Hanging Services: An Investigation of Context-Sensitivity and Mobile Code for Localised Services

Evi Syukur1, Dominic Cooney2, Seng Wai Loke1, Peter Stanski1

1 School of Computer Science and Software Engineering
Monash University, Melbourne, Australia
{evi.syukur, swloke, peter.stanski}@csse.monash.edu.au
2 Queensland University of Technology, Australia
dp.cooney@student.qut.edu.au

Abstract

As Web service technology evolves, the idea of context-aware services gains more interest. An idea is that different sets of services will dynamically drop into the mobile users’ devices depending on their contexts. To do this effectively requires location modelling and representation as well as spontaneity in downloading and executing the service interface on a mobile device. This paper introduces the concept and an implementation of Hanging Services that supports proactive and ad hoc context-aware services in mobile environments. This system works on top of an 802.11b Wireless network. The prototype implementation is done using Web services and highly compact mobile code applications using Microsoft .NET Compact Framework.

1. Introduction

Recent innovations in wireless networks, mobile devices, and software systems, are fueling new possibilities for tighter coupling of technology with people, in more ways than ever. The word “service” in the notion of electronic services (e.g., Web services1) suggests the use of software systems for providing assistance to users at a level that is more personal, and in ways that, even if non-technical, they can understand. Only imagination is the limit to the variety of such services that a user might employ. Ideally, a user should not have to grapple with a barrage of services but services should fit the user’s situation or context, presenting themselves in forms that the user can readily use, perhaps personalized to his/her needs.

The flurry of interest in context-aware computing (e.g., [1,2,3]) in recent years have sought to find ways to provide such a fit, though not all specifically dealing with Web services.

Contextual information such as location and user characteristics has been employed to tailor applications. In addition, parallel developments in distributed computing have given rise to the concept of mobile code,2 yielding unprecedented flexibility in terms of where computation can happen and reduction in set up time and effort for applications prior to use.

In this paper, we discuss the idea of Hanging Services, where we explore context-sensitivity and mobile code to provide useful services for the user with minimal or no effort for service set up prior to use. Context is used by our system to suggest on behalf of the user what services will be useful and relevant with respect to the user’s situation. Contextual information can include location, time, users’ intention, and device resources.

In our work, a user and location contexts are at fine granularity. The location context is represented by a model of indoor logical areas (e.g., rooms). Sensing the user’s context, the system computes a list of useful services, and this list “drops” into the user’s device. The user on selecting a service to use will have the required software for utilizing the service automatically set up, effectively “pulling” down required code.

As the user’s context changes, the list of services is automatically updated on the user’s mobile device and the downloaded code managed with respect to limited

---

1 http://www.w3.org/2002/ws/

2 http://www.cnri.reston.va.us/home/koe/bib/mobile-abs.bib.html
device resources. Context-sensing is particularly important if the system is to give the user the impression of proactive behaviour and spontaneity, and the use of mobile code is important in enabling services that can be used (almost) immediately without the burden of effortful set up.

As Hanging Services system is based on the highly compact mobile code, it is ideal for delivering mobile applications that make use of context-sensing information. Some interesting mobile applications that can be delivered as Hanging Services system are (a). Handheld Tourist guide application that displays different types of information on the device as the tourist enters different rooms in a museum or areas in a shopping mall. (b). Timetable system: as the user B walks into a user A’s room, the system then spontaneously displays a list of user A’s activity into user B’s device. (c). E-Notes service, where the mobile user can leave an electronic message for a particular user in a particular room. (d). Library service, where the system can guide the user to find the specific book in the library.

The rest of this paper is organised as follows. In section 2, we discuss the conceptual model of the system and some issues in providing context-sensitivity and mobile code in ubiquitous environments. In section 3, we unify our design and implementation of Hanging Services system with a logical view of the components architecture. In section 4, we discuss the evaluation of the system. In section 5, we discuss related work. In section 6, we present some possible future work and finally, in section 7, we conclude.

2. Conceptual Overview

Below, we provide a short introduction to the most important concepts and a high level architecture of the Hanging Services.

