DISTRIBUTED USER INTERFACES FOR AMBIENT INTELLIGENT ENVIRONMENTS

D A Sanders*, H Liu* and DJ Harrison**

[*] Systems Engineering Research Group, University of Portsmouth, Portsmouth, PO1 3DJ.
[**] Cleaner Electronics Research, Brunel University, Kingston Lane, Uxbridge, Middlesex, UB8 3PH.

Email: *david.sanders@port.ac.uk*

Abstract
This paper presents a model-based user interface for ambient intelligent environments. A task-centred approach is used to design interactive systems. The paper focuses on user interfaces supporting distributed tasks and a visualization of context influences on distributed, user interfaces. Support is considered using a visualization of the environment together with the user interface configurations. The concepts can narrow the gap between HCI design and software engineering.

Introduction

Modern computer solutions allow mobile and embedded software components to communicate with each other while residing on heterogeneous platforms. Modern middleware also offers automatic mechanisms to locate software and hardware available in a ubiquitous environment. While this can be considered as a step towards ubiquitous computing (Sanders, 2006; Weiser, 1991), there is still a gap between the actual tasks a user should be able to perform and the user interfaces exposed by ubiquitous systems to support those tasks. This paper is based on work by Luyten *et al* (2006) and employs a model-based and a model-driven approach to integrate the creation of a user interface and the development of the application logic.

Distributable user interfaces enable a user to exploit new possibilities in ambient computing by allocating tasks to resources that best support those tasks. A resource is any I/O channel that enables communication between user and system. The I/O channel is only supported from user to system or from system to user; it is limited to a single modality. Examples are keyboards and mice (Bergasa-Suso, 2005), joysticks (Sanders and Baldwin, 2001), pointers (Sanders, 2005), all sorts of screens, speech synthesizers, force feedback devices (Sanders, 2007), etc. Usually, an interaction resource was advertised in an environment through the computing device it is attached to an interaction cluster and manages input from or output to interaction resources attached to it. A multi-modal user interface is composed of different interaction resources, not necessarily located on the same interaction cluster.

Many research papers have been published discussing requirements, frameworks and models for distributed user interfaces (Balme, 2004; Larsson, 2004; Savidis, 2002; Vandervelpen, 2004) but there is still a lack of tools to allow designers to create such interfaces. The design of a user interface that can be distributed over several resources in a ubiquitous computing environment is a tedious task and has not yet been addressed, except perhaps by Luyten (2006). Tools to support the design of ambient intelligent user interfaces are essential. Design tools can hide the technical details and complexity. For example, distributed interfaces support interaction in ambient intelligent environments but require detailed knowledge of networking and distributed systems (something to hide from the designer).

In this paper a task-centred methodology for the design and the deployment of ambient intelligent user interfaces is presented. A mobile and distributable interface engineering tool is used that relies on human–computer interaction models introduced by model-based user interface design. By integrating support for UML based models, a rigorous software engineering process can be established. The approach provides the opportunities of UML-based modeling and tools whilst bridging the gap between traditional software engineering models and models from model-based user interface development. Such integration can be a first step towards the integration of model-based design within model-driven engineering approaches (Luyten, 2006).

This has been used to develop multi-device and context-aware user interfaces (Eisenstein, 2001; Clerckx, 2004; Mori, 2004). Different abstract models such as the task model and the domain model highlight different aspects of the user interface independent of details of the target devices. Concrete models such as the presentation model and navigation model will contribute more specific details towards the presentation of the interface. Typically, the complexity of these models is proportional to the complexity of the target domain. For ambient intelligent environments, models tend to be complicated and to require translation into intuitive interactive tool support.

The rest of this paper is structured as follows… following a brief background in Ambient Intelligence and the associated techniques; the paper provides an overview of some related work that defines the underlying concepts. Then different aspects to support a task-centred approach are considered (for ambient intelligent environments) and the way that context can influence a task is discussed before the prototype design tool is presented that has been used to support the design process. That is followed by a discussion of the opportunities available when integrating UML-based modeling and a conclusion.

Background

Ambient intelligence is a vision of consumer electronics, telecommunications and computing that was originally developed in the late 1990s for the time frame 2010–2020. Although a little behind the USA (and MIT in particular), the Fifth European Framework Program began work in Europe in 2001. The IST Program Advisory Group of the European Commission (Directorate General on Information Society and the Media) introduced the concept of ambient intelligence by publishing a report titled “Scenarios for Ambient Intelligence in 2010” (Ducatel, 2001; Ducatel, 2003).

