Policy Based Control of Context Aware Pervasive Services

Evi Syukur[1] and Seng Wai Loke[2]

[1]School of Computer Science and Software Engineering
Monash University, Melbourne, Australia
evi.syukur@csse.monash.edu.au
[2]Department of Computer Science and Computer Engineering
La Trobe University, Melbourne, Australia
s.loke@latrobe.edu.au

Abstract. While mobile computing technology is becoming more and more mature, the demand of having context sensing ability on any mobile or embedded systems is increasing rapidly. Context awareness is important if we want to give a mobile user autonomous, responsive and attentive services depending on his/her current contexts. To date, much research work have been developed to address some issues regarding location modeling, system design and implementation of simple awareness scenarios. A simple awareness system displays a set of useful services to the mobile user based on the primitive context information (i.e., a user's location). It does not take into account a rule or policy that specifies when and where the user wants the particular service to be executed. Designing a context aware pervasive system with additional policy information is a new research challenge that needs to be addressed. This paper introduces the idea of using a policy mechanism to control context-aware behaviour for pervasive services. The paper also discusses the usefulness, design architecture and prototype implementation of the Mobile Hanging Services framework that supports proactive and ad hoc awareness services in pervasive environments. An approach for contextual services that uses a set of rules or policies to govern the service execution is illustrated through a sample Mobile Windows Media Player application.

Keywords: Context awareness, pervasive services, policy, mobile code, Web service-oriented development, location modeling, and handheld devices.

1. Introduction

Over the last decade advances in mobile computing, including the efforts towards reducing the size of the computer and the invention of wireless technology, have made it possible for mobile users to access information at any time and any location that they wish. Through sensor devices and software systems which are invisibly installed in the environment of our everyday lives, the embedded or non-embedded computing devices communicate and exchange messages with each other. The availability of sensor devices, plus embedded systems in the environment is known as "pervasive computing". Pervasive computing has a broad view of providing computing devices everywhere in the environment and at any time [1]. The idea is that a mobile or non-mobile user can communicate with any embedded or non-embedded computing devices, which are invisibly integrated into the environment as soon as s/he steps into that particular space. To date, a good pervasive system is no longer determined by the exceptional number of functionalities that it supports. Instead, it is determined by the design and architecture of the system itself i.e., either it is a system centric or a user centric design.

System centric here refers to a traditional computing system that delivers a set of services to the user without taking into account information regarding the user's contexts or current activities. It is more focused on the system or developer point of view. Hence, the service that is delivered may or may not be useful to the user. A user centric system, on the other hand, refers to a system that places the end-user as the main priority in deciding on what type of services should be delivered. It is designed in such a way as to serve the user's needs in different situations by taking into account the user's context information. Designing a system based on the user centric approach has now become a key factor that plays a significant role in improving the user's experience, realising the aims of pervasive computing and to empower users to be more effective in completing their daily activities.

A context aware system has the ability to usefully adapt services or applications to the user's current situation, intention, needs or environment. This would enable users to receive a relevant set of services that fit his/her current context, instead of a barrage of irrelevant services. The notion of context here refers to any information that is considered useful to the user and usually related to the user's current activities. Context is mainly used by our system to suggest on behalf of the user what services would be useful and relevant with respect to the user's current situation, as well as to control aspects of an application. Contextual information can include location, time, a user's intention, current activities, history file, device resources [2]. Some other authors including those of [4, 5, 6] have explored much richer contexts that involve a variety of different physical sensors in the environment e.g., sound, temperature, light, touch, movement and so on.

Some work has also been done in [3] that presents a dynamic conceptual model of context that supports cognitive activities other than just location, time and some other parts of physical contexts. Currently, contexts in our work comprise a user identity, location, time and system behaviour according to context that is specified in a user's policy document. A location context is represented by an indoor logical model such as room location. Sensing the user's context (i.e., a user's location), the system then proactively discovers and computes a list of services that may be useful to the user at that particular context. This list of services is then dropped into the user's mobile device. In our definition, a service simply means a software tool, which is delivered onto the user's mobile device for the purpose of suggesting or helping users to complete their tasks.

As the user selects a particular service name on the mobile device, the highly compact (with respect to limited device resources) mobile code that provides control for the service is then downloaded. This sensing ability that can be found in context-aware applications distinguishes them from existing traditional applications. Here, "traditional application" refers to a primitive stand-alone application that does not have or utilize any context sensing ability.

To clearly understand the usefulness of context sensing behaviours in the pervasive environment, we first consider the following system scenario that uses a traditional Windows Media Player as an example:

> *"Every day after lunch (around 1:30PM), user A feels like listening to the instrumental music in his office (room A). After finishing his lunch at Merlins Café, he wants his favourite instrumental music to be played on his desktop machine, as soon as he opens the door and steps into his office. However, since a traditional Windows Media Player application does not use and utilize the knowledge of the user's context information, this smart activity cannot be performed implicitly by the system. Hence, user A needs to manually start the Windows Media Player application, select the music name and start the music on his desktop machine".*

To manually start the music each time the user wants to listen to it can be quite tedious and annoying, and perhaps unnecessary, especially if there is regularity in the user's behaviours, for example, the user always listens to the same music everyday at the same time and location.

Now, we look at another scenario of a context-aware Windows Media Player application. The following scenario uses context-sensing behaviours with the traditional Windows Media Player application. By sensing the user's context i.e., a user's current location, the system then automatically starts or stops playing the music on the user's desktop machine:

> *"By identifying a user's identity and sensing a user's current location (e.g., user A is now at the corridor and walking towards his office), the context-aware Windows Media Player application will then start playing the user's favourite instrumental music on the desktop machine, as soon as the system senses that user A is now stepping into the office (room A). The music playing depends on who the user is. This instrumental music will be playing until the system detects that the user has walked out of the room (i.e., go out for a meeting)".*

Adding context sensitivity to a traditional Windows Media Player application certainly maximises the user's experience in using Windows Media Player. This is due to the intelligent behaviours of the system that frees the user from all the manual computation tasks. Hence, this empowers the novice or even the non computer literate users to be more effective in performing their daily tasks.

Now, consider another scenario of context based policy control of Windows Media Player application. This sample scenario extends the idea of intelligent context based Windows Media Player application as discussed above by allowing the end user to define a rule or policy that specifies when, where, what type of information (e.g., music) that s/he wants to be played or stopped at each particular situation.

A policy might define a set of activities that a user is allowed, prohibited or obligated to do in an organization (e.g., a staff may only be able to read the company's message but not to modify it) [25, 26]. Integrating the concept of policy into the development of a pervasive system has impact on the way entities access services. There are several main roles policy can play in pervasive environments:

a. The policy defines the visibility of the services in particular contexts i.e., two users with different roles may see different services available in that context.

b. To help the user to perform a task automatically within a certain situation (i.e., a policy rule can say "automatically start the music (service) at 12:30PM at room A, playing The First Noel"). The rules or policies here can be used to restrict the behaviour of a service (e.g., a music service) at the specific context, for example: start the music at 12PM and pause it for 15 minutes at 12.15PM. Having a policy document certainly helps to improve the user's experience, especially if there is regularity in the user's activities.

c. To constrain the behaviours of the foreign agents or visitors [27] accessing services in the user's room (i.e., to protect a user's privacy [28] and give the owner of the room the ability to control the activities of visitors in his/her room).

The scenario below illustrates the usefulness of having a policy document via which the user can specify control behaviour for the music service at each particular situation. The policy is applied when a context is satisfied.

> *"User A defines the following intentions (rules) in his policy document*

***Intention**: During lunch time from 12PM to 12:30PM start the instrumental music service at room A.*
***Action**: As soon as the system detects that user A is entering room A and the current time now shows that it is 12PM, the system then automatically starts the instrumental music on his desktop machine for 30 minutes until 12:30PM.*

***Intention**: Pause the music for 10 minutes (from 12:31PM to 12:41PM) as the user needs to go to the post office to mail a letter.*
***Action**: As the current time shows that it is 12:31PM, the system then pauses the song for 10 minutes until 12:41PM.*

***Intention**: Resume (continue playing) the music from 12:42PM to 1:00PM as the user is done with the mailing task and will arrive at the office approximately at 12:40PM.*
***Action**: The system then resumes the music at 12:42PM and plays the song for 18 minutes until 1:00PM.*

***Intention**: Stop the music at 1:01PM as the user is going to have a regular meeting at 1:05PM*
***Action**: When the current time shows 1:01PM, the music will be stopped".*

Some other scenarios illustrating the use of policy for governing use (e.g., via a user's mobile device) of mobile services.

1. Intention: *I don't mind other users starting a media player service in my room on Wednesday from 12 to 2 PM, as I am somewhere else at that time.*
Action: All visitors can start media player service and play any song that they wish on Wednesday from 12 to 2PM at room A.

2. Intention: *Don't let anyone bother me at my office from 2 to 3PM every Wednesday, as I am preparing for my meeting.*
Action: No service is allowed to be executed on Wednesday from 2 to 3PM at room A.

3. Intention: *All services especially an online browsing service or a mobile pocket pad service are not allowed to be executed by the student during the exam on 22/12/04, Wednesday, from 12 to 2PM at exam room A.*
Action: No service is allowed to execute or all students are prohibited to execute any service on 22/12/04, Wednesday from 12 to 2PM at exam room A.

4. Intention: *Every day during lunch time from 12 to 12:30PM, I want the media player service automatically start my favourite instrumental music at tea room.*
Action: The system will automatically start the specified music on any day (Monday-Friday) from 12-12.30PM at tea room.

5. Intention: *I only allow my colleagues (i.e., all other lecturers) to start media player service at my room on Wednesday, 1-3PM.*
Action: Only visitors who have the same role (i.e., a power user role), can start media player service. Other roles can only listen to the running music without being able to start his/her own music or stop, pause and resume the currently running music.

Apart from adding context awareness and a policy document to a traditionally designed Windows Media Player application, it is also useful to provide computing support to the mobile user to access the Windows Media Player application and control the music from a mobile device while being on the move (e.g., if in a museum with headphone music). Here, computing support simply means a mobile code that is proactively downloaded to the user's mobile device, whenever the system detects that the user is in the context where such code is relevant.

Such a mobile code implements a service which is enlisted as the user needs it, which takes care of how the task gets done [19]. The ability to, in an ad hoc fashion, download and execute mobile code on the mobile device where the code can be used to control the Windows Media Player application (such as selecting the target device, music name and action types: playing, pausing, resuming or stopping the music) gives the user convenient control over the application. To achieve a situation where the system can benefit and improve the mobile user's experience in an ad hoc network requires a comprehensive model and design architecture for the context aware pervasive system. Implementing the idea of enabling context-sensitive services for mobile users has raised seven challenges.