2.1 Context-Awareness

The definition of context in this paper refers to a specific entity or property in the environment that can be used to trigger a mobile application event. According to Chen [2] and Schilit [11], the definition of contexts can be divided into four categories: (a). Computing context: This can be any information that relates to computing environments such as computer resources (e.g., printers, scanners and workstations), available bandwidth, and network connection. (b). User context: User context refers to any knowledge about the users, such as a user’s profiles, intentions, location, and current social activity (e.g., in a meeting). (c). Physical context: Physical context has a general definition that involves any physical entities and properties in the user’s environment. For example, lighting, temperature, traffic conditions, and noise levels. (d). Time context: Time refers to the current time and the user’s schedule (in terms of days, weeks, months and years) that can be used to determine the mobile application’s behaviours.

Another dimension of context is the context history, which includes the information over a time span related to the user’s context [2]. The context history is considered as useful knowledge, accumulated over a certain period of time, about the user and the surrounding situation that may result in a computation of new services.

2.2 Harnessing Context-Awareness for Computing Relevant Services

Our system provides an infrastructure that mediates the interaction between the client device and the application logic via Web service calls. This high level architecture is illustrated in Figure 1 below.

![Figure 1: High level architecture of Hanging Services](image)

The Hanging Services system provides an infrastructure that simplifies the task of developing and maintaining context-aware applications for mobile clients. The system relieves the developers from the low-level task of matching services with the location as it handles the computing of the set of relevant services given contextual information about the user.

The followings describe four of the most important components of the system:

A. Services Calculator

The Services Calculator component computes a set of services based on the updated user’s location and services available in the current context. In addition, a set of services can also be computed using the information that is stored in the history log file. For example, at time t, the system takes information about the user’s current context (include device resources)
and history (e.g., a user’s previous whereabouts and services used at those places) to compute the set of services that would be useful to the user at time $t$. This computation can be modeled as a function $f$, where $f(\text{current\_context,}\ \text{history}) = \text{a new set of services}$.

There are two major challenges that need to be addressed in order to allow the system to act more intelligently. The first issue is how often or when should $f$ be computed?

To address this problem, there are two approaches that can be incorporated. The first approach is to compute the function $f$ when a change in current\_context (e.g., location) is detected. With this approach, potentially less bandwidth, memory and battery power will be consumed. However, if a new service is just added for that location, the system will not be able to detect that particular service, as the user’s location remains the same. The new service will only be detected if there is a change in the current context.

The second approach is to recompute $f$ periodically even though there might not be a change in current\_context for every 10 seconds. With this approach, the user will definitely be able to see new services that are just added to that location context. However, relatively more resources are needed for this approach.

The second issue is how do we decide what should go into the current context and history? Current context is basically related to any knowledge regarding the user, physical and time contexts. The context history is like a history log that contains any information over a time span that is related to the user’s behavior. To proactively compute a new service based on the combination of a user and history contexts, we may require a policy.

Based on this policy, a mechanism that keeps analyzing the user’s behavior over a time span will be useful. The system can then use such a mechanism to predict the user’s needs and actions proactively. For example, every Wednesday, the user comes to the office (B.3.50 room) at 3 PM. The system then records this information into the history log file. As the system works out what situation the user will be in, the system could then predict the set of required services proactively before the user gets into that situation (e.g., switch on the room light at 2.58 PM). This will avoid lag time between the user getting into a situation and the user being able to use a service. This predictive feature is currently not supported yet in the current prototype. We will continue to implement this proactive and predictive feature in the next stage of implementation.

### B. Code Cache

Code cache refers to the mobile code application that is stored on the device for future re-use. In terms of managing the code cache on the mobile device, the following function $c$ is considered, where $c(\text{current\_context,}\ \text{history}) = \text{application-code that remains in the cache}$. The function $c$ above describes that at some time $t$, $c$ takes information about the user’s current context and history, to determine what application-code should remain in the cache i.e., what should be deleted.

