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, dcsphk, oliya, g0701139, elewwc1}@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 technologies such as RFID tagging, GPS and wireless sensoring, makes IT invisible and yet ever closer and seamless to the daily life of people. However, so far, context-aware computing has not yet found large success in the commercial area, and existing context-aware mobile applications are mostly sensor-based. We argue that a context-aware middleware is indeed necessary to help develop and deployment of mobile applications as well as managing the heterogeneous context data. In this paper, we propose such a middleware named Coalition designed to address these issues. 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-aware 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, e.g. PDAs and smartphones. Through enabling technologies, such as WiFi and 3G, mobile application users may access services (e.g. mobile payment) and share information (e.g. Twitter) 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, software applications developed for 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 selected wireless sensors (accelerometer and gyroscope), 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 service-oriented middleware known as Coalition to support context-aware (mobile) application development. Through its context data management layer function, 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 application 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 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 for context-aware mobile applications; (2) facilitate mobility issues during 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 the case study for mobile applications. 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 in realization of context-awareness in SOLE is emphasized. In the end, 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. Section 4 describes the extended features of Coalition to support context-aware mobile application development. Section 5 discusses our proposed 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. More and more independent and freelance developers are being involved in the development of mobile applications besides the traditional mobile network operators and phone manufacturers. Though there are many platforms (Symbian [6], Windows Mobile [7] and Android [8]) designed to support the development of mobile applications 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 different context-aware middleware systems have been proposed and developed to model and manage context of various entities, but there is very little discussion on context modeling and managing for mobile entities that play the most important role in mobile applications. HiCon [9] is a hierarchical context-aware middleware with good scalability. By using mobile devices to collect personal context information, HiCon creates PocketMon to model and manage the context of people. However, HiCon does not provide a general context modeling and managing solution for mobile entities. Gaia [10] is a distributed context-aware middleware aimed to coordinate software entities and network devices in a physical space. By defining active space, Gaia supplies services to manage context of entities inside active space. However, most of the context entities involved in active spaces is static. Although 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 which embeds intelligent systems into physical spaces to provide pervasive computing services to users. CoBra uses context broker to acquire and fuse context information from heterogeneous sources into a Coherent model which is shared with all context entities inside smart space. However, CoBra does not consider the specialties of mobile entities for context data management and is more concerned on a single space without the discussion on relations among different spaces.

In conventional mobile applications, the mobile device carried by the person usually acts as the gateway for him to acquire information from the outside world. Nowadays, with more advanced mobile technologies, the 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 and accuracy 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, unreliable network coverage, and power constraint, would further complicate the discovery process. Moreover, issues such as security, privacy, and service administration should also be considered in the provisioning of services. Existing approaches for service provision and discovery fall into two groups: centralized and decentralized. The centralized approaches (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 when dealing with a large number of services. In the decentralized approach, scalable Peer-to-Peer (P2P) techniques such as Chord [15] and CAN [16] are often used. Nevertheless, portable devices such as 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, that makes them unsuitable for range queries, e.g. location-based search.

We are proposing a context-aware middleware – Coalition in handling all the above discussed 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-aware middleware designed for pervasive homecare. It is used 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, context reasoning (e.g. for activity recognition) and context-aware service management. For more details, interested readers refer to [3].

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, Coalition is designed as a service-oriented architecture; that is, the various system functionalities such as context data acquisition, context 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 spaces that represent the various context data sources. 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 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). Different kinds of context data can be available from a single physical space, such as the location of a device and the temperature of a room. 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 or applications to acquire context data or subscribe to event notifications. A hierarchical context reasoning scheme is employed in the physical space as well as the context data management layer to deduce the high-level context events from low-level results such as activity recognition. Note some functions of this layer can be distributed to the system servers and in some cases to the PSGs, like the storage of the historical context data collected by each individual PSG.

![Implementation architecture of Coalition](image-url)
• **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)*. LASOD implements a two-tier structure of peers for service management. The 1st-tier service management peers define and manage geographical scopes for each area under an administration and the routing of queries in the overlay-network of 1st-tier peers, while 2nd-tier service management peers of a geographical area register application services for that local area and route queries in the overlay-network of 2nd-tier peers. 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 model 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, where all the available services are deployed. 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 space to an area or to a geographical range. We will use *Sharing Of Living Experience (SOLE)* as the case study of mobile applications in the second part of this paper.