One challenge is to have a location positioning system that can determine the user's current location accurately. Another challenge is that the system needs to proactively discover services that fit the user's current context as well as spontaneously deliver and execute the relevant services on the user's mobile device. The third challenge is to create a generic mobile context framework that can support many different applications including a traditional mobile or non-mobile application. The fourth challenge is the mobile framework also needs to exploit the system design and implementation of policies or rules in pervasive context-aware environments. The fifth challenge is to develop mobile code encapsulating user interface to embedded devices or applications in the environment, e.g., which allows a user to control the Windows Media Player application from a mobile device. The sixth challenge is how the communication among a mobile device, embedded software applications and a desktop device can be performed.

The seven challenge is how we separate the control between a user and a system, once we include a user's policy into our pervasive system i.e., who should be in control? Is it a user or the system? The system is context-aware and so should

take action autonomously but the user also needs to be in control (or have the sense of being so).

The research presented in this paper attempts to tackle the above issues in our framework for Mobile Hanging Services (MHS). The MHS supports context-sensitivity and mobile code in order to provide useful services for the user with minimal or no effort for service set up prior to use [19]. The MHS is a framework that supports development of context-aware mobile services along with a policy to govern the service execution in pervasive computing environments. It also provides a generic mobile framework that can be adapted into any existing traditional application. Using our MHS system, we can easily add context sensitive behaviours to almost any existing traditional applications such as a stand-alone Virtual Network Computing (VNC) application [18, 24], and Windows Media Player application [29].

Apart from being generic, our MHS framework also permits a remote Web service call from a mobile client device to a server or vice versa and from a server to the desktop machine. Hence, with the addition of remoting mechanism on top of our existing framework and having the extra policy document for the traditional application that specifies when and where to start the Windows Media Player application, we can control the way the application is executed on the target machine. MHS supports policy mechanisms (i.e., handing the request from mobile users up to solving the conflict if any) by having several policy software components. These are policy manager, policy interpreter, policy conflict detection and policy conflict resolution which all reside on the server side. Each of these software components is published via the system as a web service, in order to support interoperability. In addition, MHS also has a simple rule that separates a control between a user and a system (i.e., when the user or system should be in control).

The rest of this paper is organised as follows. In section 2, we discuss the conceptual model of the system. In section 3, we present our MHS policy language design. In section 4, we discuss several types of conflicts that may occur in pervasive computing environments. In section 5, we present techniques used to resolve conflicts. In section 6, we present an implementation of our MHS policy based system with a logical view of the components architecture. In section 7, we present our system evaluation. In section 8, we discuss the end-user's perspective of the system. In section 9, we discuss advantages and disadvantages of using policy in pervasive systems based on our experience in designing, implementing and testing of MHS based policy system. In section 10, we present related work and conclude in section 11.

2. Conceptual Overview of Mobile Hanging Services

Having an additional policy mechanism in a pervasive system certainly benefits and maximises the user's experience. However, there are challenges to developing such a system:

a. To handle possible conflicts that may occur in pervasive environments. The possible conflicts may vary from one pervasive system to another, and depends on the system design such as the contexts, entities, policies, and services used. For example, user A has a policy to start the music service at B536, from 12-1PM, and, user B also has a policy to stop all the running music at B536 from 12-12.30PM. The environment needs to be in control, but in some situations, the user might want control (or have the sense of being so).

b. To have a scalable mobile framework that can support a number of different policies (i.e., a policy from a user, system, service or physical room) and have an interoperable framework where the policy functionalities can be easily invoked and accessed from different platforms and languages.

c. To develop a simple and robust policy language that can be easily understood and used with context aware services.

This section provides the most important components and high level architecture of our MHS system that supports context awareness for computing relating services and policy in pervasive environments. We divide our system components into two major parts: a) contextual services and b) policy software components (see Figure 1 below).

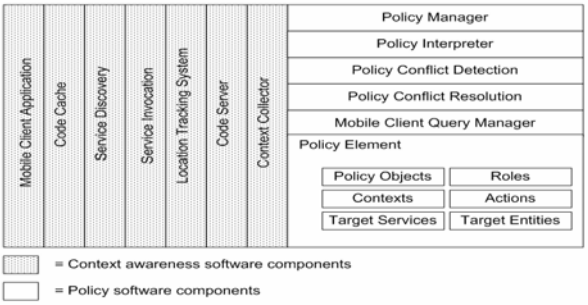

Figure 1: System Components of MHS based Policy Framework

We have discussed in detail in [19] each of our software components that support contextual services in pervasive systems. This paper focuses mainly on the policy mechanisms which are implemented on top of our existing mobile contextual framework that can be used to control the entities' behaviours in pervasive environments. The policy component focuses on the policy related queries i.e., handling a user's request whether or not s/he has a permission to perform some actions on the specified service. If the user is permitted, the system then continues to check whether there is a conflict between users or between a user and a room when s/he executes this particular service with the requested action and lastly, the system resolves the conflict if any. Most of the policy software components reside on the server side. The only policy component that resides on the client side is mobile client query manager. We now describe each of these policy components:

a. Mobile client query manager (on the client side)

The query manager handles the request from the user and sends this request to the policy manager. Once the result is returned (from the policy manager), it then performs the

action whether prohibiting the user from executing the specified action or allowing the user. For example, a user clicks on the "start button" to start music A. This query manager then contacts the policy manager to check whether the user is permitted to start the music service at the given context. If the user is permitted, the query manager then starts music A, otherwise it displays a warning message to the user.

b. Policy manager (on the server side)

Policy manager manages the interaction with the mobile client, where the mobile client sends a request to the policy manager and the policy manager processes the request and returns the result back to the client. A user's request here can be a query about the user's right, obligation or prohibition – e.g., whether or not the user can start "service A" when in room A at 3PM on Wednesday. To respond to this query, the policy manager needs to call the policy interpreter to gather the user's relevant contexts and policy.

c. Policy interpreter (on the server side)

The policy interpreter component computes a set of rights, prohibitions and obligations which are useful for the user in the particular context. The policy interpreter component first retrieves the relevant user's contexts information (e.g., a user's identity, location, current time and day). After retrieving the relevant context information, the policy interpreter then parses the policy document that specifies the user's rights, prohibitions and obligations concerning (e.g., when to start or stop) the service.

d. Policy conflict detection module (on the server side)

The policy conflict detection module is called by the policy interpreter to detect potential or actual conflicts that may occur between entities (e.g., between users or between a user and his/her environment (i.e., a room)) [14]. A potential conflict is a conflict that could happen, i.e., the conflict has not happened yet at the time the system detects that such a conflict can happen, as the context or condition for the conflict to occur has not been met. The potential conflict can be further classified into two different types: possible potential conflict and definite potential conflict.

The possible potential conflict is a potential conflict which can still not happen even in the right user contexts of location and time - the possibility of occurrence is less than definite potential conflict. For example, a system (as in its policy) allows the user to "start any service" but the room (as in its policy) only allows the user to "start media player service". "Any" here means all services which are available for the user in that context. It includes the media player service and some other services in the context. The conflict only occurs if the user starts any service other than the media player service. The conflict will not occur if the user starts the media player service. Hence, we categorise this conflict as a potential conflict with the type possible. Possible potential conflicts are only run-time detectable.

The definite potential conflict, on the other hand, refers to a conflict that will definitely occur if the context is met. For example, a system allows the user to "start media player service" but the room prohibits the user from "starting media player service". Once the context is met, the conflict will definitely occur, as one allows the user and the other prohibits the user from doing so. Definite potential conflicts can be detected statically - clear from inspection of the policies.

e. Policy conflict resolution module (on the server side)

As each entity (e.g., a user) in the system is allowed to create its own policy and each entity may have different sets of rules in their policies, there is a chance of conflicts occurring. The policy conflict resolution component here handles the conflicts between entities in the system if it occurs. There are five possible techniques that we propose to resolve the conflict: role hierarchy overrides policy, activity hierarchy overrides policy, room holds precedence policy, obligation holds precedence policy and precedence overrides policy. The details on each of these conflict resolution techniques are discussed in section 5.

To decide which technique to use, the policy conflict resolution component first analyses the type of the role that the user has. Role here refers to a level of the privileges for an entity in the system (i.e., a general user, power user and super user). Basically, the purpose of role is to group and assign different levels of authority and privilege to each entity. The grouping here is based on the type of the entity i.e., in a pervasive campus environment, an entity with a type student has a general user role, a lecturer entity has a power user role and a head of school has a super user role. A super user is at the top of the hierarchy in our system, followed by a power user and a general user. An entity with a higher level of role can do more things compared to the entity with a lower role. For example: a super user can choose either to stop the music execution at any place that s/he goes if s/he does not like the music or play his/her own favourite music. A power user can only stop the music and a general user is not allowed to stop the current playing music.

f. Policy language

A policy language is used to express rules that govern the entity's behaviours in pervasive environments. It specifies the rights, prohibitions and obligations of an entity in the specific context. We discuss in detail our policy language in section 3.

Our system provides an infrastructure that mediates the interaction between the client device and the application logic via Web service calls. The MHS 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. This high level architecture is illustrated in Figure 2 below.

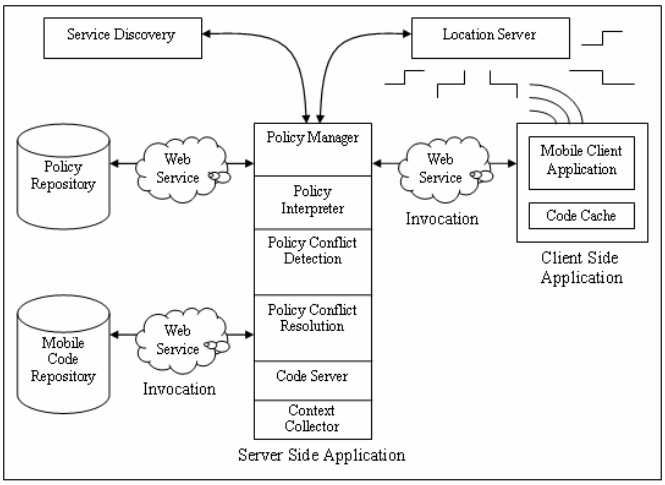

Figure 2: High Level Architecture of Mobile Hanging Services

3. Design of the MHS Policy Language

Most existing policy languages have constructs which are too domain specialized, which we feel made their direct use in our system complex. So, we developed a simple policy language for specifying policies for all kinds of entities in the system (room, user, etc). We try to keep our policy design as simple as possible so that it can be easily integrated into the existing or future implementation of context aware services. Moreover, we also focus on the interoperability aspect, in which, all the policy software components need to be easily accessed by the system itself or by external systems that have disparate platforms and languages.

