



MHS: a context-enabled regulated framework for pervasive services

MHS: pervasive services

Evi Syukur

*School of Computer Science and Software Engineering,
The University of New South Wales, Sydney, Australia, and*

Seng Wai Loke

*Department of Computer Science and Computer Engineering,
La Trobe University, Melbourne, Australia*

47

Abstract

Purpose – Pervasive computing environments such as a pervasive campus domain, shopping, etc. will become commonplaces in the near future. The key to enhance these system environments with services relies on the ability to effectively model and represent contextual information, as well as spontaneity in downloading and executing the service interface on a mobile device. The system needs to provide an infrastructure that handles the interaction between a client device that requests a service and a server which responds to the client's request via Web service calls. The system should relieve end-users from low-level tasks of matching services with locations or other context information. The mobile users do not need to know or have any knowledge of where the service resides, how to call a service, what the service API detail is and how to execute a service once downloaded. All these low-level tasks can be handled implicitly by a system. The aim of this paper is to investigate the notion of context-aware regulated services, and how they should be designed, and implemented.

Design/methodology/approach – The paper presents a detailed design, and prototype implementation of the system, called mobile hanging services (MHS), that provides the ability to execute mobile code (service application) on demand and control entities' behaviours in accessing services in pervasive computing environments. Extensive evaluation of this prototype is also provided.

Findings – The framework presented in this paper enables a novel contextual services infrastructure that allows services to be described at a high level of abstraction and to be regulated by contextual policies. This contextual policy governs the visibility and execution of contextual services in the environment. In addition, a range of contextual services is developed to illustrate different types of services used in the framework.

Originality/value – The main contribution of this paper is a high-level model of a system for context-aware regulated services, which consists of environments (domains and spaces), contextual software components, entities and computing devices.

Keywords Codes, Computer software, Modelling, Context-sensitive languages, Systems and control theory

Paper type Research paper

1. Introduction

Pervasive computing is the next generation of software systems, computing devices and environments with information, services and communication technology available everywhere in the environment for users at any time. Pervasive computing aims to create software systems that are connected, proactive, intuitive, appropriately available for users and unobtrusively embedded in the environment. Pervasive computing focuses on *mobility* in accessing information from a mobile device and so, making appropriate information available everywhere for users.

The idea is that a mobile or non-mobile user can communicate with any embedded or non-embedded software system as soon as s/he steps into a particular space or an environment. Communication here can be in the form of explicit or implicit requests



from users to the system. The implicit mode would use several sensor systems such as a location sensing system to detect a user's location, a physical sensing system to monitor a user's current activities and so on. For example, the moment a user steps into his/her office, the light switches on and when the user steps out of the office, the light turns off. This contrasts with an explicit request, where an user needs to manually turn on/off the light.

Pervasive system components are normally deployed (installed) in many server/client sides (e.g. a contextual manager component is installed on the server side and a service execution component is installed on the mobile client device). System components are also deployed in many pervasive computing environments (e.g. a pervasive shopping environment, pervasive home environment, pervasive campus environment, etc.). Although the system components are already deployed into several server/client sides and environments, there is a need to integrate them into one single platform that consists of various information, contexts, services, system configurations and business rules (policies). Ideally, these information and system components are accessible across different pervasive computing environments, different devices and different applications. As pervasive computing environments face a proliferation of users, devices and applications, integration of the information and system components become more complex.

In addition, context awareness in conjunction with pervasive software applications is emerging as intuitively suitable for being delivered as context-aware pervasive services (Schilit *et al.*, 1994, 1995; LKAA96; Dey *et al.*, 1998; CDM + 00; Gellersen *et al.*, 2002; Noble, 2000; Henriksen, 2003; Ranganathan, 2005; DSS + 06). This paper investigates the delivery of context-aware pervasive services onto a mobile device and proposes a novel context-aware regulated services model to address the specific infrastructure for pervasive computing requirements imposed on context-aware pervasive systems, by using policy (rule)-based access to govern user's behaviours in accessing and executing services.

We have implemented a prototype system for a campus environment that validates our approach and architecture for building a context-aware policy system. Our prototype system supports code on demand (run time delivery onto and execution of the code on a mobile device), Web services for contextual and policy functionalities (Syukur *et al.*, 2004a; Syukur and Loke, 2006b). We have discussed in Syukur *et al.* (2004a) how our mobile hanging services (MHS) system supports contextual services and functionalities in pervasive computing environments. We also have discussed in Syukur *et al.* (2004b, c) and Syukur and Loke (2006a), our initial policy work, where we illustrate the usefulness of having policy via scenarios to govern the execution of mobile services. This paper greatly extends our previous work, emphasizing the modelling of MHS from services, policies, to pervasive computing environments. The paper also discusses in detail the MHS context-aware policy system's design, implementation and evaluation.

This paper focuses on the development of an indoor context-aware regulated system that controls users' behaviours in accessing context-aware services in pervasive computing environments. The issues of design, infrastructure and interaction between software components that supports the discovery, delivery and execution of context-aware regulated services are considered in great detail. This paper introduces the MHS framework. The MHS framework supports awareness of both services and policies, in which the applicable services and policies would completely depend on surrounding contexts in the environment. A service in our system provides access to resources

(e.g. devices in the environment) and data/information. The system controls access to services running on shared devices (e.g. an embedded device, a workstation, etc.) and personal devices (e.g. a user's mobile device, a laptop machine, etc.).

The research contributions of this paper are twofold:

- (1) *The MHS view of pervasive computing.* The paper proposes a novel model of context-aware regulated services in pervasive computing including environments, software system components (i.e. context-aware service and policy components), entities and computing devices as shown in Figure 1. We model a system in a high-level manner, in terms of abstract system elements and parameters which are not tied to any specific environment. We view a pervasive computing environment as a collection of domains (logical boundaries) and each of these domains consists of several physical spaces. The model provides comprehensive design for context-aware regulated services.
- (2) *MHS framework architecture, design and implementation.* The second contribution is the development of a software framework with contextual software components that supports design and deployment of context-aware regulated services. This framework enables a novel contextual services infrastructure that allows services to be described at a high level of abstraction and to be regulated by contextual policies. This contextual policy governs the visibility and execution of contextual services in the environment. In addition, a range of contextual services are developed to illustrate different types of services used in the MHS framework.

Our conception of the MHS system employs a location sensing system (e.g. using the Ekahau Positioning system, 2000), and a Web service conceptual design that allows method invocation in disparate platforms and languages. This is a way of thinking about context-aware applications, driven by end-to-end high interoperability design principles, which can move between hosts during their execution lifetime and support sharing of applications between laptop computers and handheld devices.

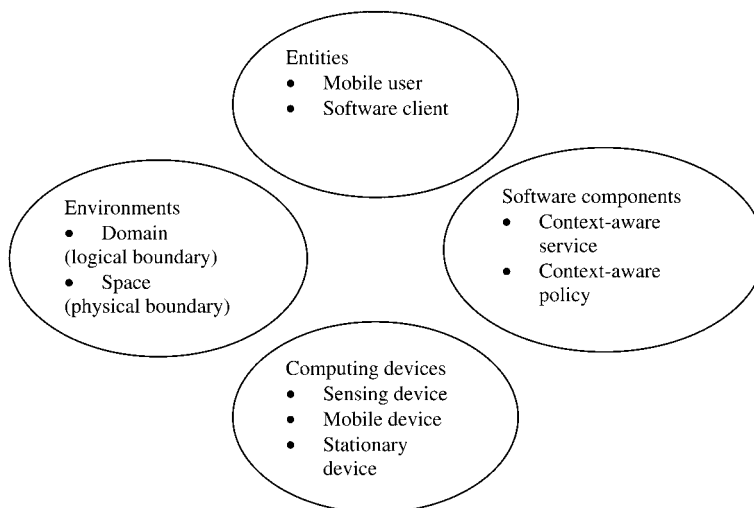


Figure 1.
The MHS view of pervasive computing

The rest of this paper is organized as follows. Section 2 provides the background necessary for context-aware regulated services. It reviews related research from the context of the work related to this paper. Section 3 presents a high-level abstraction of MHS system modelling including issues in developing localized contextual services in pervasive computing environments. The novel context-aware regulated services model proposed and developed in our research is then presented. We then provide our conceptual model and introduce our approach to building an advanced contextual system.

Section 4 discusses techniques used to implement such a system as discussed in Section 3. It presents the implementation of a prototype MHS framework with a logical view of its system components. The framework addresses the message passing between contextual software components in order to discover, deliver and execute context-aware regulated services at specific contexts. A policy specification language is then presented.

Section 5 presents the experimental results of our MHS framework. We evaluate the efficiency of the prototype system by measuring the time that it takes to execute the service, detect/update the location context change and check for policy authorization decision.

Finally, we conclude in Section 6, where lessons learnt are presented, key research contributions are highlighted and future directions of this research are discussed.

2. Related work

Many existing policy projects focus on implementing access control in the field of distributed Internet systems (Godik and Moses, 2002; Uszok *et al.*, 2003; Sandhu, 1998; Yialelis *et al.*, 1996; Damianou *et al.*, 2001; Mahon *et al.*, 2000; Moffett and Sloman, 1993; Edwards, 1996; Hengartner, 2005; Keeney and Cahill, 2003). A policy defines a set of activities that authorized users can perform (permission), cannot perform (prohibition) and must perform (obligation) within an organization depending on the position or level that the user has. For example, a boss can read and write a meeting document; however, staff can only read the document without being able to modify it.

In role-based access control (RBAC), users are assigned roles and roles determine permission that users can do. Traditional RBAC does not take into account context information. Zhang and Parashar [ZhP05] presented a dynamic context-aware access control mechanism by extending the traditional RBAC model (Gavrila *et al.*, 2001). They presented a model that dynamically adjusts role and permission assignments based on current context information. This is done by assigning each authorized user in the system a role subset from the entire role set. In the same way, services and resources in the system are assigned permission subset for each role (and its role subset). To select relevant access control based on the context, the system navigates the role subset (the subject) and then, iterates the permission subset (the object) for that particular role.

The maintenance task for this model can be quite tedious, especially if there are a number of services in the system. This is because developers need to assign certain permission (based on the role) on each of the service. In some occasions, in pervasive systems, users with different roles may have the same permission. For example, a student (with a general user role) and a lecturer (with a super user role) have the same permission to access sample examination questions during lecture time at lecture room B.123. In this case, assigning permission on each service (resource) can be redundant and may not be necessary.

There are some notable existing policy work in pervasive computing environments, however, they focus on different types of contexts such as context of an agent (SBM03; Masuoka *et al.*, 2005; Montanari and Tonti, 2002; Corradi *et al.*, 2000), networking (Lymberopoulos *et al.*, 2004; Zhuang *et al.*, 2003; Wies, 1994; Chadha *et al.*, 2003), authorization (Covington *et al.*, 2001; ZhP05; Gavrilu *et al.*, 2001; Giuri and Iglu, 1997; Moyer *et al.*, 2000; Feinstein *et al.*, 1996) and security (Kagal *et al.*, 2001; Covington *et al.*, 2002; Al-Muhtadi *et al.*, 2003; Noble and Corner, 2002; Tripathi *et al.*, 2004; Massachusetts Institute of Technology, 2004; Campbell *et al.*, 2002; Becker and Sewell, 2004). There is a little policy work done in pervasive systems, especially in the field of context-aware pervasive services. Nine closely related policy projects that deal with users' contexts are discussed as follows.

The *Automatic device project* (Connelly and Khalil, 2004) discusses a preliminary architecture for automated device configurations in smart computing environments. It takes into account context of users (e.g. location, identity and preferences). Users need to register their set of preferences in the system (e.g. what actions or activities that they wish to perform in particular context). Based on this information, a system then proactively performs actions or tasks as specified in the preference list. For example, by sensing a user's current location (e.g. walking towards a cinema), the system automatically switches the user's mobile phone to a silent mode. When the user walks out of the cinema, the system switches back the phone to a normal mode (ring tone mode).

The *Satchel* system (Flynn *et al.*, 2000) provides mobile access to documents via services. The service is aware of a user's current location. For example, when a user is in front of a printer device, Satchel presents a printer service. The service is accessible to users via Web browsers (i.e. using HTML) for desktop PC users and mobile Web browsers (i.e. using handheld device markup language – HDML (Unwired Planet, 1997)) for mobile phone users. Satchel employs policy mechanisms to securely access to documents (information). This is done via tokens issued by a Satchel browser and these tokens are signed using public key cryptography. In this case, Satchel uses two keys, one public (i.e. available to any user who wishes to access the service) and one private (that resides in the user's browser).

The current prototype system only focuses on the location context. It has not taken into account other contexts (e.g. day, time and activity running in the space). The access control is based on a simple database. The database stores a list of users with certain permission (e.g. read, write and execute) to work with documents. The permission given is relatively static and only depends on the user's identity and location. In this case, the permission remains the same regardless of days, times and activities which are running in the space. The policy document only contains permission information, it does not contain obligation of what users or the system needs to do in certain contexts.

The *policy-based agent model* (Harroud *et al.*, 2003) proposed a system that supports personalized multimedia services to mobile users and provides them with a home like environment at any visiting place. Each registered user in the system has an agent that stores, manages, maintains and enforces the user's specifications of what, how and when to execute the service. A policy contains a set of actions that needs to be performed by a subject agent on the specified target services, providing the conditions have been satisfied. The user only needs to specify one policy document which is a home policy. This policy will be applied globally to any place that a user visits based on interagent negotiation.

