The MHS Methodology: Analysis and Design for Context-Aware Systems

Evi Syukur\(^1\) and Seng Wai Loke\(^2\)

\(^1\)Caulfield Institute of Technology, Monash University, Australia
evi.syukur@csse.monash.edu.au

\(^2\)Department of Computer Science and Computer Engineering, La Trobe University, Australia
S.Loke@Latrobe.edu.au

Abstract

This paper presents a methodology (is called MHS) for analyzing, and designing pervasive systems starting from designing pervasive environments, entities to pervasive software components. Our MHS methodology is general, in that it is not only specific to one particular domain, it is applicable to a wide range of pervasive environments, comprehensive, and in a way that it deals with pervasive environmental and contextual software related system aspects. One pervasive campus case study is introduced to illustrate the MHS methodology that supports modularity of context-aware system design and is applicable in different pervasive environments.

1. Introduction and Related Work

In general, there are two technical hurdles to the extensive development and use of a pervasive system. First, there is a lack of a proven methodology enabling designers to clearly model and structure applications as contextual services starting from analyzing the requirements, design to implementation. Having a methodology helps and gives an idea to any developers (mobile computing, agent or networking) to develop a pervasive context-aware system from scratch to suit dynamic emerging of system elements. Second, there is still no clear specification of system elements and components required in pervasive systems.

There are several proposed methodologies in the field of: (a) Object Oriented systems [3] however, the guideline does not help much, particularly in creating a pervasive system that employs proactivity in sensing and delivering contextual services that takes full advantage of the convergence of advanced computing devices and connected software systems via wireless technology, the Internet and Web services\(^1\); (b) multi agent systems [1, 2, 5, 6, 7]. Multi agent design has provided a strong foundation upon a pervasive system which to build. However, multi agent methodologies are not directly applicable to pervasive systems. This is mainly due to pervasive systems typically consist of many system elements and components (many objects/agents), and so, it becomes increasingly complex in analysing, modeling and implementing the implicit interactions between these contextual pervasive components.

To the best of our knowledge, there is currently no research done in pervasive context-aware system methodologies. Most of the work done is on low-level descriptions and concepts than on a high level design methodology and without any guidance (step by step guidelines) on how to build the system from scratch. This makes the system design is too focus on a particular environment/platform/implementation, and so makes it difficult to apply or borrow these pervasive concepts to other environments that may have different system elements, architecture and modeling.

This paper focuses on the methodology that provides step by step high level guidance for building a context-aware pervasive system. It guides developers through an entire software development lifecycle starting from problem description, system requirements, constructing system elements, graphical design to functional design. Our methodology is built upon the work of MHS pervasive system [4, 8, 9] that supports context sensitivity in an indoor pervasive campus environment. We also illustrate how our methodology is used in a real pervasive environment (i.e., in a campus environment).

The rest of this paper is organised as follows. In section 2, we discuss our MHS methodology that consists of analysis and design phases. In section 3, we give a case study that illustrates how our methodology

\(^1\)http://www.w3.org/2002/ws/
is used in a pervasive campus environment. In section 4, we conclude with future work.

2. MHS Methodology

The primary purpose of MHS methodology is to guide a system designer or a developer through the pervasive software lifecycle from a specification, analysis, design and to an implemented pervasive system. MHS methodology is independent of particular context-aware pervasive system architecture, programming language and platforms used for the implementation. Ideally, a pervasive system modeled and designed in MHS could be implemented in many different ways (techniques) as our guideline provides a high-level abstraction and conceptual overview. This allows a development of a pervasive system that targets many different environments (e.g., a pervasive campus, pervasive shopping mall, pervasive recreation park, pervasive home environments, etc.). MHS also provides the ability to track changes (forward or backward) throughout different phases of the methodology and its corresponding constructs.

Our methodology provides a modularity and clear-cut design by grouping the similar system components into one module. This allows a developer to modify the existing elements or concepts on a particular module only with less impact on other modules, and in the future, we can still add new modules into a system, though not currently specified in the MHS methodology.

In addition, MHS methodology gives a systematic view for developers starting from problem descriptions, requirements, to a design that is reasonably detailed that can be implemented directly. In applying MHS guidelines, the developers move from the conceptual overview (abstract concepts) to more concrete and detailed concepts. The analysis and design phases can be considered as a process of developing detailed models that give developers the concrete idea towards the system implementation. In our methodology, we identify three basic principals/models that a pervasive system needs to have: (a) Software system elements that consist of contextual element, service element, interaction/message passing element, (b) Environment element, and (c) Entity element. Most of the existing pervasive systems only focus on designing and implementing context-aware software related components such as context-sensing, context-collector, context-interpreter, service discovery, etc., we recognize it is useful to clearly model the target environment and entity in order to deliver better pervasive systems. The figure 1 below illustrates our methodology.