Having an interoperable policy system further supports an extensibility of the framework as we can easily add more services, entities and policies in the future or perhaps integrating the existing system with some other external systems (assuming they both have similar design and architecture). In our system, a policy is a rule that is defined by an entity (i.e., a user, room or system) and it bounds any associated entities as specified in the policy document. The rule specifies a list of activities that the entity can perform (permission), must perform (obligation) and can not perform (prohibition) in the given contexts. Having a set of rules or policies in pervasive environments is considered useful as it can help to restrict the user's behaviors or constrain the activities that the user can do in the particular context as specified in the policy document. In addition, the policy can put a user in control of the surrounding activities that happens in his/her room. For example, a user that owns an office (e.g., B556 room) can write a room policy to control the activities of other users (visitors) in his/her room.

Several criteria that need to be taken into account in designing a good and robust policy language are:

a. Easy to use and understand
b. Has a simple policy design that is not tied to any specific applications or services
c. Has a loose coupling policy designed architecture (separate the abstraction layer between the policy and the service implementation). Hence, updating the system functionalities on one component will have minimal impact on the rest of the system components.
d. Supports both simple and complex situations
e. Extensible and allows a new rule to be added without any modification to the existing rules
f. Can be used to describe situation in expressive and concise way
g. Have a standard policy language i.e., conforms to a policy language schema.
h. The policy constructs should be flexible enough to not over constrain the domain (should be reusable across domains).
i. Supports basic deontic logic concepts (i.e., permissions, obligations and prohibitions).
j. Supports consistency in answering the user's query (every request is either allowed or denied, not both).

3.1 Policy Design Considerations

There are several factors that need to be considered in designing a comprehensive policy model for context-aware services. Some of these factors are:

1. Who should be in control in the system i.e., whether the user, the system, or the room?

2. How to support partial control between entities in the system? For example, at a particular situation, we would like to let the user be in control but at some other situations, we may want to let the room, or the system is in control.

3. The best technique to detect the conflict? For example, we can detect the conflict reactively or proactively. In deciding on which technique to use, we need to consider the:

- System performance i.e., how long it takes to detect the conflict.
- System resources i.e., how often we need to update the conflict detection result.
- Accuracy i.e., how accurate we want the conflict detection result to be.
- Scalability i.e., can the conflict detection technique still be used to detect the conflict if the number of entities in the system increases (i.e., we have more entities in the system than before)? If so, would it have any impact on the performance?
- Extensibility i.e., can the conflict detection technique still be used to detect new conflicts that may happen in the future?
- Simplicity i.e., how easy is it to maintain the conflict detection result?

4. The best way to resolve the conflict? For example, we can resolve the conflict as soon as the system detects that there is a conflict or resolve it only when the conditions for the conflict to happen have been met. Several considerations in deciding which technique to use are

- System performance i.e., how long it takes to resolve the conflict
- System resources i.e., do we need to store the conflict resolution result for future re-use or remove it from memory as soon as it has been used or when a system detects that the conflict resolution result may no longer be relevant to the user's current contexts?
- Satisfaction i.e., how do we ensure that the resolution technique that we use will satisfy both conflicted entities?
- Scalability i.e., can the conflict resolution technique handle a number of conflicts at a time or only a limited number of conflicts?
- Extensibility i.e., can the conflict resolution technique resolve new conflicts that may occur in the future (due to more services and contexts are employed)?
- Simplicity i.e., how easy is it to maintain the conflict resolution result?

5. Decide the best technique for communication between a mobile client that is requesting to execute a service or requesting to perform some actions on the service and the server. For example, whether the client needs to request periodically (polling method) or the server pushes the updated result to the client (publish subscriber method).

3.2 Policy Elements

Our policy language is written in an XML language that conforms to a policy schema. Our policy includes several elements for describing permissions, prohibitions and obligations of an entity in the system. Each of these elements is described as follows:

a. Policy objects

Policy objects are based on the logic of norms for representing the concept of rights, obligations and prohibitions in our system. Right (R) refers to a permission (positive authorization) that is given to the entity to execute a specified action on the service in the particular context. Obligation (O) is a duty that the entity must perform in a given context. Prohibition (P) is a negative authorization that does not allow the entity to perform the action as requested in the given context.

b. Target services

Target service refers to a particular service that the user wants to execute. Some interesting mobile applications that can be delivered as mobile services are (a). Handheld Tourist guide service 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 service: 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). Pocket pad 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. (e) Mobile VNC service that allows a user to teleport his/her desktop information to any nearby machine in the current location. (f) Mobile media player service that allows a user to start/control any music on a desktop machine from his/her mobile device.

c. Actions

Currently, there are five different actions with services that our system employs. These actions are start, stop, pause, resume and submit. The types of actions applicable to one service might vary from another, and would depend on what the service does. For example, media player service that is for playing music on any target machine supports four different actions (start, stop, pause and resume), teleporting service that is for displaying a user's desktop information on any nearest machine may only require to have start and stop actions, and pocket pad service that allows users to type in any information and stores that information back to the server only requires submit action. The five actions we identified, we found, occurs frequently among a number of MHS services we considered. Depending on the policy that has been defined by the room or the system for a particular user in the specific location, two different users may have different privileges for a certain action on the same service at the same place. Specifying different privileges to legitimate users is a way to restrict the mobile user's behaviors in a pervasive environment. For example, user A that has a general user role, is only allowed to start a media player service. User B who is a power user has the privileges to start and stop any services at that location.

d. Contexts (domain constraints)

Contexts are conditions that must be met before a list of services can be displayed on the mobile device or before the user's request to perform an action is approved. In our work, contexts consist of a user's identity, location, day and time. The location context is represented by a model of an indoor logical area (e.g., a room). Sensing the user's context, the system computes a list of useful services, and these lists are effectively "dropped" into the user's mobile device.

e. Target entities

There are three types of entities in our system.

i. The System: The system entity controls users and rooms' behaviours in accessing or executing services in the system. The system policy is created by the developer or the top user in the system. It specifies what users and rooms can do in the specific location and time contexts depending on the role of an entity. This default system policy will be automatically inherited by all registered users in the system according to the role that s/he has.

ii. User: A user is an active entity that is always on the move (able to move from one geographical place to another). By default, the system imposes on the user certain rights (denoted by sRu), obligations (sOu) and prohibitions (sPu) for each physical location in the system based on the users' role. On

top of this default system policy, users can also specify a set of obligations on the system (denoted by uOs), created via a user policy application that we have. Imposing a set of obligations on the system is a means for the user to have the system perform tasks automatically on behalf of the user at certain location, day and time. This then helps to reduce the user's task especially if there is regularity in the user's activities. Our system also checks for the conformance of uOs against the permission that the system gives to the user for that location. This is to ensure that the uOs is still within the scope of the user's permissions. In summary, depending on the user's current location and current contexts, each of the users in the system will have: **sRu_i, sPu_i, sOu_i, and u_iOs**

Note: **i** denotes a specific user i.e., user **i**, say. We now explain each of the policy object notations above: **sRu_i** is a set of permissions (**R**ights) that a **s**ystem gives to a **u**ser (say **i**). In contrast, **sPu_i** is a set of actions which the **s**ystem **p**rohibits the **u**ser from doing. The permissions or prohibitions given to the user depend on the role that s/he has, location that s/he is visiting and the ownership of the location. This would mean that two users who have the same or different roles will have different permissions in executing a service depending on the place that s/he is visiting and whether or not the user is the owner of the place. For example, the user with the higher level of role that is visiting a public place will have more rights to perform any services compared to the user with a lower level of role that is visiting the same place.

Another sample is two users with the same level of role will have different permissions when they are visiting a place that is belong to one of those users. The owner of the room has more permissions compared to the visitor of the room. **sOu_i** is a set of actions that the **s**ystem **o**bligates the **u**ser (say **i**) to (manually) do in a particular context. The obligation from a system to the user is useful to allow the system or the environment to be in control of the surrounding situation. For example, to ensure that all services are off before the user left the school, the system obligates the user to stop all the running services every day at any time before 6PM.[1] **u_iOs** is a set of actions that a **u**ser **o**bligates the **s**ystem to perform in a particular context, representing what the user wants the system to do automatically. For example, user A is always checking her email when she first arrives at school at 9.30AM. Instead of she needing to manually open her email account by clicking on the email button every day, the user can ask the system to perform this task automatically. So, when the user enters her office, Outlook Express has already started and the user is ready to check her incoming e-mails. This is possible by imposing an obligation from a user to the system (a uOs).

iii. Room: A room entity is represented by a geographical location - e.g., room B548. The room entity has its own policy that can be used to restrict the visitors' actions on mobile services in the room. Generally, the room's policy is created by the owner of the room. The public place in our system (i.e., tea room, corridor, or seminar room) is owned by the system. Hence, the public policy is created by the system (e.g., developer/administrator). Just like the system, the room also imposes certain rights (rRu), obligations (rOu) and prohibitions (rPu) to the user who is visiting the place and the owner of the place. The owner of the place generally has more rights and less prohibition compared to the visitor, even if the visitor has a much higher role than the owner of the room. In a case where the owner of the room requires the system to perform some tasks automatically, the room can also impose an obligation to the system (rOs). The purpose of rOs and uOs are quite similar here as they both ask the system to perform some tasks automatically on behalf of the user or the room. The only difference is rOs can only be created by the owner of the room and the uOs can be created by any users (i.e., a visitor) of the room. Typically, the owner of the room that has already imposed an obligation to the system (rOs) does not need to have any uOs at his/her own room. S/he only needs to impose an obligation to the system (uOs) for other locations (other than his/her room) if s/he wants to.

As the room gives certain permissions to the visitor, and often, if there is a conflict, the room wins, the visitor is free to challenge the room if s/he is not satisfied with the conflict resolution result considering his/her current situation. Allowing the visitor to challenge the system is a way of allowing the user (visitor) some control as the user may need control (win the conflict) in certain situations. By challenging the room, given the current situation of the user, the system will re-compute the conflict resolution, and if the reason is sufficient, the challenger may win. For example, user A is a student that is going to give a demo who requires starting a teleporting service at seminar room A. By default, the room only allows the power user or super user to start the teleporting service on any day between 12-2PM, as only lecturers need to use the teleporting service at this time. This is mainly due to the room needing to restrict other users (e.g., a student from performing some actions on the teleporting service, as conflicts may arise if more than one user in the room try to start or stop the service).

By default, the room does not allow the student to perform any actions on the teleporting service between 12-2PM. User A that has a permission given by the system to start a teleporting service at this location, may want to challenge the room if s/he is not satisfied with the conflict resolution result (given his/her needs to use the teleporting service). By analyzing the challenger's reason (the current activity of the challenger), the room may agree with the challenge and let the challenger start the teleporting service. In addition, as our system allows room entities to define a set of policies for the owner and the visitor of the room, conflicts can occur i.e., the system may allow the visitor to start any music at room A, but the room owner may prohibit the user from doing this. We can reduce the number of conflicts in the system by statically conforming one policy against another. The policy conformance here is to check whether one policy is behaving according to the source policy (in this case, whether the room's policy conforms to the system's policy). This conformance process can be done statically at compile time when the owner of the room finished creating the policy for his/her room.