The purpose of security control in the *Active Space* project (Sam *et al.*, 2002; Sampamane, 2005) is to ensure that entities (e.g. users) only use and access resources (e.g. devices) in authorized ways. The space is shared by different groups of users for different purposes (activities). Each virtual space is configured for a particular activity. These virtual spaces may have different policy requirements. Active Space aims to allow seamless transitions between virtual spaces and applying the relevant policies.

Ether (Argyroudis and Mahony, 2004) aims to secure communications in the Smart Home by having a policy embedded in each device. This project focuses on access control to devices and levels of authorization for each user. The access control is based on public and private keys, in which all legitimate users own a public key of a device. It is not based on RBAC and therefore, there is no notion of hierarchical roles in deciding what permission to give to certain users. There is a mapping between a public key (e.g. an owner of a public key and a list of users who know the public key), security context (e.g. only valid within certain time duration) and permission to devices which are allowed (e.g. access to a printer machine).

Cerberus (Al-Muhtadi *et al.*, 2003) is a ubiquitous security policy mechanism that integrates context awareness with automated reasoning for preventing unauthorized access to resources in ubiquitous computing environments. It aims to develop context-aware security policies unobtrusively in ubiquitous environments. Cerberus consists of three major components:

- (1) service access (authentication and authorization),
- (2) security policy documents; and
- (3) a reasoning engine that performs automated reasoning and enforces security policies.

The *Spatial policy framework* (SBM03) aims to control execution of a mobile agent in pervasive computing environments. Depending on the location that a user enters, a mobile agent has different states of executions. The state of an agent is specified by a user's policy. In this case, each user specifies agents that need to be started or terminated at certain time and location contexts, as well as actions that need to be performed if there is an external agent that comes across the place (e.g. an action can be to continue the execution or kill the external agent). The framework also defines a concept of role. The role refers to a position of a user in the system (e.g. a boss, manager or staff).

The *Rei Policy Language* was developed by some researchers at HP Lab (Kagal, Rei, 2002; Kagal, 2004). The aim of the Rei policy language is to provide flexible constructs based on deontic concepts (Mally, 1926) that are reusable across domains (such as networking and access control security). Task computing (Masuoka *et al.*, 2005) uses Rei (Kagal, Rei, 2002) as a policy engine for enforcing the authorization decision (e.g. whether to permit or prohibit a user from the requested action to use a printer at Fujitsu Lab).

Trust Context Spaces (Robinson and Beigl, 2003) focuses on security in pervasive computing environments. It aims to provide and maintain trust based context spaces in an interactive environment. The proposed solution is monitoring user and environment contexts, and acting implicitly based on the context changes (explicit parameters of physical human-human interaction). The resources (information or data) are only available to authorized users in the right context. For example, consider a meeting that

consists of a group of people and various tasks: three audiences, one presenter and one narrator.

Many existing projects focus only on one or a few elements of a pervasive computing system as discussed in the previous sections. Some existing projects only focus on the location context, contextual service and policy information. Only a few projects focus on a combination of the following context-aware system elements: a user context, contextual service and contextual policy. Two notable projects are GAIA (Roman *et al.*, 2002) with Active Space (Sam *et al.*, 2002) and Cerberus (Al-Muhtadi *et al.*, 2003) with Rei (Kagal. Rei, 2002).

Having introduced the concept, design and operation of a system for context-aware regulated services, we present a model that illustrates the operations of such a system in the next section. This facilitates a discussion on the high-level modelling of context-aware pervasive systems, interactions between contextual software components and an infrastructure that supports the aim of pervasive services.

3. Conceptual design of MHS

Our pervasive computing system consists of entities (e.g. mobile users and software clients), logical boundaries (domains), physical boundaries (spaces), contextual services, policies and computing devices. A pervasive computing environment is dynamic; a new domain, space or subspace can be added to or removed from the system. A mobile user is an end-user who is always on the move. In addition, sensing devices and software systems can be embedded everywhere in the environment. The software systems can be a server component that resides on a server side and a client component that is installed on a user's mobile device or embedded in objects (e.g. wall, fridge and so on) in the environment. The client component responds to users' requests by interacting or communicating with server components. The idea is to provide autonomous and proactive systems that could suggest and deliver useful services as soon as users step into a particular space in the environment.

The discussion in section 2 concluded with the motivation to include policy (rule) for controlling users' behaviours in accessing services in pervasive computing environments. This is mainly due to a pervasive computing environment consisting of many users who are always on the move and can access services at any time and any location. In some situations, the system wants to limit users' behaviours by specifying policies. This avenue imposes new requirements and constraints that challenge the development of context-aware regulated services.

From our investigation of context-aware architectures and systems in section 2, it was found that context-aware regulated systems that control visibility of services, as well as users' behaviours in accessing services are useful for operation in dynamic and heterogeneous environments, where services are aware of users' contexts. This is especially important as more than one mobile user can access and control services running on public devices. The public device is a shared device that is accessible to all authorized users in the system. This then results in conflicts, if there is no proper management in terms of who, what and when services can be accessed.

In this paper, we present the MHS model that addresses the requirements of systems for context-aware regulated services in pervasive computing environments. Our MHS explores user-centricity, context-sensitive information, service-awareness, policy-awareness and a pervasive computing environment model, in order to provide context-aware regulated services for users with minimal or no effort for service set-up prior to use. It is a context-aware regulated services framework that assists developers with

development of context-aware services along with policies for controlling visibility of services, as well as access to services in certain contexts.

3.1 The MHS view of pervasive computing environments

In MHS, our view of an environment consists of logical boundaries (domains) and physical boundaries (spaces). A domain represents a logical boundary of an environment that is not perceptible. A *domain* is considered as a top level hierarchy of an environment followed by *spaces* and *subspaces*. Our view of a pervasive computing environment is not only limited to one domain (see Figure 2). Instead, users can move freely between spaces, which have been grouped into a domain. For example, an office building belongs to a campus domain and an entertainment room belongs to a home domain.

Samples of domains are a campus domain, shopping domain, home domain, etc. (as shown in Figure 2). These domains offer unique services that differentiate them from each other. For example, services within a campus domain are related to education purposes that target students and staff; a shopping domain is related to shopping (buying/selling) items that targets buyers and sellers. A domain can contain zero or more (sub) domains. For example, different types of parks such as Rose Garden, Natural Park are considered as subdomains of a park domain. A domain also consists of one or many physical spaces (see Figure 3).

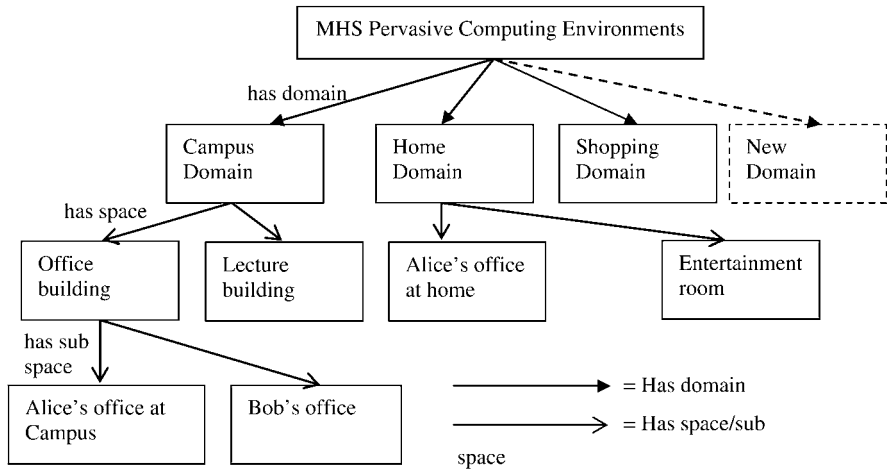


Figure 2. MHS pervasive computing environments

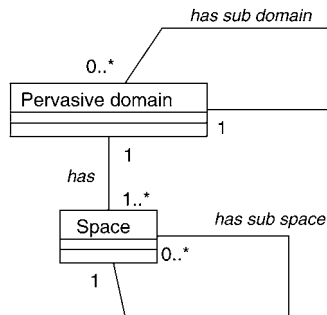


Figure 3. A pervasive computing domain and its spaces

A space is a physical location (boundary) that is conceivable (e.g. a building, a lecture room, a railway station, etc.). A space contains zero or many subspaces (e.g. a building space consists of several sub spaces: floors, rooms, corridors, etc.). A user can move between spaces in the same domain or move to different domains. For example, at time t_1 , a user can be at a lecture room (within a campus domain) and at time t_2 , the user moves to a grocery shop within a shopping domain. A space needs to be associated with at most one domain. Each space has its own policy (also known as a space policy). The scope of this policy is valid within the space.

With our conception of an environment, we can draw logical interconnections between similar physical spaces in the system. For example, for a house domain, there is a logical interconnection between different types of houses (e.g. a villa house subdomain, residential house subdomain, etc.). The idea is that mobile users do not need to manually specify their locations to the system when accessing services, instead, the system needs to proactively detect the users' current locations and find the mapping of relevant services for users in that particular context.

3.2 *The design of MHS context-aware services*

The notion of *service* suggests software systems for providing assistance to users to complete their daily tasks, such as automatically configure a printer according to a user's profiles (e.g. print document in single sided using a colour printer). Such manual tasks can be handled implicitly by the system via services. In this case, a user who is a subject of a system, can say "I want to print my paper in colour (single sided) by 10 a.m. on Friday". Upon receiving this command, the service then places the paper document in a queue, configures the printer according to the user's preferences and prints it out before Friday, 10 a.m. It then notifies the user when the printing is done.

Delivering only the right service to users is considered necessary, as they are always on the move and often limited in time to explore service functionalities. Ideally, a user should see the service interface as soon as it is downloaded on the mobile device and does not need to know the detail on how to compile and execute the service. This all needs to be handled implicitly and proactively by a system. In our work, a service to developers refers to a mobile code application that contains a service interface and data associated with it.

The code application is mobile and compact in size. Each code bundle handles one particular task/service; more services mean more code existing in the system. Our system stores code on a server and manages the downloading and uploading of code from the server to a client and vice versa. To a user, a service is a tool that is enlisted as the user needs it, which takes care of how tasks get done. A service can just display information to users or perform tasks. For example, by scanning a cup tag ID, not only information about the cup will be displayed, but also, a service interface appears allowing the user to perform an action (e.g. buying a cup, emailing the owner of the cup, etc.).

The service in our system is context sensitive, mobile and regulated by policies (rules). Having context-aware regulated services is considered important, as in some situations a system may want to be in control by asking all users to obey policies which have been pre-specified. The policy controls the visibility of services, and actions permitted on the service, depending on the user's role and current contexts. For example, during exam time, all users who sit for closed book exams are unable to access services. The users could access services as per normal before and after exam time. This example shows how services and policies can be aware of (or vary depending on) contexts.

Our system supports the creation of services that targets mobile devices (e.g. pocket PC, and laptop devices) and desktop devices. Ideally, users do not need to know where

s/he is (or under what domain s/he currently is), relevant services should be seen across domains and spaces. In addition, a user should be able to see relevant information related to a space, regardless of a domain that s/he is currently in. For example, a user who owns offices in a few domains (e.g. an office at home domain, office at work domain and office at campus domain) could see the same information related to the office in different domains. This is possible as we store service information per user. This information can then be retrieved by the user in any domain and within a related space.

In general, there are two approaches for implementing context-aware services:

- (1) *Without the need to download the service onto users' devices.* For example, as soon as the system senses a user enters a room, the light is automatically turned on and when the user steps out of the room, the light is turned off.
- (2) *By downloading the service onto users' devices.* This then allows users to control service behaviours (e.g. start, stop the service) from their devices. For example, when a user steps into a room, the light is automatically turned on, but also, the user can control this service, by manually turning the light off from the device.

The MHS system focuses on the development of context-aware services as in approach "no. (2)". The service provides a way of controlling the running services from mobile devices (e.g. a user would be able to control the execution of a music service that is running on a desktop machine). It has a user interface which a user could interact with.

We recognize five aspects that would affect the visibility of services. These can be based on the scope of services, a user's role, a history log file, a user's preferences and space policies. Each of the aspects that affect visibility of services is described as follows:


- (1) *The scope of services.* A service is visible only within a certain scope (boundary). As illustrated in Figure 4, depending on the boundaries, different services would be visible. For example, in a car exhibition, a service can be delivered depending on the types of cars (i.e. specific to the car) or a generic service for the exhibition (e.g. an exhibition map service). A specific service is termed a local service and a generic service is termed a global service.
- (2) *A user's role.* Users with different roles would see different services in the same contexts (e.g. location, day and time). A user with a higher role would generally see more services compared to users with a lower role. For example, a lecturer in the campus could see more services (e.g. an exam service, a lecture note service, a student mark service and a student attendance service) compared to students.
- (3) *Information that is stored in the history log file.* For example, at time t , a system takes information about a user's current context and history such as a user's previous whereabouts and services associated with those places. This history information is useful for computing a set of services that would be useful to the user at time t . This computation can be modelled as a function f , where

$$f(\text{current_context}, \text{history}) = \text{a new set of services}$$

- (4) *The user's preferences.* A user specifies a list of services that s/he might need in a specific situation or context. As a result, two users may see different sets of services, although they are in the same context. This is due to different sets of

| | | |
|-----------|----------------------|-------------------|
| Corridor | Mercedes Car Display | Volvo Car Display |
| Reception | BMW Car Display | Corridor |

Legend:

 = This boundary receives specific information (i.e., a local service)


 = This boundary receives generic information (i.e., a global service)

Figure 4.
The visibility of services depends on the boundary

preferences that they have specified. System management for this technique is quite complex, as the system needs to maintain all users' preferences and display services differently for each user depending on his/her preferences.