Some caching policies may also be required to address the above situation. The policy may remove the code if the user has not used it for a certain period of time (e.g., two days) or code is removed if the user has not been in the space where the code is useful for two days. The policy can also decide when the function $c$ should be evaluated, such as when there is a need to make space for a new mobile code.

### C. Code Server

Code server refers to the mechanism that handles a user’s request from a device, responding to that request by transferring the relevant mobile code. In this case, the code server retrieves the mobile code that matches with the user’s request and sends it back to the mobile client.

### D. Client Application

Client application resides on the mobile client side that manages the incoming mobile code and prepares to execute the service interface on the device.

#### 2.3 Mobile Code

Within our framework we employ mobile client code applications to provide a simplified interface to XML Web services. The usage of contextual mobile applications that migrate to devices presents a powerful paradigm. The development of mobile applications that targets resource constraint devices does incur greater development effort. As this requires an agent programming model which differs from traditional desktop development. This is further complicated by attempting to build applications that substantially simplify the mobile users’ experience. Within our model, devices permit the reception of mobile code which presents the user interface (an end-point) to a contextually available XML Web service.

---

3 The set of services available for a location is not considered part of the context but used in the computation of $f$. 

---

Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM’04)
0-7695-2070-7/04 $20.00 © 2004 IEEE
Based upon the user’s selection and profiles, different types of mobile code applications may be downloaded to the mobile device for execution. Thus the mobile client application delivers a responsive on device user experience and permits remote Web service calls.

2.4 Localised Web Services

The idea of localised Web Services is to set a relevance and utility boundary on the services. In this case, a particular service will only be available in a certain contextual environment. For example, the mobile users will get different set of services as they move to a different physical location. Although the definition of the context has been clearly defined and understood by the programmers, there are many outstanding issues on how to efficiency and effectively support the above contexts within mobile applications and pervasive environments. Numerous issues must be addressed in order to deliver a consistent user experience and a framework supportive of frequent user and application context switching. Two of the major issues related to the localised Web services are:

A. Web Service API issues

Fundamentally, one of key issues to address is how a user (client) may interact with an XML Web service. By manually generating SOAP messages and interpreting responses requires intimate knowledge of the service by the client. This is an unrealistic model as an infinite amount of services providers may exist. Hence, knowledge of Web service APIs must be somehow delivered to a user and the client device up-front. Common solutions to this have been performed by a Web application representing the user interface to the related service offering. However, such browser based systems lack a real time nature and the richness of on device applications. Therefore, this creates an opportunity for mobile code, or mobile agents, to be downloaded to devices on demand.

These computational units of code may be obtained on demand or proactively within a user’s context, activated to service the user’s needs and later discarded or cached for future re-use. By supporting mobile code in mobile devices, we have a model whereby arriving code is knowledgeable of XML Web service APIs and their respective communication protocols. Hence, these code payloads referred to as ‘Picklets’ provide a rich user interface and application logic. For each service only one picklet is required and many picklets may be in existence on one or numerous client devices at any one time.

B. Localised service issues

The localised service challenges related to the situation where a user is in an area, is given a list of services to use, and the user invokes a service from this list. However, before the service completes, the user goes out of the area into a different location where the service is no longer relevant.

One possible solution to address this issue is to apply some expiry time or maximum suspension time for service invocation. In this case, the service will not be valid after exceeding the amount of specified time, akin to a leasing mechanism. There is also another challenge in adding a new service to the system. This issue is related to the effort required to add the mobile code application for a new service and mapping the available service with location context.

3. Prototype Implementation

We present our prototype where some of the above issues have been tackled.

3.1 Architectural design

The system consists of users with handheld devices or laptop computers, a Web service that determines the location of a user, and a set of location-based services that are published via the system (see Figure 2 below). This section gives a high-level description of these parts of the system, and how the parts interact.

Figure 2: System architecture of Hanging Services

Five of the main components of the system are first discussed:

Mobile Client Software: We have used Microsoft .NET Compact Framework technology which natively supports XML, SOAP and remote code loading [14]. Users with mobile devices run software that continually polls a central Web service to discover the services which are available at the device’s location context. When the user selects a service, the mobile
device contacts the central Web service and downloads an application for interacting with the selected service. Services can also be delivered to laptop computers via the system. Laptop users go to a Web site that presents them with a list of services that are available in the current context. From there, they can download service applications. These are launched via the browser.

