reference refers to the mean to trigger the service in case it is a software service. It can be a function reference, a Web service URL or a TCP/IP socket. For local service registration, the reference to the newly created SM object is returned back to the application, so application can publish its associated service using the mediator. Note that once the application service is bound to a particular SM instance, it may utilize the functions provided by the SM instance to communicate with LASOD and to use its resources and functions. Figure 11 illustrates the relationships and interactions among the components – Application, ApplicationService, ServicePeer and ServiceMediator.
4.2.3 Mobile Service Provision

As mentioned before, the application service is not necessarily hosted on the same machine as where the service peer is running. When services are supposed to be provisioned on a separate machine (e.g., for mobile service provision), the responsible service peer is acting as a proxy. In this case, two issues must be resolved: (1) the discovery of the proxy and (2) the communication method. For (1), we have designated a port for each service peer capable of being a proxy to listen on, so that if the MSP and the proxy are in close proximity, the proxy can be discovered via WiFi broadcast. In this case, the MSP does not necessarily possess a public IP for its service provision. Alternatively, a list identifying the address of potential proxies can be retrieved from our dedicated Web server. For (2), all kinds of invocations are through the Web service mechanisms. Therefore, the Web service URL of the mediator will be returned when registering application services to the service peer.

5 CASE STUDY

To demonstrate the design, integration and deployment of mobile applications in the Coalition environment, we have chosen Sharing Of Living Experience (SOLE) as the case study because of its richness in application and user behaviors. Moreover, we can easily extend SOLE to support other types of applications such as service recommendation based on people’s experience rating. Our SOLE application framework will be differentiated from the existing information sharing work (e.g. RevisiTours [22], Sentient Graffiti [23], foursquare [24] and Gowalla [25]) in the following aspects: (1) it is generic and scalable, by not assuming a specific application scenario, e.g. museum or tourism. It allows users to share and retrieve experiences about any entity at any time (without requiring RFID tags); (2) it 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 mobile services from these devices; (3) it is context-aware by considering the location, time, and preferences of the user in discovery and delivery of experience information. In this section we will briefly explain the design of SOLE, but with special emphasis on Coalition as a platform to help make SOLE development more efficient and effective.

5.1 SOLE Overview

The detailed description of SOLE application can be found in [26]. The application consists of three participants, which we call them as SOLE Application Server (SOLE-AS), SOLE Application Service Provider (SOLE-ASP) and SOLE Application Service Consumer (SOLE-ASC). 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 shop), and the value describes certain aspects of the experience, e.g. who shared it, when and where it was shared. SOLE facilitates sharing of experiences through distributed SOLE-ASs. Each SOLE-AS has an ID and is responsible for a part of the keyword space based on the Distributed Hash Table (DHT) technique: for storing the index of the experience data, the name of the entity of interest is hashed to a key and is assigned to the first SOLE-AS whose ID is equal to or greater than the key value.

Users use SOLE-ASP/ASC to share or retrieve experience information. As SOLE allows the actual experience to be stored on the mobile device, when retrieving data from the device, the respective SOLE-ASP is also considered as a Mobile Service Provider (MSP). In the current design of SOLE-ASP, the functions to support experience retrieval on the SOLE-ASP's mobile device are all exposed as Web services, which is similar to the case for the SOLE-AS. The interactions among SOLE components for experience retrieval are illustrated in Figure 12.

5.2 SOLE Application Service in Coalition

With Coalition, SOLE can rely on the service management layer (i.e. LASOD) for the organization and discovery of its services. There are two major types of services in SOLE: (1) services provisioned by SOLE-ASs; (2) services provisioned by SOLE-ASPs. The former represents the conventional client-server model where SOLE-ASPs/ASCs share or retrieve experience through dedicated SOLE-ASs; while the latter considers peer-peer model and SOLE-ASC could directly retrieve data from SOLE-ASP. Figure 13 shows the two service provisioning models for SOLE in LASOD. Sample service data for SOLE-AS service is shown as follows:


Figure 12. Illustration of experience retrieval by SOLE-ASC. Step 4 is involved when the experience data is stored on the SOLE-ASP.
Indeed, when SOLE is integrated with LASOD, it can use the service browsing and discovery features of LASOD. More specifically, if an entity of interest has a corresponding service registered with LASOD, 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-ASCs due to the location-awareness of LASOD.

5.3 SOLE Mobility

If the experience data is stored on a mobile device (instead of a distributed server), efficiently locating the device by the SOLE-AS/ASC can be challenging. In Coalition, such a problem can be handled in two phases: (1) retrieving the current location of the SOLE-ASP by issuing the query “SELECT location FROM PERSON WHERE id = asp_id,” where asp_id is the SOLE-ASP’s ID; (2) sending a service discovery query to LASOD limiting the search scope to the proximity of SOLE-ASP’s location. Once the reference of the SOLE-ASP is retrieved, its service can be invoked, but if the reference corresponds to a proxy, the proxy will instead direct the query to the right SOLE-ASP.