- (5) *Space policies (rules)*. Developers and owners of physical spaces could set policies (rules) that specify list of services which are available (visible) in the space in certain contexts. As a result, two users in different contexts (e.g. location, day and time) would see different services.

Our MHS system uses a combination of a user's role and space policies in determining visibility of services in the space. We discuss in detail in Section 4 how context-aware policies control the visibility of services in pervasive computing environments.

3.3 The design of MHS context-aware policies

Our policy design takes into account an interoperability aspect, in which policy software components need to be easily accessed within a system itself and by external systems that have disparate platforms and languages. In our policy model, the policy enforcement depends on a user's current contexts. For example, when a user enters a campus domain such as the entrance of the campus, the campus policy is applied. When the user enters Blackwood building which is under a campus domain, the building policy is applied and when the user enters a lecture hall in the Blackwood building, then the lecture hall policy is applied. Note that a lecture hall might also have different policies at different times and occasions.

However, if the space does not have a policy, the system then applies the policy which belongs to the closest logical or physical boundary to this space. For example, as shown in Figure 2 above, if a space named "Bob's office" does not have a policy, the office building policy is then applied in the space. In short, the system first looks for a relevant policy in the space of where the user is currently in. However, if the space does not have a policy, the system then takes the policy that belongs to another space/sub domain/domain, which has a higher level of the environment structure compared to the current space.

Our policy design takes into account the reusability aspect. This is possible by storing policy documents and having policy software components on the server side. The relevant policy for the space is applied to all users in the space. Our system also separates rules from contexts. We do not explicitly specify context information in the policy document (e.g. a space or location as well as the exact date and time of when and

where activity occurs). Instead, we store the context and activity mapping separately in an external XML file. The mapping here works like a scheduling system, where it stores a list of spaces' activities in particular contexts.

In addition, our policy system controls behaviours of services which are running in the space. Our policy design separates the mobile code application (service application) functionalities and interfaces from the policy. Another approach is to embed the rule itself, as well as policy mechanisms into the application in a tightly coupled way. This complicates both the application's design and run time adaptation to change context or execution environment. We have separately encoded service application's functionalities, and context-aware service components from context-aware policy components.

MHS differs from all other approaches in its ability to specify the policy strategies (mapping between policy, context information and services) at a high level of abstraction. It also supports policy conflict resolution during application execution at run time (Syukur and Loke, 2007, 2006b). When 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 a rule per activity, in which only in certain occasions, the space will be in control. Different policies are applied depending on the activities running in the space. Ideally, it should allow seamless transition between different activities which are happening in the space. The right policies should be easy to configure and enforce with or without a delay.

Ideally, the end-user would still be able to access services as per normal in the spaces 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). For example, during exam time, the space (the exam room) makes an "online browsing service" visible only to hall supervisors and this service is invisible to all students who are sitting for a closed book exam. In short, the idea of having policy in pervasive computing environments is like a user who has a car, where she can drive anywhere that she likes. However, she needs to follow the road and traffic rules. Having the rules here do not stop the user from driving.

In general, implementing a policy for controlling access to services would benefit a certain entity (i.e. the one who is in control). However, by taking into account convenience versus controlling behaviours in designing policies, the system would allow an entity to give as much permission as possible to other entities in most of the contexts and only in some situations when it is necessary, developers or owners of spaces give partial control to users to access services in specific contexts.

In designing a policy, there are several questions that need to be addressed, such as:

- (1) Who are we trying to control (e.g. the system, end-user)?
- (2) What are we protecting (e.g. resources, network information, services)?
- (3) What are the different policy objects (e.g. permission, obligation, prohibition) used in the policy system?
- (4) What is the target computing environment (e.g. networking, distributed, pervasive computing environment, etc.)?

3.4 MHS context-aware policy concepts

Figure 5 illustrates an overall picture of policy concepts in our system and how these concepts are related. Four main policy concepts are described below:

- (1) *Policy objects.* Policy objects are based on the logic of norms for representing the concepts of rights, obligations and prohibitions in our system. Right (R) refers to permission (positive authorization) that is given to an entity to execute a specified action on the service in a particular context. Obligation (O) is a duty that an entity must perform in a given context. Prohibition (P) is a negative authorization that does not allow an entity to perform the action as requested in the given context.
- (2) *Policy actions.* Currently, there are five different actions 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. The five actions we identified, we found, occur frequently among a number of MHS services. For example, a Mobile Pocket Pad service that allows users to retrieve and submit their information online, would require two types of actions (Start or Retrieve, and Submit). A Media Player service that allows users to listen to any music at any nearby desktop machines supports four different actions (Start or Play, Pause, Stop and Resume) the music. A Mobile VNC service that allows users to teleport their desktop information to any nearby desktop machine in a location would require Start and Stop actions.

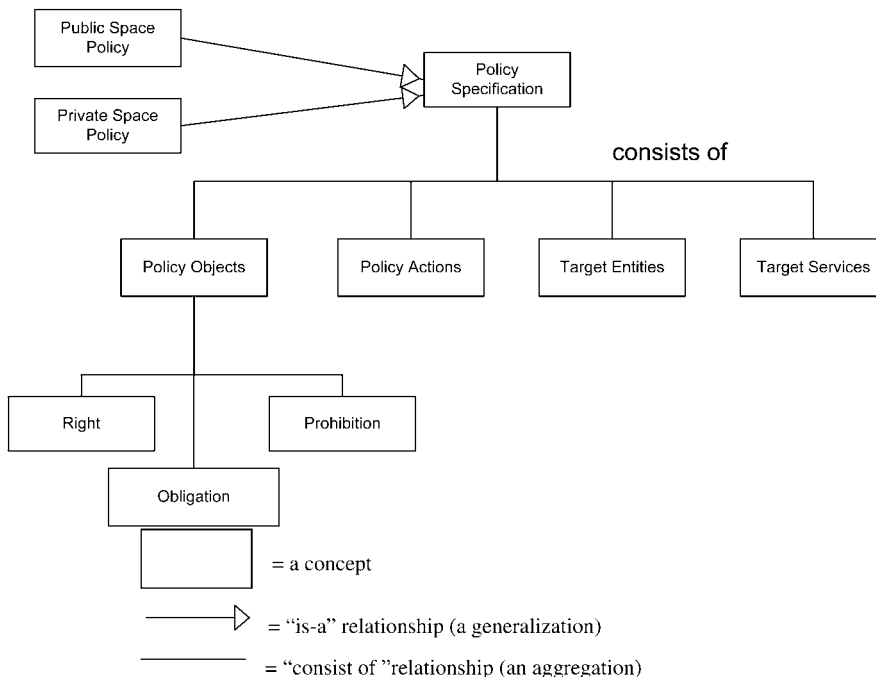


Figure 5.
MHS context-aware policy concepts

Our system stores a semantic definition of an action, as one action could have different names, but, the purpose is the same (see Figure 6). Hence, we group the action based on its purpose. For example, in a Media Player service, a start button is called “play”, but in the Mobile Pocket Pad service, a start button is called “retrieve”. In this case, they both have the same purpose (i.e. to start a service), though the action name is different.

In addition, our system could support many types of actions. An addition of a new action or a modification of an existing action only needs to be updated in one place, which is in the synonym action document (as shown in Figure 6). Therefore, it is not an issue if the number of actions employed increases in the future. Figure 6 describes the synonym of actions in our MHS system. For example, an action with the name “start” has the same meaning with “play”, “turn on”, “retrieve”, “calculate”, “print” and “search”. As for actions that do not have synonyms (e.g. pause, resume and submit actions), they do not have a synonym tag as part of their child elements. The action name “any” means all actions are allowed (i.e. start, stop, pause, resume and submit). The action name “None” means none of the actions is allowed, and hence, it does not have a synonym tag.

- (3) *Target services.* Target service refers to a particular service that a user is allowed to access or obligated to access or prohibited from accessing. Our MHS

```
<ActionSynonymDescription>
  <Action Name="Start">
    <Synonym>Play</Synonym>
    <Synonym>TurnOn</Synonym>
    <Synonym>Retrieve</Synonym>
    <Synonym>Calculate</Synonym>
    <Synonym>Print</Synonym>
    <Synonym>Search</Synonym>
  </Action>
  <Action Name="Stop">
    <Synonym>TurnOff</Synonym>
    <Synonym>Close</Synonym>
    <Synonym>Upload</Synonym>
  </Action>
  <Action Name="Pause">
  </Action>
  <Action Name="Resume">
  </Action>
  <Action Name="Submit">
  </Action>
  <Action Name="Any">
    <Synonym>Start</Synonym>
    <Synonym>Stop</Synonym>
    <Synonym>Pause</Synonym>
    <Synonym>Resume</Synonym>
    <Synonym>Submit</Synonym>
  </Action>
  <Action Name="None">
  </Action>
</ActionSynonymDescription>
```

Figure 6.
Synonyms of actions in
the MHS system

system deals with two types of services: shared resource services and non-shared resource services.

MHS: pervasive services

- (4) *Target entities.* A user is an active entity who is always on the move (able to move from one geographical place to another). Our system recognizes two types of entities: (a) an embedded or non-embedded object (a client software system) that is installed in the environment and (b) a mobile user who is always on the move. A mobile user interacts with the system (e.g. requests to deliver music service) via a software client that is installed on the user's mobile device or embedded in the wall in the environment. A software client monitors a user's current contexts by requesting contextual information from context sensing devices and passing on the result to a context interpreter. The software client implicitly performs tasks on behalf of users, such as contacting software system elements to retrieve users' information and automatically executing tasks based on the users' preferences.

61

Each user in the system is assigned a specific role. The role is associated with privileges that determine actions that users can perform and visibility of services in particular contexts. There are three different types of roles in our system: super entity, power entity and general entity. Depending on the role that an entity has, s/he may have different privileges in accessing the service. For example, a user with higher role can do more things and may have more services available in the context compared to the user with lower role. Figure 7 illustrates the role and entity mapping in our system. As shown in Figure 7, an "instance" element (that is a child of "role" element) refers to a sample role in a particular domain. For example, in a campus domain, instances of a general entity role are undergraduate student and postgraduate student.

4. Implementation of MHS: detailed design and prototype

The prototype implementation of our MHS system as published in Syukur *et al.* (2004a), and Syukur and Loke (2006b, 2007) supports the needs of context-aware regulated services by:

- *Using a combination of mobile code and a client-server model.* This is to perform contextual functionalities that result in overall minimum user wait time. The MHS system supports code on demand execution, which allows a computational

```
<RoleCollections>
  <Domain name="CampusDomain">
    <Role name="GeneralEntity">
      <Instance>UndergraduateStudent</Instance>
      <Instance>PostgraduateStudent</Instance>
    </Role>
    <Role name="PowerEntity">
      <Instance>Lecturer</Instance>
      <Instance>AdministrativeStaff</Instance>
      <Instance>TechnicalStaff</Instance>
      <Instance>Supervisor</Instance>
    </Role>
    <Role name="SuperEntity">
      <Instance>HeadOfSchool</Instance>
      <Instance>Dean</Instance>
    </Role>
  </Domain>
</RoleCollections>
```

Figure 7.
Role collections document
in a campus domain

component to load code from a remote service repository when needed. Basically, the MHS system retrieves service applications from a service repository on the server side, when there is a request from a user. When the mobile client application finishes downloading and compiling the service application, it then displays the service interface.

- *Identifying a high-level abstraction of a system.* This is done by deriving concrete system components from a pervasive computing environment model. The MHS system logically groups spaces into domains. The system functionalities are stored on the server and client sides.
- *Supporting remote execution.* The MHS system enables remote execution of a process on one machine from another (e.g. to remotely start a teleporting service that is installed on a desktop machine and controlling this process from a mobile device).
- *Supporting a policy mechanism.* It is for controlling access to services in particular contexts.

This paper discusses in detail basic components of the MHS system and how the parts interact. The system consists of users with a laptop or handheld device, a Web service that determines the location of a user, a set of contextual-based services and policy components which are published via the system (see Figure 8). A handheld device user

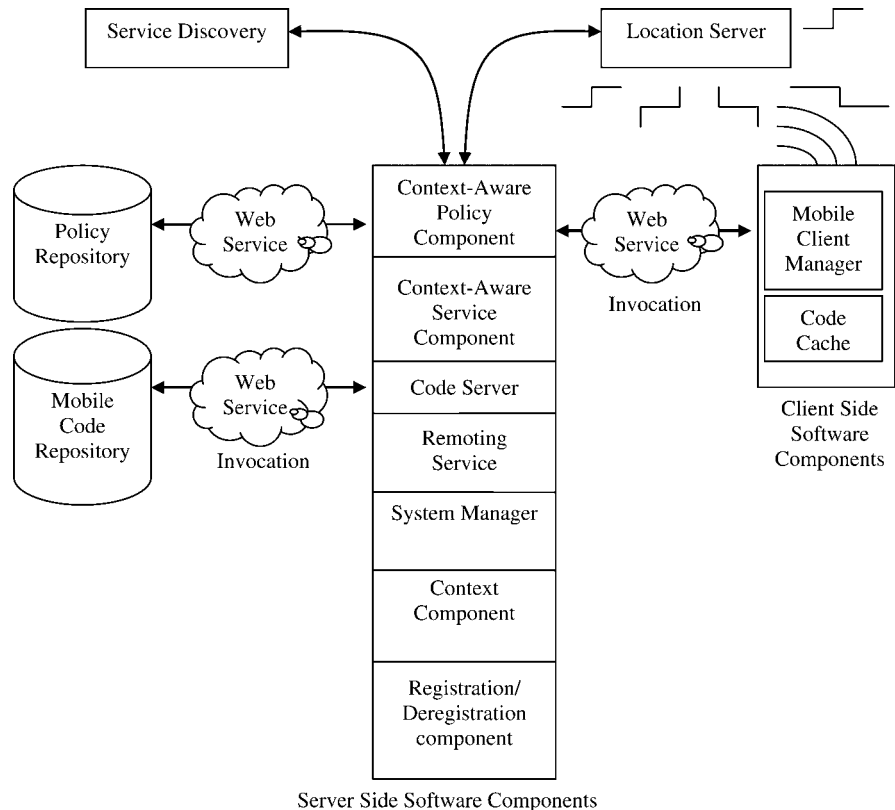


Figure 8.
A high level architecture
of the MHS system

accesses the system via a mobile client application that has been installed on a device and laptop users access the system via a Web browser.