In the following, we highlight the mechanisms used for context data management 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 operating 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, e.g. sensors and computing devices, 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. For instance, a typical context attribute in the home space can be the temperature whose data can be retrieved from the temperature sensor; while the current activity attribute in the person space should rather be reasoned by the person’s calendar or the data collected by his body sensors. Since the collection of sensors within a physical space is normally under the same administration domain, this in another way shows that our space-based context management scheme is reasonable and feasible.

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. The main function of the PSG is to bridge the various data sources within a physical space with our Coalition middleware or applications. Thus, it must contain a set of components that manages and manipulates context data in the space. Figure 2 shows the functional components of a PSG, where these components are divided into three categories:

• **Local Service Management.** A PSG maintains different kinds of services using its service manager. The core of the services is a set of context data services that encapsulates various operations of context data acquisition in the physical space. In addition, a PSG can have system services that monitor and control sensor networks/computing devices, as well as application-oriented services that perform local processing tasks for the applications. Each of these services can be exposed as a Web service.

![Figure 2. Functional components of a PSG.](image-url)
through the HTTP server at the PSG, and registered in the service management layer of Coalition for discovery and invocation. The plug-in manager allows applications to deploy and share libraries of reusable plug-in modules. A plug-in is designed to work with the Web services to alleviate the processing burden of applications.

- **Context Query Processing.** A PSG accepts declarative, SQL-based queries, e.g. from applications. The syntax of the queries is based on a context schema kept by the schema manager which specifies all the context attributes that the physical space currently possesses. The queries are executed by the query processing engine provided as part of the PSG. The engine employs a context reasoner to deduce activities in the space. A local context database is used by the engine to store historical context data as well as by the reasoner to store the training datasets and reasoning results.

- **P2P Connectivity Management.** The connection manager of a PSG has two sub-components: the P2P manager and the mobile neighbor locator. The P2P manager links the PSG to a number of other PSGs to form a global P2P network, which enables the inter-gateway communication between multiple physical spaces and is managed by the middleware servers. Furthermore, a PSG can use its mobile neighbor locator to dynamically detect mobile PSGs in the same proximity and establishes connections with them for mutual data exchange. A PSG uses its registration manager to register to the most appropriate context space at the middleware servers and then join the corresponding semantic P2P cluster.

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 schemas. In Coalition, each context space has a space schema which describes the characteristics of the physical spaces it represents. 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. For instance, a home may be equipped with voice sensors so that the owner can use them to control the start/termination of electronic devices such as lights; while others don’t. 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. Each semantic cluster in a ring overlay corresponds to a context attribute in the space and is implemented as a a P2P network in which each peer is a PSG.

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 space are created and maintained by the context space manager 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 additional features by using new keywords. For instance, a CONT keyword can be used to specify a data acquisition query 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. However, as this is not our main focus of this paper, interested readers may refer to [3] for details.

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 communicate or 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 divides the world into a set of geographical areas in which resident service providers are managed by a local administrator. The area classification allows for local administration in terms of access control, policy making, and information management. Furthermore, any management approach taken within the local areas deals with a reasonable number of service providers. In LASOD, areas and the service providers within each area are organized in two distinct tiers.

- **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 (e.g. Walmart Shopping Center). 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, we are currently using rectangles to model each area and an irregular area could be approximated by multiple rectangles in practice. 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. In any case, we find our proposed indexing mechanism exert insignificant workload to the computing devices of a superpeer. We will demonstrate this in Section 6.

- **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: arealID and peerID. As an example, the identifier for service peer c in Figure 3 is 1110 1001. The arealID is to differentiate peers in different areas so to enable cross-area routing. Its length is \( d \log(bf_{\max}) \) 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 4 illustrates the allocation of areaID for the areas in the first tier of Figure 3. 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 in terms of decimal value. This approach maximizes the flexibility of area organization; most importantly, it ensures a fixed number of connections are maintained by a superpeer irrespective to changes of 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. For this purpose, we deploy the Hilbert Space Filling Curve (Hilbert SFC) for each local area. The Hilbert SFC is a continuous fractal curve (self-similar) that can cover a 2-dimensional space through several iterations [21]. We assign the peerID for each service peer based on its position on the Hilbert SFC. 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). The peerID
length is $2r$, where $r$ is the depth of the Hilbert construction 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. After assigning identifiers, the connections among service peers in each local area are based on their positions on the Hilbert SFC. That is, if two peers follow each other on the curve, they maintain a connection to each other.