![Figure 1. MHS Methodology](image)

2.1 Analysis Phase

Our analysis phase consists of capturing system requirements, refining scopes and constructing system elements. The objective of this phase is to give developers an idea of the target pervasive environment and its constituents by analyzing the system scenarios and requirements. It is a first step towards modeling the system. At the end of this phase, we hope developers could come out with system elements required in pervasive computing environments (e.g., it has mobile users, services, contexts, etc.). We now describe steps required in this analysis phase:

A. Capturing system requirements. Developers need to analyse the situation, and problem of pervasive computing environments by studying an existing pervasive organizational setting. The output of this phase is an organizational model, which includes the system requirements and system goals. We recognize four system requirements in building a pervasive system: (a) Have sensing systems and ability to sense users’ contextual information accurately, (b) Proactively detects and delivers relevant services to mobile users, (c) Ability to access service information at any time and any place while on the move, (d) Support modularity, interoperability, extensibility and scalability of a system.

B. Refining Scopes. Developers then need to clearly specify the scope of a system (i.e., the aim of developing a system such as sensing and collecting a user’s contextual information, covering delivery services to users and as well as, controlling users’ behaviours in accessing mobile services.
C. Constructing System Elements. A pervasive system consists of a collection of elements or concepts, which have a number of social dependencies with other elements in their environment. We model each of these elements as a module that is independent from other modules, but they can interact or communicate to pass information. We recognize three basic system elements in building a pervasive system: a pervasive environment (that consists of domains and spaces), mobile users (entities), and contextual software systems (that consist of mobile services, and context). Figure 2 below describes the relationships between system elements in a pervasive environment.

![Figure 2. Pervasive System Elements](image)

We now briefly explain each of the system elements in pervasive computing environments:

1. **Environment.** Environment is an atmosphere surrounding a system that influences concepts, elements of the system and how end-users should behave. The environment could be a distributed environment, pervasive environment, agent environment, etc. Depending on the nature of an environment, each of these environments will have different conceptual view of the system, different system elements, architectures, modeling and implementation details. Therefore, identifying the target environment early would help to better architect and model the system.

2. **Entity.** Entity is an end-user who is the target of a system. This can be a physical embedded or non-embedded object (a client software system) in the environment, and a mobile user who is always on the move. Each entity in the system has a role and depending on its role, different services, permissions and responsibilities are imposed. The permission here governs the entities’ behaviours in accessing, requesting and performing certain actions on the service. For example, an entity might be allowed to see “Service A” but not “Service B”. The responsibility is a task that an entity needs to perform. The task can be performed automatically by the system or manually by the end-user. For example, an entity is responsible for registering his/her personal device, organization details, preferences, etc.

3. **Contextual software systems.** Software system here refers to a system component that may reside on the client side or server side for delivering contextual functionalities (e.g., a contextual software module, a service module, etc.). Just like an entity, a software system also has a role. Role here refers to a feature or functionality that a software system needs to deliver. For example, a contextual software module has context sensing role, context collector role, context interpreter role, etc. In addition, each role has its own permissions and responsibilities. For example, the context collector role is allowed to retrieve the user’s location information; it has the permission to obtain the user’s current proximity. Responsibilities are tasks that need to be satisfied. For example, for context collector role, it needs to ensure that it collects all entities’ context information accurately and with a minimum amount of time. One software system/ component can also interact with another software system such as: a context sensing passes the sensed information to context interpreter and then, a context interpreter passes the information onto a system manager to analyse and decide the relevant services for this particular context information.

2.2 Design Phase

We divide the design phase into two sub phases: conceptual modeling and functional modeling.

2.2.1 Conceptual modeling

A. Environment modeling

Our view of a pervasive environment consists of logical boundary (domain) such as a campus domain, shopping mall domain, etc. that may contain several physical spaces and sub spaces. Each domain offers a unique feature, in which the system elements may be different from another domain. Basically, in all pervasive domains, we have the same basic elements (e.g., have mobile users, services, contexts, etc.), but the instances could be different. For example, in a campus domain, we have students, and staffs as target mobile users, but, in a shopping mall domain, we have buyers, and sellers. Type of services delivered for each domain are also different, this is to suit the purpose of a domain (e.g., a campus domain mainly delivers services which are related to an education field).