The current implementation of MHS uses the Microsoft .NET framework (Thai and Lam, 2001) that natively supports Web services and remote method invocations. The .NET framework targets a number of applications such as a smart device application (e.g. for a pocket PC), a Web application (e.g. for a desktop, or laptop, in which the application is accessible via a Web browser), a mobile Web application (e.g. for a smart phone where the application is viewable via a micro browser) and a stand alone application (e.g. for a desktop device). The .NET framework provides stable development for both a traditional system (e.g., a standalone application) and a distributed system that involves various system components such as a server, client, Internet service and database objects, as well as high interactions between them.

4.1 Architecture of the MHS framework

This section discusses a high-level architecture and abstraction of our MHS system. Our system provides an infrastructure that handles interaction between a client device that requests a service and a server which responds to the client's request via Web service calls. The MHS system simplifies the task of developing and maintaining context-aware applications for mobile clients. The system handles and caches all incoming services or code applications on the client side, as well as executing services when downloaded. This is possible as the MHS software components reside both on the client and server sides and each of them handles and manages a particular task (see Figure 8). Design properties in our contextual system include modularity, extensibility, genericity and interoperability.

The system is *modular* as it has a loose coupling architecture. Updating the system functionalities in one component will have minimal impact on the rest of the system components. Moreover, addition or modification of system functionality only needs to be done in one place (e.g. to add a new function to the policy manager, only the policy manager needs to be updated).

Our system is *extensible* as we can still add additional services in the future. This is possible as we store all services on the server side (not on the client side). The service is only downloaded onto the client side when the user is in a particular context where such service is considered useful or when the user requests it.

Our system is *generic* as it can employ existing or new traditional and non-traditional applications. The proposed system is also highly *interoperable* as we implement all the contextual methods or functionalities of the system as Web services. Implementing system functionalities as Web services allows Web methods to be accessed and invoked easily across disparate platforms and languages via the HTTP protocol.

Figure 8 illustrates a high-level architecture of the MHS system that consists of server software components and client software components.

Our system simplifies interaction between a user and system starting from sensing a user's current context, to downloading and executing a service. Ideally, users do not need to know low-level details of matching, selecting, downloading and executing services. All these interactions need to be handled implicitly by a system. The framework exploits services present in the environment, provides functionality to manage, proactively download services from a server side to a client side and upload the modified information to the server.

MHS also supports the ability to control services by having a policy that specifies a set of rules of what services are visible to users and what actions are allowed, prohibited and obligated in particular contexts. Our policy design balances the convenience and compliance aspects of users in the system. As a result, mobile users still have the convenience in accessing services, although s/he has been bound to a set of rules.

We describe 11 main software components in our system as follows.

4.1.1 Server side software components. Server side software components are stored on the server. These components continue to monitor a user's current contexts and obtain relevant services and rules. Each of these software components is discussed as follows.

4.1.1.1 System manager. A system manager manages the interaction between a mobile client manager and server side software components. As soon as the system detects a user's location, the system manager proactively computes a set of services and actions allowed on a service in the particular context. This is done by first contacting the context collector to gather the user's current context and then, the policy interpreter is called to parse the relevant policy document and interpret its contents (i.e. permission, obligation and prohibition given).

The system manager manages the interaction between a client and the server. It is responsible for retrieving the available set of services in the specific context as well as responding to the user's request regarding the space policy decision results (e.g. what actions that a user can perform on a target service in the specific context). Figure 9 illustrates how relevant services are retrieved given a set of contexts.

4.1.1.2 Code server. Code server refers to a mechanism that handles a user's request, responding to that request by transferring the relevant mobile code. In this case, the code server retrieves the mobile code that matches with the user's request and

```
//Web method to get a list of services in the space
[WebMethod]
Public ArrayList GetService (string deviceIP, string userRole)
{
    //Initialisation of ArrayList
    ArrayList ListOfServices = new ArrayList();

    //Add a Web reference to the Location Web service and creates an
    //instance of it
    LocationService locationService = new LocationService();

    //Add a Web reference to the Policy Web service and creates an
    //instance of it
    PolicyService policyService = new PolicyService();

    //Call the Location Web service to get the updated user's location //and
    store the result in a userLocation variable
    String userLocation = locationService.getAreaLocation(deviceIP);

    //Call the policy Web service to get a list of available services //in
    the space. The input parameters are a user's current location, //a user's
    role, day, and time contexts. If there is a service //found, then assigns
    the result to ListOfServices variable
    ListOfServices = policyService.getSpaceAvailableServices
        (userLocation, userRole, Day, Time);

    //Finally, return a list of services.
    return ListOfServices;
}
```

Figure 9.
Get service web method

sends it back to the mobile client. Within our system, we employ a Web service method invocation such as Get Mobile Code Web method, in order to retrieve a mobile code application that matches with the service name, returning the particular mobile application to the user's device. An example of Get Mobile Code Web method on the Code server class is illustrated in Figure 10.

4.1.1.3 Remoting service. Our system permits a remote Web service call from a client device to the server or the server calls a Web service on the client. Using remoting permits remote execution of a process of an application on a specified target machine via a Web service call. Our remoting Web service is developed in the .NET environment using VB.NET and C# languages. This remoting service can be easily invoked by other processes which are running on different platforms.

4.1.1.4 Context component. Contexts are conditions that must be met before a list of services is displayed on a mobile device or before a user's request to perform an action is approved. In our work, contexts consist of a user's identity, location, day and time. Activity information (e.g. activity name = "Meeting") is derived from a mapping between location, day and time contexts. For example, by knowing a user's current location, day and time, the system can retrieve the activity that is happening in the space. All these information are based on manual inputs by developers and owners of the spaces.

4.1.1.5 Context-aware policy component. Most existing policy languages have constructs which only include a few pervasive computing principals (i.e. does not use contexts, or services), which we feel makes their direct use in our system difficult. Therefore, we developed a simple policy language that includes some basic principals (i.e. it has a notion of entities, spaces, services and contexts) in pervasive computing environments. We try to keep our policy design as simple as possible so that it can be easily integrated into an 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 extensibility of the framework as we can easily add more services, entities and policies in the future or perhaps integrate the existing system with some other external systems (assuming they both have similar design and architecture).

```
//Web method to get mobile code based on the requested service
[WebMethod]
Public void GetMobileCode (Service service)
{
    //Initialisation of MHSMobileCodeService Web Service.
    //The MHSMobileCodeService Web Service has a function called
    //GetApps(service name)to retrieve a mobile code from the mobile code
    //repository.
        MHSMobileCodeService mhsCode = new MHSMobileCodeService();

    //Retrieve a particular mobile code on the server and download it
    //onto the user's device
        String serviceName = service.ServiceName.Trim();
        App app = mhsCode.GetApps(serviceName);

    //Start and execute the mobile code on the device
        Start(app);
}
```

Figure 10.
Get mobile code web
method

Allowing users to specify their service preferences would certainly maximize the aim of context-aware pervasive systems, since the service delivered would more likely be useful to users. Also, users are given privileges to specify things they wish to see or do in particular contexts. However, in some situations, the system may want to be in control, by asking all users to obey the policies (rules) which have been pre-specified.

The policy in our system specifies the visibility of services, as well as controlling users' behaviours in accessing services (i.e. what action they can perform) depending on their role and current contexts. The service controls access to devices and information. Our MHS system does not embed a policy document on a client device, instead the policy is stored in a centralized database. This then supports reusability of policy information.

In general, there are two approaches used in integrating a context-aware policy into a context-aware system:

- (1) *By embedding a policy document on an executable service application.* With this approach, a client-side service application enables and disables an action according to the specified rule (e.g. only the permitted actions are enabled). Hence, no policy checking is required. However, this approach requires the system to recompile the client-side service application each time the policy is modified. This does not suit a situation where a policy is dynamic (often changes). The compilation of a service here may take some time depending on the number of services that need to be recompiled and this update has to be propagated to all mobile users.
- (2) *By having a system manager to manage all policies for services in the system.* This approach allows the policy to be reusable across services, and users. The maintenance is also simple, as any modification only needs to be done in one place. The only issue is that it takes some time to respond to the user's requests, as the policy is stored on the server side. One possible solution to mitigate this issue is by proactively computing a set of relevant policies as soon as the context is sensed, as well as, caching the policy decision results for future re-use (can be on the server or client side).

Our MHS system employs this approach in implementing context-aware policy components. We have a system manager that manages all policy software components and documents.

4.1.1.6 Context-aware service component. Location context plays an important role in deciding useful services for users. This is mainly due to each physical space having its own purpose and a user tending to be on the move from one physical space to another. Many existing context-aware systems support the delivery of context-aware services by mapping services with contexts only. Our MHS system has extended the initial context-aware services idea by including a policy (rule) that specifies visibility of services and actions permitted on the service.

Figure 11 illustrates a user's profile document. It is in an XML document. For different services, different features that a user needs to specify (e.g. to specify what tasks to automatically perform on a mobile Media Player service is different from a mobile VNC service). As for a mobile Media Player service, user A specifies music to be played (e.g. instrumental.mp3 song) on the target machine, when she is approaching room B538.Campus. This song will be played continuously, unless explicitly terminated by the user. As for a mobile VNC service, the user specifies his/her source

and target machine IP addresses. The system will look for these IP addresses, when a user is approaching room B535_Campus and requesting to start a mobile VNC service.

Our service in the form of mobile code is stored on the server side and only downloaded onto a client device when needed. This supports extensibility of a service, where we can add many services in the future. Our system computes a relevant set of services based on the policy information as specified in the space policy document. In addition, our system categorizes services into two types depending on its target execution. There is a service that is executed and run only on the mobile device (also known as *a non-shared resource service*).

There is also a service that is executed and run on the mobile device; however, it can initiate a process on other computing machines (e.g. a desktop machine) – also known as *a shared resource service*. For example, a notepad service would only run and execute on the user's mobile device, as it is only for displaying the user's online note and uploading information back to the server. On the other hand, a mobile Media Player service that allows users to start any music on any nearby target machines or any embedded devices needs to provide a way for mobile users to control the running music service from a mobile device. The user also needs to be able to initiate a music service on the target desktop machine.

4.1.1.7 Register and deregister component. This component is responsible for registering and deregistering services, policies, environments (i.e. domains and spaces), contexts, entities and computing devices to the system. The registration simply means adding new instance of the element to the system (e.g. adding new services). In contrast, deregistration means removing the existing instance from the system.

4.1.2 Client side software components. This is the second component of our MHS system. Client side software components reside on the client side. Client side software components interact with server components implicitly to help users perform their daily tasks. This section discusses each of the client software components in our system.

4.1.2.1 Mobile client manager. When a user selects a service, a mobile client manager contacts central Web services and downloads an application for interacting with selected services. The mobile client manager resides on the mobile client side (e.g. on a mobile device, a laptop or an embedded device) and manages interactions between a client and server such as:

- Between a mobile client and system manager. This can be retrieving and displaying a list of available (permitted) services on the device and setting permission on the service (i.e. what actions that a user can perform).

```
<UserProfiles id="user01">
  <Service name="MobileMediaPlayerService">
    <Location name="B538_Campus">
      <MusicToBePlayed>Instrumental.mp3</MusicToBePlayed>
      <RepeatPlayed>Yes</RepeatPlayed>
      <TargetMachine>192.168.0.8</TargetMachine>
    </Location>
  </Service>
  <Service name="MobileVNCService">
    <Location name="B535_Campus">
      <SourceMachine>192.168.0.1</SourceMachine>
      <TargetMachine>192.168.0.2</TargetMachine>
    </Location>
  </Service>
</UserProfiles>
```

Figure 11.
A user's profile document

- Between a mobile client and code server. The mobile client manages the incoming mobile code and prepares to execute the service interface on the device. Our mobile client manager also caches services or mobile code for future re-use. Each downloaded application is specialized for interacting with a specific service and can deliver a rich user interface and functionality, even if the device is temporarily disconnected from the network. This is possible as the client device software caches downloaded applications and stores them locally on the mobile device. The cache contains code and metadata describing applications. Hence, if a downloaded application is running, its cache code also exists in the client memory.

Our mobile client software application is developed using the Microsoft .NET Compact Framework technology which natively supports XML, SOAP and remote code loading [MPS02].

4.1.2.2 Code cache. Code caching refers to mobile code or service applications which are stored on the mobile device for future re-use. In our current implementation, we clear the code cache when there is an explicit request from users. To remove code from the cache implicitly, we need to consider the following function c , where

$$c(\text{current_context}, \text{history or preferences}) = \text{application_code}$$

that remains in the cache. The function c above describes that at some time t , c takes information about the user's current context and history or preferences (depending on how do we compute a set of relevant services), to determine what application code should remain in the cache (i.e. what should be deleted).

