Tools for Designing and Prototyping Activity-based Pervasive Applications

Pascal Bruegger, Denis Lalanne, Agnes Lisowska, Béat Hirsbrunner
Department of Informatics
University of Fribourg
Fribourg, Switzerland
{first.lastname}@unifr.ch

ABSTRACT

This paper proposes a new approach for modelling, testing and prototyping pervasive, possibly mobile, and distributed applications. It describes a set of tools aimed at supporting designers in the conceptualisation of their application and in the software development stage, and proposes a method for checking the validity of their design. The article also presents a pervasive application implemented and evaluated using our approach. It concludes with propositions for improvements in order to build a complete modelling, prototyping and testing framework for pervasive applications.

Categories and Subject Descriptors

H.4 [**H5.2**]: D.2.8

General Terms

Pervasive computing

Keywords

Mobile computing, Pervasive Computing, Activity-based computing, User Centered Design

1. INTRODUCTION

For some years already, research in the domain of pervasive computing systems and mobile computing has focused on concepts and frameworks for the development of end-user applications that take into consideration the user's context and location to do the right thing at the right time. Two important difficulties during the design and development of such applications, distributed by nature, are the integration of a variety of heterogenous technologies (e.g. mobile phone, PDA, back-end server, communication protocols) and testing the validity of the technological choices.

There has been much discussion about the use of prototypes of pervasive applications in order to elicit user requirements from potential end-users to help feed the design or to evaluate options for the front-end of the application that the user will actually be interacting with [1, 6, 10, 16]. In order to support this type of requirements elicitation and evaluation, toolkits such as [2, 14] exist to encourage rapid prototyping. But, these toolkits are often too restrictive, either in the types of applications that they can be used to create, or in the flexibility of adding and modifying code. Consequently, difficulties might be encountered when trying to transfer the initial prototypes that they produce to more concrete and finalized products.

We have noticed that there is a lack of tools, guidelines and methods to help designers in the definition, validation, prototyping and development of their design choices in a holistic manner, where all of the steps can be done within the same framework and using a cohesive set of tools. In this paper, we propose a new approach for designing and implementing pervasive applications that aims to fill this gap.

In our approach, shown in Figure 1, the KUI model and its respective uMove conceptual framework enables designers to specify the architecture and content of their pervasive system. The IWaT methodology is then used to test the overall functional design of the system (e.g. behavior of the system, message passing between modules) in order to validate the design. Finally, the uMove API enables rapid-prototyping at a fairly high level of fidelity and the eventual full-scale implementation of the system by directly transferring the design specification made using the uMove conceptual framework. It is important to note that the design lifecycle proposed in our approach supports the functional specification of the system and should carefully complement a regular user centered design approach to elicit user requirements (beforehand) and to evaluate the design from end-user perspective once a prototype is available.

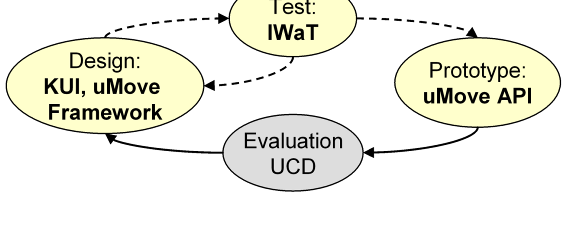

Figure 1: Project development phases and tools

In the rest of this paper, we will focus on describing the KUI semantic model (section 2), the uMove conceptual model (section 3) and the iWaT validation method (section 4), and will only briefly discuss the uMove API (section 3). A case study applying our approach is described in section 5 and conclusions are drawn in section 6.

2. THE SEMANTIC MODEL: KUI

The concept of the Kinetic User Interface (KUI) [4] is a way of grouping Weiser's Ubiquitous Computing vision [19] and Dourish's Embodied Interaction vision [9]. In the early '90s, Marc Weiser predicted that computers would disappear and computing power would fade inside the network infrastructure. Paul Dourish investigated how to move the interface "off the screen" and into the real world. In his model, users can interact with physical objects which have become augmented with computational abilities. KUI-based systems are intended to enable the merging of these two visions, where the motions of users or objects in physical space are recognised and processed as meaningful events. The KUI model is a conceptual framework helping in the design of pervasive systems including mobile applications or server-based systems integrating user's locations and activities.