The ambient intelligence paradigm builds upon ubiquitous computing. At the moment, users consciously engage single devices for specific specialized purposes. Someone using ubiquitous computing engages many computational devices and systems simultaneously and may not necessarily even be aware that they are doing so. Ubiquitous computing is a post-desktop model of human-computer interaction in which information processing is integrated into objects and activities. This paradigm is described as ambient intelligence.

Human-centric computer interaction design needs systems and technologies that are: embedded (that is many networked devices are integrated into the environment); context aware (these devices can recognize you and your situational context); personalized (they can be tailored to specific needs); adaptive (they can change in response to you); and anticipatory (they can anticipate your desires without conscious mediation). These are each considered here.

Embedded. An embedded system is a special-purpose computer system designed to perform one (or a few) dedicated functions, often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer can do many different tasks depending on programming. Since an embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale. Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights or factory controllers. Complexity varies from low (with a single microcontroller chip) to high (with multiple units, peripherals and networks).

Context aware. Context awareness originated in computer science. It is a way of linking changes in the environment with computer systems. Although it originated in computer science, it has also been applied to business theory within business process management. In computer science it refers to the idea that computers can sense and react based on their environment. Devices may have information about the circumstances under which they are able to operate and based on rules, or an intelligent stimulus, react accordingly. The term context-awareness in ubiquitous computing was introduced by Schilit (1994a and 1994b). Context aware devices may also try to make assumptions about the user's current situation. Dey (2001) defined context as "any information that can be used to characterise the situation of entities. While the computer science community has initially perceived context as a matter of user location, this notion has been considered not simply as a state, but part of a process in which users are involved; thus, sophisticated and general context models have been proposed, to support context-aware applications which use them to adapt interfaces, tailor sets of application-relevant data, increase precision of information retrieval, discover services, make user-interaction implicit, or build smart environments. Context aware systems are concerned with the acquisition of context (that is using sensors to perceive a situation), the abstraction and understanding of context (that is matching a perceived sensory stimulus to a context) and application behaviour based on the recognized context (that is triggering actions based on context). Context awareness is regarded as an enabling technology for ubiquitous computing systems. Context awareness is used to design innovative user interfaces, and is often used as a part of ubiquitous computing. It is also beginning to be felt in the internet with the advent of hybrid search engines. Human factors related context is a subset of this area and is often structured into three categories: information on the user, the user's social environment, and the user's tasks. Likewise, context related to the physical environment is structured into three categories: location, infrastructure and physical conditions.

Personalized. Personalization is based on user attributes. Personalization models include rules-based filtering, based on if then rules (Sanders and Rasol 2001), and collaborative filtering, which serves relevant material to customers by combining their own personal preferences with the preferences of like-minded others. Recently, another method, Prediction Based on Benefit is proposed for products with complex attributes such as apparel (Haag et al, 2006). Many companies already offer services for web recommendation and email recommendation that are based on personalization or anonymously collected user behaviours and some research is investigating Intelligent browser-based systems to assist Internet users (Bergasa-Suso, 2005) and electronic multi-media assessment systems (Chester, 2006). Web personalization is closely linked to the notion of adaptive hypermedia. The main difference is that the former would usually work on what is considered an Open Corpus Hypermedia, whilst the latter would traditionally work on Closed Corpus Hypermedia. However, recent research directions in the adaptive hypermedia domain take both closed and open corpus into account. Thus, the two fields are closely inter-related. Personalisation is also being considered for use in less overtly commercial applications to improve the user experience online.

Adaptive. An adaptive system is a set of interacting or interdependent entities, real or abstract, forming an integrated whole that together are able to respond to environmental changes or changes in the interacting parts.

Feedback loops represent a key feature of adaptive systems, allowing responses to changes; examples of adaptive systems include: natural ecosystems, individual organisms, human communities, human organizations, and human families. Some artificial systems can be adaptive as well; for instance, robots employ control systems that utilize feedback loops to sense new conditions in their environment and adapt accordingly (Stott, 1995). Adaptive behaviour is often characterized by a kind of behaviour that allows an individual to substitute an unconstructive or disruptive behaviour to something more constructive. These behaviours are most often social or personal behaviours. For example a constant repetitive action could be re-focused on something that creates or builds something. In other words the behaviour can be adapted to something else. A maladaptive behaviour is a behaviour or trait that is not adaptive (it might be counterproductive to the individual). Maladaptivity is frequently used as an indicator of abnormality or mental dysfunction in human beings, since its assessment is relatively free from subjectivity. However, much behaviour considered moral can be apparently maladaptive, such as dissent or abstinence.