There are two different conformance techniques that we employ:

a. **Full conformance.** Full conformance means one policy document needs to be fully conformed or matched against other policy documents (i.e., the system's policy document). We use full conformance to check the conformance between u_iOs against sRu_i. This is mainly because our system only allows the user to perform a task which is permitted by the system. For example, if sRu_i specifies that user i can only start the music service on Monday, from 12-2PM, this means that the u_iOs has to be within that condition (Monday, from 12-2PM). The user is not allowed to impose an obligation on the system to start a media player service, other than Monday, 12-2PM (other than the permission given by the system).

b. **Partial (subset) conformance.** As it is partial, the conformance here only checks the subset value of one policy document against the system's policy. We use partial conformance to check the conformance between rRu_i against sRu_i, rOu_i against sOu_i and rPu_i against sPu_i – effectively, room should be subset of system. For example, if sRu_i allows the user to start a media player service at any day and any time, the room can further restrict the user's permission at this room, by just giving the user a permission to start a service at a certain day and time. The reason of using a subset conformance here is because we want to let the room to be in control at some situations by further restricting the permissions given by the system to the user for that location. For example, during the exam, the room may prohibit users from executing an online browsing service or perhaps, reduces some of the permissions given by the system.

The following describes the policy objects relevant to a room S for a user i in the system: **r_sRu_i, r_sOu_i, r_sPu_i and r_sOs**, where S denotes a specific room, e.g., room B536, and i denotes a specific user. We now describe policy objects for a location that consists of one or more users. Let m be the number of users currently in a location (say, room A).

Let $sRu = \{sRu_1, sRu_2, sRu_3, ..., sRu_m\}$ be the set of rights given by the system to m number of users in a location (say room A).

Let $sOu = \{sOu_1, sOu_2, sOu_3, ..., sOu_m\}$ be the set of obligations from the system to m number of users in a location.

Let $sPu = \{sPu_1, sPu_2, sPu_3, ..., sPu_m\}$ be the set of prohibitions from the system to m number of users in a location.

Let $r_ARu = \{r_ARu_1, r_ARu_2, r_ARu_3, ..., r_ARu_m\}$ be the set of rights given by room A to m number of users in a location (room A).

Let $r_AOu = \{r_AOu_1, r_AOu_2, r_AOu_3, ..., r_AOu_m\}$ be the set of obligations from room A to m number of users in a location.

Let $r_APu = \{r_APu_1, r_APu_2, r_APu_3, ..., r_APu_m\}$ be the set of prohibitions from room A to m number of users in a location.

Let $uOs_A = \{u_1Os_A, u_2Os_A, u_3Os_A, ..., u_mOs_A\}$ be the set of obligations from m number of users to the system at room A (involving services for context where location is room A).

Let r_AOs be the obligations of room A imposed on the system.

The combination of policy objects in room A with m number of users are:

Rights for the users: $\mathbf{sRu \cup r_ARu}$

Obligations on the users: $\mathbf{sOu \cup r_AOu}$

Prohibitions on the users: $\mathbf{sPu \cup r_APu}$

Obligations on the system imposed by the users, and room A (or owner of room A): $\mathbf{uOs_A \cup r_AOs}$

f. Role. Role is associated with a level of privileges that determine the actions that a user can perform and the visibility of the services in a particular context. Depending on the role that the entity has, s/he may have different privileges in executing the service. For example, a user with higher level of role can do more things and may have more services available in the context compared to the user with lower level of role. In our system, we classify users into three different roles: a super entity, power entity and general entity. Each of these roles has different scope of service visibilities and activities that the entity can perform depending on the place that s/he is visiting and whether s/he is the owner of the place.

3.3 Policy language Notation

In designing a policy language, it is important to balance the convenience and compliance aspects, where a system has control over users' actions or activities, but does not overly restrict or control users' behaviours. This is possible by specifying rule per activity, in which only at certain occasions, the space will be in control. Ideally, the end-user would still be able to access services as per normal in all public places and circumstances, and only in some situations (e.g., during exam or meeting time), the space takes full or partial control over the service from users (e.g., allowing users to perform certain actions on the service or prohibiting users from performing any action on the service) as illustrated in Figure 3a below.

In addition, our policy design also takes into account the reusability aspect, in which the policy is stored on the server side and can be shared with other spaces in the system. This is possible, as in creating rule per activity, we do not explicitly specify the context information (e.g., space/location as well as the exact date and time of when and where the activity occurs). Instead, we store this context and activity mapping in an external file (see Figure 3b below). The mapping here works like a booking system, where it stores the user's schedule (in this case the owner of the space's or the public space's activities). The system then refers to this location_activity document to have an idea of the activity

running in the space. After that, it retrieves the relevant rule that matches this activity. It then enforces the rule to all users who visit the space when the contexts elapse.

This activity information can also be retrieved from sensing devices installed in the environment (e.g., using smart camera that could detect the user‘s activities and movements). We will continue to integrate this smart sensing device into our contextual system in the future. After describing the elements of our policy language, we now present our policy language notation in EBNF.

Subject ::= “SUBJECT(“ entityIdentity “)” “WITH” role “HAS” rules

entityIdentity ::= DATA

role ::= DATA

Rules ::= Rule | Rule “,” Rules

Rule ::= Activity policyObject “(“policyImposedBy policyImposedOn “)” targetService action

Activity ::= DATA

policyObject ::= “RIGHT” | “PROHIBITION”| “OBLIGATION”

policyImposedBy ::= DATA

policyImposedOn ::= DATA

targetService ::= DATA

action ::= DATA

**(DATA is a string in some format.)*

For example:

(1) SUBJECT (GU01) WITH ‘General Entity’ HAS $rule_1$, $rule_2$, $rule_3$

GU01 who has a general entity role has three rules at exam room.

$Rule_1$ = having lunch RIGHT(‘GU01 user’ ‘All users’) ‘any’ ‘start’

This rule is specified by GU01 user that allows other users to start any service during lunch time at her office (e.g., room B338).

$Rule_2$ = study PROHIBITION(‘GU01 User’ ‘All users’) ‘any’ ‘start’

The rule prohibits all visitors with any level of role to start any service when GU01 is preparing for her meeting (studying) at her office.

$Rule_3$ = having a meeting OBLIGATION(‘GU01 User’ ‘Space (roomB338)’) ‘any’ ‘stop’

The rule obligates the space (room B338) to automatically stop any running service during a meeting at room B338.

(2) SUBJECT (PU01) WITH ‘Power Entity’ HAS $rule_1$

PU01 who has a power entity role has one rule.

$Rule_1$ = having an exam PROHIBITION(‘Space (exam room)’ ‘General user’) ‘any’ ‘any’

The rule prohibits all users who have general user role (e.g., a student) and are sitting for exam to start any service at exam room. Other students who are not in the exam room (e.g., at school lounge) are still be able to access the service. At the exam room, the space only restricts students access to services, other users (e.g., staffs or hall supervisors) are still be able to access services as per normal.

The following XML document is a sample of a space policy based on activities that may occur in a space (as illustrated in Figure 3a below). We also give a sample of how the mapping between location, activity, day and time in our system (see Figure 3b below). The mapping and policy are created by the developer or owner of the space. The mapping is done per space (to customize the activities that may occur in the space), but, a generic policy rule can be shared. The advantage of separating the rule and context details is the rule does not have to be changed when the activity and contexts change, only the mapping needs to be updated when there is a new event or modification of an existing event. The rule can also be re-used by other spaces which have the same activity. This is possible as we have a consistent naming of activity throughout all spaces.

In a case where the activity at certain day/time is not specified in the mapping document (e.g., between 12-1PM and after 2PM as shown in Figure 3b below), the system then looks for activity name=”any” in the policy rule. During this time (activity=”any”), all visitors are given flexibility to access any service and perform any action. This then balances the convenience and compliance aspects in our system, where the space is only in control at some situations (activities), and the rest users could still access services as per normal. In addition, “any” on *service allowed* means any service as described in the user’s preferences for that particular contexts, “any” on action means any action that a service supports (e.g., a media player service has start, stop, pause and resume actions). “None” simply means no services will be visible or no actions are allowed at certain activity.

```
<Rule>
 <Activity name="Meeting">
  <Has policyObject="Right" by="System"
on="General_User">
      <Service allowed="Mobile Pocket Pad
Service">
     <Action allowed="Any"/>
  </Service>
  </Has>
  <Has policyObject="Obligation" by="System"
on="General_User">
  <Service obligated="any">
     <Action obligated="stop"/>
  </Service>
  </Has>
  <Has policyObject="Prohibition" by="System"
on="General_User">
  <Service prohibited="any">
        <Action prohibited="any"/>
  </Service>
  </Has>
 </Activity>
 <Activity name="Any">
```

```
    <Has policyObject="Right" by="System"
on="General_User">
          <Service allowed="any">
        <Action allowed="any"/>
    </Service>
    </Has>
    <Has policyObject="Obligation" by="System"
on="General_User">
    <Service obligated="none">
       <Action obligated="none"/>
    </Service>
    </Has>
    <Has policyObject="Prohibition" by="System"
on="General_User">
    <Service prohibited="none">
             <Action prohibited="none"/>
    </Service>
    </Has>
   </Activity>
</Rule>
```

(a)A sample rule document

```
<Location_Activity for="RoomB530" createdBy="Alice">
  <Activity_Details day="Monday" time="9-12PM">
    <Activity name="Meeting"/>
  </Activity_Details>
  < Activity_Details day="Monday" time="1-2PM">
    <Activity name="Out to lunch"/>
  </Activity_Details>
</Location_Activity>
```

(b) A sample location_activity mapping

Figure 3a and 3b: Sample policy and mapping documents

4. Policy Conflict Sources and Situations

This section discusses several possible sources and types of conflicts that may occur in pervasive environments, based on the policy design discussed in section 3. As each entity is assigned different specifications depending on the role that it has, there will be a chance of conflict occurrence. Conflicts arise due the differences including:

a) The differences in specification between entities on how the entities should behave. These differences lead to a potential or definite conflict that needs to be resolved as soon as the conflict is detected or just when the conditions for the conflict to happen are satisfied. We deal with two types of resources: shared resource services and non-shared resource services. A shared resource service refers to a software tool that is enlisted as the user needs it and it helps users to accomplish the tasks by downloading the application or mobile code onto a shared machine (usually a desktop PC machine). Some samples of shared resource services that we have developed are Mobile VNC [18] and Mobile Media Player Applications [29]. This shared resource service can be controlled and accessed by all legitimate users from their mobile devices in that specific location. Hence, there is a high chance of conflict occurrence here as there may be more than one user in the location trying to access or control the same shared resource service with different interests or specifications on what action to perform (i.e., start, stop, pause or resume) and when to perform this particular action. For example, one user may want to start music A, but another user in the same location wants to stop music A and start music B instead. A non-shared resource service, on the other hand, is a service that is downloaded and compiled onto a user's mobile device only. This service is running on the user's personal device and only accessible to that user (i.e., Mobile Pocket Pad Service [19]). Hence, there is a less chance of conflict here- conflict can still happen here between a user and a room (even if not between users).

