[2] Journal of Mobile Multimedia, Vol. 0, No. 0 (2010) 000–000
© Rinton Press

Camera Ready

ENRICHING THE DESIGN AND PROTOTYPING LOOP: A SET OF TOOLS TO SUPPORT THE CREATION OF ACTIVITY-BASED PERVASIVE APPLICATIONS

PASCAL BRUEGGER
pascal.bruegger@unifr.ch

AGNES LISOWSKA
agnes.lisowska@unifr.ch

DENIS LALANNE
denis.lalanne@unifr.ch

BEAT HIRSBRUNNER
beat.hirsbrunner@unifr.ch

Department of Informatics, University of Fribourg, Perolle 90
1700 Fribourg, Switzerland

Received (received date)
Revised (revised date)

This paper proposes a complete approach to support the modelling, testing and prototyping of pervasive 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. For each step of the development life-cycle (design, prototyping and evaluation), we position the proposed approach with respect to related tools and methods. We also present a pervasive application implemented and evaluated using the proposed approach and tools, which facilitates further discussion of the benefits and disadvantages of using the proposed framework. We conclude with propositions for improvements to our comprehensive modelling, prototyping and testing framework for pervasive applications.

Keywords: Mobile computing, Pervasive Computing, Activity-based computing, User Centered Design
Communicated by: to be filled by the Editorial

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 in order 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, 2, 3, 4]. In order to support this type of requirements elicitation and evaluation, toolkits such as [5, 6] 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 into 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 the related uMove conceptual framework enable 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 the end-user perspective once a prototype is available.

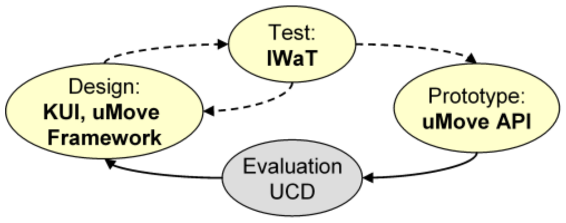

Fig. 1. Project development phases and tools

In the rest of this paper, we will focus on describing the KUI semantic model (section 2), the uMove conceptual model, will discuss the uMove API (section 3) and the IWaT validation method (section 4), . 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) [7] is a way of grouping Weiser's Ubiquitous Computing vision [8] and Dourish's Embodied Interaction vision [9]. In the early '90s, Marc Weiser predicted that computers would disappear and computing power would fade into 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 which can help in the design of pervasive systems including mobile applications or server-based systems integrating a user's locations, activities and other contexts.

2.1 A systemic approach

The KUI model is based on General System Theory (GST). GST defined by von Bertalanffy [10] gives the framework and the concepts to model specific systems studied in sciences such as biology or chemistry. There exist different types of systems such as inert or dead, living or evolutionary, open (exchanging matter with its environment) or closed. Systems can be made of sub-systems, for instance the earth being an extremely complex living system and also being one component of the solar system (hololic concept). 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 ([11], 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, everything is an entity within a system. Systems are open and dynamic (living). 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. Usually a system such as a cell in biology or a solar system in astronomy are observed from a certain point of view. The observer and the point of view are not included in the system itself. We consider the *observer* (who/what is observing the system) and the *viewer* (the observer's point of view) as part of the system because conceptually if a system is not observable, it does not exist.

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: actors, observers and viewers* (Figure 2).

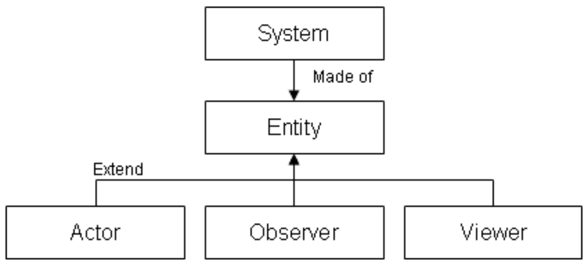

Fig. 2. System diagram

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 Actors

Actors 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 [12] define a context as any information that can be used to characterise the situation of an actor. 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 actor within the system.
- Location. It gives the physical and logical position of the actor.
- Geometry. An *Actor* can be considered as a point or an atom (e.g. human, robot, object) or a place (e.g. rooms, house, boat). A place has a geometrical dimension (polygonal, circular).
- Status. An entity has two possible statuses: mobile (motion capabilities) or static (fixed).
- Structure. It defines the contained actor.
- Relations. Two types of relations between actors: 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 actor 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 viewers

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 [13], 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

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.