To support keyword-based search, we map the set of service keywords (e.g. WSDL description) to the Hilbert curve by using SHA-1, which is a consistent hash function. Each keyword is hashed to a $2r$-bit key and is assigned to the first service peer on the Hilbert curve whose peerID is equal to or greater than the key value. We also incorporate the service scopes in the process of service indexing. Consider, for instance, a weather report Web service which is hosted in US 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 US. Such a service is then labeled as remote service.

LASOD adapts the small world model [19] in the novel Source Sampling method to make the resulting network model adaptable and navigable. 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. 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. Note that the two peers can reside in different areas. After sampling, a greedy routing mechanism is applied; that is, the service peer with the closest areaID (or peerID) to the target area (or peer) is chosen as the next hop for cross-area (or local-area) routing. Using the method, the routing efficiency is improved, when compared with early approaches.

We further developed a scheme named long-range link indexing [4], to improve the fault resilience of the framework. The scheme takes advantage of the long-range links created during the source sampling process. It lets other service peers within the area know the existence of long-range links created by a certain service peer. The index maps the target areaID of the long-range links to the service peerID in the local area. The distribution and maintenance of the index are similar to that of service keyword indexing when using target areaID as the key. Besides improving the system resilience, the long-range link index scheme also 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, interested readers may refer to [4] for details.

4 COALITION FOR MOBILE APPLICATIONS

The Coalition middleware can be used to support context-aware application development such as pervasive homecare as shown in our previous work [3]. Nevertheless, it is not optimized for mobile applications. Considering the increasing number of mobile users, we anticipate the following two main issues to be addressed if we are going to make Coalition as a platform for context-aware mobile application development.

- Mobility Issues. If the user accesses the middleware through a mobile device, the connection or the session required by the application will be inevitably lost during the movement. The conventional way to work around this problem is to let the user re-initiate the connection or session. A better way would be to let the application handle this situation automatically so as to make the mobility issue transparent to the user. Nevertheless, all mobile applications should cater for this issue...
themselves. Moreover, when application services are hosted by Mobile Service Providers (MSPs) which themselves are moving around, it is hard to re-setup the connection or session unless there is a specific mean defined to discover the MSP or Mobile Service Consumer (MSC). Currently, Coalition models all context entities into different physical spaces with different attributes, specialties of mobile entities, such as network changing due to mobility, are not embodied by the current data management mechanism.

- **Mobile Service Provision.** 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. The latter factor also affects the availability of the services they intend to provide. Besides, flexible security and privacy protection plans should be designed for these MSPs. Finally, as each MSP may use different mobile OSs and APIs, this would make the accessing of the services (e.g. retrieve the context data from a particular device) difficult. Fortunately, with existing Web service standards such as SOAP and WSDL, the interoperability issues may be solved by running Web services on the devices of MSPs. Nevertheless, public IP addresses are required for all the MSPs.

   To lessen the burden for mobile application developers by considering the above two issues. We have extended Coalition with additional features to make it an application-independent platform to support mobility and mobile service provision. The developed 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 better support context-aware applications for mobile entities, Coalition employs *mobile space* to model and aid the context data management of mobile entities. More specifically, for any physical space mobile context entities, if its PSG can also be mobile with it, we call this physical space as *mobile physical space*, the corresponding context space as *mobile context space*, and the PSG as *mobile PSG* (M-PSG). The following context attributes are designed for mobile context spaces as common context attributes to reflect the mobility:

- **location:** This indicates the current location of mobile entities, which is important to identify surrounding situations of entities.
- **visitingSpace:** This indicates the surrounding space of mobile entities, which may affect the current running applications.

- **powerPSG:** This indicates the power level of PSG device, which is always an important constraint for mobile devices.

   Besides the above common mobile attributes, certain private context attributes should be defined for different mobile context space so that special properties can be reflected. As a result, the whole schema of a specific mobile context schema can be divided into two parts: *common mobile attributes* and *specific context attributes*.