**Location – Web service**: To realise the location-aware services, this system employs the current release of the Ekahau Positioning Engine (EPE). The EPE is a software solution that keeps track of a user’s location based on the signal strength measurements. It also targets several devices such as: wireless PDAs, laptops and any 802.11b-enabled devices [8]. Ekahau consists of three main parts: the Ekahau server, Ekahau client, and Listener-application.

The EPE server includes a standalone manager application, and a Java Software Development Kit (SDK) that can be used for tracking client’s position (X and Y coordinates or latest logical area). In order for the Ekahau server to keep track of the client device, the Ekahau client software needs to be installed on the mobile client device. The listener-application refers to the application code that implements the listener interfaces (to obtain the location estimate, logical area and status) to accurately track wireless devices.

Moreover, to allow interoperability with other platforms and languages, our system implements the listener-application as a Web service. This location service is deployed on the Axis Apache Web server environment. The service returns the user device’s position in X and Y coordinates as well as the logical area.

**Code server Web service**: Within our system, we employ a Web service method invocation (such as GetMobileCode Web method) to retrieve a mobile code application that matches the service name, returning the particular mobile application to the client device. An example of GetMobileCode Web method on the Code server Web service class is illustrated in Figure 4 below.

**Calculator Web service**: In our work, the Calculator Web service computes a proactive service using a combination of a user and location contexts. Different users get different types of services in the specific location. Our current implementation has not incorporated other types of context parameters (e.g., history, users’ intention, profiles and device resources) in deciding the combination of services that will drop into the user’s device.

In addition, our Calculator Web service can also be used to personalise a list of services for each user by adapting to their profiles. However, this personalisation feature has not been implemented in the current prototype. We will continue to develop this feature in the next stage of implementation.

A simple GetService Web method of the Calculator Web service class is described in Figure 3 below. This method returns a list of available services for a particular location and takes into account the identity of the user that is currently logged on.

**XML Repository**: XML database contains the mappings information between a user, location and services available. For example:

```xml
<?xml version="1.0"?>
<User name="Peter" location="B3.50">
  <Service>
    <Name>e-Post it Note</Name>
    <Description>an e-message</Description>
  </Service>
</User>
```

We now describe each of the steps in Figure 2 above:

1a. **Send Access Point Information**: Once the Ekahau mobile user device is switched on, the EPE server then starts tracking the position of the mobile client. 1b. **Start Tracking**: Our system provides a login mechanism to the Hanging Services. The user needs to enter the credentials information such as a user name and password. The system then validates these credentials against the user’s information which is stored in an XML database. The system will only redirect the user to the main service form, if all information that he/she enters is valid. If the user is valid, the system then invokes a Web method of the Calculator Web service called “Start Tracking” by providing the IP address of the device.

2a, 2b and 3. **Get a User’s Logical Area, Call the Ekahau Server and Return Logical Area**: The Calculator Web service then continues to invoke the “get logical area” Web method of the Location Web Service and again passing in the IP address of the device to this Web method. The Location Web service then fires the Logical Area Listener Application on the Ekahau Server. The Listener application then continuously listens for the mobile client’s movements. Finally, this Web method returns the most accurate user’s logical area to the method caller (e.g., to Calculator Web service class).

4 and 5. **Find and Return the Available Services**: Once the logical area is returned, the system then searches for the available services that match with the user who is currently logged on and the logical area from the XML database.
If there is a service associated with this logical area and a user, the service information (service name and description) is then returned.

6. Send a list of Services to the Mobile Client: If the services are found, a list of services will then be sent to the mobile client. The mobile client application then displays these set of service names.

7. 8 and 9. Request Mobile Code (or ‘Picklet’), Get and Send Mobile Code: When the user chooses a service from the list, the code server Web service is again contacted to provide code for invoking the selected service. The code provided to the mobile device software subsequently requests the associated mobile code application from the mobile code repository.