One pervasive domain may consist of none or more pervasive sub domains. A domain or sub domain may also consist of several physical spaces. An environment can be considered as a domain if the
scope or boundary is large enough and it offers a unique feature (e.g., a campus is for education that is different from shopping mall for shopping or entertainment purpose). In contrast, a sub domain is a subset of a domain boundary (e.g., a campus domain may consist of sub domains: IT department and Business department).

The space is a physical boundary that is conceivable. For example, a park domain that consists of badminton court space, garden space, walking path space, barbeque area space, etc. Each of these spaces may offer different services/functionalities depending on its purpose, but they all still offer similar services (as per a recreation domain).

b. Entity modeling

Our view of an entity consists of mobile users who are always on the move or a software client that is embedded into the environment. We recognize five behaviours of an entity in pervasive environments: (a) enter/move in or out of a domain, physical spaces, (b) access services, (c) perform actions on the service, (d) request services, and (e) perform responsibility tasks. In addition, when designing an entity object, there are several data/information associated with it, such as: a user’s identity, history file, preferences, and ownership information (e.g., device, room, etc.).

c. Mobile service modeling

The notion of “service” suggests the use of software systems for providing assistance to users to complete their daily tasks. As illustrated in [8], the service is a software system that is enlisted as we need them. Ideally, context-aware pervasive systems must hide all the low level details from retrieving the available services to performing actions on the service. So, instead of end-users manually managing their computing environment by knowing, for example, details regarding location of a service, how to call a service (i.e., service API), deliver and execute it on the mobile device, all of these are handled implicitly by the system.

In modeling a contextual mobile service, we also associate certain data/information and behaviours into the service. Such behaviours are a service can be downloaded - from a server to a client’s device; compiled - this depends on how the service is created, it can be using a mobile code that requires code compilation, or just a normal web page that only requires a web browser to display the service information (and so, it does not require a service compilation); cached - a service can also be stored temporarily on a client device for future re-use; uploaded - a service (e.g., a mobile pocket pad service [8]) provides an ability for users to write their information and pin it back to the server). A service also has a set of information or data associated with it, such as: a service description; actions associated with a service, and role of a service.

d. Context modeling

We view context as a parameter to suggest on behalf of an entity what services will be useful and relevant with respect to the entities’ situation. Context information can be a user identity, location, day, time, activities, physical objects surrounding users (e.g., RFID object tags), etc. A location context is represented by an indoor/outdoor logical model. Sensing the user’s current context information, a pervasive service would be able to trigger and deliver relevant set of mobile services which are considered useful for the user in that particular context. With the use of context, a pervasive system would have the ability to usefully adapt services or applications to the user’s current situation, intention or environment, as well as, presenting services in forms that the user can readily use, perhaps personalized to his/her needs. Some behaviours associated with the context are context is sensed by sensor systems, context is collected by a context-collector and context is transferred to a system manager for further analysis to retrieve relevant services. Context also has context description (e.g., name) and data/information associated with it.

2.2.2 Functional Design

We now discuss the functional design of each software system mentioned in section 2.2.1 above.

A. Context module

Context module has four properties that need to be addressed: (a) Accuracy properties – does the sensor system capture the user’s context accurately?; (b) Context delay properties – how long it takes to retrieve the user’s updated context information?; (c) Usefulness properties – having this additional context, would it be useful in deciding a list of relevant services for entities?; and (d) Dynamicity properties – context information is often changed dynamically, does the contextual module cope with this?

In addition, a context module has several sub roles:

- **Sensing task.** A context-aware system needs to employ several sensor systems to sense user/environment context information.
- **Collector task.** An important design question in pervasive computing environments is how to obtain or collect context information to enable adaptive service behaviors. One solution is by requiring each service application to independently collect the necessary context information. This technique, however, places a burden on system developers in which, they have to manually add context-collector behaviours to every
single application that they are developing or will develop in the future. The maintenance is also costly and time consuming as developers need to manually update or modify the same changes on each service application. Another solution is to have a shared software component that handles and collects context information for use by all services in the system.

- **Interpreter task.** The context interpreter then analyses the retrieved context information and find the relevant services that match to this context.

**B. Service module**

A contextual service has two properties: **relevancy property** and **service delay property.** The contextual system must enable users to receive a relevant set of services that fit his/her current context, instead of a barrage of irrelevant services and these relevant services need to be delivered onto a client’s device in minimum amount of time (with minimum delay). In general, a service is considered relevant to users if the services enlisted as users need them at right context information. Functional details of a service are:

- **Discovery task** is discovering a list of available services in the environment.
- **Interpreter task** is filtering the discovered services according to a user’s needs (e.g., user preferences)
- **Delivery task** is transferring services onto a client device.