Anticipatory. Anticipatory means that systems can anticipate desires without conscious mediation. For example, anticipatory scheduling is an algorithm for scheduling input/output. Anticipatory scheduling can yield significant improvements for some work. Another example is when a design program anticipates the next set of icons or buttons that a user might want to use and presents them to the user on the screen.

Early developments in Ambient Intelligence took place at Philips (Sanders, 2006). In 1998, Philips commissioned a series of internal workshop to investigate different scenarios that would transform the high-volume consumer electronic industry from the current "fragmented-with-features" world into a world in 2020 where user-friendly devices support ubiquitous information, communication and entertainment. In 1999, Philips joined the Oxygen alliance, an international consortium of industrial partners within the MIT Oxygen project (Oxygen, 2006) aimed at developing technology for the computer of the 21st century. In 2000, plans were made to construct a feasibility and usability facility dedicated to Ambient Intelligence. This opened in 2002. Along with the build up of the vision for Philips, a parallel track was started to open up the vision. Following the advice of the Information Society and Technology Advisory Group, the European Commission used the vision for the launch of their sixth framework (FP5) in Information, Society and Technology and the Commission played a crucial role in further developing Ambient Intelligence. More recently, several major initiatives have been started in the USA, Canada, Spain, France and the Netherlands.

Ambient intelligence emphasizes people and user experience (Sanders, 2006). The interest in user experience grew in importance in the late 1990s because of the overload of products and services in the information society that were difficult to understand and hard to use. A strong call emerged to design things from a user's point of view. Ambient intelligence is influenced by user-centred design where the user is placed in the centre of the design activity and asked to give feedback through specific user evaluations and tests to improve the design or even co-create the design together with the designer or with other users. Ambient intelligence needs further development in a number of key technologies: Unobtrusive hardware; Seamless communication and computing infrastructure; Dynamic and distributed device networks; and human-centric computer interfaces.

Crafter (Ponnekanti, 2001) is a framework of services and their user interfaces for use in a ubiquitous environment. Services can register their user interfaces with the ICrafter Interface Manager. This way, other service can request those interfaces through the Interface Manager for rendering purposes. Once the interface is instantiated, interaction with the associated service becomes possible. This service-oriented approach provides a uniform and location-independent access to the functionality of the system. Dynamic composition or on-the-fly aggregations of user interface components are central to this approach. However, there is no design support to constrain the dynamic behaviour so it is difficult to ensure the user interface is usable while supporting the envisioned tasks.

Heider and Kirste (2002) proposed a goal-driven approach to decide which interaction resources to use. In their approach a planning algorithm is used for developing strategies to reach the predefined goals. An execution control component can execute a strategy and manages the resources that are necessary for the selected strategy. This approach is useful to cope with the complexity of designing a user interface that should work in an ambient intelligent environment. A task-centred approach could benefit by using a planning algorithm to calculate an optimal strategy for executing the required tasks with the interaction resources that are available. Look *et al* (2003) proposed a similar approach, where the importance of the user's goals is recognized as input for deciding on an optimal resource allocation to support the user.

Distribution of a user interface among different interaction resources or multiple surfaces is also gaining importance: unlike traditional desktop computing, a user interface in an ambient intelligent environment is no longer limited to one device that is the centre of interaction. In (Coutaz, 2003) an ontology for multisurface interaction is proposed by Coutaz et al. This ontology offers a unifying framework for reasoning about distributed user interfaces. Because of the complexity of the covered types of problems, this ontology can only show its full potential when it is used in a human computer interaction design tool.

Balme *et al* (2004) presented the CAMELEON-RT Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. Some middleware was provided (the Distribution-Migration-Plasticity middleware) to allow smooth integration of user interfaces that reside on different physical locations. This type of support for distributed user interfaces is required to deploy a user interface for an ambient intelligent environment. In (Vandervelpen, 2005) it was shown that conventional interactive websites can be distributed among different interaction resources. This proved a structured high-level user interface description language (HTML in the case of the website) is a suitable way to create distributable user interfaces.