10. Return Mobile Code to the Mobile Client: This returns the mobile code used to access the service to the client device. Upon its arrival, the mobile client application then loads and processes this service application, executing and displaying the service interface on the mobile device.

3.2 Implementation Aspects

Caching applications on mobile devices: each downloaded application is specialised for interacting with a specific service, and can deliver a rich user interface and functionality, even if the device is temporarily disconnected from the network. This is possible as the handheld device software caches downloaded applications (stores it locally on the mobile device). The cache contains code, and metadata describing applications. Hence, if a downloaded application is running, its cache code also exists in the memory.

The .NET Compact Framework includes a garbage collector, so when physical memory is limited, unreachable objects are discarded [14]. If memory becomes extremely tight, the .NET Compact Framework pitches the native code of JIT-compiled methods. This means that performance will suffer if the user downloads a lot of applications because of the increased memory pressure, and increased likelihood of methods needing re-JITing. The .NET Framework
handles caching downloaded applications for laptop computer users. Downloaded applications can use a small amount of isolated storage for client-side cookies.

.NET Framework security: The .NET Compact Framework provides security guarantees for mobile codes [14]. Validity and verifiability ensures that programs can only 'go wrong' at runtime in specific, predictable ways. Like the full .NET Framework, the .NET Compact Framework has code access security (CAS), a runtime security system with declarative and imperative develop-time declarations and administrator-controlled policies.

CAS stops, for example, .NET applets accessing the user's file system by restricting access to sensitive parts of the System.IO API. This simply means that applications downloaded to the user's device cannot attack their system. In this case, the user must trust in the prototype implementation, the mobile device user must trust the mobile device software to enable an appropriate security policy. For laptop computer users, various security restrictions are imposed by the .NET Framework on the downloaded applications [15].

For example, downloaded applications can listen on sockets, and are only permitted to make socket connections to their originating server. This security restriction means that each downloaded application must poll to determine if the user's location has changed. The Web server that the application was downloaded from must act as a proxy for any Web requests the application makes, for example, to invoke a Web service.

Data persistence on the .NET Compact Framework: The .NET Compact Framework has no built-in binary serialisation. To support caching application-specific data on mobile devices, we have used the Pickle framework which provides lightweight binary data persistence [9]. Pickle requires more effort to implement than .NET Framework binary serialisation; however, Pickle has the advantage of being reusable across the .NET Compact Framework and in Web-deployed laptop applications where .NET Framework serialisation is normally not available.

Mobile User’s Respective of the System: The layout of the interface is critical for the effectiveness of the system. We have implemented a simple interface for both laptop and handheld devices. The interface is divided into three main regions, the label that describes the user who is currently logged on, the current location context of the user and the type of services available for the particular user and location. Figure 5 below demonstrates the consistent looks and feel of the Hanging Services on a mobile device. It also illustrates that the services displayed will vary depending on the user and the current location contexts.

![Figure 5: Accessing Hanging Services system from a Mobile Device](image)

Figure 6 (a) below illustrates the Hanging Services system from a laptop device that is accessed via an Internet Explorer Web browser and (b) shows the sample of mobile application used in the system. It is called Mobile Pocket Pad which allows users to type in any information and stores that information back to the server.

![Figure 6: (a) Accessing Hanging Services System from a Laptop Device and (b) a Sample of Mobile Code Application](image)

### 4. Evaluation

The framework has given promising results in obtaining the list of services, keeping track of the user’s location, downloading and executing the mobile code on both laptop and handheld devices. The evaluation starts from the Web service call to get a user’s location up to the service activation. The evaluation aspects on Hanging Services system are described in Figure 7 below.

In our evaluation and testing, results were collected for five times of service activation (see Figure 8 below).
We measured each of the aspects in executing the mobile code on the handheld device. The timing variations in performing the seven main evaluation aspects of the system are illustrated in the Figure 8 below.