However, the SOLE service can be continuous sometimes, e.g. streaming for sharing a video. As a result, the transaction can be affected by the movement from SOLE-ASP/ASC. How to make the mobility influence transparently for users is another challenging task. As mentioned in Section 4.1, Coalition creates two special system services to solve this problem: Availability Updating Service (AUS) and Application Callback Service (ACS). On one hand, AUS helps a user detect availability (namely network connection) changes and update the changes with middleware. On the other hand, ACS memorizes previous halted application sessions and activated by AUS to generate and distribute application callback notifications on behalf of the halted applications. As a result, previously halted sessions can be resumed and proceeded. In addition, the above two services can work automatically so that users will not be disturbed by mobility effect.

5.4 SOLE Context-Awareness

Location, as a relevant context, is already embedded in SOLE by leveraging on LASOD. More specifically, the SOLE-ASC/ASP may browse for entities within the local range or issue keyword-based queries to select an entity of interest.

The context of a SOLE-ASC can be exploited for pushing relevant experience information. The pushing of information can also be personalized by taking the person’s preference into consideration. For instance, the SOLE-AS can subscribe with Coalition to detect the event of users entering a mall, e.g. “SELECT name FROM PERSON WHEN LocationChange WHERE new_location = ION Orchard”. Later when the LocationChange event occurs, Coalition will notify the SOLE-AS with the name of the person and then the SOLE-AS can get the preferences of the mobile users by issuing another query “SELECT preference FROM PERSON WHERE name = person_name AND app_type = SOLE”.

Finally, the experience meta-data can be used in a number of ways. For instance, “Time” can be used to filter old experience data; “Friend List” can be used to filter irrelevant SOLE-ASCs. Note that the proposed data schema is extensible so that applications may define their own attributes to be matched in the SOLE-AS.

6 EXPERIMENTAL RESULTS AND PROTOTYPE IMPLEMENTATION

We present our current performance evaluation results on the Coalition middleware for the support of mobile application development. This section includes three major parts: the experimental results for mobility support in the context management layer; the simulation studies for routing behaviors and service provision in the service management layer; and the prototype implementation for real-time tests of the SOLE mobile application.

6.1 Mobility Support

6.1.1 Mobile PSG Registration and Deletion

We first studied the overheads introduced by the session id during registration process. PSG registration time is defined as the time interval between the submission of the registration request and the reception of the registration reply. The PSG registration time was measured and compared for two middleware versions – one using the session id and the earlier version without session id. Figure 14 shows the measurements carried out using PSGs for ‘PERSON’ context domain. The results indicate that
a small overhead is introduced in the registration process due to the introduction of the session id mechanism. However, the observed overhead is not very significant compared with the entire registration time, so it can be safely concluded that the introduction of session id mechanism does not cause an appreciable change in the registration process.

6.1.2 Callback Registration and Deletion
We next analyzed the time required for callback registrations and deregistration for all the possible scenarios. Table 1 outlines the four different test cases we considered for the experimental measurements. The results in figure 15 indicate that the registration time for the first and second cases are larger than the time required for the third and fourth cases. This is expected as the creation of new callback instances should take more time than just appending the new callback information to the existing instance. Also, comparing the first and third case to the second and fourth case clearly shows that the registration time increases when the pair of caller and callee belongs to different context domains. This increase is due to the processing time required to transfer information between two different context domains. The results also indicate that the callback registration process does not incur any significant additional processing overhead.

Table 1. Cases for Callback registration and deletion.

<table>
<thead>
<tr>
<th>Case</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Both the caller and callee belong to the same context domain</td>
</tr>
<tr>
<td>2</td>
<td>The caller and callee belong to different context domains</td>
</tr>
<tr>
<td>3</td>
<td>Same as case 1, a callback has already registered upon the callee</td>
</tr>
<tr>
<td>4</td>
<td>Same as case 2, a callback has already registered upon the callee</td>
</tr>
</tbody>
</table>

6.1.3 Availability Updating and Callback Notification Management
We next studied the overheads incurred in the processes of availability updating and the callback notification distributions. We measure the overhead as the total time taken for the middleware to receive a mobility update and distribute the notifications to the registered callers. The measurements are carried out by varying the size of the callback list and the results obtained are shown in figure 16. The results indicate that there is an increase in the time required for the availability updating and the callback notification distribution as the size of the callback list increases. This is intuitive as an increase in the callback list size will also require more processing to distribute the callback notifications. However, it can be observed that the increase curve becomes smooth after the initial increase. This indicates that the proposed callback mechanism does not create any large additional overhead and is stable with respect to the increase in the callback list size.