b) The differences in the privilege that the entity has. For example, one user (with higher privilege) can execute more types of services at any time and any place compared to other users (with lower privilege) that can only execute certain number of services at certain place and time. In our system, the level of privilege is determined based on the level of positions or roles that the user has. As each entity has a different level of privileges, a user with a higher level role may override the execution of the shared resource service that has been started earlier by a user with lower role. This then leads to a conflict. The occurrence of the conflict further increases as we are dealing with mobile entities, in which, the entity can move freely from one geographical location to another (i.e., from one place to another place), and the entity carries its own role and rule on how the service should be executed (i.e., what action that s/he can perform) in the designated place. A conflict can happen if, for example, one user has started the service (i.e., start the music) and another user wants to stop the execution of the running music or perhaps start a different music. This type of conflict can occur for both shared resource services and non-shared resource service. We start from conflicts involving non-shared resource services:

- Between a user's obligation and a user's own action i.e., user A has a right to start music service and she is starting the music now. However, this user has also obligated the system to stop the music after some time. This leads to a conflict if the user obligates the system to stop the service but the user him/her-self manually starts the service from the device. This conflict can go on and on as the system will keep stopping the service the user starts, as the system would detects that the service is running. Another example is the user manually stopping the running music from the device which is just started by the system as the room obligates the system to do so.
- Between a system's obligation and a user's action i.e., user A has a right to start any service and s/he is starting the music service now. However, the system is obligated by the room to stop all the running services, including the service that the user has started on the shared machine or on her mobile device earlier.
- Between a system's obligation and a room's obligation i.e., a system is obligated by the user to start the music. At the same time, the room (or its owner) imposes an obligation on the user (visitor to the room) to stop the music. Another example is the system obligates the user to stop the music, but the room obligates the user to start the music.

Accessing shared resource service can also create conflicts as discussed above as well as inter-user conflicts, e.g. conflict

arises if user A has started music A and user B wants to stop the currently running music.
c) Conflict also occurs when more than one user try to access the same service but have different specifications on what to do with the service i.e., one wants to start a music service but another wants to stop a music service. The conflict in modality occurs between users, between a user/system and the room, between user and his/her manual execution from a mobile device. We start from accessing non-shared resource service:
- Between a system and a room i.e., one allowing the user to start the service (system) and the other is prohibiting a user from starting the service (the room) or one is obligated to start the service by the room and at the same time, the user is obligated by the system to stop the service.
- Between a user and his/her manual execution from a mobile device. For example, a user imposes an obligation to the system to stop the currently running Mobile pocket pad service from 12-2PM at B558 (during lecture time), however, as soon as the system does this (i.e., stop the service), the user manually starts this service again, because s/he wants to look up his/her online note i.e., the lecture contact details and this information is stored at online pocket note. The infinitive conflict occurs here as the system will automatically stop this Mobile pocket pad service if it detects it is still running.
For shared resource service would be the same as above plus the following: between two users with the same or different role in which, one user would like to start the service but others want to stop the service. This conflict comes from manual execution of the service (not from u_iOs, sOu_i or rOu_i).

5. Policy Conflict Resolution

We propose several conflict resolution strategies described as follows:

a. Role hierarchy overrides policy

The role hierarchy overrides policy is used if the conflict occurs between users that have different roles, in which a user with a higher role has much higher level of priority, and the conflict happens at a place which is not owned by a user with the low level of role and priority.

b. Activity priority overrides policy

This technique is used if a conflict occurs between two users that have different roles and the user with lower level of role has much higher level of priority of activity (assuming there are sensors and mechanisms to detect such activities) compared to the user with a higher level of role. For example, user A (student) is having an exam at room B (hence, high priority), user B (head of school) is in relaxing time (low priority). At this situation, a student's policy will be given a higher priority than user B (hence, we can override user B's rule).

c. Precedence overrides policy

This technique is used if a conflict occurs between users, in which one user has much higher role and higher level of priority than another; however, it occurs at a place where it is owned by a user with lower level of role and lower level of priority.

d. Room holds precedence over visitor

This technique is used if a conflict occurs between a user and a room. For example, the system permits a user to start a service at room A, but room A prohibits the user from starting this service. If there is a conflict, the room (representing its owner) always wins, regardless of the level of roles of the visitor. The user or visitor can also choose to challenge the room if s/he is not satisfied with the conflict resolution result.

e. Obligation holds precedence over rights

This technique is used if a conflict occurs between an obligation and the right. An obligation always wins over the right. For example, if the user is permitted by the system to start a media player service, but a room obligates a user to stop the media player service.

6. Implementation

MHS uses the Microsoft .NET Compact Framework technology that natively supports XML Web service calls. Our system consists of users with handheld devices and Web services that determine the location of a user, collect the user's context information and interpret the policy document, which are published via the system. This section gives a high-level description of these parts of the system, and how the parts interact. The system architecture is illustrated in Figure 4 below.

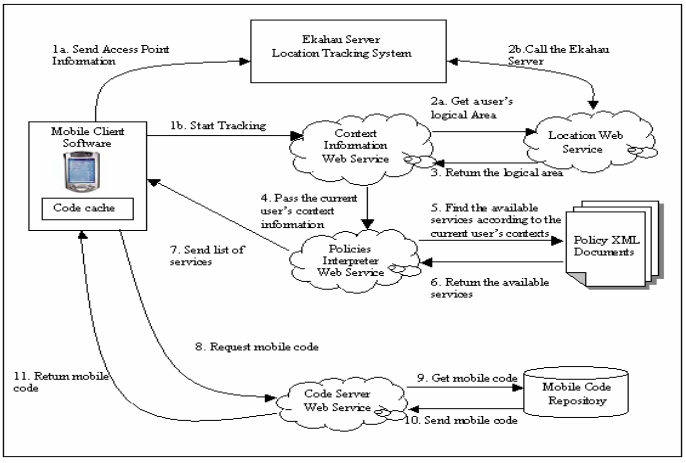

Figure 4: System Architecture of MHS system

Five of the main components of the system are discussed in the following sections:
a. Mobile Client Software. Users with mobile devices run software that continually polls a central Web service to discover services and policy information that are available at the user's current contexts (i.e., location, day and time). A push-based architecture will be considered in the future (see

conclusions). When the user selects a particular service, the mobile device contacts the central Web service and downloads an application for interacting with the selected service. The mobile client software also caches downloaded applications. The cache contains code and metadata describing applications. Hence, if a downloaded application is running, its cache code also exists in the memory.

b. Location – Web service. To realise location-aware services, this system employs the current release of the Ekahau Positioning Engine (EPE). The EPE is an indoor positioning system that keeps track of a user's location based on signal strength measurements. It also supports devices such as wireless PDAs, laptops and any 802.11b-enabled devices [23]. 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.

c. Context information Web service. Context information Web service is a context collector that collects all users' contexts information, which are specified by the system i.e., a user's current location, a user's identity, day and time. The context collection process is done by calling the respective Web service i.e., to get a user's current location, the context collector needs to invoke the location Web service method. The updated current day and time are obtained by checking the current system day and time. The user's identity is retrieved from the login form, once the user logs on to the system. After retrieving all the required contexts, the context collector then passes these contexts information on to the Policies interpreter Web service. The reason for separating the collection tasks from the policy interpreter is we want to have a modular and extensible system, in the sense that we only need to update a single component (e.g., the context collection component) if there is a new context added in the future. Allowing a system to add additional context information sources is considered as a feature of an extensible system.

d. Policy Interpreter Web service. The policies interpreter component is called after the system retrieves all the contexts information. The policies interpreter parses the user's policy document that specifies when and where to start the service. This interpretation process is done on the server side and it takes into account information regarding the user's current contexts i.e., a user's current location, day, time and a user's identity. If there is a service associated with these contexts, the service information (service name, service location, action type, music name, day, time to start and end service) is then returned.

e. Code server Web service. Within our system, we employ the Web service as a method invocation to retrieve a mobile code application that matches the service name and this service method then returns the particular mobile application to the client device.

The following paragraphs describe each of the steps in Figure 4 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 Mobile Hanging Services system. 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 s/he enters is valid. If the user is valid, the system then invokes a Web method of the Context Information Web service called "Start Tracking" by passing 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 the IP address of the device to this Web method. The Location Web service then fires the Location Listener Application on the Ekahau Server. The Listener application then is continuously listening to the mobile client's movement. Finally, this Web method returns the most accurate user's logical area to the method caller (e.g., Context Information Web service). Besides retrieving the user's location information, the Context Information also retrieving other contexts used in the system i.e., current day, time and a user's identity.

4. Pass the current user's context information. Once, the context information retrieves all the contexts information, the context information Web service then continues to call Policies Interpreter Web service by passing these context details.

5 and 6. Find and return the available services. Once the policy interpreter Web method is called, the system then interprets the user's policy document that matches with the user's current context. If there is any context that matches with the user's policy document, these service information is then returned. In this implementation, a user's policy document that specifies when and where to start the service is described in an XML language.

7. Send a list of Services to the Mobile Client. If the services are found, a list of services and its policy details will then be sent to the mobile client. The mobile client application then displays these set of service names as specified by the policy.

8 and 9. Request a Mobile Code, Get and Send a Mobile Code. When the user chooses a service from this list, the code server Web service is contacted to provide code for invoking the selected service.

10. Return a Mobile Code to the Mobile Client. This returns the mobile code applications to the client device. Upon its arrival, the mobile client application then loads and

processes this service application, finally executing and displaying the service interface on the mobile device.

6.1. Partial control between users and systems

As our system supports both manually and automatically execution of the service, it is important to clearly separate the control between users and systems i.e., when control should be given to the user and when the system should be in control. The control becomes extremely important especially, when the user performs some odd activities during the day, which is different from the tasks that s/he has specified in the policy document. For example: a user is having a group meeting at room A (the user's office). He specifies in the policy document to start the music at his office at 3PM (basically, after finish the meeting). However, what happens if the meeting has not finished at 3PM. As discussed before, as soon as the current time shows it is 3PM, the system will automatically start the music in room A. The system does this automatically by interpreting a user's policy document and it will not be able to tell whether the meeting is over or not. If such situation happened, most likely, the user will want to manually stop playing the music from his/her mobile device. This is done through selecting the service name i.e., Remote Media Player on the mobile device and a mobile code with respect to this Remote Media Player service will then be downloaded to the user's mobile device.

Once the service interface is displayed, the user then clicks on the stop button to manually terminate the playing music. Once the system detects that the user is manually performing the task and this task is different from the activity that s/he has specified in the document, the full control is now given to the user. The system will not perform any further policy interpretation (and music execution) until the system detects that the user has closed the Remote Media Player service form. Once, the form is closed the full control is now returned back to the system. The system then continues to interpret the user's policy document and automatically start, pause, resume or stop the music. In summary, our current control scheme is as follows: the user takes control of the service by requesting and using the mobile code application (containing the user interface) for the service and control is returned to the system when the user closes this application.