Based on Figure 8, we can see that the time required to call the Web service: get a user’s location and a list of available services, decreases for the 2nd, 3rd, 4th, and 5th times of service calling. The first call of the Web service takes a longer time, as the system needs to compile and download the local host Web service proxy object to the device. The proxy object allows the Web service to be treated like other .NET classes. The 2nd and subsequent calls to the service will have much shorter times as it reuses the service proxy object already on the device.

In addition, the amount of time required to display a list of services on the mobile device is consistent throughout the executions as the number of services used for the evaluation is the same. Our system gets an updated service list for each user and location from an XML database.

The time required for downloading and displaying the service interface on the device depends on the size of the mobile code itself. The larger the size of the mobile code, the longer it will take to download and execute the service application. Note that the mobile code only needs to be downloaded once to the mobile device; subsequent service executions will reuse the code that has been stored on the device. Therefore, the downloading time for subsequent service executions is zero.
For the purpose of testing, we have developed one mobile code application: Mobile Pocket Pad (10 KB). Basically, Mobile Pocket Pad is like a Note Pad that the user can use to type in and store data back to the server. As we have kept the size of the mobile code constant in our runs, and we have been reusing the same application throughout the repeated executions, the time to display the service’s interface remains the same. Moreover, the time required to get the compiled mobile code from the JIT cache is constant throughout the executions (1s).

Compilation of the code that implements a service in the .NET Framework involves two stages: (1) the framework compiles the source code (e.g., C#, VB.Net or C++) into the Microsoft Intermediate Language (MSIL), (2) when the application gets executed (at run time), the Just In Time (JIT) Compiler compiles the MSIL code to native code and stores this native code in the JIT cache for future re-use.

Note that service activation at the first time means causing the relevant downloaded mobile code or picklet to be compiled and then when executed, displaying the service’s interface to the user. Subsequent activations will not involve the service compilation. Subsequent activations will retrieve the compiled code from the mobile code storage, i.e. the JIT cache (see Figure 7 above). Therefore, the compilation time for subsequent service activations is zero.

Finally, we present a diagram to illustrate the total time delay to seeing the updated services when the user moves to a new location (i.e. from Location A to Location B) and how long the user should wait before a selected service’s interface is displayed. As shown in Figure 9 below, the time delay for the location context change comprises the Ekahau delay in detecting the user’s new location and the time that it takes to update the list of available services on the mobile device, when the user moves to a new location.

Based on the formula above, we can conclude that the worst-case scenario for the location context change delay is the first time of service execution, which takes 6.5s (= 2.5 + 3.5 + 0.4 + 0.1). The 2.5s is the Ekahau delay in detecting the user’s movement, the 3.5s is the time that it takes to get a current user’s location from the Axis Apache server via a Web service call and the 0.5s (=0.4+0.1) is the time that is required to get and display the updated list of services on the device. The best case scenario, i.e. the minimum time delay to see the updated services for the current location is in any execution which is not the first. In such a case, the delay time is 5.5s (=2.5+ 2.5 + 0.4 + 0.1). The delay time to detect subsequent location context change decreases to 5.5s, because, the Web service calls in a subsequent context change, re-uses the local proxy object, which has been downloaded and compiled previously.

In addition, the time delay (or service execution time) between the user’s selection of a service and the display of the selected service’s interface varies depending on the execution number (see Figure 8 above). The first time a service is executed by the user, a much longer time delay is experienced compared to the subsequent executions. The time delay for subsequent service executions is consistent throughout the subsequent activations. Moreover, as we are using multi-threading in executing the service application, we experience some delay in displaying the service interface. This delay we call the thread task.

Now, we give the formula to measure the service execution:

\[ T_{\text{Service Execution(s)}} = T_{\text{Download mobile code}} + T_{\text{Mobile code compilation}} + T_{\text{Get compiled code}} + T_{\text{Display service interface}} + T_{\text{Thread task}} \]

Based on the testing and evaluation results, we conclude that the worst-case scenario to see the service interface is the first time of service execution which takes 5.1s (=1+2+0+0.9+1.2). The best-case scenario for service execution is any execution number, which is not the first. It takes 3.1s (=0+0+1+0.9+1.2), as the mobile code has been downloaded and compiled in the first run. These figures were obtained on a 206MHz iPAQ on a wireless Wifi network.