4.1.3 Mobile code. Within our framework we employ mobile code to provide a simplified user interface (an endpoint) to a contextually available XML Web service (Cooney and Roe, 2003). The highly compact mobile code application is built using the Microsoft .NET Compact Framework technology. It is compact in size and stored on the server side. The code contains data and service functionality. Based on the user's selection, different types of mobile code applications may be downloaded onto the mobile device for execution. Thus the mobile code delivers a responsive user experience and permits remote Web service calls.

4.1.4 Repository. The system repository resides on the server side. We store service applications (mobile code) in a SQL server database; mapping between context, service and policy is stored in an XML document. The temporary cached service is on the mobile device.

4.1.5 Controlling users' behaviours. Our policy language is implemented using XML. XML-based standardization is a key to interoperability in many systems and platforms. This is due to the fact that XML is a simple and flexible text format that can be easily read in and manipulated in any languages and platforms. Moreover, many existing languages (e.g. Java, VB.NET and C#) support an XPath query language (W3C, 1999) for XML-based documents. With XPath, we can query or search for a particular node or element. Once the result is returned, we then parse XML elements into a local system data representation and manipulate it accordingly. Querying for certain elements can also be done in a minimum amount of time. Our policy document contains a mapping between rules, services and contexts. It shows differences in deontic permission, obligation, prohibition of actions and service visibility depending on the user's role.

Our policy implementation is modular and interoperable. We separated policy tasks according to its functionality such as a policy manager, policy conflict detection, policy conflict resolution and policy execution. Each of these policy functionalities is implemented as Web services. We also separated the implementation of context-aware policy from context-aware services. Our system stores the context-aware policy software components and the policy documents on the server side. This then supports extensibility of the system, as we can add additional policies in the future.

In addition, in order to know the relationship between a domain and spaces (e.g. *a mobile computing room* is within *lecture hall building* and the lecture hall building belongs to the *Caulfield campus* subdomain and this subdomain belongs to a *campus domain*), the policy manager queries the environment document. This relationship information is useful for applying a relevant policy in the space (e.g. what policy document should be used for this space?). The relevant policy of where the user is currently in, is dominant over a global policy. The global policy refers to the policy which is in the higher hierarchy of an environment (a domain and spaces) structure. For example, if a user is currently in the mobile computing room, the upper environment structure is a lecture hall building, followed by a Caulfield campus subdomain, and lastly, followed by a Campus domain. In this case, if the space does not have a local policy, then the closest global policy is applied. For example, if the mobile computing room does not have a policy document, then the lecture hall building policy is applied in the mobile computing room.

We now discuss how our run time policy mechanism works.

The steps in Figure 12 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 music on a Media Player service by clicking on the “play button”.

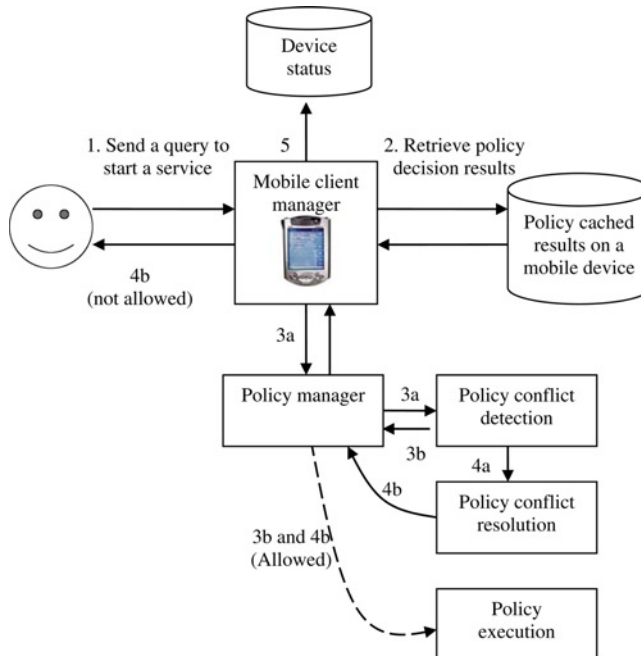


Figure 12. The MHS context-aware policy implementation

- (2) *Retrieve policy decision results.* The policy decision checking is computed at run time, just when a user requests to perform certain actions on the service. It is done by a mobile client manager by reading a local cached policy decision result that has been downloaded onto a mobile device. The policy decision checking takes into account a user's current contexts and checks the requested action against the cached policy decision result. The policy decision result is either allowing or disallowing users from performing the requested action.

The owner of the space can access and execute any services with any actions that s/he likes in his/her room. We exclude the owner of the space from the policy decision checking. The mobile client manager looks for the policy when a user requests a service (e.g. at Context *C*, user *A* is allowed to start service *S* with any action). In this case, the system will not perform any checking for subsequent action requests on the service. However, if the policy said that user *A* is only permitted certain actions on Service *S* (e.g. only allows to start, but not to stop), the system then needs to check for the policy decision result for the subsequent requests. This is possible in situations (e.g. during exam time) where the space only gives students partial control (access) to the service (i.e. a mobile pocket pad service). For example, they can post their notes, but they are not allowed to retrieve their notes. In this case, the policy checking is done per action, where for each action requested, policy checking is executed.

We employ two types of policy decision checking: per service and per action depending on the policy decision result. Employing only a single policy decision checking (e.g. checking per action) may not be necessary due to:

- (1) users clicking on the same action more than once; and
- (2) for some services (e.g. a mobile VNC service), allowing users to start the service, would also require permission to stop the service.

By default, this service gives permission to users to perform any action (i.e. start or stop) on the service. Hence, for this service, checking does not need to be done per action, instead, we check it per service or per collection of related actions on a service. This simply means when the service is allowed, all actions on it are permitted by default.

In addition, the requested action is only allowed if there is permission given by a space. If there is no permission given (e.g. service allowed = "none"), this simply means the requested action is not allowed. Our system employs both right and prohibition policy objects to determine the service visibility and actions allowed to users. If the permission does not specify the requested action (absence in the permission) such as service allowed = "a particular service" and users are requesting other services than in the permitted list, the mobile client manager then continues to consult the prohibition rule. If it is prohibited, then the requested service is not allowed. If the prohibition rule does not say anything about the requested service (absence of the prohibition rule) such as service prohibited = "a particular service", which is not the requested one, then the system permits the user to perform the requested action. We summarize our policy decision checking in Table I.

Our system checks for consistency of rights, obligations and prohibitions of the same activity within the same policy document. This is to avoid unnecessary inconsistent value of permission, obligation and prohibition. For example, things that are permitted should not be prohibited. Likewise, things that are prohibited should not be permitted. When the internal consistency of policy document is achieved, the

| Right Service allowed | | | Prohibition Service prohibited | | | Result |
|--------------------------|------|-----------------------|-----------------------------------|------|-----------------------|------------------------------------------------------------------------------------------------|
| Any | None | Particular service | Any | None | Particular service | |
| x | - | - | - | x | - | Any services are visible and users are allowed to access any visible services. |
| - | x | - | x | - | - | It further checks the prohibition. As it prohibits any, none of services are visible to users. |
| - | x | - | - | - | x | None of services are visible to users. |
| - | - | x | - | x | - | Only a particular service (as listed in the right) is visible to users. |
| - | - | x | - | - | x | Only a particular service (as listed in the right) is visible to users. |

Table I.
Policy decision checking

system then validates the policy document against the policy schema (as illustrated in the Appendix).

Our system also checks for the conformance of uOsp against the permission that the space gives to the user in that specific location. This is mainly because our system only allows the user to perform a task which is permitted by the space. For example, if $spRu_i$ specifies that user i can only start the music service on Monday from 12 to 2 p.m., this means that the u_iOsp has to satisfy that condition (Monday, from 12 to 2 p.m.). The user is not allowed to impose an obligation on the system to start a Media Player service other than Monday, 12-2 p.m. (other than the permission given by the space).

As they move from one space to another, they need to adapt to rules specified by a space. This is to ensure that the uOsp is still within the scope of the user's permission. The system will review whether or not the requested user's obligation is allowed. This is to avoid a conflict between a user and a space, where a user has specified things to automatically perform, but a space does not allow such a task to be executed due to out of scope permission.

In addition, our system further checks on the consistency of the value of attribute name "*on*". This is to ensure that the rule is imposed to all users in the system (not only to some users). This then helps to avoid the misinterpretation of how the policy object should be interpreted and unnecessary redundancy. The value of attribute *on* = "any" means the policy object is applied to any users in the system. The value of attribute *on* = "a particular entity" (e.g. *on* = "GeneralEntity") means the policy object is applied only to the specified entity. The value of attribute *on* = "OtherEntities" means the policy object is applied to other specified entities. Therefore, if developers or owners of the rooms have specified the value of attribute *on* = "any", they do not need to explicitly impose the rule for each of the entities. Specifying the rule for each of the entities is considered redundant.

Moreover, if they have specified attribute *on* = "a particular entity", to specify imposed on other entities, they should use *on* = "OtherEntities" instead of *on* = "any". This simple means for other entities that is absence in the policy object, the system then reads the rights, obligations or prohibitions from *on* = "OtherEntities". The attribute *on* = "OtherEntities" needs to be used in conjunction with attribute *on* = "a particular entity". We describe in Table II the policy consistency checking of the "imposed on" value.

As for non-shared resource services, when the service is permitted and the requested action is allowed by a space, the mobile client manager then performs the

Table II.
Policy consistency
checking of “imposed
on” value

| Imposed on value “Any” | Imposed on value “A particular entity” | Imposed on value “OtherEntities” | Validity |
|---------------------------|----------------------------------------------|----------------------------------------|----------------------------------------------------------------------------------------|
| x | – | – | True |
| – | x | x | True |
| x | x | x | False (redundant) |
| x | x | – | False (redundant) |
| x | – | x | False (redundant) |
| – | x | – | False (incomplete. It needs to specify other users in the system too) |
| – | – | x | False (incomplete, other entities can only be used if there is a particular entity) |

requested action (e.g. retrieving a user’s information on a Mobile Pocket Pad service), and no further checking needs to be done (i.e. policy conflict detection and resolution). As soon as the system permits a user’s request to perform an action, the mobile client manager then contacts the policy manager.

In addition, a mobile client manager also continuously runs the thread that checks the occurrence of a list of obligations which are imposed by a space on a user (spOu). When the time elapses, the mobile client manager then alerts users (if they are currently logged on to the system) by popping up the message box on their computing devices. This acts as a reminder for users to fulfill their obligations. As for obligations which are imposed on the space by a user (uOsp), the system manager (on the server side) periodically checks for a user’s task list (i.e. uOsp) to look for opportunities to automatically provide assistance to users (Syukur *et al.*, 2004a; Davies *et al.*, 1999). For example, automatically play a user’s favourite music at room B538 from 12 to 12:30 p.m., when the context is satisfied. The context information of when and where to perform such tasks can be retrieved from context_activity document.

When a system detects that the user (the one who specifies uOsp) is currently approaching the destination space, where the obligation needs to be fulfilled and when the current day and time are matching to the context of uOsp policy object. The system refers to a user’s profile document (see Figure 11) to find out the details of the task to be automatically performed. Such details are what music to be played and how to play the music. The details of the task are retrieved by matching the location name.

Having a proactive task assistant is considered useful as the information or task that users required are ready before the user reaches the place. The above proactive sample scenario extends the idea of having context-based services by allowing the end-user to define a rule or policy that specifies when, where and what type of service that s/he wants to be automatically displayed or started at each particular situation. The rules or policies here can also be used to restrict the behaviour of the service in the specific context. For example, start the music at 12 p.m. and pause it for 15 min at 12:15 p.m. It also helps to improve the user’s experience, especially if there is regularity in the user’s activities.

- (3a) *Call the policy conflict detection.* The policy manager then calls a policy conflict detection component to detect whether or not there is a conflict when the requested action is performed. As we detect the conflict only at run time, the only way to detect whether or not there is a conflict is by checking the status of the shared resource device (e.g. not running – idle or running – in use). If it is in idle

mode, the user is then permitted to start the requested service and action. There is no conflict here. If it is currently in use and a user requests to execute the same service with different actions, this then creates a conflict. The system needs to resolve this conflict by using one of the proposed conflict resolution strategies.

- (3b) *Send the result back to the policy manager and call the policy execution.* If no conflict is detected when requesting a shared resource service, the conflict detection module then sends a message to the policy manager (e.g. allowing user A to execute the specified action as no conflict has been detected and so, no resolution is required). The policy manager then calls the policy execution to perform the specified action on the service (e.g. start playing “First Noel” at any nearby desktop machine at room A).
- (4a) *Call the policy conflict resolution.* If there is any conflict detected in step 3 above, the conflict detection result is passed onto the policy conflict resolution component. Our conflict resolution resolves conflicts as soon as they are detected. All conflicts which are detected are actual conflicts, as our system detects conflicts only at run time and the contexts for the conflict to occur have been met. In addition, depending on the result of the policy conflict resolution, the requested action may or may not be permitted.
- (4b) *Send the result back to the mobile client manager and call the policy execution.* This conflict resolution result is then passed back to the policy manager. If the requested action is permitted, the policy manager then executes the requested action. However, if there is a conflict and the result stated that the user is not allowed to perform the specified action, the policy manager then sends back this message to the mobile client manager. The mobile client manager then notifies the user that the requested action is not allowed.
- (5) *Update the shared device status.* The shared computing device status only gets updated when there is permission to perform an action and the requested action is permitted. This then changes the state of a service from “not running” to “running” or from “running” to “not running”. This is particularly useful for shared resource services where services can be accessed by many users simultaneously from their mobile devices.