6.2 Service Management
In [4], we have demonstrated the network navigability and framework resilience by using Source Sampling and the long-range link indexing mechanism. The simulation is carried out on a 1024m*1024m map with the geographical tree generated randomly. In a trial run, the depth of the tree is 6 and the number of tree nodes (i.e. areas) is 364. The service peers are randomly distributed on the map (certain areas may be empty if having a low-density distribution of service peers) and for each round of query routing, two service peers are randomly selected as the source and the target of the query. In this paper, we further study the effect of long-range link indexing mechanism in affecting the network routing behavior. More specifically, the routing efforts imposed on superpeers and service peers are compared. The routing effort is defined as...
the average number of peers (superpeers/service peers) involved in the routing of a single query, and it reflects the required processing workload. Figure 17 presents the results. We observe that in our approach (Figure 17(a) and 17(c)), 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 long-range indexes are created. It justifies that the superpeer function in LASPD can be performed by any non-dedicated server and the failure of a superpeer will not affect cross-area routing severely [4]). We also note that with larger $k$ value (i.e. number of long-range links augmented for each service peer), it requires less number of queries to achieve the same effect in reducing the routing effort. The reason is simply due to the fact that more long-range links are allowed to be created with the same number of queries routed. However, the performance gain becomes smaller when $k$ increases. This is also observed in the navigability test in our previous paper, where $k=3$ is the optimal value. For the conventional hierarchical routing approach (Figure 17(b) and 17(d)), the routing effort required on superpeers does not change considerably. Figure 17(b) shows that for the case of low-density distribution of service peers ($2^{15}$ 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 $k=1$. Note that overall our approach imposes a higher routing effort on service peers; for instance, when $k=1$ an average of 26 service peers after $10^6$ queries (Figure 17(c)) as compared to 15 in the conventional approach (Figure 17(d)). Nevertheless, the routing path-length does not differ much, i.e. 1+26 versus 13+15 after $10^6$ queries.

We also measured processing overhead on proxies using a sample Web service which accepts a number of parameters (words) as input and echoes them to the client. We used the kSOAP2 library [27] for generation and processing of SOAP messages. The processing overhead on the proxy is mainly due to the direction of a SOAP message to the corresponding mobile device. Note that the time remains almost constant (around 100 ms) even though the number of Web service parameters increases.

### 6.3 SOLE Mobile Application

We have developed a prototype of SOLE in our laboratory to emulate the scenario illustrated in Figure 12. Figure 18 presents the user interface for the SOLE-ASP/ASC. For evaluation of SOLE performance, we have run the SOLE-AS on both desktop PC (Intel Dual-Core E8400) and a mobile device (HTC Hero) to emulate the scenarios for client-server and peer-peer models (Figure 13). Several metrics based on real-time evaluation are compared and shown in Table 2.

#### Table 2. Performance comparisons for SOLE.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Time (Milliseconds)</th>
<th>Desktop</th>
<th>Mobile Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP Deserialization</td>
<td>12</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>SOAP Serialization</td>
<td>5</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Application Execution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experience Insertion</td>
<td>17</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>Experience Deletion</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Experience Search</td>
<td>4</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

### 7 CONCLUSION AND FUTURE WORK

In this paper, we present the design and implementation of a service-oriented context middleware – Coalition to support context-aware mobile application development. The middleware is extended from our previous work for pervasive homecare with two additional features: the mobility support and the service management for mobile application services. The fundamental goal of Coalition is to introduce an open, programmable and
reusable platform for context-aware application developers to leverage on and effectively relieve the complexity and workload associated with application development. The detailed APIs and mechanisms that allow such easiness by using the aforementioned two features are discussed in the paper. Furthermore, the working principles of Coalition have been demonstrated via a case study, namely Sharing Of Living Experience (SOLE) mobile application.

So far, context-aware computing hasn’t found commercial success and our present prototype is still far from usable though we have carried experiments in the laboratory environments. However, we believe that a good initiative to advance context-aware computing is by having a robust and powerful middleware to fulfill context related tasks. In the next stage, we will deploy Coalition in a large field trial, such as in the university campus, to do more rigorous study and evaluation of its performance. In addition, we are currently working on the context realization mechanisms to further ease the utilization of the Coalition middleware and mechanisms.

8 ACKNOWLEDGMENTS

This work is partially supported by project grant NRF2007IDM-IDM002-069 on “Life Spaces” from the IDM Project Office, Media Development Authority of Singapore.

9 REFERENCES