Modeling user interfaces is an important part of the design of complex interactive applications. The user interface, however, must also be coupled to application logic. This together with the fact that often programmers must ultimately also produce working code for the user interface, led to several approaches that describe models that originally came from the MBUID community using Unified Modeling Language.

Some of the earliest results in pairing Unified Modeling Language diagrams and model-based user interface design were discussed in (Nunes, 2000). UMLi and Wisdom were among the discussed approaches. UMLi (da Silva, 2003) focused on the description of user interfaces and defined two new diagram types for the description of user interfaces. The presentation model was described using a diagram similar to the deployment diagram; while the user interface logic was described using a notation based on the activity diagram. UMLi is probably the most technically mature proposal for interface development in Unified Modeling Language; it is becoming accepted as the Unified Modeling Language for Interactive Applications. The Wisdom approach (Nunes, 2000) used a light-weight software engineering method for small businesses, used a set of stereotypes to extend the Unified Modeling Language for interactive systems development. Among others it used an extended version of the class diagram to express the presentation model and the task model. CanonSketch (Campos, 2005) offered specialized tool support for the presentation model offering views using Unified Modeling Language, Canonical Abstract Prototype notation (Constantine, 2003) and HyperText Markup Language.

Properties of ambient task modeling

Paternò's ConcurTaskTrees (Paternò, 2000; Mori, 2002) is a notation for task modeling that provides temporal operators between tasks. This notation offers a graphical syntax, a hierarchical structure and a notation to specify the temporal relation between tasks. Four types of tasks are supported in the ConcurTaskTrees notation: abstract tasks, interaction tasks, user tasks and application tasks. These tasks can be specified to be executed in several iterations. Sibling tasks, appearing in the same level in the hierarchy of decomposition, can be connected by temporal operators like choice, independent concurrency, concurrency with information exchange, disabling, enabling, enabling with information exchange, suspend/resume and order independence (Luyten, 2006). Paternò and Santoro (2002) specify the following priority order among the temporal operators: choice > parallel composition > disabling > enabling.

The ConcurTaskTrees notation allows the extraction of task sets where each task set contains tasks that should be "active" during the same period of time in order to reach a (sub) goal. This concept is called enabled task sets (Paternò, 2000). For a given task model several enabled task sets can be identified: each set contains tasks that execute within the time frame defined by the set and do not overlap with other tasks from other sets. We can describe this process by the function $f{:}M \rightarrow TS1,TS2,\ldots,TSn$ that maps a task model M on the set of enabled task sets, $TSi, 1 \leqslant i \leqslant n$; which is a set of subsets of the model M. Several design tools exist that provide this TS extraction functionality by means of the CTT notation and their use is described in existing literature (Mori, 2002; Luyten 2003; Vanderdonckt 2003). Each task set, $TSi, 1 \leqslant i \leqslant n$, contains a subset of tasks t1, t2, …, tm from the task model M. A task set requires a distribution configuration for the tasks it contains: the representation of a task set is distributed among different devices that are available in the environment. Temporal relations between different tasks, together with the fact there are no two task sets that can overlap in time, allow the construction of a sequence of task sets that the user(s) should execute to reach their goals as specified by the task specification. Luyten depicts an example of such a sequence of enabled task sets (Luyten, 2006).

Task set properties. A first property is completeness (Luyten, 2006). User interface completeness indicates that all interaction tasks needed to reach a goal at a particular moment are made accessible to a user regardless of the interaction resources available in the environment (including the interaction resources exposed by the user's personal devices). The use of task sets to guide a design process ensures this property: all tasks of the active TS need to be allocated to interaction resources that can handle these tasks. From a given task model the number of task sets that can be found is exactly the minimal number of logically different interfaces, a designer should provide to allow the user to access the complete functionality of a system. Luyten (2006) shows how tasks in an active task set are distributed over interaction resources in the environment. Task Sets can be ordered in time; this ordering is also referred to as the dialog model (Luyten, 2006).