4.1.2 Availability Management for Mobile Spaces

To support mobility among mobile spaces, *availability* must be described for mobile spaces. We define the availability of physical spaces, more specifically PSGs (including M-PSG), as the ability of reaching-out to the PSGs through communication networks, such as the Internet and mobile network. Additionally, we also define the concept of *callbacks* as a service that provides notifications for changes of status of PSG’s availability. The availability of a PSG can be affected by mobility-related events such as temporary loss of network connectivity due to poor network coverage or handover, transmission related events such as signal interferences, network congestions, low power or even intentional power down of the PSG. Its availability therefore fluctuates with time and space, and will affect current applications. In order to enable applications better dealing with some of these abnormalities, we introduce two new functions to Coalition middleware: 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 server during registration which is defined as *Middleware Session ID* (MSID). Coalition contains a *MSID table* to maintain availability information of registered PSGs with using the MSID as the lookup key. As a result, any changes in the status of availability can be updated automatically and other mechanisms can benefit accordingly. ACS handles the disruption and resumption of an application due to the availability issue of PSGs. Each application launch is given an Application Session ID (ASID) by the respective host. By combining MSID and ASID, a unique Session ID (SID) is generated to identify a particular application running in a host of PSG. By using this ACS, an application in a PSG can register a callback upon changes of availability of a particular PSG. Whenever a change is detected by the AUS, all callback registered upon that particular PSG are retrieved in the form of a callback list. The updated availability information of the PSG is retrieved via the middleware, and then sent to those PSGs in the callback list by application callback notifications.
4.1.3 Availability Updating Service (AUS)

MSID is one key element of AUS that contains two main information fields: context domain indicator field showing the context domain that PSG belongs to, and a counter field that uses a numerical value to uniquely identify the PSG. The entire service can be divided into two parts: Middleware Session Processor inside Coalition server and Availability Updating Processor inside PSG (shown in Figure 6).

- **Middleware Session Processor.** It generates and records the MSID information for each PSG during registration. Meanwhile, it updates and manages the MSID information during the session life cycle of the PSG. This component contains two sub-components: **MSID table** and **query handler**. MSID table aims to record the availability information of each PSG according to MSID. Consequently, MSID table contains two information fields: **MSID** and **network information**. The network information contains the IP address and port number of a PSG can be easily queried and retrieved by taking MSID as the key. To increase the scalability, MSID tables are distributed into different CSGs that each CSG maintains its own MSID table containing availability information of all PSGs registered with this particular context domain. Query handler handles different kinds of MSID related queries issued by PSGs that including following operations: insert, update, retrieve and delete. By extracting and utilizing the context domain indicator field information of an MSID, information of CSG where the PSG is registered can be obtained. Accordingly, the MSID table inside this particular CSG is identified and can be looked up for the network information of the particular PSG. Meanwhile, by leveraging on the property that MSID of a PSG contains the information of its current context domain, a PSG can enquire the status of another PSG registered to a different context domain by providing its MSID. The same property can also be used to manage callbacks between a pair of PSGs from different context domains.

- **Availability Updating Processor.** It detects the availability changes of a PSG, and then updates the changes with Middleware Session Processor. This component contains two sub-components: **Network Monitor** and **Availability Updater**. Network Monitor collects the access network information (mainly the IP address) of the PSG continuously. By comparing it with previous stored information copy, availability updating alert will be generated accordingly and forwarded to Availability Updater. Availability Updater contains the MSID information of this PSG that is returned from middleware during registration. As a result, Availability Updater can communicate and update the availability information of the PSG with middleware according to the MSID.

4.1.4 Application Callback Service (ACS)

ACS aims to resume halted application sessions caused by unforeseen unavailability of related PSGs. As discussed previously, a callback is defined as a service that provides notifications about the status in availability of PSGs. In addition, we define the PSG requesting the callback as a **Callback Caller** and the PSG for which the callback is issued upon as a **Callback Callee**. An application can issue a callback through ACS when session disruption is detected. Fired by AUS, callback notifications will be sent to callers when the callee updates its availability information. The entire ACS can be divided into two parts: **Application Callback Processor** operating inside Coalition middleware and **Application Callback Handler** operating inside a PSG or application server (Figure 7).