2.1 A systemic approach

The KUI model is based on General System Theory (GST). GST defined by von Bertalanffy [18] gives the framework and the concepts to model specific systems studied in sciences such as biology or chemistry. We consider that any user or object moving in their environment is part of a system made up of different components such as buildings, rooms, streets, objects and other users. For Alain Bouvier ([3], p.18), a system (a complex organised unit) is a set of elements or entities in dynamic interaction, organised to reach a certain goal and differentiated within its environment. It has an identity and represents a "finalised whole". Each of these entities exhibits "behaviour", action or change, and this behaviour is considered to be related in some way to the environment of the entities, that is, with other entities with which it comes into contact or into some relationship. The important points are that 1) the entity's actions (activities, behaviour) are related to their environment and that 2) entities form relationships with one another. In the KUI model, systems are open and dynamic. Their complexity evolves over time with respect to their components. Components can join and leave systems, increasing or reducing their size. We have included two concepts which are not present in the definitions above: the *observer* (who/what is observing the system) and the *view* (the observer's point of view).

We define a system as a set of observable, interacting and interdependent objects, physical or virtual, forming an integrated whole. The system includes different types of objects: entities, observers, and views (Fig. 2).

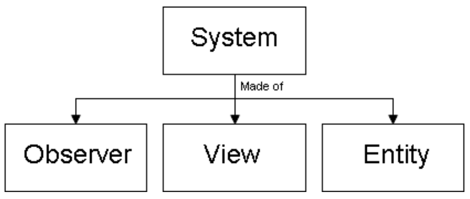

Figure 2: System diagram

This model offers a simple way for pervasive computing system designers to describe their system, which is made up of components, objects and rules that once defined will be programmed. We will now define the objects from which the system is composed.

2.2 Entities

Entities are the observable elements of the system. They can be physical or virtual, living things (users, humans or animals), moving objects (cars or planes) or places (rooms, floors or buildings). An entity is made of *contexts* and does *activities.*

2.2.1 Contexts

A. Dey et al. in [8] define a context as any information that can be used to characterise the situation of an entity. Context includes location, identity, activity and time. In our model, contexts are used to define the attributes of an entity. Contexts do not include the activity. The activity is influenced by the environment and therefore by the contexts in which it is done. We will see later that contexts provide relevant information to the observer in the situation analysis. We mainly use the following contexts in our model:

- Identity. It allows to uniquely identify the entity within the system.
- Location. It gives the physical and logical position of the entity.
- Role. Entities have 2 roles: *Actor* (e.g. human, robot, object) or *Place* (e.g. rooms, house, boat). An actor is said to be atomic and cannot contain other entities. A place is said to be complex because it can contain other entities (actors or places).
- Status. An entity has two possible statuses: mobile (motion capabilities) or static (fixed).
- Structure. It defines the contained entities.
- Relations. Two types of relations between entities: spatio-temporal relations (physical connections between entities like "inside" or "next to") and interactional relations between actors (needed to carry out complex activities).

Other contexts can be defined for an entity and would depend on the characteristics of the system. For instance, temperature or light intensity can be useful contexts for observers.

2.2.2 Activities in places

Activities are controlled within a place. Places have rules that determine the authorised, forbidden and negotiable activities. We introduce the concept of activity lists. White-listed activities are the authorised activities within a place. Black-listed activities are, on the contrary, forbidden and provoke an immediate reaction from the observer. We also take into consideration what we call the "grey" list. If an activity is not explicitly declared in the white or black lists then it is "negotiable" and gives the freedom to evaluate it and make an inference from the situation. Activity lists allow the observer to quickly react when something is going on in a place.

2.3 Observers and views

The second part of the system consists of observing the entities. Observers are the agents which collect and analyse information (activities and contexts) about actors and places and possibly react to particular situations. Observers have specific roles and analyse one or a small number of situations. To illustrate this concept, let us take the example of a family house where several rooms (kitchen, living room, bedrooms) afford different activities. Observers shall be placed in each room in order to evaluate situations taking place in them, taking into consideration the actor's activity and context.

2.3.1 *Non-intrusive behaviour of observers*

Weiser, in [20], introduced the concept of calm technology. In his concept, the user is increasingly surrounded by computing devices and sensors. It becomes necessary to limit the direct interaction with computing systems in order to avoid an unneeded cognitive load and let the user concentrate on their main activity. Our concept of observer is inspired by Weiser's idea. There is no interference with actors and places: the observer only reports situations to the higher level and lets the application decide what to do.