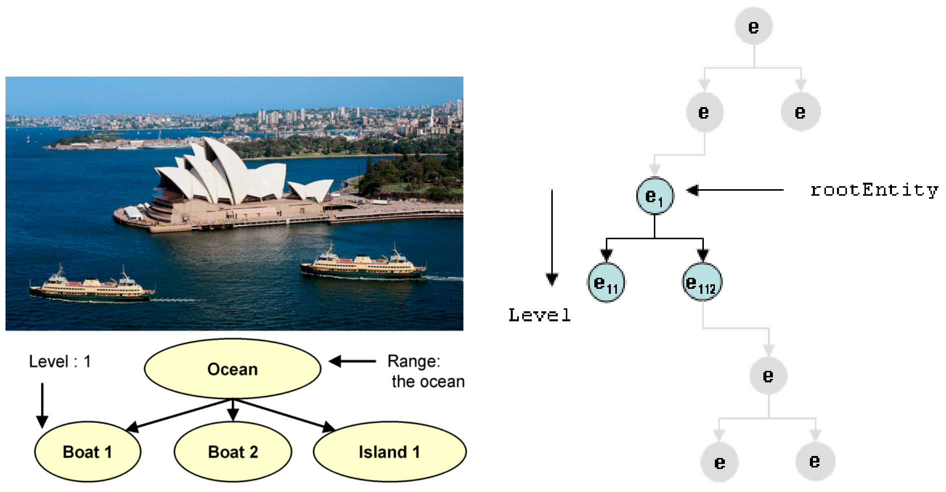

Fig. 3. Concept of view in KUI: Range Level parameters and partial view of an entity tree

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). A viewer set with a local root entity (actor or zone) and a level allows the observation of a part of the entity tree.

2.4 An activity-based model for situation awareness

We now define how the kinetic information from the different actors 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. For instance, the activity of running in high temperature conditions and along a road with heavy traffic will lead to a more dangerous situation for the actor (e.g. physical and security conditions) than doing the same activity on a quiet track in a temperate forest.

2.4.1 Situations

We have noted two points of view for describing situations. In [6], 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 [14], 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* (Figure 4).

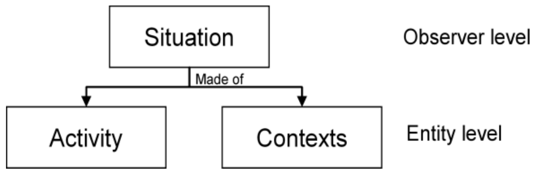

Fig. 4. Situation model in KUI

2.4.2 Activity

For Loke [14], activity typically refers to actions or operations undertaken by human beings such as "cooking", "running", or "reading". For Y. Li and J. Landay [6], an action like "running" is not considered as an activity because it focuses on an immediate goal. For Kuutti, in [15], 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 (Figure 5 a), 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 which allows the definition and imple-

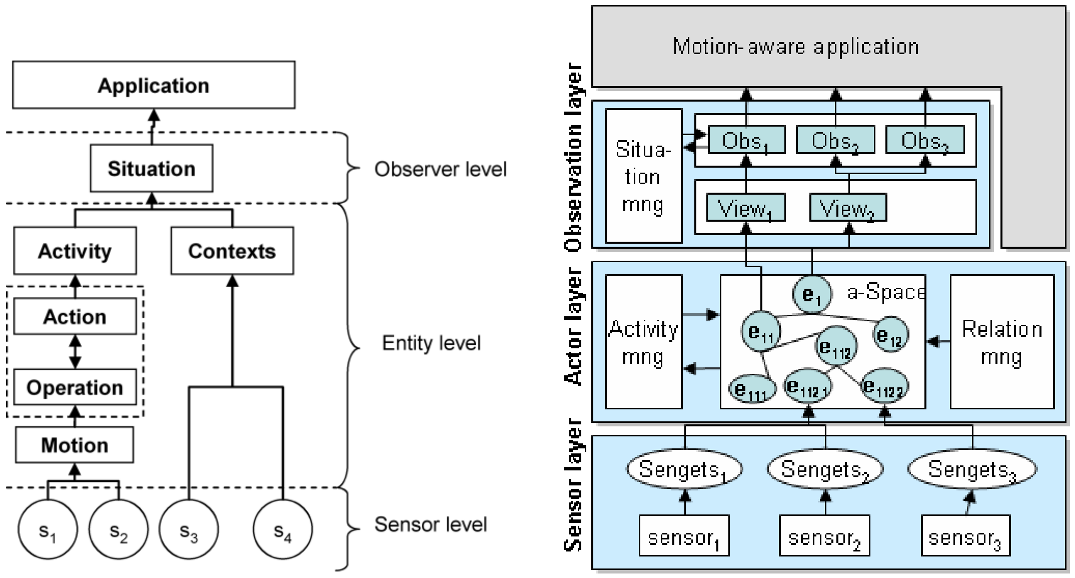

Fig. 5. a) Data flow from motion to situation in KUI, b) uMove framework

mentation of 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.