- **Application Callback Processor.** It manages application callbacks, handles callback queries and processes 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 PSGs. Meanwhile, by leveraging on the property that MSID of a PSG contains the information of its current context domain, a PSG enquire the status of another PSG registered to a different context domain by providing its MSID. The same property can also be used to manage callbacks between a pair of PSGs from different context domains.

Figure 6. Availability Updating Service architecture.

Figure 7. Availability Callback Service architecture.
callers with respect to this callee from corresponding callback table. After getting the caller list, it retrieves the up-to-date network information of all callers from middleware, and then generates and distributes applications callback notifications to all caller PSGs according to their network information.

- **Application Callback Handler**. It issues application callback queries to application callback processor as well as process application callback notifications from application callback processor so that previously halted applications can be resumed. This component contains three sub-components: Query Controller, Notification Controller and Network Controller. Query Controller receives messages from applications, generates and issues callback queries to application callback processor. Notification Controller receives and analyzes application callback notifications on behalf of halted applications. Whenever receives an application callback notification, it extracts the ASID, MSID and network information of callee, and then interprets the ASID to identify the application and passes the MSID and network information to corresponding applications to resume previously affected application sessions. Network controller delivers queries to applications callback processor as well as receives application callback notifications from application callback processor.

### 4.1.5 APIs for Mobility Support

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

During the normal operating of a PSG, the **Network Monitor** component (Figure 8 top) continuously monitors the network information by using the function of “getIP()”. Function of “updatingAlert()” checks and detects the changes of network information, and then forwards updating message to the **Availability Updater** component which runs the function “availabilityUpdatingPSG()” to activate the application callback service and availability updating processes by sending the new network information with MSID to **Application Callback Processor** and **Middleware Session Processor** respectively. Inside Middleware Session Processor component, function of “availabilityUpdate()” refers to the **MSID Analyzer** component to get context domain (namely CSG) information by using the function of “analyzerCSG()”. Consequentially, new network information is updated with corresponding **MSID table** by using the function “put(MSID, IPPort)”. Application callback service is activated when a PSG updates its network information through the availability updating service (Figure 8 bottom). The **Availability Updating processor** component inside a PSG runs function of “availabilityUpdating()” which sends a messaging containing MSID with new network information to **Callback Notifier** components inside **Application Callback Processor**. The component runs function “callbackNotification()” to initialize the application callback service and uses the function “validMSID()” to check the validity of coming MSID. A valid MSID is forwarded to **Caller Searcher** component which runs function of “getCallers()” to retrieve the caller list from **Callback table** with respecting to this MSID. Meanwhile, this component also gets network information of all callers from the **Netinfor Retriever** component through the function “getIPPort()” which retrieves network information upon MSID from **Middleware Session Processor** component. Consequently, **Caller Searcher** component sends information to the **Notification Manager** component which runs two functions “notification()” and “notificationDistribution()” to generate and distributes application callback notifications. **Application Callback Handler** components uses the function “notificationReceiver()” to receive application callback notifications from middleware, and then leverages on **Notification Controller** to analyze the notification and forwards information to corresponding applications.

Figure 8. Illustration of API usage for mobility support.
4.2 Mobile Service Provision

4.2.1 Three-tier Service Provision Architecture

LASOD is designed as an application-independent platform for any application to leverage on so to publish and discover application services. However, to utilize the functions of LASOD, the application needs three logical software components (as represented in the service peer model in Figure 9), including (1) user interface specific which interacts with the user to allow functions of the application service to be invoked; (2) application specific which performs application specific logics plus session management; 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 in LASOD, which we formally name as Service Mediator (SM). More specifically, it mediates the application service with the LASOD functions. On one hand, the SM allows applications to utilize the indexing and routing mechanisms of LASOD to publish their application services or discover available ones (in case the application requires dynamic service composition); on the other hand, SM keeps the service information of each application service (e.g. WSDL reference of a Web service) so that it helps LASOD to direct an application-specific query to the responsible application. Indeed, the three components defined for the application may reside in one or 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, along with performing application specific functions. This is because mobile devices usually have limited memory, processing power, and battery life; besides, their availability is intermittent. 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 service mediator, it maximizes the flexibility of LASOD, as the service registration does not need to differentiate between local service and mobile service provision.