2.3.2 *Views*

The entities are observed from certain points of view. Observers can select different points of view to analyse the same situation. Each point of view represents a focus on the situation. Many observers can use similar views for a different situation analysis. A view is a multi-dimensional filter placed between an observer and the entities. We have 2 dimensions in our model of view: range and level.

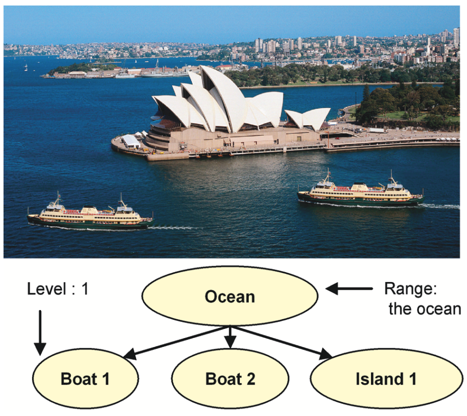

Figure 3: Concept of view in KUI: Range and Level parameters

As shown in Figure 3, the range is a parameter that influences the scope of the observation (e.g. the ocean or only a cruise boat) and the level is the parameter which gives the granularity of the observation (e.g. decks or decks and cabins or passengers).

2.4 An activity-based model for situation awareness

We now define how the kinetic information from the different entities is processed and how a situation is derived from a simple motion. Context-aware systems often take into consideration the user's external parameters such as location, time, social information and activity to characterise a situation. In our model, we bring a new point of view to situation characterisation by considering the activity as separated from a context. An activity must be interpreted in given contexts in order to fully understand the situation (e.g. driving in the snow or on a dry road).

2.4.1 *Situations*

We have noted two points of view for describing situations. In [14], Y. Li and J. Landay propose a new interaction paradigm for Ubicomp based on activity (activity-based ubiquitous computing). In their model, the relation between activity and situation is defined as follows: An activity evolves every time it is carried out in a particular situation. A situation is a set of actions or tasks performed under certain circumstances. Circumstances are what we call contexts in our model. For Loke in [15], the notion of context is linked to the notion of situation. He proposes the aggregation of contexts (perhaps varieties of) in order to determine the situation of entities. In that sense the situation is thought of as being at a higher level than context. Loke makes a difference between activity and situation and considers an activity as a type of contextual information to characterise a situation. Our model of situation combines the two visions and we define it as follows: *A situation is any activity performed in contexts* (Fig. 4).

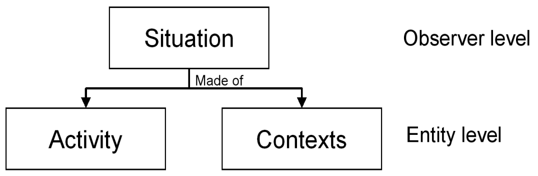

Figure 4: Situation model diagram in KUI

2.4.2 *Activity*

For Loke [15], activity typically refers to actions or operations undertaken by human beings such as "cooking", "running", "reading". For Y. Li and J. Landay [14], an action like "running" is not considered as an activity because it focuses on an immediate goal. For Kuutti, in [13], an activity is the long-term transformation process of an object (e.g. a user's body) oriented toward a motive (e.g. keeping fit). The notions of "long term" and "immediate" allow the separation of activities and actions. In our model (Fig. 5), we consider an activity to be made of detected motions aggregated into operations and actions, and it is an input for observers.

3. UMOVE: THE DEVELOPMENT FRAMEWORK

In this section we describe the uMove framework that allows to define and implement a KUI system on top of which specific applications will be developed. The framework contains two specific parts: the conceptual framework and the Java API.