We now summarise the analysis and design phases of our methodology:

1. **Analysing pervasive scenarios**
   Output: system requirements and scope

2. **Identify elements in the system.** After identifying a list of system elements, we then need to architect, model a relationship between the system elements and specify detailed design (both conceptual and functional designs). A domain in a pervasive environment typically corresponds to: a logical boundary that offers a unique feature. This logical boundary can be a logical scope/area, an organization itself, a department within an organization, or an abstract/virtual collection of tasks. Within one domain, it consists of several elements. Element in a system typically corresponds to:

- an actor that plays an important role in a domain or space environment
- an individual, either within a system or organization or acting independently.

Output: An independent element model, which describes the system and its structures.

3. **Identify the role of software systems.** The role typically corresponds to a functionality/task that it offers within a space/domain environment

Output: Software systems’ role model.

4. For each software system, **elaborate the properties and identify its functional design.**

Output: A fully elaborated element model, which documents the property and functional tasks that it needs to satisfy.

5. **Iterate stages (1)-(3)**

**3. MHS on Campus: Analysis and Design**

As discussed earlier in introduction, our definition of a pervasive computing environment consists of entities, spaces, services, mobile devices, workstations and contexts. The details of each of these concepts depend on the target pervasive system and its environment. For example, an entity in a campus domain refers to a student, a lecturer and a head of school, however, in a shopping mall domain, it could mean something different i.e., a customer and a seller. This paper mainly focuses on the pervasive concepts in a campus domain.

We now discuss the system elements of our campus system:

**a. Environment element.** We view a campus domain as a top hierarchy that consists of many sub domains such as a collection of faculties, departments and schools. Within the department itself, we have another sub domain (e.g., IT R&D department). The IT R&D sub domain also consists of several physical boundaries such as floors, and rooms (e.g., meeting rooms, office rooms, etc.). Noted: services and contexts are shared across domains/sub domains. We can also have an instance of MHS pervasive system in each domain/sub domain.

**b. Entity element.** Entities here refer to mobile users which are always on the move (move from one geographical space to another). Three types of entities in our system are a student, a lecturer, and a head of school. By default, our system imposes certain rights (denoted by sRe), obligations (sOe) and prohibitions (sPe) to each of these entities depending on the role that the entity has and the physical space that the entity is visiting. While on the move, the user also accesses services from a mobile device. The mobile user can request to execute a service or impose a certain obligation to a system (eOs) to help users to auto perform the services just when the system detects the user’s contexts [9, 10].

**c. Software system element**

Our MHS explores context-sensitivity and mobile code in order to provide useful services for users with minimal or no effort for service set up prior to use. As the user’s context changes, the system proactively computes a new set of services which are useful for the
user in that context and this list of services is automatically updated on the user’s mobile device.

As the user selects the particular service name on the mobile device, the highly compact (with respect to limited device resources) mobile code that provides control for the service is then downloaded. The execution of the service or mobile code on the mobile device is done in minimum amount of time and so, the user would be able to see the service interface immediately or with minimum delay. Our MHS system addresses two main properties in modeling a context: (a) accuracy property: we employ an indoor location sensing system to detect the user’s movement. We increase the location sensing accuracy by employing a number of access points in a location. We also calibrate the logical area with several tracking paths, and we model a location as a logical area that consists of several physical spaces. Therefore, by detecting the logical boundary, the system then knows roughly relevant services for that boundary (e.g., teaching logical area that consists of seminar room, lab room, lecture theater, etc. Basically, each of these spaces within the logical area offers similar services); (b) context-change delay: Ideally, the user should see the service interface as soon as it is downloaded on the mobile device and no need to know the details on how to compile and execute the service on the mobile device. This needs to be done and handled proactively by the system [8]. Our system addresses the delays by employing prefetching techniques that prefetches services ahead of use (based on guessing on what the user is going to do next) [11].

4. Conclusions and Future Work

We had discussed our initial MHS methodology for developing context-aware pervasive systems. Our methodology is modular as we group each of the software functionalities into a module. This then allows an addition, modification of a particular module with less impact on other modules. We also recognize two main modules in delivering contextual pervasive services: contextual and service modules. As part of our future work, we will extend our guidelines (a) to cover an access control (policy) module that governs entities’ behaviours in accessing services in particular contexts, (b) dynamic emerging of system elements (e.g., dynamic context information, users, services, environment) and software systems.

References