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. the first is the design of dedicated context-aware applications such as GUIDE [16] or ubiCicero [17] 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 [6], Bannach et al. [5] and Ghiani et al. [18]. These toolboxes offer generic elements to create a more specific application, and often include pre-defined 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 [16, 17] who focus only on specific applications or prototypes, uMove allows programers to easily create all entities, the relations between them and the connected sensors, and to load the activity and situation recognition modules (algorithm or classes). However, uMove does not provide, as [5] does, the activity (task) recognition modules or algorithms. Instead, it allows them to be separately developed and connected to the entities active in the system. It is also not dedicated, like [16] or [17] are, to one kind of application but can be considered as basic and independent system to which any application can be 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 Figure 5 b), a KUI system has three layers responsible for the different objects interacting together.

3.2.1 Sensor Layer

The sensor layer contains the *sengets*[a]which are the logical abstractions of the sensors connected to the system. For instance, an application tracking the movement of users within a building may need to connect location sensors, independant of their type. The location senget connects the sensor to the system and provides the entity location to the higher level. The advantage of the senget is that even if the type of sensor technology changes (e.g. RFID versus Bluetooth or wifi), the senget remains the same: it will still provide an entity location to the entity layer. Only the interface (the driver) between the physical sensor and senget 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 sengets. Each entity is attached to an activity manager which aggregates the data received from the different sengets and determines the current activity.

3.2.3 Observation 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 forwards 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 (sengets), users or objects (entities) and their management (creation, removal, modification).

3.3 uMove API

After the design of the system with the uMove conceptual framework, the developers use the uMove Java API to set up the different objects. The uMove library offers all the necessary classes to create and manage the entities active in the system. We will now describe the main classes used during the implementation phase.

[a]stands for sensor gadget similar to the concepts of widget (Windows gadget) or phidget (physical gadget)

Any system is created by the instantiation of the main object called `KuiSystem`. It encapsulates all the methods allowing the creation of the entities (actors, observers, viewers) as shown in the following example.

```
KUISystem kS = new KUISystem();
Actor zone0 = kS.createPhysicalZone("Gruyere","Rgion de Gruyere", IDTags, null, null,
new Coordinates(575000, 165000, 800, CoordinatesFormatEnum.CH1903), eGeom, null);

Viewer v1 = kS.createViewer(zone0, 3);
Observer o1 = kS.createObserver(v1, null, new ObserverMessageProcessor());
```

Listing 1 Code sample of KuiSystem instantiation

The management of the entities is done through `KuiSystem` and this object is the ”handle” of the system. For instance, `KuiSystem` connects the sensors and the applications and also launches all threads when the system is started.

3.3.1 Types of entities

After the creation of the system, the programmers instantiate its structure. There exist three types of actor objects in the system which are used to do this: zones, actors and groups.

Zones. Zones are typically places (building, rooms) or objects (cars, boats) and have a dimension or a geometry. They can be physical or logical, mobile or static. A zone is defined by giving a location point called ”point zero” and either a set of vectors if it is a polygonal zone or a radius if it is a circular zone (Listing 1). A zone must be located in a parent zone. If the zone is the root of the system, its parent is considered as the ”universe” and the system manages it as such. For instance, a system modeling a building will have the building with its physical dimension as the root zone. This zone ”building” will be made of floors and floors will contain sub-zones (offices). The result will be a tree of zones representing the structure of the building.

Actors. Actors represent the physical users or the mobile objects of the system. The actors are considered as atomic entities located in a point with no dimension and no geometry. They move in the tree of physical or logical zones.

Groups. Groups are special actors which have no logic and no location but contain other actors. Users can belong to different groups, for example a student group or a teacher group. Groups are a simple way to manage relations between actors and help determine situations at the observer level.

3.3.2 Observers and viewers

Observers are the interface objects between the applications and the actor structure (Listing 1). They provide information about 1) any contextual changes in the system and 2) the current situations of actors. Each observer receives all events from the actors it observes and filters the events before sending them to the applications according the rules defined by the programmer. The processing of the events is described in detail in 3.3.4.

The actors are observed through viewers (cf. 2.3.2). Viewers are the objects that represent the tree of actors (zones and actors) from a local root (a zone) and down to a given level

(Listing 1). The observer has a partial knowledge of the system and receives only the events of actors contained in the structure of the local root zone (Figure 3).

The view on the system can be changed by modifying the two parameters `Root` and `Level`.

3.3.3 Sensors

In a KUI system, the contexts are acquired through different sensors connected to actors. The main sensors considered in the different prototypes developed so far were location sensors such as GPS, RFID or Bluetooth. However, any kind of sensor can be connected (temperature, light intensity, accelerometer). The driver of the physical sensor must implement the interface `Sensor` which provides `attachSenget()` and `detachSenget()`.

The sensors are connected to the actor through a `Senget` object. The senget is the interface between the sensor driver sending the raw data such as a GPS coordinate, an RFID tag or the 3-axis acceleration value, and the actor object which will store the integrated values in its context object. The main purpose of the senget is to allow the programmers to:

- filter and integrate data such as acceleration or temperature values,
- retrieve, for instance, an actor object from a tag (RFID) or a coordinate received from a location sensor. The senget will process the received tag and location and send a message to the concerned actor with 2 parameters: the previous location (zone object) at which it was located and the new location.