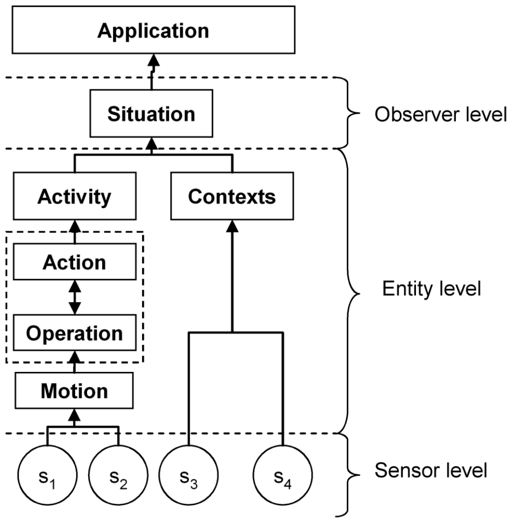

Figure 5: Data flow from motion to situation in KUI

3.1 Related work

Pervasive computing aspects such as context-awareness (i.e location-awareness) or activity-based application and rapid prototyping toolboxes are nowadays often addressed in academic and commercial projects. They tend to fall into one of two categories. One is the design of dedicated context-aware applications such as GUIDE [7] or ubiCicero [11] which offer solutions to contextualise information for users according to their location (e.g. in museums). The other is rapid prototyping toolboxes for activity-based applications, such as those proposed by Li and Landay, Bannach et al. and Ghiani et al. [14, 2, 12]. These toolboxes offer generic elements to create a more specific application, and often include predefined algorithms to help generate the applications.

With the uMove framework, we propose a way to represent and manage a complex system made of different kinds of entities taking into consideration their location, activity and situation and on top of which a variety of different types of individual applications can be plugged. Unlike [7, 11] who focus on only specific applications or prototypes, uMove allows programers to easily define all the entities, the relations between them and the connected sensors, and to load the activity and situation recognition modules. However, uMove does not provide, like in [2], the activity (tasks) recognition modules or algorithms but allows them to be separately developed and connected to the entities active in the system. It is also not dedicated, like in [7] or [11], to one kind of application but can be considered as the basic and independent system to which applications are connected (like using a shared database). For instance, a university campus with buildings, rooms, students and professors can be represented with uMove, and different applications such as a tracking system for safety, an activity-based smart alert for students or phone call transfer can be connected to the same system, receiving data from different types of sensors and applying specific activity recognition algorithms.

3.2 Conceptual framework

The conceptual framework is the tool used by developers to theoretically design the system that will be observed. As shown in Fig. 6, a KUI system has three layers responsible for the different objects interacting together.

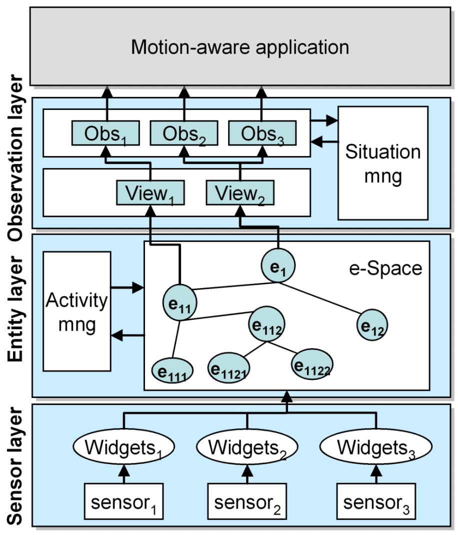

Figure 6: uMove framework

3.2.1 Sensor Layer

The sensor layer contains the *widgets* which are the logical abstractions of the sensors connected to the system. A widget is the generic interface to a category of sensors. For instance, an application tracking the movement of users within a building may need to connect location sensors, whatever type they are. The location widget connects the sensor to the system and provides the entity location to the higher level. The advantage of the widget is that if the type of sensor technology changes (e.g. RFID versus Bluetooth or wifi), the widget remains the same: it will still provide an entity location to the entity layer. Only the interface (the driver) between the physical sensor and widget needs to be adapted.

3.2.2 Entity Layer

In the *entity layer*, we find the logical representation of the physical entities (i.e. users, places, objects) being observed. Each entity (actor or place) is defined by its identity, its role, its location within the system and its current motion and activity. The entities are organised in an n-ary tree and they all have a parent node except for the root of the system (e.g. the world or the building). Entities get their locations, motions, activities and other contexts updated from the connected widgets. Each entity is attached to an activity manager which aggregates the data received from the different widgets and determines the current activity.

3.2.3 Observer Layer

The *observation layer* analyses the current situation of the entity based on the activity and contexts. The observers are the last filter between the entity and the application. They listen for any entity changes and forward them to the situation manager in order to have the new situation analysed. Depending to the result received from the situation manager, the observer forward or not a message to the application (e.g. a ”warning” message or ”critical situation” message).

The uMove model allows developers to concentrate on the application, the specific activity and situation algorithms without worrying about the communication between sensors (widgets), users or objects (entities) and their management (creation, removal, modification).