A second property is continuity. User interface continuity ensures the user can interpret and evaluate the internal state of a system while using different interaction resources. Consequently, the user does not lose track of the current task. When the distribution of the interface parts changes at run time, this property must hold. Consistency of the user interface across different platforms can support a better continuity of the user interface. In this sense continuity is a broader concept: it refers to minimization of interruption of the user by the changes in the user interface. Providing support for the preservation of continuous interaction will pose a difficult challenge for a design methodology (and tool) that uses tasks, activities and temporal relations (Faconti, 2000). Continuity is supported by constraining the possible task-distribution strategies. For example; a constraint to support continuity is the fixed task constraint which is formalized as follows: if the tasks in TSi are enabled and $\exists t \in M{:}t \in TSi \wedge t \in TSj$ then t will not be re-distributed to another device when a transition from TSi to TSj is executed (Luyten, 2006). A task that reoccurs in different task sets can be restricted to the same device when the transition to the following task set is made. An example of the application of such a fixed task constraint is included in (Luyten, 2006).

More specific constraints could be added depending on the properties of the devices.

Properties such as bandwidth should be made explicit. This allows a designer to use these properties while modeling the interactive system. The device model and constraints can be combined in an environment model and used in a tool to support task modeling for ambient intelligent environments. The environment model can also be represented as a universal modelling language deployment diagram that encodes the available interaction resources, relations between interaction resources and properties of both resources and relations.

Task migration paths. The previous section discussed the distribution of tasks of each individual task set taking into account continuity and completeness of the user interface. Once an appropriate set of task set distributions is found for each task set, the designer should be able to constrain the transitions from one task set to another.

Task representations. Each interaction task from the task specification should be presented in the environment so the user can interact with it. Interaction tasks can be annotated by different ways they can be presented to the user(s). For each task $t \in M$ an abstract user interface description $x \in \{X1,X2,\ldots,Xn\}$ can be retrieved, the set of related user interface descriptions is referred to as the presentation model.

Based on the findings in related research (Luyten, 2004 and 2006), a user interface description is specified using an XML-based notation. Luyten (2006) shows how the high-level user interface descriptions of all tasks available in a task set are distributed among different appropriate interaction resources available in the environment while the user continues their interaction with the application, from one active task set to the other.

For each task set there are different possibilities of how a user interface representing the set of tasks can be divided.

Contextual task constraints

The allocation of a task to a set of interaction resources can also constrain the execution of the task. For example: a task can only be valid within a certain physical range because the interaction resource it is allocated to has to maintain a communication channel with another device that executes a parallel task exchanging information with the first task. Luyten (2006) shows this scenario.

Two different constraints can be identified for these two tasks:

(1) both tasks should be observable at the same time by the user;

(2) both tasks should be able to exchange data using some kind of communication channel.

The first one depends on the designer's intentions and should be part of the task specification; the second one can be derived from the task-device allocations. With either one (or both) of these constraints there is only one possibility: the device that represents t2 has to be located in the predefined area of the device presenting t1. t1 and t2 belong to the same task set, since they can be executed during the same time period.

The prototype platform

The models discussed in this paper have been integrated by a prototype platform running on a Sony Vaio laptop computer. The prototype platform supports a simple user interface for designing within ambient intelligent environments. The central model is the task model, describing the set of tasks the (ubiquitous) application supports. Other models include the environment model that describes all available interaction resources in the environment of the user, a dialog model containing the task sets derived from the task model, a presentation model that is related to the tasks in the task model and an interaction model describing the interaction between the user interface and the application logic. Every view offers direct manipulation of these different models and visualizes the relations between different models.

The prototype architecture consists of three modules:

- Design Tool
- Environment Repository
- Distribution Manager

When the Environment Repository is started, it probes the ambient intelligent environment to look for interaction clusters that can be used in an interactive distributed user interface session. This has been implemented using Universal Plug and Play (Luyten, 2006). The Environment Repository uses Resource Description Format and can use existing tools to parse the received documents and merge them into one in-memory resource description format model (an environment model). That this model was based on the ontology presented in (Preuveneers, 2003).

Once the environment repository is started, the design tool can interface with it locally or through a network to get information about the environment model. The design tool uses this information to build a visual representation of the environment and to check constraints of interaction resources while the user interface designer constructs the distribution model.

The last important module of the prototype platform is the Distribution Manager which can be used to deploy the user interface in the real ambient intelligent environment according to the distribution model constructed by the user interface designer (using the design tool) and the state of the environment.

The distribution manager can be invoked by the design tool. In that mode, the necessary models are passed to the distribution manager and the state of the task distributions at a particular moment is communicated back to the design tool. The design tool used this information to update its visualizations.