Each instance of senget is connected to a sensor class and processes specific data with its message processor as described in the next section.

3.3.4 Message, activity and situation processors

One advantage of the uMove architecture is that the processing logic of each component has been separated from the objects (sengets, actors, observers, viewers).

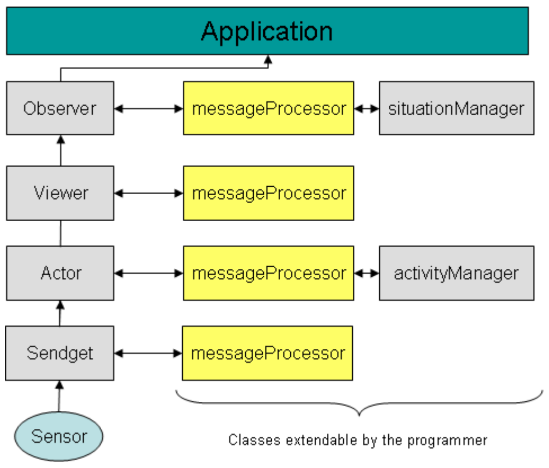

Fig. 6. uMove: message and activity/situation processing architecture

Message processor. As shown in Figure 6, each type of system object has a separated message processor. This message processor allows to specify and implement the algorithm that will handle each message coming from the object below. For instance, an actor receives messages from the different sengets. Each senget is interfacing and sending a certain type of data or context (location, temperature, acceleration). The following is an example of an algorithm processing messages received at the actor level. In this algorithm, the type of messages considered (`KUIMessage`) are location changes and context changes (temperature, light, acceleration) coming from sengets and location changes coming from a child actor (sent to its previous and new physical parent).

```
public class ActorMessageProcessor implements MessageProcessor {

private ActivityManager activityManager;
...
public ActorMessageProcessor(ActivityManager activityManager) {
    this.activityManager = activityManager;
}

public IMessage processMessage(IMessage message, AbstractRunnableEntity entity) {

    IMessage pRetval = null;

    // Checks if the message is a KUIMessage
    if (message instanceof KUIMessage && entity instanceof Actor) {

        KUIMessage m = (KUIMessage) message;

        if (m.getKeyType().equals(KUIMessageKeyType.SENSOR_LOCATION_CHANGED)) {
        ...
        } else if (m.getKeyType().equals(KUIMessageKeyType.ACTOR_LOCATION_CHANGED)) {
            pRetval = sendActorStructureChangedMessage(m, entity);
        } else if (m.getKeyType().equals(KUIMessageKeyType.SENSOR_CONTEXT_CHANGED)) {
            pRetval = sendContextChangedMessage(m, entity);
        }
    }
    return pRetval;
}
```

Listing 2 Code sample of an Actor Message Processor algorithm

uMove provides standard message processor classes for each object. For the senget, the location message processor is the main processor used as it is mandatory for the consistency of the system tree to have each actor located somewhere at all times. This processor handles different kinds of location data such as coordinates (GPS) and zone and actors tags (bluetooth, RFID tag). From this low level information, it sends the corresponding zone objects to the actor.

Each message processor is extendable by programmers using uMove. They can simply extend or replace the class `MessageProcessor` with their own algorithm:

```
Observer o1 = kS.createObserver(v1, null, new MyObserverMessageProcessor());
```

Listing 3 Creation of an observer with a specific Message processor

Activity and situation managers. There are two special classes at the actor and observer levels (cf. 6) which can be developed in order to process the current actor activity and its situation (cf. 2.4). Those two classes can be ignored if the system does not consider activities or situations (user location tracking only for instance). But, if uMove is used to prototype applications in research projects specifically treating a user's activity or modeling a user's situation, it gives the advantage of having a running system where activity or situation management classes can be plugged in on the fly to test different algorithms. Those classes must implement the `ActivityManager` and `SituationManager` interfaces and develop the methods `checkActivity()` and `checkSituation()`. As shown in Figure 6, the classes are attached to the message processor and receive messages from it and not directly from the entities.

Relation manager. As described in 2.2.1, relations are important contexts that are considered at the situation management level. This aspect is managed in uMove by providing a relation manager object which is automatically instantiated when the system is started. It is based on the singleton design pattern [19] and returns the current relations of any actor. The relation manager provides methods such as `getInsidePhysicalRelationOf(actor)` and it allows the situation manager to query the relations according to the activity and rules applied to the zone where the actor stands. This type of querying becomes an on "demand" process. An advantage of this concept is that relations, being dynamic by nature, are not stored within the actor. If they were stored within the actor, they would need to continuously check the actor state and those of all other actors and zones it is in relation with, which would result in large quantities of needless calculations.