3.3 uMove API

Once the system has been designed, the developer uses the uMove Java API to create the different objects of the system in a simple manner as shown in the following example.

```
KUISystem kuiSystem = new KUISystem();
Entity room403 = kuiSystem.createNewZoneEntity(
           "A403", "Meeting Room", floor4,
           null, room403Area, robinActivityManager);
kuiSystem.attachEntityToLocationWidget(
           locationWidget, room403);
```

The system is represented by an object (`KuiSystem`) which links the other objects described above (Entity, observer, view, ActivityManager). `KuiSystem` offers the necessary methods to access the system, to connect the sensors and the applications and also launches all threads when the system is started. The communication between objects is based on the message listener concept. For instance, the observer listens to the entities and automatically receives all messages when any entity changes occur. The same concept is applied between an entity and its connected sensors. This type of asynchronous communication allows the system to be dynamic, possibly distributed, and guarantee that all object processes run in parallel.

4. THE IWAT EVALUATION METHOD

Literature in the area of evaluation of pervasive systems shows a trend towards using low-fidelity prototypes to both gather user requirements and to evaluate potential system designs. Some recent methods are particularly innovative and encourage the evaluation of designs as soon as possible in order to maximize the amount of information gathered and minimize implementation overhead. For instance, Abowd et al. [1] used paratypes (situated experience prototypes) to test models of interaction in real situations. While standard methods are conducive to testing functional prototypes in lab experiments, paratypes allow testing of design concepts in real life.

In order to create prototypes for the types of testing mentioned above, visual programming tools such as those proposed by Bannach et al. [2] have been suggested. Most evaluations of these tools are concerned with their usability and aim at measuring the effectiveness of a tool to support the prototyping phase. However, most of these prototyping tools favor rapid development of prototypes rather than supporting the design of the application itself, and for this purpose developers need to understand the design challenges. To fill the gap between the worlds of end users and developers Reilly et al. [16] suggest encouraging designers to immerse themselves in the target population’s current practices to gain a better understanding of user needs and thus be more effective in the design of the pervasive system itself.

User evaluations of designed products and prototypes are generally very useful to help understand a user’s expectations and requirements for a certain task, to measure the usefulness of a pervasive system to support a task, or to evaluate the usability of its interface. However, even though the product or prototype might use various hidden distributed modules, which is often the case in pervasive computing, users only perceive the tip of the iceberg since they interact with a single front end. As such, the pervasive system can be evaluated only as a whole, making it difficult to deduce an error or problem in the functional design of some of its modules.

We argue that once user requirements have been translated into a system design, an evaluation of that design at a functional level, and at a point before actual implementation has begun, can greatly help to alleviate this problem. This type of functional evaluation can help answer questions such as: Is the architecture of the system well designed and robust? Do the individual modules allow for the necessary behaviours? Do the modules communicate with each other as expected? Does the global behavior of the pervasive system meet technical and end-use requirements?

The IWaT (Interactive Walk-through) evaluation method was designed to fill this gap and was inspired by the family of walkthrough methods from User Centered Design (UCD). The method can be used to test the design and components architecture of a pervasive application to ensure that the various algorithms, strategies, inferences (of activities or context) and measurements (for example from sensors) chosen by the designers/developers operate together smoothly and form a coherent and comprehensive system. This is particularly critical in pervasive computing systems where the components, although distributed and independent, are often developed by different teams who may have limited contact with one another once development work begins.

IWaT is intended to complement existing UCD methods by allowing results from UCD studies to be quickly incorporated into a new or existing design and tested for feasibility and appropriateness before any time has been spent on implementation, which in turn can greatly speed up the design-implementation-evaluation loop. Implementation is often costly in terms of time and manpower and it is always difficult to modify code and/or the entire structure of the project if the design is scrutinized only through evaluation of an implemented system (even if this system is only an early prototype). IWaT is intended to be used between the design and implementation phases (Fig. 1) in order to reduce the risk of encountering design problems that are usually detected only during the prototype or system evaluation phases.

Moreover, we believe that IWaT is a novel and interesting method which fosters team work and favours the exchange of information and ideas between development teams. It makes the different teams more aware of the impact of their own designs and ideas on the work of other teams, and gives developers of individual components a more holistic view of the system and the exact role that their component plays in it. Combined with KUI modelling and the uMove framework, IWaT can reduce the time needed for the development and refinement processes.

4.1 How it works

IWaT requires two elements: 1) the conceptual models designed in the uMove framework and 2) the interactive components test.