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 mobile service providers 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 (mobile) service registration and publication. In the case there are services delegated by mobile service providers, 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.

```java
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 that later application-specific query received by the \( sp \) can be directed to the correct service. The 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. The 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. For local service registration, the reference to the newly created SM object is returned back to the application, so application can let the mediator to publish its associated service. The service indexing function of the \( sp \) will then be triggered to index the service in LASOD. 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 LASOD and to use its resources or functions. Figure 11 illustrates the relationships and interactions among the software components Application, ApplicationService, ServicePeer and ServiceMediator.

### 4.2.3 Mobile Service Provision

Indeed, as mentioned before, the application service is not necessarily hosting 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 mobile service provider and the proxy are in close proximity, the proxy can be discovered via WiFi broadcast. (In this case, the mobile service provider 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 service mediator will be returned when registering application services to the service peer.

### 4.2.4 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 frameworks unless the query is flooded to all peers of the area. In our framework, such queries can be easily handled due to the usage of locality-preserving Hilbert SFC. The query is only matched with those relevant service peers in the range but not to all local peers. In the proposed approach, we first discover segments of the Hilbert curve involved in the range search (Figure 12). 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 \(<\text{currentStartID}, \text{currentEndID}>\) is kept to trace the current segment to be discovered, initially assigned with the value of the first segment. Once the query reaches the target peer which has the least greater peerID compared to \(\text{currentStartID}\), the query is passed through and matched with all service peers along the curve until the current peer has a peerID exceeding \(\text{currentEndID}\). 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.

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 are contained in the search bound. It utilizes the process for the Hilbert SFC construction. Each sub-area \((r1,r2,r3,r4)\) is checked with the bound in the ascending order of their startIDs (line 10-21) so that they can be added into the list orderly. Besides, if the sub-area is already contained by bound, a segment can be directly constructed and added to the list. Due to space limit, only the processing of one area is illustrated in Algorithm 2.

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. RevisiTou [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 anywhere, anytime (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 13.

5.2 SOLE in Coalition

With Coalition, SOLE can be designed as a mobile application that relies 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 14 shows the two service provisioning models for SOLE in LASOD. For the detailed provision of SOLE-ASP services, as mentioned in Section 4.2, the proxy is required in LASOD to help these mobile services to perform tasks such as service registration and query redirection. Therefore, once the SOLE-ASP discovers the nearby proxy (e.g. through either local broadcast with designated port or contact with our dedicated Web server), it register its service with the proxy, and the proxy will create the specific SM module. The SM module contains the service data (e.g. WSDL reference) about the mobile service. Sample service data used to describe the SOLE-ASP service is shown as follows:

```
```

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 Context-awareness

Context-awareness in SOLE is mostly realized by making use of the context data management layer of Coalition. More specifically, a mobile personal device is considered as the gateway (M-PSG) to the PERSON context space. The gateway caters for context data/resource management as well as all the interactions with the outside world. All the contextual details (e.g. name, location) of a person can be retrieved through the SQL-based query interface of Coalition. We consider the following applications of Coalition in the development of context-aware SOLE mobile application, with special emphasis on mobility support.

5.3.1 Mobility of the SOLE-ASP

If the experience data is stored on a mobile device (instead of a distributed server), efficiently locating the device by the SOLE-AS or SOLE-ASC can be challenging. In Coalition, such a problem can be handled in two phases: i) retrieving the current location of the mobile PSG by issuing the query “SELECT location FROM PERSON WHERE id = asp_id”, where asp_id is the SOLE-ASP’s ID; ii) 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 take care of the request by directing it 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 movement from SOLE-ASP or SOLE-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 and application callback service. On one hand, availability updating service helps a user detect availability (namely network connection) changes and update the changes with middleware. On the other hand, application callback service memorizes previous halted application sessions and activated by availability updating service 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.

For instance, Jane is currently retrieving a piece of image experience data (e.g. a short video) from the poster – Jenny who is currently moving from Shopping Mall A to Shopping Mall B. At the moment of existing A, Jenny encounters network connection loss, and the on-going transaction is broken. SOLE running in Jane’s device issues callback registration through application callback service upon Jenny’s availability. After entering into B, Jenny rejoins to network. The availability updating service running inside Jenny’s device updates this new availability information with Coalition, and fire the application callback service which generates and distributes a notification to Jane’s device to resume the previous transaction. As a result, Jane successfully obtains the experience data without disturbed by Jenny’s movement.

5.3.2 Experience “Pushing” to the SOLE-ASC

The context of an 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 WHERE name = person_name AND app_type = 'SOLE'
```

5.3.3 General Context-awareness

The location-awareness 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. Other types of context-awareness can be realized by using the experience meta-data. 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. These attributes can also be opted not to be shared publicly. We've developed three flexible data storage schemes for user to choose for their experience sharing.