3.3.5 Inter-objects communication

The communication between uMove objects is based on the message listener concept. For instance, an observer listening to an actor automatically receives all messages when any actor change occurs. The same concept is applied between an actor 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. Each object (senget, actor, observer) is connected to a port object which is dedicated to listening to a channel and playing the role of new message sender or receiver (Figure 7).

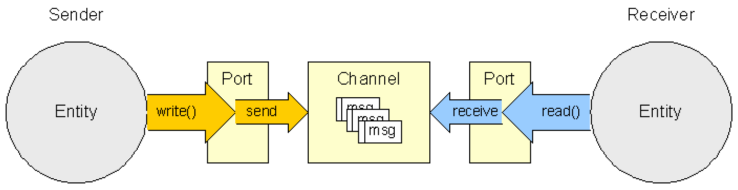

Fig. 7. uMove: port communication between objects

3.3.6 Attaching an application to a KUI system

Once the KUI system is set up and running, programmers can attach applications on top of it in order to process high level information. An example of a developed application is a context-aware service provider. In this application, a building is equipped with a server-based KUI system and users are tracked using their Android phones and Bluetooth-based location management. The mobile phone also runs uMove and transmits context changes such as location, acceleration and activity to the server. The server application provides the user with contextual services chosen according to a set of rules. For instance, if the user is passing by the cafeteria, he/she may or may not receive the menu on their phone, depending on their speed and agenda. In this case, the rules applied would typically be: `if speed > 2 m/s and activity == meeting in 5 min -> don't send menu text`.

A KUI-enabled application must implement an interface using the concept of a message listener (provided with uMove) also based on the observer design pattern [19]. The application is attached to the observer(s) instantiated in the KUI system. This configuration allows to have one application attached to one or more observers or one observer attached to one or more applications. In the case study described in section 5, the Robin application is divided into two specific parts (Figure 9): 1) the tracker which listens to the observer and 2) the graphical user interface (GUI) which receives the information to show from the tracker.

The framework that we have presented not only provides a way of designing a pervasive system but also a programming library which simplifies implementation of the KUI concepts. In the next section, we will see how to evaluate the design of the system before starting to completely implement the different modules and algorithms.

4 The IWaT Evaluation Method

Literature in the area of evaluation of pervasive systems shows a trend towards evaluation at one of two abstraction levels: low and high. At the low-level is the components evaluation, where the individual components of a pervasive system are evaluated to ensure that they are functioning according to design specifications and that they are free of bugs. This type of evaluation is obviously necessary to ensure that each component in the system performs its function correctly and generally follows standard software engineering test methods.

At the higher level is evaluation using prototypes (low and high fidelity) to both evaluate potential system designs and to gather additional user requirements in order to improve the design. Some researchers such as Abowd et al. [1] encourage the evaluation of designs as soon as possible in order to maximize the amount of information gathered and minimize implementation overhead. For example, they used paratypes (situated experience prototypes) to test models of interaction in real life situations rather than in laboratory settings. Those wishing to test a functioning prototype but who face limitations on the amount of time spent on implementation or the infrastructure that is available for testing use Wizard of Oz prototypes [20, 6, 21, 4] and sensor simulation [12, 22, 23, 24] to complement missing or unavailable components. Others have used innovative methods such as immersive video [25, 26] to simulate changing and variant outdoor environments in a controlled lab setting, which allows for comparison of different variants of a single system or a comparison of different systems.

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 a 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 is evaluated only as a whole, making it difficult to deduce an error or problem in the functional design of some of its modules.

Both of the evaluation levels mentioned above are useful and necessary, but we argue that another level of evaluation should also be considered. 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 reduce the existence of errors in the overall functional design. 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 conceived to fill the functional evaluation 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 or developers operate together smoothly, satisfy user requirements, take into account technical and infrastructure limitations and form a coherent and comprehensive system. This type of evaluation is particularly critical in pervasive computing systems where the components are distributed and independent, and are often developed by different teams who may have limited or no 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 (Figure 1) in order to reduce the risk of encountering design problems that are usually detected only during the prototype or system evaluation phases. Moreover, it can be used at any iteration in the design process, although its use in early stages of design and development is the most fruitful.

4.1 How it works

The IWaT evaluation method assumes that each component of the pervasive system is being developed by a different team. Therefore, for the evaluation, each team comes with the model (for instance the algorithms) they have developed for their component. The goal of the evaluation is to create a physical interaction between the components where the developers become the "processors" and interpret their algorithms. For example, the team responsible for the mobile phone component 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 (Figure 10) in pseudo-code and possibly sending a message back to the mobile phone team. A log of the events is kept on a board where a process sequence diagram is represented (Figure 8).

The method clearly shows the flow of information or messages between the components and quickly gives a good picture of how the system runs in general.