4.1.1 Component models

In the development process, independently of the method used, there are steps that define the software architecture, possibly the design pattern to be used, algorithms that need to be developed and the technological and hardware choices. A uMove conceptual model helps to rapidly get the main functionalities such as the objects involved, the activity or situation algorithms (possibly in pseudo-code) and the sequence of operations. The model is important because 1) it gives an idea of how the components will behave and 2) it encourages reflection on the design [17]. However, the model itself is not sufficient to assure the validity of the design within the complete project and needs to be tested with the other component models.

4.1.2 Interactive component test

We now propose a method to test all of the different components together. Each team comes with the model (for instance the algorithms) they have developed. The goal is to create a physical interaction between the components. The developers become the "processors" and interpret their algorithm. For example, the team responsible for the mobile phone manually runs their application and sends paper-based messages to the team responsible for the back-end (server) application. Then these messages are interpreted by applying the back-end application algorithm (Fig. 9) in pseudo-code and possibly sending a message back to the mobile phone team. The log of the events is done on a board where a process sequence diagram is represented (Fig. 7).

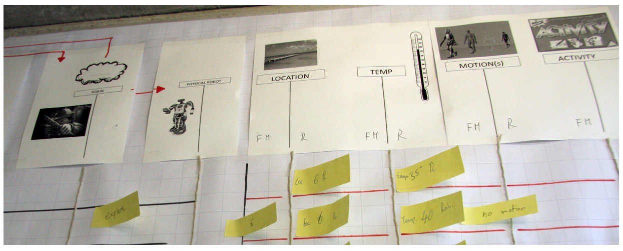

Figure 7: The events are logged on a sequence diagram board

The groups must prepare the environment in advance and define: The physical position of the components (the teams), the message flow between the components, the board for the sequence diagram of the different components and the initial state of the global application.

This method clearly shows the flow of information or messages between the components and quickly gives a good picture of how the project runs in general. In the next section we present a case study in which the KUI model, uMove and IWaT were applied together.

5. CASE STUDY: PROJECT ROBIN

The case study we used to evaluate the efficiency of the KUI and uMove tools and IWaT is a project developed in the scope of a master's course on Pervasive Intelligence given at the University of Fribourg. As part of the course project, the students had to develop a system where user activities and motion were taken as main input. The project used the KUI model to define the objects of the system, the uMove conceptual framework to plan the initial design, IWaT to validate the design and uMove API to implement it.

5.1 Robin: activity-based rescue staff safety management

The goal of the project was to develop an application able to detect the motion and activity of rescue staff such as firemen or policemen. Depending on their movements within a building, the system controls a robot sent ahead of the person or the team to gather information (such as the state of a room, temperature, fire or smoke) that might represent or signal a potential physical danger for the rescue team. For example, a fireman rescuing an injured person trapped in a room might need to be aware of where danger areas are. According to the robot's data, the system is able to transmit useful information to the fireman (fire alarm or smoke density for instance) in order to help them perform the appropriate rescue operation and to avoid a critical situations. The project was aimed at familiarising the students with pervasive technologies (sensors, programming framework) and pervasive concepts such as context-awareness, situation, activity-awareness, mobile computing and wireless communication. It also introduced the notions of implicit human-computer interaction and user-centered design.

5.2 The prototype

The project was developed in JAVA using uMove as the core application, SunSPOTs[1] as sensors for the motion and temperature data and LEGO Mindstorm NTX[2] for the robot. The general design and decomposition of the application into components was done together by all the students. Each student actively participated in the definition of the project needs and proposed solutions. Then, two groups were created and each one had specific components to design and develop. The first group was responsible for developing the robot, the motion detection and the activity detection classes (Fig. 8). The second group was in charge of the observers, views, the situation management and the Robin application including the robot control and feedback sent to the mobile device carried by the fireman.

The two groups had to first work on the interfaces between the different components. For instance, they defined the type of interaction between the Robin application and the robot or the type of motions processed in order to derive the activities. Then each group worked on the algorithms for the motion detection, activity detection and situation analysis. Once ready, an IWaT session was organised to test their work.