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

For mobility support, the entire evaluation includes three different parts: PSG registration and deregistration evaluation, Callback registration and deregistration, availability updating and notification distribution. The evaluation measurement was conducted under wireless networks.

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 15 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 16 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 17. 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

6.2.1 Query Routing Behavior and Efficiency

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 18 presents the results. We observe that in our approach (Figure 18(a) and 18(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 18(b) and 18(d)), the routing effort required on superpeers does not change considerably. Figure 18(b) shows that for the case of low-density distribution of service peers (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. 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 18(c)) as compared to 15 in the conventional approach (Figure 18(d)). Nevertheless, the routing path-length (i.e. routing efficiency) does not differ much, i.e. 1+26 versus 13+15 after 10^6 queries.

6.2.2 Mobile Service Provision
For mobile service provision and discovery, preliminary experiments have been carried out to measure the processing overhead on proxies and mobile devices. The results are plotted in Figure 19. We used an Intel Dual-Core E8400 machine as the proxy and the HTC Hero as the mobile device. The experiments are based on 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 overhead on the proxy is mainly due to the direction of a SOAP message to the corresponding mobile device; while for mobile devices the SOAP deserialization/serialization process is the major concern. The overhead for directing SOAP messages on the proxy remains almost constant, even though the number of Web service parameters increases. The processing overhead of the HTC phone remains in an acceptable range. It is to be noted that we ignored the processing time of services, as this overhead is affected by a number of uncontrollable parameters such as data size and implementation efficiency. In the next section, we will demonstrate the overhead of application specific operations in mobile devices, e.g. insertion of experience in SOLE application.

6.3 SOLE Mobile Application
6.3.1 Application Prototype Implementation
We have developed a prototype of SOLE in our laboratory to emulate the scenario illustrated in Figure 13. The prototype consists of several SOLE-ASs which rely on MySQL [28] for the data management. Their functions of experience indexing and retrieval are exposed using Web services. By registering these Web services in LASOD, the process of discovery of the designated SOLE-AS (i.e. using DHT) is transparent to SOLE users. The SOLE-ASP/ASC is implemented using HTC Hero Android handphone. Differ from the SOLE-AS, the private data management in the mobile phone is implemented using the SQLite library [29]. Figure 20 presents the user interface for the SOLE-ASP/ASC.

6.3.2 Real-time Performance Evaluation
For a more rigorous study and evaluation of
SOLE performance, we have run the SOLE-AS on both desktop PC (Intel Dual-Core E8400) and mobile device (HTC Hero) to emulate the scenarios for client-server and peer-peer models (Figure 14). Several metrics based on real-time evaluation are compared and shown in Table 2. In the table, “SOAP Deserialization/Serialization” is referring to the time spent for parsing/composing SOAP messages. 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. We further divided it into three categories of operations. As for a SOLE-AS, the three most important functions are to support experience insertion, deletion and search. Although all these operations are strongly related to the data being manipulated, such as the size of the data, we are currently testing based on simple texts that can be possibly issued by people. With the real-time results shown in the table, we can see that as compared to the conventional desktops, applications running on mobile devices costs more overhead due to their slower CPU and smaller memory, the performance is still acceptable, which verifies the previous study shown in Section 6.2.2. Indeed, with more and more matured mobile technologies, we can foresee service provision will be popular in the near future.

Table 2. Performance comparisons for SOLE running on desktop and mobile device in real-time.

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

7 CONCLUSION AND FUTURE WORK

In this paper, we present the design and implementation of a service-oriented middleware to support context-aware mobile application development. The proposed middleware, known as Coalition, 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...
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 support by project grant NRF2007IDM-IDM002-069 on “Life Spaces” from the IDM Project Office, Media Development Authority of Singapore.

9 REFERENCES

tele-web-server.