An important aspect to consider when preparing an IWaT evaluation is how to prepare

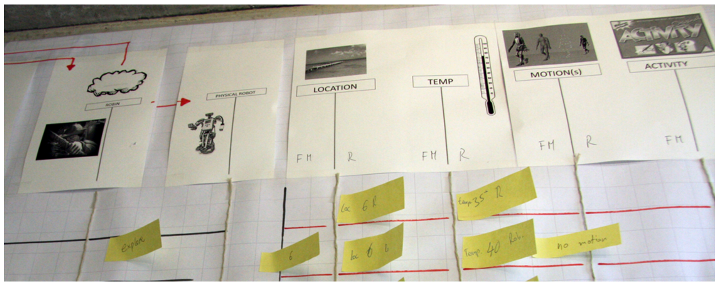

Fig. 8. The events are logged on a sequence diagram board

the evaluation environment. In particular, careful thought should be given to the physical distribution of the components (the teams) within the space, taking into consideration the message flow between the components since it will have a manifestation in the physical space. For example, two components that send messages to one another on a regular basis should not be placed in physically distant locations in the evaluation environment since the team members from those components will have to move on a regular basis as well. Moreover, thought must be given to what types of physical artefacts of the evaluation are necessary. For example, is a board for the sequence diagram of the interaction between the different components necessary? How can the initial and global states of the application be represented? Are extra people necessary to perform tasks such as updating the sequence diagram?

4.2 Using IWaT with KUI and uMove

The IWaT evaluation was conceived in relation to KUI and the uMove frameworks, and therefore easily fits into the process of designing pervasive system with these tools. In any development process, independently of the method used, there are steps that define the software architecture, possibly the design pattern to be used, the algorithms that need to be developed and the technological and hardware choices. A uMove conceptual model helps to rapidly get the main aspects and functionalities of a system, 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 [27]. However, a single model itself is not sufficient to assure the validity of its design within the whole system. It therefore needs to be tested together with the models of the other components. The IWaT evaluation method allows for just this type of testing. Once the evaluation has been completed and the conceptual model has either been validated or refined and retested the system design is ready to be implemented, and the implementation can be done directly within the uMove framework.

4.3 Advantages and drawbacks of using IWaT

As with any other evaluation methodology, there are both advantages and drawbacks to using IWaT to evaluate a pervasive system. Again, the overall goal of the method is to evaluate the general design of a system, and of its components, without having to implement it. IWaT enables evaluation of the design of a system, that its overall behavior works as expected, and that its components collaborate smoothly. A by-product of the IWaT methodology is that it encourages discussion and collaboration among the components' developers and as such favors team building. The method requires functional descriptions of the components, which can be simulated, and takes about a day to do - half a day to set up and another half to run the evaluation.

4.3.1 Disadvantages

In order to do a thorough evaluation, the whole system will need to be run, and as many test cases as possible will need to be taken into consideration. This implies three things. The first is that at least one member of each of the teams for all of the components needs to be present at the evaluation in order to play the 'role' of the component. It can be hard to arrange for a time when all of the teams can be represented. Moreover, depending on the size of the system and the number of teams involved, a sufficiently large physical space will need to be found in which the evaluation can take place. Second, the process can be slow since a human will be stepping through the algorithm and not a machine. For large and complex systems, this might mean that the evaluation will not be completed in just a few hours, but rather might require a whole day or more. Third, being able to accurately record the steps and artefacts of the evaluation might be difficult for large systems because the number of steps and cases required could become too large to note explicitly in a physical space. Related to this is the issue that the data that is recorded will need to be easy to understand, particularly given the potentially large quantity. These three factors imply a lot of overhead and careful planning which might not be feasible for some situations such as projects which have very short production times or which are being developed by large numbers of dispersed teams.

4.3.2 Advantages

Despite the disadvantages presented in the previous section, we believe that IWaT has several advantages which can outweigh the inconveniences in some situations. The first advantage is that this type of evaluation can save a lot of time and effort during later stages of development if it is done early enough in the system lifecycle. Since the evaluation primarily focuses on testing the inter-operability of the different components, it allows system designers to quickly and accurately pinpoint problems with the information flow within the system and to determine which components are involved in or are causing the problems. This is something that is hard to do when testing individual components, and is even harder to do when testing the system as a whole using prototypes and end-users if careful and detailed logging capabilities are not built into the system from the start. Having information about problem areas available before implementation begins allows designers to reconsider or appropriately modify their design before significant time and effort has been put into the development process. Once development has reached an advanced stage, most stakeholders in the system are very reluctant to make changes except when they are critical, which is understandable given the complexity of pervasive systems, but can also be detrimental to the overall usability and acceptance of the system by end-users once the system is launched. Moreover, this type of evaluation does not require any type of prototype, nor does it require any type of technical infrastructure, such as a wireless network to be in place, which means that it can be done at any time and in virtually any location without having to worry about network failures or other types of technical problems.