5. Evaluation of the MHS prototype

The MHS system takes into account the concepts proposed in the previous sections to support the delivery of context-aware services that incorporates policy for controlling access to services. The system performs selection of relevant services and delivery tasks based on a user’s current contexts. This paper evaluates the time that it takes to:

- detect a location context change and update service and policy information as per current location context;
- activate mobile code; and
- check policy authorization decisions.

The prototype system needs to be able to respond to users’ requests with little delay or in a minimum amount of time. All requests from clients to a server and information returned from a server to clients needs to be handled and delivered in a minimum amount of time. That way, we can minimize users’ wait time, thereby, allowing them to perform their tasks efficiently and effectively.

The framework has given promising results in obtaining a list of services, keeping track of a user's location, downloading, executing the mobile code, as well as checking policy decision authorization results. The user wait time for service execution varies depending on the type of services: shared or non-shared resource services. In general, the user wait time is longer for executing a shared resource service, as it involves extra steps to initiate a service process on the target machine. We illustrate our evaluation aspects from retrieving a user's location up to service activation and checking the user's permission in Figure 13.

In our evaluation, experimental results were collected for ten times of service execution on a 206 MHz iPAQ on a wireless Wi-Fi network (54 Mbps). We employ the Ekahau Positioning System to keep track of a user's location. Our server side software components are stored on a 2.09 GHz desktop machine.

5.1 System performance

As shown in Figure 14, as soon as a user moves to another location (i.e. from location A to location B), the Ekahau location system needs to detect the user's current position. The system then updates a list of services and policy decision results for that location. When the user selects a particular service name on a mobile device, the system then

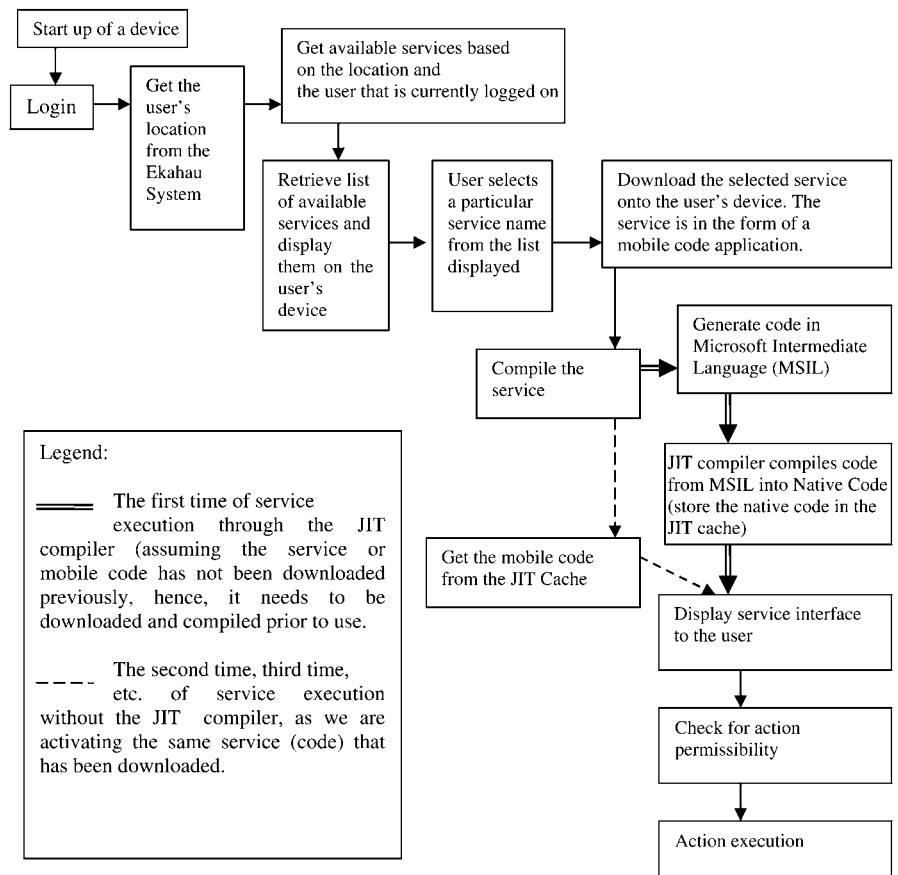


Figure 13.
Evaluation stages of the MHS system

downloads the relevant mobile code, compiles it and displays the service interface. After this, the system then checks the space policy to determine whether or not the user is permitted to perform the requested action. This is done by interpreting the relevant space policy document (i.e. the space policy of where the user currently is) for the specific contexts (e.g. current day, time and activities).

Three main aspects that we evaluate in this section are discussed as follows.

- (1) *Location context change delay.* As we associated services and policies to a geographical location (i.e. a room), when a user moves from one location to another, the system needs to update two things (a) a list of services on the mobile device and (b) a relevant policy document for the location. Both of these updating processes require a confirmation from Ekahau regarding the user's current location (i.e. the user's exact location as s/he moves from one place to another). Hence, the time delay in displaying the updated list of services to users comprises the Ekahau delay in detecting the user's new location, plus, the time that it takes to update a list of available services on the mobile device. In addition, to update policy decision results on the mobile device, the system first needs to know the user's current location (the updated one), then retrieving the relevant policy document for that location and downloading it onto the user's mobile device.

We now describe a formula to calculate the time required for updating a list of services on the mobile device.

$$\begin{aligned}
 T_{\text{location context change delay}(s)} = & T_{\text{Ekahau delay}} + T_{\text{get a user's location}} \\
 & + T_{\text{get a list of services}} \\
 & + T_{\text{display a list of available services on a device}} \\
 & + T_{\text{retrieve or update the policy decision result}}
 \end{aligned}$$

Based on the formula above, we conclude that the worst-case scenario to update a list of services on the mobile device and retrieve policy decision results is the first time of service execution (first time of Web service calling i.e. to get a user's location, to get a list of services), which takes 8.6 s (=2.5 + 3.5 + 0.5 + 0.1 + 2). The 2.5 s is the Ekahau delay in detecting the user's movement, the 3.5 s is the time that it takes to get a user's current location from the Axis Apache server via a Web service call and the 0.5 s

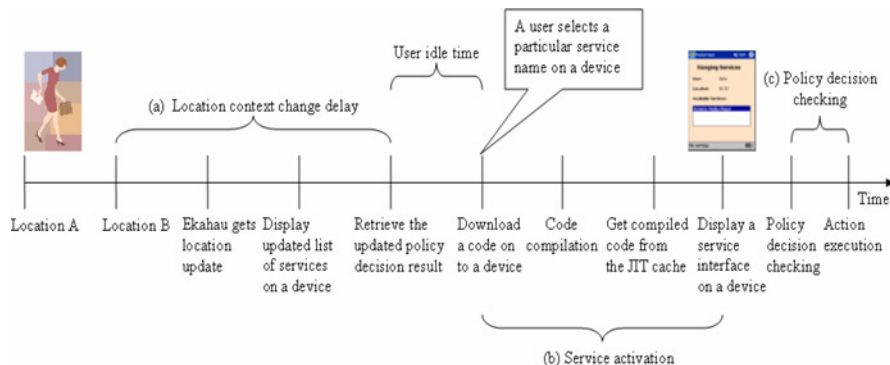


Figure 14. Location context change delay, service activation and policy decision checking on a time line (distances between vertical bars are not indications of time scales)

is the time that is required to get a list of services and the 0.1 s is to display the updated list of services on the device. The 2 s is the time to retrieve the policy decision result.

The best case scenario (i.e. the minimum time delays to see the updated services for the current location) is in any execution which is not the first. In such a case, the delay time is 6.5 s ($=2.5 + 2.5 + 0.4 + 0.1 + 1$). Note that 1 s is the time to update the policy decision result (assuming the policy has been modified and therefore the system needs to update the local policy decision result). The time to detect subsequent location context change decreases to 6.5 s, because the Web service calls in a subsequent context change re-uses the local proxy object, which has been downloaded and compiled previously. The time delay between the user's selection of a service and the display of the selected service interface varies depending on the execution number (see Figure 15).

As we cached the downloaded policy decision result for a location on a device, when a user re-visit the space (room) and we have previously cached policy for that location and assuming the policy has not been changed, there is no policy required to be downloaded and so, for subsequent requests, we can reduce the delay from 6.5 to 5.5 s ($=2.5 + 2.5 + 0.4 + 0.1 + 0$). Replacing Ekahau by some other faster location positioning engines in the future can reduce delays further.

- (2) *Service activation.* It is necessary to display the service interface in a minimum amount of time. Ideally, the user should be able to see the service interface immediately or with minimum delay, as soon as clicking on the service name. In this section, we measured each of the aspects in executing the mobile code on the handheld device. It starts from retrieving the code, downloading, compiling and up to displaying it on the mobile device. The timing variations in performing each of the service activation aspects are illustrated in Figure 15.

Based on Figure 15, we can see that the time required to call Web services: get a user's location, get a list of available services and download a mobile code, decreases for subsequent Web service calls (second, third, fourth, fifth, etc. times of service calling). The first call of the Web service takes a longer time, as the system needs to download and compile the local host Web service proxy object on the device. The proxy object

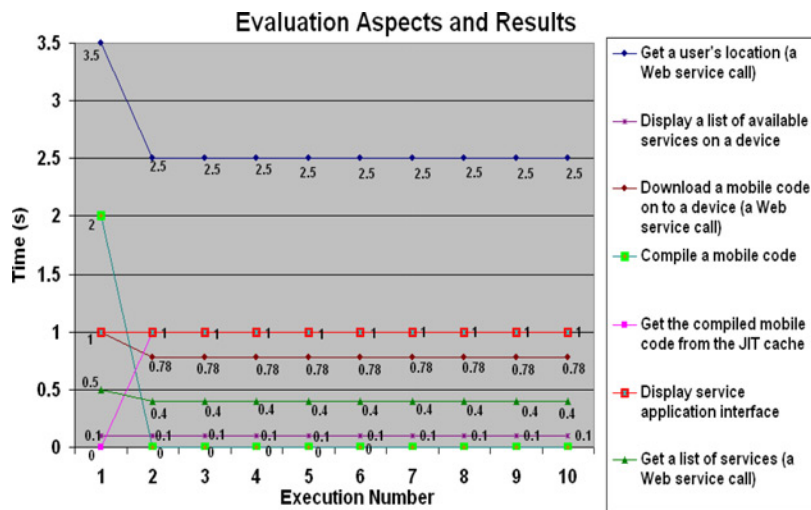


Figure 15. Experimental results

allows the Web service to be treated like other .NET classes. The second and subsequent calls to the service will have much shorter times as they re-use the service proxy object which is already on the device.

In addition, the amount of time required to display a list of services on the mobile device is consistent throughout the executions as the number of services used for the evaluation is the same. Our system gets an updated service list for each user and location from an XML database. The time required for downloading and displaying the service interface on the device depends on the size of the mobile code itself. The larger the size of the mobile code, the longer it takes to download and execute the service application.

Note that the mobile code only needs to be downloaded once to the mobile device; subsequent service executions (i.e. execute the same service application) will reuse the code that has been stored on the device. Therefore, the downloading time for subsequent service executions is zero (for the same service application). However, if the subsequent executions is different from previous runs (i.e. it executes different services which have not been downloaded previously), it would require a client to download the particular mobile code. In this case, it takes around 0.78 s to download the mobile code from a server to a client's device (assuming the code size is 10 kB).

We have developed several mobile services in our system. For testing, we focus on one mobile code, which is a Mobile Pocket Pad service that is 10 KB in size. As we have kept the size of the mobile code constant in our runs and we have been reusing the same service application throughout the repeated executions, the time to display the service interface remains the same. Moreover, the time required to get the compiled mobile code from the JIT cache is constant throughout the second and subsequent executions (1 s). There is no retrieving compiled code from the cache in the first execution (0 s) as the first time of service execution directly compiles the code and displays the compiled code interface on the device.

Compilation of the code that implements a service in the .NET Framework involves two stages:

- (1) the framework compiles the source code (e.g. C# or VB.Net) into the Microsoft Intermediate Language (MSIL); and
- (2) when the application gets executed (at run time), the Just In Time (JIT) Compiler compiles the MSIL code into native code and stores this native code in the JIT cache for future re-use.

Note that service activation at the first time means causing the relevant downloaded mobile code to be compiled and then when executed, displaying the service interface to the user. Subsequent activations of the previously activated service will not involve the service compilation (the compilation time is 0 s). These subsequent activations will retrieve the compiled code from the mobile code storage, i.e. the JIT cache (see Figure 15). However, if the subsequent activations require a service (code) which has not been previously downloaded on to and compiled on the device, it requires the code to be first downloaded and compiled prior to use. The compilation time here varies depending on the size of the mobile code.

The time to compile the code will be longer if the code size is larger. Our framework that supports mobile code and targets mobile devices aims to create a compact mobile code by separating the service interface and functionalities from the system functionalities (i.e. contextual and policy functionalities). In this case, the mobile code only contains the service interface and its functionality. In average, our mobile code

sizes (including the mobile VNC and Media Player service) are 10 kB and the time it takes to compile the code is similar – 2 s.

In general, the first time a service is activated (by the user or by the system), a much longer time delay is experienced compared to the subsequent activations. The time delay for subsequent service activations that execute the service that has been previously downloaded and compiled in the previous runs is consistent throughout the subsequent executions. Moreover, as we are using multi-threading in executing the service application, we experience additional delay in displaying the service interface. This delay refers to a thread task.