Our policy implementation is developed on top of our previous prototype as described in [20]. The policy software components only get called when the service interface has been displayed and the mobile user is requesting to execute a certain action on the service i.e., by clicking on the start button on the media player service interface on the mobile device. Our policy implementation is modular, interoperable and extensible. We separate the policy tasks according to its functionality i.e., we have a separate web service method for policy interpreter, conflict detection, resolution and manager. Hence, we only need to update a single component (i.e., the context collection component) if there is a new context added in the future.

In addition, we also separate the policy implementation from the services (or their mobile code) implementations. This allows our system to easily add additional services in the future and we may need to have only one policy document for all services or applications that we have in the system. Moreover, as we create each of our software components as web services, this makes our software functionalities accessible in disparate platforms and languages. Figure 5 below describes in detail how our policy mechanism works.

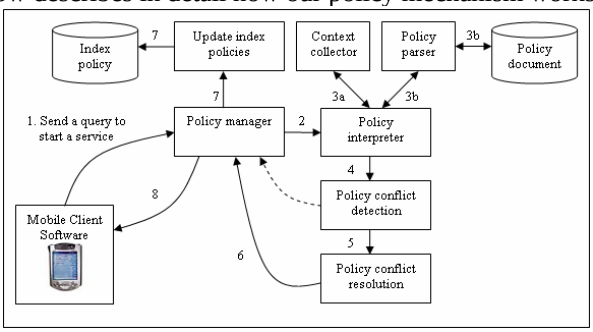

Figure 5: MHS Policy implementation

The steps in Figure 5 are:

1) Request an action (e.g., start) on a service.

Once the service interface is displayed on a mobile device, a mobile user can request to start a music on a media player service by clicking on the "start button". When there is a request from the user, the mobile client query manager then passes this query on to a policy manager (i.e., to decide whether or not the user is permitted to start the service with a particular song name).

2) Call the policy interpreter

There are a few steps needed to be performed by a policy manager in order to answer the user's query such as calling the policy interpreter to collect information regarding the user's current context and the relevant policy documents.

3a) Retrieve the context information

The policy interpreter first calls the context collector to collect all users' contexts information i.e., a user's current location, a user's identity, current day, time, and who else in the location. The context collection process is done by calling the respective Web service i.e., to get a user's current location, the context collector needs to invoke the location Web service method. The updated current day and time are obtained by checking the current system day and time. The user's identity is retrieved from the login form, once the user logs on to the system.

3b) Retrieve the entities' policy documents

After knowing who the requesting user is and how many users in the location for the given context, the policy interpreter then gets and parses the relevant entities' policy documents that specify when and where to start the service.

4) Pass the information on to conflict detection

After collecting all the required information, the policy interpreter then passes this information on to a policy conflict detection component. The conflict detection looks for potential or actual conflicts between a user and system and between a user and the room for that location context. The strategy used to detect the conflict here is based on the reactive detection strategy, in which the procedure for detecting conflicts is only initiated when there is a request

from a user to perform an action (in this case, when the user is clicking on the action button on the service interface). This technique is considered easy to develop and maintain. However, there are two issues that we need to consider:

1. When to trigger the conflict detection procedure, and

2. When to update the conflict detection result.

We can certainly trigger conflict detection when there is a request from a user. However, this leads to system performance and resources issues, as the conflict detection is called each time there is a request from a user, regardless of the user's current location, the name of the service and action that the user is requesting. Hence, the performance will slow down (the user needs to wait a long time to see the response result) and resources may be wasted (e.g., the conflict detection result that is just computed may be the same as the previous result, due to the user's location context being still the same).

One solution to this issue is caching the detection results which have been computed earlier. Caching the result is useful to avoid calling the same method with the same action and context again and again. The conflict detection component only needs to be called once, when it is the first time a user is selecting a particular action. Subsequent requests for the same action will not trigger the conflict detection process. Instead, it will read the result from the cached file. With this technique, we only need to update the cached result if the system detects that the user's location context has changed (i.e., user A has moved from B536 to B558). Hence, the system needs to redetect conflicts for the new context (or location, e.g., B558).

The following is the procedure that we employ to detect the conflict:

i). Checking a user's request

When there is a request from the user to perform some actions on the service, the system then checks this request against the permission that the system gives to the user for that particular context. If a permission rule specifies the intended user's action, this means the user is permitted to perform the action. If this happens, the system then continues with conflict detection and does not have to check it against the prohibition rule as we assume that the policy objects within one policy document are consistent (i.e., the tasks which are permitted are not also prohibited by the same entity). Note that this assumption is only valid within one policy document. We still need to check the consistency of one entity's policy against another as the system may permit the user to perform the action but the room or other users may prohibit the user from doing this. However, if there is an absence of permission (the permission rule does not mention the user's intended action), the system then has to look up prohibition rules. If the prohibition rule specifies the user's intended action, the system would forbid the user from performing the action. In addition, if there is an absence of permission and prohibition (none of these says anything about this intended action), the system then prohibits the user to perform the action (taking a conservative view). If there is permission given by the system to perform the specified action, the conflict detection then continues to step ii. However, if the system does not permit the user to execute the specified action, the system does not have to continue to detect further conflicts. The policy interpreter then passes the result to the policy manager (continue to step 6).

ii). Checking against room's policy

If the user is permitted to start the service by the system, we then continue to check this against the room's policy (whether or not the room gives permissions to this user to start the service). If the room does not permit the user, this conflict detection result will then pass onto the conflict resolution to be resolved (continue to step 5). By default, if there is a conflict between a user and room, the room always wins. However, the visitor is free to challenge the room if s/he is not satisfied with the conflict resolution result considering his/her current situation.If there is no conflict between a user and room or if there is a conflict, but the user wins the challenge, we then continue with step iii.

iii). Checking against other users in the current location

After checking the request against the room's permission, our system also further checks this against all other users' policies in that location. This is done by creating an index policy document at run time that indexes all the relevant entities' policies based on the service name or day or location. In our system, we index the policy based on the service name. Our index policy document contains information regarding all the entities' name, roles, actions, condition days, times, durations and the state of the services (i.e., running or not running) for that location context. By default, all services are in idle state. The state changes to "running" when there is a request from the user to start the service and this request is approved by the system (as there are no potential conflicts detected when executing this service). Again, when this user stops the service, the system then updates the service state to "not running". Keeping track of the service state here is useful to avoid the conflict in effect of service. For example, if user A has a right to start the service and user B also has a right to stop the service. As soon as user A started the service, the system will keep track of the duration of the service that is just started. Hence, during this period, user B is not allowed to stop the service (although s/he has a permission to do so). The only request that can modify the state of the currently running service is the obligation - our system places a higher priority on obligations than on rights. In addition, we also update our cached index policy document periodically i.e., every 5 seconds as there may be a new user entity moving into a place or the existing user has moved out of a place.

iv). Execute action only if no conflict is found

If there is no conflict found in the index policy document, the system then allows the user to execute the specified action. We then continue to step 6.

5) Call the conflict resolution

If there is any conflict detected in steps i, ii, iii, or iv above, this conflict detection result will be passed onto the conflict resolution to be resolved. Our conflict resolution resolves all the potential and actual conflicts detected and caches this resolution result for future reuse. In addition, the conflict resolution technique that is used here depends on the contexts of the conflicts (see section 6).

6) Send result to the policy manager

If no conflict is detected, the conflict detection module then sends a message to the policy manager (i.e., allowing user A to execute the specified action as no conflict has been detected). However, if there is a conflict, the conflict resolution result is then sent to the policy manager.

7) Update the index policy

The policy manager then updates the index policy based on the conflict detection or resolution result i.e., changing the state of the service from idle to "running" or from "running" to "not running".

8) Send a message back to the client

After updating the index policy document, the policy manager then sends a message back to a mobile client manager. This can be either allowing or disallowing a user to execute the service. If it is allowed, the user is then permitted to perform the requested action (i.e., start music A at B536). In addition, the system also sends back to the client the conflict detection and resolution results. The mobile client manager then caches these results on the mobile device for future re-use. The mobile client manager updates the cached results periodically - every 5secs.

Figure 6 shows the user's perspective on the system. The conflict detection and resolution procedures are initiated when the user clicks "play" (requesting an action to be performed).

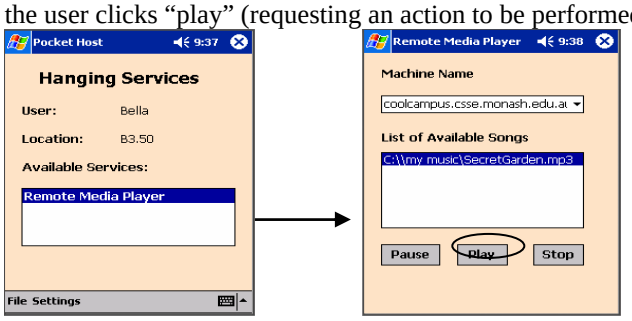

Figure 6: Mobile user perspective screen shots

7. Evaluation

We have discussed in [17, 19, 30] our MHS performance evaluation results starting from 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. In this section, we evaluated several aspects of policy checking starts from detecting a user's current location, retrieving a policy based on that location, downloading and caching the location policy result on the user's mobile device, up to detecting whether or not a user is given a permission to perform the requested action on the service, as well as detecting and resolving a conflict with other entities if any. We also measured the total user wait time in various scenarios such as:

(a) How long a user should wait to receive a response from a server regarding the requested action, when it is the first time entering a location and first web service call,

(b) How about if the user moves to another location, how long does it take to update the policy cached result on the device?

(c) How about if the user closes the MHS application and after some time, the user decides to start the application. How long does it take to respond to a user's request if all policy results for that location have been previously cached on the mobile device?

(d) How about if a location policy gets modified by an owner of a place or a developer, how long does it take to update a copy of that policy on each of a user's device?

The policy evaluation results are illustrated in Figure 7 below. Based on Figure 7, we can see that the time required to call Web services: send a query from a client to policy manager, retrieve and download a relevant policy decision document, detect conflict between users (check against location-service status document), resolve conflict dynamically, and send back result to the mobile client manager decreases for subsequent Web service calls. As explained in the previous evaluation section, the first call of the Web service takes longer time, as the system needs to download and compile the local host Web service proxy object on the device. In addition, the first time a Web method on a Web service is called causes the SOAP client to reflect over the Web service proxy object. To mitigate this delay, we declare the Web service object globally within a class and call a simple Web service method asynchronously on the form load or constructor. The subsequent calls of this Web method will not incur this reflection overhead and so, it takes a much shorter time to complete the process. Moreover, the subsequent calls of other web methods on the web service will not incur downloading and compiling of the web service proxy object (as it only needs to be done once, when the first time calling a Web service), and so, in general, it also helps reducing the time to call other web methods on the web service.