5. Related Work

This section provides a brief overview about the research work that has been done to date that also concentrates on determining the location of a mobile
user and implementing an infrastructure for context-aware systems. Some earlier research relating to location models which have directly accessed the needs for context-awareness in an ad hoc environment are Hybrid Location Model [4] and Personal Help Desk (PhD) [5].

The Hybrid Location technique presents a model of physical environment and representation of locations by using a combination of the hierarchical and coordinate approaches. The hierarchical location model refers to the hierarchy spaces and the coordinate system is used to define the coordinate points within the specified location. The Hybrid Location Model also emphasized the performance and flexibility of handling the spatial queries for context-aware applications by introducing the Aura Location Identifier (ALI).

The ALI utilises the SQL like queries to handle the information retrieval and storing. Although this project can speed up the process of handling the spatial queries for context-aware applications, there is also a constraint that limits its functionality. For example, the ALI can not directly store or integrate the highly dynamic attributes (e.g., a user’s location). These particular attributes have to be maintained separately by custom services. Our project implementation has employed the Ekahau Positioning Engine (EPE) for the purpose of tracking mobile clients’ location coordinates. The EPE which is based on the signal triangulation maintains the dynamic information related to the user’s position (e.g., X and Y coordinates).

PhD from Carnegie Mellon University is an example of a location model that uses signal strength from wireless access points to obtain the user’s position. This model, to some extent, has a similar philosophy to the EPE, since it provides the most accurate user’s location based on the signal strength measurements. However, the PhD application requires relatively more extra hardware for acquiring results in a high degree of accuracy. The Ekahau, on the other hand is a pure software application that does not involve any hardware requirements, apart from the ordinary WLAN infrastructure.

Another interesting aspect that will be looking at this section is the system infrastructure of context-aware applications. This involves the techniques on how to load and display the service interface into the mobile device. There are two different approaches that can be used to address this issue. The first approach is to implement mobile code which contains information about the service interface and application. This mobile code is stored on the server side and only downloaded into the client device whenever it is triggered. The second approach is to embed the service application and user interface into the mobile device, for example, as in [12]. As the mobile device is very limited in the resources (memory and speed), the second approach is less resource smart.

The Hodes system [6] also recognised the need to use mobile code in downloading a service interface and application into a mobile device. This system aims at providing variable network services in different network environments, which involves changing connectivity. Hodes also introduced an open service architecture with minimal assumption about standard interfaces and protocols to support heterogeneous client devices. However, the implemented prototype application has not incorporated the services for per-user location based interfaces. Our system has implemented different types of services for each user on the particular location. For example, the lecturer that is visiting the administration office may be interested in different services than the student in the same location.

The Mobile Shadow [13] project focuses on prototyping applications for proactive cell-based location aware services. The Mobile Shadow uses the concept of mobile agent to keep track of the user’s location. This project also exploits the concept of mobile code in downloading the service interface to the client device. Our system extends the idea of mobile code here by supporting data sharing of mobile applications between laptop and handheld devices.

Another work that employs the concept of mobile code is Hive [7]. Hive is a distributed software agent platform that uses a combination of wearable and ubiquitous computing to address the concept of location-aware services. In this project, the computation and information are shared between the environment and the wearable. Rhodes claims that implementing the location-aware systems in both pure ubiquitous and wearable have presented fundamental difficulties [7]. This is due to the ubiquitous computing tending to have trouble with personalisation and privacy, whereas, the wearable system has some drawbacks with localised information, resource control and management.

Hive is a Java-based agent architecture, that relies on the Remote Method Invocation (RMI) distributed objects and mobile code. This system only targets the small device application. In contrast, our system is implemented on the highly compact mobile environment (.NET Compact Framework) and employs the concept of Web service for the purpose of retrieving the updated user’s location and a list of available services in a particular location. Moreover, our Framework is not only able to download the
service application and interface; it is also able to pin the information back to the server. This updated information is available from both laptop and handheld devices.