Now, we give a formula to measure the service activation:

$$T_{Service\ Activation(s)} = T_{Download\ mobile\ code} + T_{Compile\ mobile\ code} + T_{Get\ compiled\ code} + T_{Display\ service\ interface} + T_{Thread\ task}$$

Based on the testing and evaluation results, we conclude that the worst-case scenario to see the service interface is the first time of service execution which takes 5.1 s (=1 + 2 + 0 + 0.9 + 1.2). The best-case scenario for service activation is any execution number, which is not the first. It takes 3.1 s (=0 + 0 + 1 + 0.9 + 1.2) – assuming the subsequent executions activate the same service application, in which, *the mobile code has been downloaded and compiled in the first or previous runs.*

- (3) *Policy decision checking:* 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 a user’s mobile device, up to detecting whether or not a user is given a permission to perform the requested action, as well as detecting and resolving conflicts with other entities (if any).

The policy evaluation results are illustrated in Figure 16. Based on Figure 16, we can see that the time required to call Web services: send a query from a client to the policy manager, retrieve and download a relevant policy decision document, detect conflict

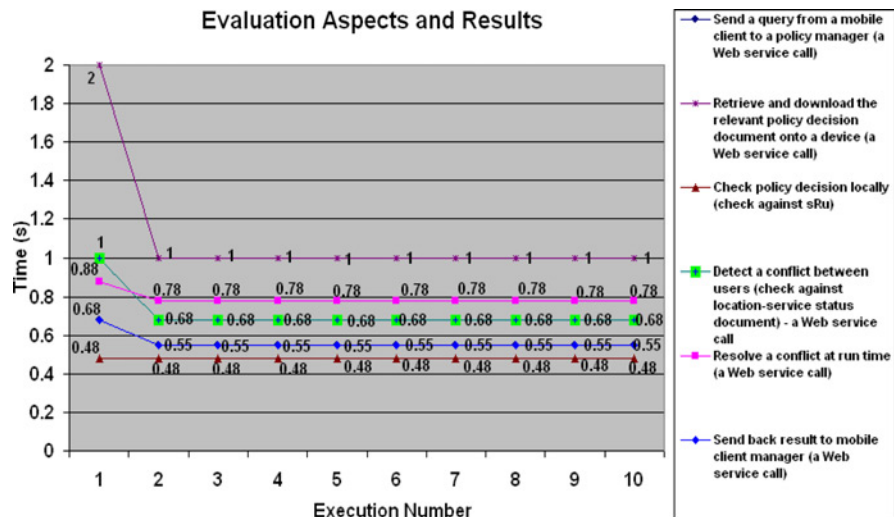


Figure 16. Policy evaluation results

between users, resolve conflict dynamically and send back resolution results to the mobile client manager decreases for subsequent Web service calls.

As discussed in the previous 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, the SOAP client needs to reflect over the Web service proxy object. To reduce this delay, we declare the Web service object globally within a class and call a simple Web service method asynchronously on the constructor. The subsequent calls of this Web method will not incur this reflection overhead and so, it takes 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.

When a user moves from one location to another, our system continuously monitors a user's location and retrieves the policy information accordingly (different spaces uses different policies). 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 2 s 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 being loaded, only takes 1 s. 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 (e.g. MobileComputingLabPolicy.xml) 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 spRu). 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 (0 s). Instead, it will just check the local cached policy results. 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 1 s.

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 data set and hence, the checking can be done in a minimum amount of time. In addition, 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 1 s 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 2 s to read 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 does 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 an action against 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:

$$T_{policy\ checking(s)} = T_{checking\ against\ spRu} + T_{checking\ conflicts\ between\ entities}$$

Based on the testing and evaluation results, it takes 0.48 s to check the policy decision against spRu for the first and subsequent requests. This time remains the same throughout the 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 processing speed of a device and a number of applications running at the time. The faster the speed, the shorter time it takes to complete the checking. Moreover, more applications running during the checking would result a much slower response from CPU to perform 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 method and hence, it requires a Web service call to a server. The first time of Web service calling takes much longer (i.e. 1 s) than the subsequent requests (i.e. 0.68 s).

As for executing a non-shared resource service, it only takes 0.48 s (=0.48 + 0) to check policy decision results for the first and subsequent requests. The shared resource service would take 1.48 s (=0.48 + 1) for the first time checking and reduces to 1.16 s (=0.48 + 0.68) for subsequent checking. Our system performs two types of policy checking: per action and per service (as discussed in Section 4.2.3). For example, by allowing users to start the music service with any action, that would allow users to stop, pause and resume the music. Therefore, for subsequent requests (e.g. stop, pause or resume the music), which are related to the requested action (e.g. start the music), we assume we do not require the policy decision checking. This then reduces unnecessary checking for every single requested action.

Finally, we present a formula to calculate the total user wait time to request 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, checking a policy decision result and resolving conflicts if any. As shown in Figure 16, to resolve one conflict for the first time of a policy conflict resolution web service called, it takes 0.88 s. Resolving subsequent conflicts would take 0.78 s. The time to resolve different conflict resolution techniques (e.g. role hierarchy overrides policy, space holds precedence over policy, obligation holds precedence over visitors, and maximum execution time policy) is generally the same, which is 0.88 s for the first time policy conflict resolution web service call and subsequent calls take 0.78 s.

In our test case scenario, we have two users (e.g. a lecturer and a student) with different roles (e.g. a power entity and a general entity). A lecturer specifies policies for a lecture room for his lecture. A student owns a personal room (e.g. room B538). There is one conflict occurring between these users in a public space (e.g. a lounge room), as they want to start the shared resource service (e.g. a music service) with different songs. To resolve this conflict, our system then applies a role hierarchy overrides policy. Apart from performing these tasks, a lecturer also starts a non-shared resource service (e.g. a Mobile Pocket Pad service) on his pocket PC device. A student also starts

a non-shared resource service (e.g. a Mobile Pocket Pad service) on her pocket PC device. There is no conflict occurs for this non-shared resource service. Therefore, it does not require a conflict resolution for the non-shared resource service (0 s).

We now present the formula to calculate the user's wait time:

$$T_{user\ wait\ time(s)} = T_{location\ context\ change\ delay} + T_{policy\ checking} + T_{policy\ conflict\ resolution}$$

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, is 10.96 s (=8.6 + 1.48 + 0.88). For a non-shared resource service, it is 9.08 s (=8.6 + 0.48 + 0). 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). We assume here that there is a conflict between users to execute a shared resource service, requiring a resolution.

In such a case, the time delay is 0.48 s (=0 + 0.48 + 0) for a non-shared resource service and 2.26 s (=0 + 1.48 + 0.78) for shared resource services. 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 the policy and no location context change is required. The time to detect subsequent requests decrease to 0.48 s for non-shared and 2.26 s for a 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 does not need to update the list of services.

In addition, when the user moves to another location (e.g. from a lounge room to a seminar room), the user wait time is 6.98 s (=6.5 + 0.48 + 0) for non-shared services. It takes 7.98 s (=6.5 + 1.48 + 0) 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 a subsequent Web service call. There is also no conflict between users in a seminar room. If there is a conflict, the time to resolve the conflict is 0.78 s as it is the subsequent policy conflict resolution calls.

In a case where only the policy gets modified and the user is still in the same location, the system only needs to update the policy document. The formula to calculate the user wait time is

$$T_{user\ wait\ time(s)} = T_{retrieve\ or\ update\ policy\ decision\ result} + T_{policy\ checking} + T_{policy\ conflict\ resolution}$$

We assume here the user has been in the location and requested to perform some actions. Therefore, it is not the first time of calling Web services (i.e. location, policy checking and policy conflict resolution Web services). The best case scenario for user wait time is 3.26 s (=1 + 1.48 + 0.78) for shared resource services. We assume there is one conflict between users when executing a shared resource service. It takes 1.48 s (=1 + 0.48 + 0) for a non-shared resource service. In addition, in a situation where the policy decision result is already on the device; the user re-visits the location and

there is no policy modification, the user wait time for a shared resource service would only be 1.48 s ($=0 + 1.48 + 0$) – assuming, there is one conflict between users when executing a shared resource service. It takes 0.48 s ($0 + 0.48 + 0$) for a non-shared resource service.

6. Conclusion and future work

The MHS framework addresses six main requirements for building a context-aware regulated system as discussed in the previous sections:

- (1) *Use a location positioning system.* The MHS system is independent of any indoor location tracking systems. This simply means we can employ any indoor location positioning systems with different sensing techniques (e.g. active badge, radio frequency, etc.). As for the implementation purpose, we employ current version of the Ekahau location positioning system (version 2.0) in order to keep track of a user's location. This is because, the Ekahau does not require any extra hardware for installation and it is based on the signal strength triangulation.
- (2) *Proactively capture contexts, discover and deliver relevant services.* Our system proactively delivers relevant services upon detecting a user's location. When a user selects a particular service name on the device, the system proactively downloads, compiles and displays the service interface to the user. The system also caches the service application for future re-use.
- (3) *Centralized services.* Our system stores services in a centralized place that is accessible by all legitimate users from their mobile devices. This design supports reusability of services across computing devices in pervasive computing environments.
- (4) *Generic mobile framework.* Our system supports the ability of creating mobile code (mobile service) via the MHS framework and adding existing traditional applications to the system. Having a remoting mechanism enables the system to add existing traditional applications. This is done via a system primitive action execution by using a `system.dagnostic.process` to start and stop the service or via the application API.
- (5) *Interoperable contextual software components.* Our system develops contextual software functionalities as Web services. The Web methods are widely accessible from any platform and language.
- (6) *Support contextual policy to control access to services.* Our system controls behaviours of users in accessing services, through the notion of policy. The policy is a set of rules that specify what an entity can do with services in a specific context. The MHS framework supports policy mechanisms for detecting and resolving conflicts if any, as well as enforcing relevant policies. Each of these policy software components is published via the system as a Web service, in order to support interoperability.

Based on the initial concept and implementation of our system, there is still room for improvements, to add, enhance and nurture concepts of context-aware regulated systems. This section presents future work involved in iterating over the conceptual and logical design model, as well as investigating any other possible approaches to improving the efficiency and effectiveness of the development of context-aware regulated services.

- *Scope of the service.* The existing context-aware system implementations only sense and deliver relevant services based on a user's and an environment's current contexts. Our current implementation only delivers services based on a user's current contexts in indoor environments. We plan to investigate and develop a larger scope of contextual services that also target outdoor environments. This gives user flexibility in terms of scope of services that users may be able to see, considering a user is within multiple domains at the same time. For example, in a room which is in a Caulfield campus, which in turn, is in the University, which in turn, is in Melbourne. This requires a comprehensive location model (both indoor and outdoor environments) and we may implement a "zooming" model to address the above situation.
- *Dynamic service discovery.* We have not as of yet incorporated a dynamic service discovery (e.g. using Jini, UPnP and JXTA). In the future, this remains as a critical area of the research in the field.
- *Semantic representation.* The current prototype implementation uses an XML language to represent a mapping between services, contexts and rules. An XML language is a purely text format that does not add any semantic value to its representation. It limits its ability to support complex reasoning such as requests to display a list of students who are currently using a Media Player service and whose supervisor is a professor. Moreover, an XML language does not support reusability of the language in different systems (e.g. security, management, distributed system, etc.). This simply means to include the existing XML elements in other systems; we need to copy and paste whole elements and include them manually.

As a result, a comprehensive representation and reasoning of contexts (i.e. using Semantic Web ontology is required). We plan to explore Semantic Web ontology (Dean *et al.*, 2002; Sadeh *et al.*, 2003; Xiaohang, 2003; Chen, 2004; Gu *et al.*, 2005) to deal with richer contextual information other than those currently supported by the system. Ontologies allow reusable concepts across domains as well as supporting complex query and reasoning (e.g. to display only a list of shared resource services which have been started earlier by users in room H7.93).