[1]http://www.sunspotworld.com/
[2]http://mindstorms.lego.com/Products/Default.aspx

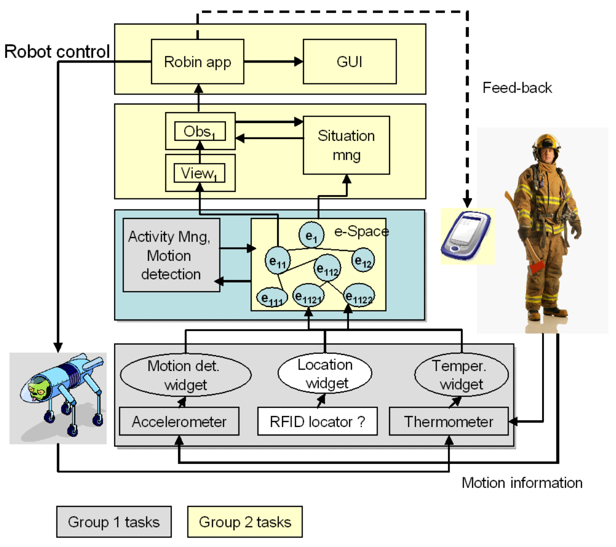

Figure 8: Robin project architecture and group tasks assignment

5.3 IWaT session

The goal of the session was 1) to test the preliminary algorithms of the robot, motion recognition, activity and situation detection, i.e. identify the possible deadlocks and 2) to validate the general design and decomposition of the project before starting the implementation. The general scenario to be tested was the situation where a fireman and a robot are searching for heat sources on a floor that is possibly on fire. The robot always has to be in front of the fireman and send temperature readings to the Robin application in order to inform the fireman of potentially dangerous situations.

First, the environment and the number of required participants, the physical distribution of the participants (Fig. 9) and the sequence diagram board representing the different components of the system was defined. The evaluation involved a total of 8 people: 2 students for the motion and activity recognition, 2 students for the situation manager and Robin application, 1 student for the Robot, 2 assistants for the uMove components (entity and observer), 1 assistant for the sequence diagram board management.

Figure 9: Students applying their algorithm during the IWaT session

All participants where grouped per component around a table. The distribution of the components depended on their inter-communication. For instance, the observer was next to the situation manager and next to the Robin application in order to facilitate physical message passing. The sequence board was hidden from the participants and the virtual robot represented on a floor map was also hidden. No discussion was allowed during the run of the scenario. The idea was to put the participants in the exact situation of their component and avoid human interpretation bias. Only the algorithms were interpreted. The initial situation was “the fireman and the robot reach the floor and start to search. The robot is still close to the fireman and no heat source is detected. The fireman knows the layout of the floor.”

A sequence was considered to be the complete processing of a message. For instance, when footsteps were detected, an event (message) was generated and sent to the entity which sent it to the activity manager and waited for an answer. The answer was forwarded to the observer and then to the situation manager. Finally the situation manager processed the activity and the Robin application produced the actual detected situation level (e.g. normal or critical) and sent a new command to the robot and feedback (if needed) to the fireman. At that moment, the next sequence began. During a sequence, each group manually applied their algorithm(s), processed the input message and sent the result to the next component.

5.4 Results

The scenario was played for about an hour and about 20 sequences were completed. The session revealed some important points that would have to be modified and/or adjusted in the project. The students highlighted the following sources of problems undiscovered during the design phase: 1) Some situations could not be analysed because the activity was not defined properly, 2) the motion detection algorithm was insufficient to detect proper movements, 3) the robot was not autonomous enough and did not give enough feedback on its location, 4) the fireman is delayed by the robot, who got stuck quickly.

During the debriefing, the students talked about the general behaviour of the application, and for instance, the idea of removing or replacing the robot was raised. They also naturally considered the decomposition of the application and fine tuned the type of input and output each component must receive and provide.

From the method evaluation point of view, we noticed that the overall student experience was good and the discussions following the session showed the motivation of the groups to interact and exchange information in order to adjust the different components. It also allowed to note major problems and bugs and possibly reconsider the pertinence of some components. The most important point is that this method made possible an important test before starting the concrete implementation of the project.

6. CONCLUSION

In this paper we addressed the problem of designing and evaluating pervasive computing systems in the early development phases. These types of applications require an integration of different heterogeneous technologies, often distributed and it is difficult to evaluate the validity of the technical choices as well as the project decomposition. We proposed a new approach for modelling, validating and prototyping pervasive applications. Our approach consists of a KUI model [4] which allows developers to define the components (users, environment, activities and situations) of the future application, a JAVA API (uMove) for the prototyping and a method (IWaT) to test the validity of choices such as conceptual decomposition or technologies at an early stage. We also illustrated the use of this approach by presenting a case study applying the tools and method and we have in particular analysed the impact of the IWaT method on the design of the application.