In addition to helping to improve and refine the system design and reduce implementation overhead, the IWaT evaluation method also have several benefits that can be felt at the social level. We believe that the nature of the method 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. This type of awareness can help improve communication between teams. Not only will the team members have met in person, which gives more context to their communication, but they will also have a better understanding of the role of the other team's component, and potentially of how they view their component in relation to other components. Within large companies this type of communication between teams, which sometimes never meet in person during the development of a system, can help create a healthier work environment.

For those who are new to the field, for example new employees at a company or even university students working on a project, the type of inter-team interaction proposed in the IWaT evaluation can help them get a better understanding of the roles of the different teams within the system design and to familiarize themselves more quickly with the design of pervasive systems. An example of just this case is presented in the next section, where a group of university students working on a group project to design and implement a pervasive system using the KUI and uMove tools and the IWaT evaluation method to validate their design.

5 Case study: project Robin

The case study we used to evaluate the efficiency of the KUI and uMove tools, as well as the IWaT methodology, is a project developed in the scope of a master's course on Pervasive Intelligence given at the University of Fribourg, Switzerland. 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 the 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 situation. 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[b] as sensors for the motion and temperature data and LEGO Mindstorm NTX[c] 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 (Figure 9). The second group was in charge of the observers, the views, the situation management and the Robin application including the robot control and feedback sent to the mobile device carried by the fireman.

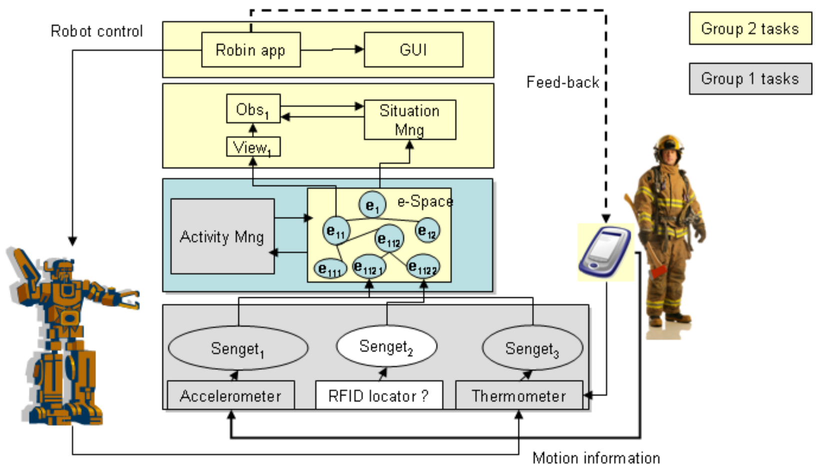

Fig. 9. Robin project architecture and group tasks assignment

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.

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 (Figure 10) and the sequence diagram board representing the different

[b]http://www.sunspotworld.com/
[c]http://mindstorms.lego.com/Products/Default.aspx

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) and 1 assistant for the sequence diagram board management.

Fig. 10. 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 1 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:

- Some situations could not be analysed because the activity was not defined properly;
- The motion detection algorithm was insufficient to detect proper movements;
- The robot was not autonomous enough and did not give enough feedback on its location;

- The fireman was 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 integration of different heterogeneous technologies, which are 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 modeling, validating and prototyping pervasive applications. Our approach consists of a KUI model [7] 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 found that there exists a naming ambiguity in the definition of the Actor as a type of entity (like Observer and Viewer) and the "zone" and "actor" as instances of the type Actor. We actually have to differentiate between them since a zone has a physical dimension but can also be called an "actor" as it can move within the system (e.g. boat, car). In the uMove framework, we also need to investigate the propagation of context changes between entities, e.g. how the change of temperature in a room might affect the temperature of the actors and objects present in the room. We will also propose a graphical editor and monitoring application in order to facilitate the development and maintenance of the KUI system. 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 [28].