References

- Al-Muhtadi, J., Ranganathan, A., Campbell, R. and Mickunas, D. (2003), "Cerberus: a context-aware security schema for smart spaces", *Proceedings of PerCom 2003, Dallas-Fort Worth, TX, March*, pp. 489-96.
- Argyroudis, P.G. and Mahony, D.O. (2004), "Securing communications in the smart home", *Proceedings of the Embedded and Ubiquitous Computing (EUC) Conference, Aizu-Wakamatsu City, August*, pp. 891-902.
- Becker, M.Y. and Sewell, P. (2004), "Cassandra: flexible trust management, applied to electronic health records", *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pp. 139-54.
- Campbell, R., Al-Muhtadi, J., Ranganathan, A. and Mickunas, M.D. (2002), "A flexible, privacy-persevering authentication framework for ubiquitous computing environments", *Proceedings of the 22nd International Workshop on Smart Appliances and Wearable Computing, Vienna*, pp. 771-6.
- Chadha, R., Cheng, Y.-H., Cheng, T., Gadgil, S., Hafid, A., Kim, K., Levin, G., Natarajan, N., Parmeswaran, K., Poylisher, A. and Unger, J. (2003), "PECAN: policy-enabled configuration

- across networks”, *Proceedings of the Fourth International Workshop on Policies for Distributed Systems and Networks, June*, pp. 52-62.
- Chen, H.L. (2004), “An intelligent broker architecture for pervasive context-aware systems”, PhD thesis, University of Maryland Baltimore County, Baltimore, MD.
- Connelly, K. and Khalil, A. (2004), “On negotiating automatic device configuration in smart environments”, *Proceedings of Perware 2004 Workshop, Orlando, 14-17 March*, pp. 213-8.
- Cooney, D. and Roe, P. (2003), “Mobile agents make for flexible web services”, *Proceedings of the Ninth Australasian World Wide Web (AUSWEB) Conference*, available at: ausweb.scu.edu.au/aw04/papers/refereed/cooney/paper.html (accessed 10 October 2006).
- Corradi, A., Montanari, R., Lupu, E., Sloman, M. and Stefanelli, C. (2000), “A flexible access control service for Java mobile code”, *Proceedings of the 16th Annual Computer Security Applications Conference, New Orleans, LA*, pp. 356-65.
- Covington, M., Fogla, P., Zhan, Z. and Ahamad, M. (2002), “A context-aware security architecture for emerging applications”, *Proceedings of the 18th Annual Computer Security Applications Conference, December 2002, USA*, pp. 249-58.
- Covington, M., Long, W., Srinivasan, S., Dey, A.K., Ahamad, M. and Abowd, G.D. (2001), “Securing context-aware applications using environment roles”, *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies SACMAT’01, May*, pp. 10-20.
- Damianou, N., Dulay, N., Lupu, E. and Sloman, M. (2001), “The ponder policy specification language”, *Proceedings of the Second International Workshop on Policies for Distributed Systems and Networks, Bristol*, pp. 18-38.
- Davies, N., Cheverst, K., Mitchell, K. and Friday, A. (1999), “Caches in the air: disseminating tourist information in the guide system”, *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications (WMCSA’99), New Orleans, LA*, pp. 11-19.
- Dean, M., Connolly, D., Harmelen, V.F., Horrocks, I., McGuinness, D., Schneider, P.F. and Stein, L.A. (2002), “OWL web ontology language 1.0 reference”, W3C Working Draft, July 2002.
- Dey, A.K., Abowd, G.D. and Wood, A. (1998), “CyberDesk: a framework for providing self-integrating context-aware services”, *Journal of Knowledge Based Systems*, Vol. 11, pp. 3-13.
- Edwards, W.K. (1996), “Policies and roles in collaborative applications”, *Proceedings of the ACM Conference on Computer Supported Cooperative Work, Boston, MA*, pp. 11-20.
- Ekahau Positioning Engine™ 2.0 (2000), *Developer Guide*, available at: www.ekahau.com/ (accessed 18 October 2003).
- Feinstein, H., Sandhu, R., Coyne, E. and Youman, C. (1996), “Role-based access control models”, *IEEE Computer*, Vol. 29 No. 2, pp. 38-47.
- Flynn, M., Pendlebury, D., Jones, C., Eldridge, M. and Lamming, M. (2000), “The Satchel system architecture: mobile access to documents and services”, *Mobile Networks and Applications*, Vol. 5 No. 4, pp. 243-58.
- Gavrila, S., Kuhn, D.R., Ferraiolo, D.F., Sandhu, R. and Chandramouli, R. (2001), “Proposed nist standard for role based access control”, *ACM Transactions on Information and System Security*, Vol. 4 No. 3, pp. 224-74.
- Gellersen, H.W., Schmidt, A. and Beigl, M. (2002), “Multi-sensor context-awareness in mobile devices and smart artifacts”, *Proceedings of the Mobile Networks and Applications (MONET), October 2002*.
- Giuri, L. and Iglío, P. (1997), “Role templates for content-based access control”, *Proceedings of the Second ACM Workshop on Role Based Access Control, Virginia, VA*.
- Godik, S. and Moses, T. (2002), OASIS eXtensible Access Control Markup Language (XACML), OASIS Committee Specification cs-xacml-specification-1.0, November.

-
- Gu, T., Pung, H.K., Zhang, D.Q. (2005), "A service-oriented middleware for building context-aware services", *Journal of Network and Computer Applications*, Vol. 28 No. 1, pp. 1-18.
- Harroud, H., Ahmed, M. and Karmouch, A. (2003), "Policy-driven personalized multimedia services for mobile users", *IEEE Transactions on Mobile Computing*, Vol. 2 No. 1.
- Hengartner, U. (2005), "Access control to information in pervasive computing environments", PhD thesis, Carnegie Mellon University, Pittsburgh, PA, August.
- Henricksen, K. (2003), "A framework for context-aware pervasive computing applications", PhD thesis, The University of Queensland, Brisbane.
- Kagal, Rei, L. (2002): "A policy language for the Me-Centric Project", Technical Report HPL-2002-270, HP Laboratories, Palo Alto, CA, September.
- Kagal, L. (2004), "A policy-based approach to governing autonomous behavior in distributed environments", PhD thesis, University of Maryland Baltimore County, Baltimore, MD.
- Kagal, L., Finin, T. and Joshi, A. (2001), "Trust-based security in pervasive computing environments", *IEEE Computer Magazine*, December.
- Keeney, J. and Cahill, V. (2003), "Chisel: a policy-driven, context-aware, dynamic adoption framework", *Proceedings of the Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'03)*, pp. 3-14.
- Lymberopoulos, L., Lupu, E. and Sloman, M. (2004), "PONDER policy implementation and validation in a CIM and differentiated services framework", *Proceedings of the Ninth IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, May*, pp. 31-44.
- Mahon, H., Bernet, Y., Herzog, S. and Schnizlein, J. (2000), "Requirements for a policy management system", IETF Internet Draft, available at: www.ietf.org/Internet-drafts/draft-ietf-policy-req-02.txt (accessed 10 October 2004).
- Mally, E. (1926), *The Basic Laws of Ought: Elements of the Logic of Willing* (Mally, Ernst, 1926, *Grundgesetze des Sollens: Elemente der Logik des Willens*), Leuschner und Lubensky, Universitäts-Buchhandlung, Graz.
- Massachusetts Institute of Technology (2004), The Trusted Software Proxies project, available at: <http://oxygen.csail.mit.edu/Network.html#k21> (accessed 10 October 2005).
- Masuoka, R., Chopra, M., Labrou, Y., Song, Z., Chen, W.I., Kagal, L. and Finin, T. (2005), "Policy-based access control for task computing using Rei", *Proceedings of Policy Management for Web Workshop in Conjunction with WWW2005, Chiba, May*, available at: <http://cs.umbc.edu/pm4w/papers/masuoka10.pdf> (accessed 8 October).
- Moffett, J.D. and Sloman, M.S. (1993), "Policy hierarchies for distributed systems management", *IEEE Journal on Selected Areas in Communications*, Vol. 11 No. 9, pp. 1404-14.
- Montanari, R. and Tonti, G. (2002), "A policy-based infrastructure for the dynamic control of agent mobility", *Proceedings of the IEEE Policy Workshop*, p. 206.
- Moyer, M.J., Covington, M.J. and Ahamad, M. (2000), "Generalized role-based access control for securing future applications", *Proceedings of the 23rd National Information Systems Security Conference (NISSC 2000), Baltimore, MD, October*.
- Noble, B. (2000), "System support for mobile, adaptive applications", *IEEE Personal Communications*, No. 1.
- Noble, B.D. and Corner, M.D. (2002), "The case for transient authentication", *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, September 2002*, pp. 24-9.
- Ranganathan, A. (2005), "A task execution framework for autonomic ubiquitous computing", PhD thesis, University of Illinois.

- Robinson, P. and Beigl, M. (2003), "Trust context spaces: an infrastructure for pervasive security in context-aware environments", *Proceedings of the First International Conference on Security in Pervasive Computing*, pp. 157-72.
- Roman, M., Hess, C.K., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K. (2002), "Gaia: a middleware infrastructure to enable active spaces", *IEEE Pervasive Computing*, pp. 74-83.
- Sadeh, N.M., Chan, T.-C., Van, L., Kwon, O.B. and Takizawa, K. (2003), "A semantic web environment for context-aware M-commerce", *Proceedings of ACM Conference on Electronic Commerce 2003, San Diego, CA, June*, pp. 268-9.
- Sam, G., Naldurg, P. and Campbell, R.H. (2002), "Access control for active spaces", *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC'02), Las Vegas, NV, December*, p. 343.
- Sampamane, G. (2005), "Access control for active spaces", PhD thesis, University of Illinois.
- Sandhu, R.S. (1998), *Role-based Access Control. Advances in Computers*, Vol. 48, Academic Press.
- Schilit, B., Adams, N. and Want, R. (1994), "Context-aware computing applications", *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, December*, pp. 85-90, available at: <http://seattleWeb.intel-research.net/people/schilit/wmc-94-schilit.pdf> (accessed 3 October 2003).
- Schilit, B.N. (1995), "A context-aware system architecture for mobile distributed computing", PhD thesis, Columbia University, USA, May 1995.
- Syukur, E., Loke, S.W. and Stanski, P. (2004a), "Hanging services: an investigation of context-sensitivity and mobile code for localised services", *Proceedings of the IEEE International Conference on Mobile Data Management, Berkeley, 19-22 January*, pp. 62-73.
- Syukur, E., Loke, S.W. and Stanski, P. (2004b), "The mobile hanging services framework for context-aware applications: the case of context aware VNC", *Proceedings of WIS (Wireless Information Systems) Workshop, Porto, April*, pp. 81-8.
- Syukur, E., Loke, S.W. and Stanski, P. (2004c), "The mobile hanging services framework for context aware applications: an experience report on context aware VNC", Technical Report No. 151/2004, Monash University, Australia.
- Syukur, E. and Loke, S.W. (2006a), "Context-aware regulation of context-aware mobile services in pervasive computing environments", *Proceedings of the Second International Workshop on Ubiquitous Web Systems and Intelligence (UWSI 2006) in Conjunction with ICCSA Conference 2006, Glasgow, 8-11 May*, pp. 138-47.
- Syukur, E. and Loke, S.W. (2006b), "Implementing context-aware regulation of context-aware mobile services in pervasive computing environments", *Proceedings of the International Journal of Web and Grid Services (IJWGS), Inderscience Publisher*, Vol. 2 No. 3, pp. 260-305.
- Syukur, E. and Loke, S.W. (2006c), "Augmenting everyday objects with context-aware services", CoolCampus summer projects, December 2005–March 2006, available at: www.infotech.monash.edu.au/promotion/coolcampus/summer/summerproj05-06/index.html (accessed 18 August 2007).
- Syukur, E. and Loke, S.W. (2007), "Policy based control of context-aware pervasive services", *Journal of Ubiquitous Computing and Intelligence (JUCI)*, Vol. 1 No. 1, pp. 110-31.
- Thai, T.L. and Lam, H. (2001), *NET Framework Essentials. O'Reilly Programming Series*, 1st ed., O'Reilly, Sebastopol, CA, June.
- Tripathi, A., Ahmed, T., Kulkarni, D., Kumar, R. and Kashiramka, K. (2004), "Context-based secure resource access in pervasive computing environments", *Proceedings of the Second IEEE Percom Workshop (PERCOMW'04)*, pp. 159-63.
- Unwired Planet (1997), *Handheld Device Markup Language Specification*, Unwired Planet, Redwood City, CA.

-
- Uszok, A., Bradshaw, J., Hayes, P., Jeffers, R., Johnson, M., Kulkarni, S., Breedy, M., Lott, J. and Bunch, L. (2003), "DAML reality check: a case study of KAoS domain and policy services", *Proceedings of the International Semantic Web Conference (ISWC 03)*, Sanibel Island, FL, available at: www.ihmc.us/research/projects/KAoS/SemanticWeb03.pdf (accessed 20 August 2005).
- Wies, R. (1994), "Policies in network and system management – formal definition and architecture", *Journal of Network System Management*, Vol. 2 No. 1.
- W3C (1999). *XML Path Language (XPath). Version 1.0*, available at: www.w3.org/TR/xpath (accessed 3 October 2003).
- Xiaohang, W. (2003), "The context gateway: a pervasive computing infrastructure for context-aware services", Research Proposal, Nanyang University of Singapore and Context-Aware Dept., Institute for Infocomm Research, November 2003, available at: www.comp.nus.edu.sg/~wangxia2 (accessed 8 May 2004).
- Yialelis, N., Lupu, E. and Sloman, M. (1996), "Role-based security for distributed object systems", *Proceedings of the Fifth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'96)*, p. 80.
- Zhuang, W., Gan, Y.-S., Loh, K.-J. and Chua, K.-C. (2003), "Policy-based QoS management architecture in an integrated UMTS and WLAN environment", *IEEE Communication Magazine*, Vol. 41 No. 11, pp. 118-25.

Further reading

- Fox, A., Gribble, S.D., Chawathe, Y. and Brewer, E.A. (1998), "Adapting to network and client variation using active proxies: lessons and perspectives", *Proceedings of the IEEE Personal Communications, Special issue on adapting to network and client variability, August*.
- Godefroid, P., Herbsleb, J.D., Jagadeesan, L. and Li, D. (2000), "Ensuring privacy in presence awareness systems: an automated verification approach", *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, Philadelphia, PA*, pp. 59-68.
- Gu, X., Nahrstedt, K., Messer, A., Greenberg, I. and Milojicic, D. (2003), "Adaptive offloading inference for delivering applications in pervasive computing environments", *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (Percom'03), March*, pp. 107-14.
- Lara, E., Wallach, D.S. and Zwaenepoel, W. (2001), "Puppeteer: component based adaptation for mobile computing", *Proceedings of the Third USENIX Symposium on Internet Technologies and Systems 2001, March*.
- Messer, A., Greenberg, I., Bernadat, P., Milojicic, D., Chen, D., Guili, T.J. and Gu, X. (2002), "Towards a distributed platform for resource constrained devices", *Proceedings of IEEE Second International Conference on Distributed Computing Systems (ICDCS 2002), Vienna, July*, p. 43.
- Snekkenes, E. (2001), "Concepts for personal location privacy policies", *Proceedings of the Third ACM Conference on Electronic Commerce, Tampa, FL, October*, pp. 48-57.

Corresponding author

Seng Wai Loke can be contacted at: s.loke@latrobe.edu.au