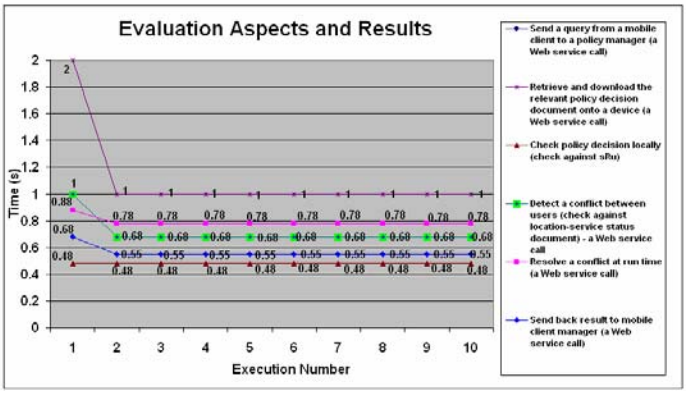

Figure 7: Policy Evaluation Results

When a user moves from one location to another, our system continuously monitors a user's location and retrieves the policy information accordingly. Based on the testing and evaluation results, the worst case scenario to retrieve a policy decision result is the first time of Web service calling. This means, when the first time a user enters a location and the first time a Web service calling, it takes 2s to retrieve and download the policy decision result onto a user's mobile device. However, as our system has previously called this web method on the constructor, retrieving the policy decision result upon the MHS client application is loaded only takes 1s. This can be considered as a subsequent call of this Web method as the first one has been done during the initialization in the constructor.

Upon retrieving the location policy document, the mobile client manager stores this temporarily on the local system data set and writes it to an external XML file on a device, when a user closes the MHS client application. Storing the policy decision result locally is considered useful in order to speed up the policy decision checking (check against sRu). As a result, the subsequent policy checking does not need to contact the Web server to re-download the relevant policy document, and so, no policy needs to be re-cached (0s). Instead, it will just locally check on the local data set. Moreover, when a user moves to another location and the policy decision has not been previously downloaded onto a device (this may be because the user has not visited the location previously), the client then contacts the system to retrieve and download the specified policy decision result. In this case, it takes 1s.

In a situation, where the user moves back to a location where a policy has been cached, the subsequent policy checking does not incur a policy downloading process, instead it just reads from the local policy result dataset and hence, the checking can be done in minimum amount of time. In addition to it, the policy decision result which has been stored on the mobile device only needs to be updated, when the policy for that location is modified by a developer or an owner of the place. In this case, it only takes 1s to retrieve and download the updated policy document from a server to a client (assuming the user's location has been known or detected by the system earlier). In addition, when the user has closed the MHS client application and decided to open it after some time, it takes 2s for reading the cached policy XML file. As we also cached the policy decision result and store it as an XML file, the subsequent policy checking after a user closes the application still do not require a policy to be downloaded from the server and so, reduces the user wait time.

In general, the time that is required to check for the policy depends on the type of services that a user wishes to execute. It takes longer time for executing an action on the shared-resource service compared to a non-shared resource service. This is mainly because there is more checking that needs to be done. The policy checking here is triggered when a user clicks on the action on the service. The policy checking formula comprises of:

$$T_{\text{policuy checking(s)}} = T_{\text{checking against sRu}} + T_{\text{checking conflicts between entities (check against location-service status document)}}$$

Based on the testing and evaluation results, it takes 0.48s to check the policy decision against sRu for the first and subsequent requests. It remains steady throughout our executions as checking only needs to be done locally and does not involve any Web service calling. The only aspect that influences the amount of time required to check for the policy decision locally is the device's processor speed and the number of applications running at the time. The faster the speed, the shorter it takes to complete the checking. Moreover, more applications running during the checking would result a much slower response from CPU to handle the checking. The amount of time required for checking whether or not there is a conflict if the specified action of the service is executed, varies depending on the number of executions. This is mainly because, it is implemented as a Web service, and hence, the first time of Web service calling takes much longer time (i.e., 1s) than the subsequent requests (i.e., 0.68s).

As for executing a non-shared resource service, it only takes 0.48s (=0.48+0) to check for policy decision result for the first and subsequent requests. The shared resource service would take 1.48s (=0.48+1) for the first time checking and reduces to 1.16s (=0.48+0.68) for subsequent checking. We can further improve this policy checking by implementing a group or compound checking, in which, checking is not done only per action. The system may group the checking here as per service, and so, we can reduce the unnecessary checking for every single action. For example, by displaying a service to a user that would mean all actions on the service are allowed or by allowing a user to execute a service (i.e., service A) would also allow a user to execute another service (i.e., service B).

Finally, we present a formula to calculate the total user wait time to request to execute an action on shared or non-shared resource services till the system responds back to the user. This requires a system to detect the user's location, display a list of available services, and download the relevant policy document up to checking a policy decision result. This formula is illustrated as follows:

$$T_{\text{user wait time(s)}} = T_{\text{location context change delay}} + T_{\text{retrieve or update policy decision result}} + T_{\text{policy checking}}$$

Based on the formula above, we conclude that the worst-case scenario for the user wait time when the user first enters a

location (first time calling a policy result web service) and wishes to perform an action on the shared resource service is the first time of requesting the service, which takes 10.48s(= 8.6 + 1.48). For non-shared resource service is 9.08s (= 8.6 + 0.48). The best case scenario i.e., the minimum time delay to get a response back from the policy manager is in any execution which is not the first (assuming the location context for subsequent requests is still the same, and so there is no need to retrieve or update the policy decision result as well as a list of services). In such a case, the delay time is 0.48s (=0+0.48) for non-shared resource service and 1.48s (=0+1.48) for shared resource service. The delay can be minimized as the system does not need to retrieve the updated services and policies as the location is still the same. Here, we only need to check for the policy and no location context change is required. The delay time to detect subsequent requests decrease to 0.48s for non-shared and 1.48s for shared resource service, because, the subsequent requests re-use the local cached of policy decision results which have been previously downloaded (in the first run) for that location and no need to update the list of services.

In a case, where only policy gets modified (a list of services in a location still remains the same) as the user is still in the same location, the system only needs to update the policy document. In this case, the user wait time is 2.48s (=1+1.48) for shared resource services and 1.48s (=1+0.48) for non-shared resource service. In addition, when the user moves to another location (i.e., from location A to location B), the user wait time is 6.98s (=6.5+0.48) for non-shared and 7.98s (=6.5+1.48) for shared resource service – assuming here, the policy decision result has not been previously cached on the device as it is the first time a user visits the location and it is considered as subsequent Web service calling. In a situation where, the policy decision result is already on the device and the user re-visits the location and there is no policy modification or list of services that need to b e updated, the user wait time for shared resource service would only be 1.48s (=0+1.48) and for non-shared resource service is 0.48s (0+0.48).

8. Lessons Learnt

Designing a system based on the user centric approach has now become one of the key factors that plays a significant role to empower users to be more effective in completing their daily activities. In addition, on top of having a user centric designed system, employing a policy mechanism to utilize contextual services is also considered important in pervasive systems. This is due to users in pervasive environments tending to be always on the move and is allowed to access services at any place and any time that s/he wishes to. However, in some circumstances, the space where the users are visiting to, may want to be in control by asking all visitors (foreign users) to obey rules which have been pre-specified. Perhaps, the space has the intention to restrict the behaviours of foreign users in accessing services in that particular space.

Having an additional policy mechanism in pervasive systems certainly benefits and maximises the user's experience. This is mainly because entities are given a privilege to control or restrict the behaviours of other entities in accessing services in particular contexts by defining a rule or policy which specifies when, where and what type of services that s/he permits, obligates, or prohibits others from accessing. However, there are challenges to developing such a system: **(a)** we need to detect and handle all possible conflicts that may occur in pervasive environments. The possible conflicts may vary from one pervasive system to another, and depends on the system design such as the contexts, entities, policies, and services used. **(b)** to have a scalable mobile framework that can support a number of different policies (i.e., a policy from a user, system, service or physical room) and have an interoperable framework where the policy functionalities can be easily invoked and accessed from different platforms and languages.**(c)** to develop a simple and robust policy language that can be easily understood and used with context aware services in pervasive systems.

After designing, implementing and testing a sample policy application, we conclude that integrating a context-aware system with a policy mechanism offers several advantages and disadvantages. The main advantage is the system or space having control over users' (e.g., visitors') behaviours in accessing services in particular contexts (e.g., specific day, time and location) and hence, can limit conflicts that may occur between users who are trying to access the same service (e.g., a music service) on the same target device with different actions (i.e., one wants to stop and another wants to start the music service). Also, it restricts users from performing prohibited actions at specific contexts (e.g., during exam time, all students are not allowed to retrieve their online notes on the "e-note service"). A drawback here is a user would not have as much freedom, and flexibility as s/he is used to having in accessing and executing actions on the service. This drawback in integrating policy with a context-aware system can be solved by balancing the convenience and compliance aspects, where a system has control over users' actions or activities, but does not overly restrict or control users' behaviours. Ideally, the end-user would still be able to access services as per normal (depending on his/her role) in most places and circumstances, and only in some situations (e.g., during exam time or meeting time), the space takes full or partial control over the service from users i.e., only displays certain services, allowing users to perform certain actions on the service or prohibiting users from performing any action on the service.

9. Related Work

This section provides a brief overview about the research work that has been done to date that also concentrates on developing a framework for context aware application. While many authors have acknowledged the usefulness of context-awareness in the pervasive environment, only little work has been done to-date that supports a mobile framework in this

field. Some earlier mobile context aware frameworks are the Hodes system [7] and Hive [8].

The Hodes system [7] introduces a concept of mobile context aware framework by employing the mobile code for 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 from the student in the same location.

Another mobile context aware framework in the field is Hive [8]. Hive is a distributed software agent platform that uses a combination of wearable and pervasive computing to address the concept of context-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 pervasive and wearable have presented fundamental difficulties [8]. This is due to the pervasive computing tend to have troubles 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. In contrast, our system is implemented on the highly compact mobile environment (.NET Compact Framework) that 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 also enhances some policies used for service execution.

There are also some various other context-aware applications surveyed in [2]. However, none of them are using mobile code and positioning technology like the way we do. Among previous work on exploiting a framework for context aware applications, much work has been done on location aware systems. Some existing projects, which have successfully developed an application that is aware of the user's location are Mobile Shadow. The Mobile Shadow [9] project focuses on prototyping applications for proactive cell-based location aware services with mobile code. Another project that also makes use of location sensing is Virtual Tour Guide [10]. A Virtual Tour Guide project uses a GPS system to detect a user's location in an outdoor environment. As soon as the user walks past or enters an area, which is delimited by the GPS coordinates, a relevant stick e-note to a physical location is delivered to the user's mobile device. A Cool town project from Hewlett Packard also developed a project that makes use of the knowledge of the user's physical location [11]. The interesting part from this project is the creation of a mobile WWW infrastructure with respect to the physical location. As the mobile user moves toward the physical space, the relevant WWW pages are displayed on the user's mobile device. In addition, some other projects in location context-aware field are described in [12, 13].