The case study raised some points about the KUI model, the uMove framework and the IWaT method that need to be addressed in future work. In the KUI model, we have found that there exists a conceptual ambiguity in the definition of the actor, the place and the role they can have in the system. In the uMove conceptual method, we need to process the propagation of the context changes in an more defined manner, e.g. the change of temperature in a room might affect the temperature of the actors and objects present in the room. Lastly, we consider that the IWaT method is at an early stage of definition and we need to now formally define it and test it with other projects such as Hestia [5].

7. ACKNOWLEDGMENTS

We would like to thank Daniel Ostojic for his contribution to the IWaT method and the session we organised to evaluate the Robin project. Many thanks also to the students of the Pervasive Intelligence course, Benjamin Hadorn, Adriana Wilde, Fouad Murr and Tom Forrer, as well as our colleagues Amos Brocco and Apostolos Malatras for their active participation during the evaluation session.

This work is supported by the Swiss National Fund for Scientific Research Grant n.116355

8. REFERENCES

[1] G. Abowd, G. Hayes, G. Iachello, J. Kientz, S. Patel, M. Stevens, and K. Truong. Prototypes and paratypes: designing mobile and ubiquitous computing applications. *Pervasive Computing*, 4(4):67–73, Oct.-Dec. 2005.

[2] D. Bannach, P. Lukowicz, and O. Amft. Rapid prototyping of activity recognition applications. *Pervasive Computing*, pages 22–31, April-June 2008.

[3] A. Bouvier. *Management et projet*. Hachette, Paris, 1994.

[4] P. Bruegger and B. Hirsbrunner. Kinetic user interface: Interaction through motion for pervasive computing systems. In *5th International Conference, UAHCI 2009, Part of HCI International 2009*. Springer, 2009.

[5] P. Brugger, V. Pallotta, and B. Hirsbrunner. Optimizing heating systems management using an activity-based pervasive application. *JDIM - Journal of Digital Information Management*, (ISSN 0972-7272), 2009. [To Appear].

[6] S. Carter and J. Mankoff. Prototypes in the wild lessons from three ubicomp systems. *Pervasive Computing*, 4(4):15–17, Oct-Dec 2005.

[7] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA, 2000. ACM.

[8] A. Dey, E.D Abowd, and G.D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction Journal*, Vol 16:pages 97–166, 2001. Anchor article of a special issue on Context-Aware Computing.

[9] P. Dourish. *Where the Action Is: The Foundations of Embodied Interaction*. MIT Press, Cambridge, 2001.

[10] D. Fitton, C. Cheverst, C. Kray, A. Dix, M. Rouncefield, and G. Salsis-Lagoudakis. Rapid prototyping and user-centered design of interactive display-based systems. *Pervasive Computing*, 4(4):58–66, Oct-Dec. 2005.

[11] G. Ghiani, F. Patterno, C. Santoro, and D. Spano. A location-aware guide based on active rfids in multi-device environments. CADUI 08, Spain, 2008.

[12] G. Ghiani, F. Patterno, and D. Spano. Cicero designer: an environment for end-user development of multi-device museum guides. IS-EUD '09, Germany, 2009.

[13] K. Kuutti. *Activity Theory as a Potential Framework for Human-Computer Interaction Research*. MIT Press, 1996.

[14] Y. Li and J. A. Landay. Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1303–1312, New York, NY, USA, 2008. ACM.

[15] S. W. Loke. Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective. *The Knowledge Engineering Review*, pages 213 – 233, 2004.

[16] D. Reilly, D. Dearman, M. Welsman-Dinelle, and K. Inkpen. Evaluating early prototypes in context: trade-offs, challenges, and successes. *Pervasive Computing*, 4(4):42–50, Oct.-Dec. 2005.

[17] D. Schön. *The Reflective Practioner: How Professionals Think in Action*. Basic Book, New York, 1983.

[18] L. von Bertalanffy. *General System Theory. Foundations, Development, applications*. George Braziller, 1969.

[19] M. Weiser. The computer for the 21st century. *Scientific American 265*, Vol 3:pages 94–104, September 1991.

[20] M. Weiser and J.S. Brown. The coming age of calm technology [1], consulted: April 2007. http://www.cs.ucsb.edu/ ebeld-ing/courses/284/w04/papers/calm.pdf.