In addition, there are also some various other context-aware applications surveyed in [2]. However, we believe none of them are using mobile code and positioning technology like the way we do here.

Our conception of a “Hanging Services” is based on the Ekahau Positioning system [8], a Web service conceptual design implicitly in [10] that allows method invocation in disparate platforms and languages. Moreover, a mobile code and caching logical model expressed explicitly in the Migration Sympathetic Program model [9]. This is a way of thinking about context-aware applications, driven by end-to-end highly interoperability design principles, which can move between hosts during their execution lifetime and support sharing of simple mobile applications between laptop computers and handheld devices [9].

6. Future Work

Future work involves iterating over the conceptual and logical design model, as well as investigating any other possible approaches to improve the efficiency and the effectiveness of the prototype implementation.

(a). Service Discovery issues: We have not as of yet incorporated a dynamic Service discovery (e.g., Jini, UPnP and JXTA). In the future, this remains as a critical area of the research in the field.

(b). Wide Area issues: Our prototype implementation has been tested only in an indoor local area environment (for example, only a particular level of the building). We are investigating the bigger scope of the system (using Global Positioning System). This relates to the location tracking issues (support both indoor and outdoor of the building), query semantics and dynamically discovering a new service as the context changes.

(c). Context Parameters: We wish to add the additional context parameters to our prototype implementation, including the history log file, device resources and a user’s profiles. The current implementation only supports two aspects of the context parameters, which are a user and location. Moreover, we also wish to utilise the personalisation mechanism in delivering services to mobile users. This can be done by adapting a set of services to the user’s profiles and history log file.

(d). Caching: In general, the system may want to react intelligently in managing the code caches on the device rather than explicitly removing of the cached code by a user. This requires a management caching policy to intelligently remove codes which are no longer used by the user.

(e). Polling versus a publish-subscribe model: Currently, the supporting software for handheld devices polls the centralised Web service at a fixed time interval. More frequent polling means that the list of Web services offered to the user is more accurate. But the polling operation is redundant if the user has not moved, because the set of services available at a particular location is relatively stable. To conserve the limited processing power, memory and battery life of these constrained devices, it would be good to reduce the number of redundant poll operations. One simple solution is to increase the time interval when a device detects that it has done a redundant poll. This is effective because once a user has stopped moving, she is likely to stay in the same place for a relatively long time.

Using a publish-subscribe model instead of continual polling can eliminate redundant Web service calls. When the software starts on the handheld device, the mobile device registers with the centralised Web service. The centralised Web service would then monitor the location of the mobile device, and push service lists to the mobile device via a Web service call to the mobile device. Applications leveraging this functionality can support ad-hoc collaboration between co-located mobile device users. Although in principle this application could be supported on the existing infrastructure, the infrequency of polling would cause the list of services to adapt slowly as services entered and left the location.

(f). Scope of the service: As part of the future work, it is also important to add the functionality that gives a user the flexibility in terms of the scope of the services that the user may be able to see, considering the user is within multiple domains at the same time. For example, in a room which is in Caulfield campus, which, in turn, is in the University, which, in turn, is in Melbourne. This requires a comprehensive location model and we may implement the zooming kind of model to address the above situation. Leasing of services also needs to be implemented.
7. Conclusion

We have presented an architecture for “Hanging Services”, allowing a mobile device to adapt its functionality to exploit a set of services that it discovers depending on the user and location contexts. The system allows users to interact with services on-the-fly without the burden of effortful setup or knowing the Service API of XML Web services. We also have tested the performance of our prototype architecture using an example of service (Mobile Pocket Pad application) at Monash University campus, building C level 5. To overcome the lag times for service usage, we plan to use predictive techniques based on richer contextual information.

Although, we have successfully tested the system performance, some usability study on user aspects is required in the future. This usability study can focus on how to present localised services to the mobile user, whether the user really wants a proactively changing list of services that adapt to the user’s current context (e.g., location and time) or relatively fixed set of services depending on the user’s profiles such as, the user is only interested in seeing news, hence, a fixed set of services will always be displayed at any location that the user goes.

References