Another interesting aspect that is focused in this section is the policy. Policy is defined as a rule to govern the behaviour of the system i.e., the way the system needs to be executed. To date, most of the policy projects focus on implementing flexible and adaptive systems in the field of networking, security and distributed internet system [14, 15]. There is also some other policy works surveyed in [16]. From some research findings in policy, we believe that only little work has been done to date that implements a policy language in the location aware or context aware pervasive environment.

Although, they are dealing with the context, but their definition of context is different from us. In our work, we focus more on the user's contexts (i.e., a user's intention, profiles, behaviours, location, current time and etc). On the other hand, most of the policy works in pervasive environment focus on the context of the agent, a system, networking and access control security rather than context of the user. Some of these closely related policy projects are the spatial policy framework [20] and Rei Policy Language [21, 22].

The spatial policy framework aims to control the execution of the mobile agent in location based services environment. Depending on the location that the user enters, a mobile agent will have different states of the executions i.e., user A is currently listening to the news in his room. A "news agent" is then running at this stage. But, if user A decides to go to a different room (i.e., user B's room), different states of the agent will be encountered i.e., the agent may be frozen or killed. The state of the agent is specified by the user's policy. In this case, each user specifies what agents need to be started or terminated at a certain time and location and what actions need to be performed if there is an external agent that comes across to its place (the action can be to continue the execution or kill the external agent).

This framework also defines the concept of role. In here, role refers to the position of the user in the environment i.e., a boss, manager and staff. In the case of conflict between users that have the same role, the system will seek a policy resolution from a user with a higher level of authority. In this case, a boss which is the top level of the hierarchy can write a policy to override the other users' policies.

However, this project has not incorporated any development of user interface into a mobile device. The portable device that is used in this project is a laptop. In general, this project only focuses on the development of a security policy language that restricts the behaviour of the mobile agent. In contrast, in our system, we aim to provide policy languages for a context aware pervasive environment. The policy language here is used to govern the service execution according to the user's needs i.e., a user only wants to see a Media Player service at the certain context, then the system displays this Media Player service in that context only. Based on the policy, our system will deliver different types of services to the user depending on the user's current contexts.

The Rei Policy Language was developed by some researchers at HP Labs. The aim of this policy language is to provide flexible contracts based on deontic concepts that are

reusable across domains (such as at networking and access control security) [22]. Some parts of these projects including the policy engine are still under development. Our model to some extent has similar philosophy to the Rei Policy Language, since we aim to develop a policy language that can target multi domains in pervasive environment, but we focus on mobile services.

Many policy projects focus on implementing flexible and adaptive systems in the field of networking, security and distributed internet systems [9, 10]. There is also some other policy works surveyed in [11]. We believe that research is ongoing for policies in location aware or context aware pervasive environments. In our work, we focus on the user's contexts (i.e., a user's intention, profiles, behaviours, location, current day, time and a user's identity) but many other policy works in pervasive environment focus on the context of the agent [3, 5, 12, 20], networking [16], access control [13], and security [14, 15, 17] rather than context of the user. But two closely related policy projects are the Spatial policy framework [20] and Rei Policy Language [3, 5].

10. Conclusion and Future Work

We have presented an architecture for "Mobile Hanging Services", allowing a mobile device to adapt its functionality to exploit a set of services that it discovers depending on the user, location, day and time contexts. We proposed that adding context awareness and some rules or policies to the traditionally designed application helps to improve the user's experience in using the system, especially if there is regularity in the user's activities. We also have developed a prototype implementation of adding context awareness and a simple policy document into a traditional Windows Media Player application.

We conclude that having a policy in pervasive environments is generally useful as it can be used as a tool to govern and control the entities' behaviours. Moreover, it can also help the user to express actions to be automatically executed (as obligations the user imposes on the system). In several experiments with our prototype, the only drawback that we experience is the additional delay in responding to the user's request due to conflict checks and resolution when needed (1-3s additional delay after click). The user wait times in our policy system can be further reduced by exploiting different techniques to detect and resolve the conflict i.e., detecting and resolving the conflict proactively. *Some aspects that need to be further analysed and developed are: (a) Considering multi users in one physical location. (b)* We may want to use Semantic Web language with Ontology. (c) Have dynamic and multiple roles. (d) We need to ensure the consistency of policy objects within one policy document. (e) What is the penalty if the user forgets or does not want to complete his/her obligation?

References

1. Weiser, M., "The Compute for the 21st century", Scientific American, 94-104, Sep 1991.
2. Chen, G. & Kotz, D. (2000), "A Survey of Context-Aware Mobile Computing Research", Dartmouth Computer Science, *Technical Report TR2000-381* [online], Available: ftp://ftp.cs.dartmouth.edu/TR/TR2000-381.pdf [Accessed 03 April 2003].
3. Prekop, P. & Burnett, M. (2003), "Activities, Context and Pervasive Computing", *Special Issue on Pervasive Computing Computer Communications*, vol.26, no.11, pp.1168-1176.
4. Schmidt, A., Beigl, M. & Gellersen, H-W. (1999), "There is More to Context than Location", *Computers and Graphics*, vol.23, no.6, pp.893-901.
5. Turner, R.M. (1998), "Context-Mediated Behaviour for Intelligent Agents", *International Journal of Human-Computer Studies*, vol.48, no.3, pp.307-330.
6. Castro, P. & Muntz, R. (2000), "Managing Context Data for Smart Spaces", *IEEE Personal Communications*, vol.7, no.5, pp.44-46.
7. Hodes, T.D. & Katz, H.R. (1999), "Composable ad hoc location based services for heterogeneous mobile clients", *Wireless Networks*, vol.5, no.5, pp.411-427.
8. Rhodes, J.B., Minar, N. & Weaver, J. (1999), "Wearable Computing Meets Pervasive Computing: Reaping the best of both worlds", *Proceedings of the 3rd International Symposium on Wearable Computers*, San Francisco, USA, 18-19 October 1999, pp.141-149.
9. Fischmeister, S., Menkhaus, G. & Pree, W. (2002), "Context-awareness and Adaptivity Through Mobile Shadows", Software Research Lab, University of Salzburg, Austria. *Technical Report TR-C047* [online], Available: http://www.softwareresearch.net/reports/C47.pdf [Accessed 12 April 2003].
10. Brown, P.J., Bovey, J.D. & Chen, X. (1997), "Context-Aware Applications: From the laboratory to the marketplace", *IEEE Personal Communications*, vol.4, no.5, pp.58-64.
11. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopall, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B. & Spasojevic, M. (2000), "People, places, things: Web presence for the real world", *Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, CA, USA, December 2000, pp.19-28.
12. Beadle, H.W.P., Harper, B., Maguire, G.Q. & Judge, J. (1997), "Location Aware Mobile Computing", *Proceedings of the IEEE International Conference on Telecommunications*, Melbourne, Australia, April 1997, pp.1319-1324.
13. Davies, N., Cheverst, K., Mitchell, K. & Friday, A. (1999), "Caches in the Air: Disseminating Tourist Information in the Guide System", *Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications (WMCSA)*, New Orleans, USA, 1999, pp.11-19.
14. Moffett, J.D. & Sloman, M.S., Policy Conflict Analysis in Distributed System Management, Journal of Organisational Computing, Vol. 4 no 1, pp 1-22 (1994).
15. Marriott, D.A., "Policy Service for Distributed Systems", Thesis, June 1997.
16. Damianou, N., Bandara, A.K. Sloman, M. and Lupu, E.C., "A Survey of Policy Specification Approaches", Department of Computing, Imperial College of Science Technology and Medicine, London, UK.
17. Syukur, E., Loke, S.W. and Stanski, P. (2004), "Performance Issues in an Infrastructure for Mobile Hanging Services", *Proceedings of the First International Conference on Mobile*

Computing and Pervasive Networking (ICMU), NTT DoCoMo R&D Center, Yokosuka, Japan, 8-9 Jan 2004, pp.32-37.

18. Syukur, E., Loke, S.W. and Stanski, P., "The Mobile Hanging Services Framework for Context-Aware Applications: the Case of Context Aware VNC", WIS (Wireless Information Systems) Workshop, Porto, Portugal, April 2004.
19. Syukur, E., Cooney, D., Loke, S.W. & Stanski, P. (2004), "Hanging Services: An Investigation of Context-Sensitivity and Mobile Code for Localised Services", *Proceedings of the IEEE International Conference on Mobile Data Management*, Berkeley, USA, 19-22 Jan 2004, pp.62-73.
20. Scott, D., Beresford, A. and Mycroft, A., "Spatial Security Policies for Mobile Agents in a Sentient Computing Environment", LNCS 2621, pp. 102-117, 2003.
21. Kagal, L., "Rei: A Policy Language for the Me-Centric Project", Technical Report, HP Labs, Palo Alto
22. Kagal, L., Finin, T. & Joshi, A., "A Policy Language for a Pervasive Computing Environment",
23. Ekahau Positioning Engine™ 2.0 Developer Guide [Available upon commercial purchased].
24. Syukur, E., Loke, S.W. and Stanski, P., "The Mobile Hanging Services Framework for Context Aware Applications: An Experience Report on Context Aware VNC". Technical Report no:151/2004, Monash University, Australia.
25. Bettini, C., Jajodia, S., Wang, X.S. and Wijesekera, D., "Provisions and Obligations in Policy Rule Management", Journal of Network and Systems Management, Vol. 11, No. 3, September 2003.
26. Moffett, J.D. and Sloman, M.S., "Policy Hierarchies for Distributed Systems Management", IEEE Journal on selected areas in communications, VOL.11 No.9, December 1993.
27. Scott, D., Beresford, A. and Mycroft, A., "Spatial Policies for Sentient Mobile Applications", IEEE Policy Workshop 2003, 4-6 June 2003, Italy.
28. Godefroid, P., Herbsleb, J.D., Jagadeesan, L., Li, D., "Ensuring Privacy in Presence Awareness Systems: An Automated Verification Approach", Proceedings of the 2000 ACM conference on Computer supported cooperative work, pp. 59-68, United States.
29. Syukur, E., Loke, S.W. and Stanski, P., "A Policy based framework for Context Aware Ubiquitous Services", Proceedings of the Embedded Ubiquitous Computing Conference, Aizu-Wakamatsu, Japan, LNCS, vol. 3207, Springer-Verlag, pp.346-355, 2004.
30. Syukur, E., Loke, S.W., and Stanski, P., "Methods for Policy Conflict Detection and Resolution in Pervasive Computing Environments", Proc. of Policy Management for Web Workshop, Chiba, Japan.