Acknowledgements

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

1. G. Abowd, G. Hayes, G. Iachello, J. Kientz, S. Patel, M. Stevens, and K. Truong. (2005) Prototypes and paratypes: designing mobile and ubiquitous computing applications. *Pervasive Computing*, 4(4):67–73 [cited at p. 2, 13]
2. S. Carter and J. Mankoff. (2005) Prototypes in the wild lessons from three ubicomp systems. *Pervasive Computing*, 4(4):15–17 [cited at p. 2]
3. D. Fitton, C. Cheverst, C. Kray, A. Dix, M. Rouncefield, and G. Salsis-Lagoudakis. (2005) Rapid prototyping and user-centered design of interactive display-based systems. *Pervasive Computing*, 4(4):58–66 [cited at p. 2]
4. D. Reilly, D. Dearman, M. Welsman-Dinelle, and K. Inkpen. (2005) Evaluating early prototypes in context: trade-offs, challenges, and successes. *Pervasive Computing*, 4(4):42–50 [cited at p. 2, 13]
5. D. Bannach, P. Lukowicz, and O. Amft. (2008) Rapid prototyping of activity recognition applications. *Pervasive Computing*, pages 22–31 [cited at p. 2, 7]
6. Y. Li and J. A. Landay. (2008) 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, ACM. [cited at p. 2, 6, 7, 13]
7. P. Bruegger and B. Hirsbrunner. (2009) Kinetic user interface: Interaction through motion for pervasive computing systems. In *UAHCI '09: Proceedings of the 5th International on Conference-Universal Access in Human-Computer Interaction. Part II*, pages 297–306, Berlin, Heidelberg, Springer-Verlag. [cited at p. 2, 20]
8. M. Weiser. (1991) The computer for the 21st century. *Scientific American 265*, Vol 3:pages 94–104 [cited at p. 2]
9. P. Dourish. (2001) *Where the Action Is: The Foundations of Embodied Interaction.* MIT Press, Cambridge [cited at p. 2]
10. L. von Bertalanffy. (1969) *General System Theory. Foundations, Development, applications.* George Braziller [cited at p. 3]
11. A. Bouvier. (1994) *Management et projet.* Hachette, Paris [cited at p. 3]
12. A. Dey, E.D Abowd, and G.D. Salber. (2001) A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction Journal*, Vol 16:pages 97–166 Anchor article of a special issue on Context-Aware Computing. [cited at p. 4, 13]
13. M. Weiser and J.S. Brown. (1991) The coming age of calm technology [1], consulted: April 2007. http://www.cs.ucsb.edu/ebelding/courses/284/w04/papers/calm.pdf. [cited at p. 5]
14. S. W. Loke. (2004) Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective. *The Knowledge Engineering Review*, pages 213 – 233 [cited at p. 6]
15. K. Kuutti. (1996) *Activity Theory as a Potential Framework for Human-Computer Interaction Research.* MIT Press [cited at p. 6]
16. K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. (2000) 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, ACM. [cited at p. 7]
17. G. Ghiani, F. Patterno, C. Santoro, and D. Spano. (2008) A location-aware guide based on active rfids in multi-device environments. CADUI 08, Spain [cited at p. 7]
18. G. Ghiani, F. Patterno, and D. Spano. (2009) Cicero designer: an environment for end-user development of multi-device museum guides. IS-EUD '09, Germany [cited at p. 7]
19. E. Gamma, H. Richard, R. Johnson, and J. Vlissides. (1995) *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley [cited at p. 12, 13]
20. S. Consolvo, B. Harrison, I. Smith, M. Chen, C. Everitt, J. Froehlich, and J.A. Landay. (2007) Conducting in situ evaluations for and with ubiquitous computing technologies. *International Journal of Human-computer Interaction*, 1(22):107122 Lawrence Erlbaum Associates, Inc [cited at p. 13]
21. S. Dow, J. Lee, C. Oezbek, B. MacIntyre, J. D. Bolter, and M. Gandy. (2005) Wizard of oz interfaces for mixed reality applications. In *CHI '05: CHI '05 extended abstracts on Human*

factors in computing systems, pages 1339–1342, New York, NY, USA, ACM [cited at p. 13]

22. J. Kjeldskov and S. Howard. (2005) Envisioning mobile information services: Combining user-and technology-centered design. In *APCHI*, pages 180–190 [cited at p. 13]
23. Y. Li, J. I. Hong, and J. A. Landay. (2004) Topiary: a tool for prototyping location-enhanced applications. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 217–226, New York, NY, USA, ACM [cited at p. 13]
24. E. O'Neill, M. Klepal, D. Lewis, T. O'Donnell, D. O'Sullivan, and D. Pesch. (2005) A testbed for evaluating human interaction with ubiquitous computing environments. In *TRIDENTCOM '05: Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pages 60–69, Washington, DC, USA, IEEE Computer Society. [cited at p. 13]
25. C. Kray, P. Olivier, A. Weihong Guo, P. Singh, H. Nam Ha, and P. Blythe. (2007) Taming context: A key challenge in evaluating the usability of ubiquitous systems. In *Ubiquitous Systems Evaluation 2007 (USE'07) - Workshop at Ubicomp 2007*, Innsbruck, Austria [cited at p. 13]
26. P. Singh, H. Ha, Z. N. Kuang, P. Olivier, C. Kray, P. Blythe, and P. James. (2006) Immersive video as a rapid prototyping and evaluation tool for mobile and ambient applications. In *MobileHCI '06: Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 264–264, New York, NY, USA, ACM [cited at p. 13]
27. D. Schon. (1983) *The Reflective Practioner: How Professionals Think in Action.* Basic Book, New York [cited at p. 15]
28. P. Brugger, V. Pallotta, and B. Hirsbrunner. (2009) Optimizing heating systems management using an activity-based pervasive application. *JDIM - Journal of Digital Information Management*, (ISSN 0972-7272) [cited at p. 20]