The distribution manager was divided into the Distribution Producer (also connected to the Environment Repository) and the Distributed User Interface Session Manager. The session manager executed the scenario by sending the appropriate user interface components to the interaction clusters. The distribution producer could register itself to receive notifications from the environment repository when changes in the properties of relevant interaction resources occurred.

Tasks could be related with interaction resources of the environment model in two ways:

(1) Automatically: task can be allocated among the available interaction resources automatically by applying the different constraints.

(2) Manually: usually, there are a number of solutions that are valid with respect to the constraints defined by the different models. The design tool supports manual editing of the task allocations (which actually presents the task-environment inter-relation): the designer can relate tasks with interaction resources and observe the effects of these changes.

An important aspect of the design tool was the possibility to simulate the run-time behaviour of the distributed user interface. This simulation is considered as a view on the different models that are built with the design tool and was integrated with the other views. The simulation creates a 3D model of the environment model (using the Java3D API3), and uses the list of interaction resources to dynamically render the user environment. A simulation module aids in defining the appropriate Task Migration Paths.

Relating tasks with devices through direct manipulation on the 3D view of the environment model is obviously more intuitive than working only with diagrammatic notations. Although this model supports direct manipulation, it is also suitable to visualize existing relations already created between the other models. This way the designer will have a graphical overview of the user interface distribution and instantly sees the effect of model manipulations.

A possible extension that is investigated is to use a universal modelling language representation of the Environment Repository. This could be done in real-time within the tool or by converting snapshots of the Environment Repository to a XMI-document that could be imported.

Integration with UML-based software engineering

Universal Modeling Language 2.0 was used to provide a rigorous description of the different aspects of interactive software. It had better facilities for model-driven development which could make the design methodologies more efficient.

Integration with the universal modeling language was important because it helped bridge the gap between the human computer interface designer and the software developer. For complex domains, (ambient intelligent environments) there was no support to integrate the design of the application logic with the interaction design. Because universal modeling language was widely accepted, it also offered a more formal way to describe the functionality of a system and provided tools to relate a user interface with the functionality that was represented.

Mapping to Universal Modelling Language. Universal Modelling Language 2.0 deployment diagrams were used to describe some features in more detail, for example the communication channels, interaction clusters and composition of the interaction clusters. The deployment diagram is usually a static diagram but (Luyten, 2006) showed that dynamic discovery of available interaction clusters was possible. Since the content of the deployment diagram depends on the point in time when a discovery is executed, it is possible to have many deployment diagrams for a single application.

The distribution of a user interface can be specified by allocating parts of the user interface to specific interaction resources or interaction clusters. There is a natural mapping from interaction clusters and interaction resources to Devices in Universal Modelling Language 2.0, where the former can contain other Devices (interaction resources) and the latter cannot, due to the definition of an interaction resource.

Links between the logical structure of the user interface and the physical structure were explicitly modelled. This approach has the advantage that the realization of the user interface is explicitly and unambiguously specified using an approach already in use for the application logic.

Model-driven engineering. Abstract presentations could be derived from the task sets. These could be converted to high-level descriptions using context information (for example user-profiles).

Abstractions can be useful to derive user interfaces that are consistent and complete, but have different appearances. It would be more difficult to design a multi-cultural, multi-device user interface that is both consistent and complete using separate designs (Luyten, 2006).

Two possible tool configurations for integration of the prototype with model-driven engineering to create a complete environment for the design of distributed user interfaces. In both configurations the prototype provides the task-based distribution facilities based upon the discovery of interaction devices and resources in the neighbourhood and designer input.

The prototype works at the task-level and relies on the linking of tasks to (declarative) user interface descriptions to accomplish effective distribution of user interfaces.

In the first configuration all necessary components to create the user interfaces are integrated into one integrated tool so that the distribution created in the tool is passed to a separate part of the tool that works with universal modeling language and starts from a deployment specification. Starting from this model, several transformations can be made to transform the abstract model into different concrete models, specifying concrete user interface configurations for certain hardware configurations. An advantage of this approach is that only one environment is needed and it can be easier to get specific support for transformations.

The second configuration delivered task distribution in XMI-format to a Universal Modelling Language Tool and it could support model-driven engineering. After one or more transformations, the abstract representation that was given to the Universal Modelling Language Tool was translated into one or more concrete models of user interfaces and was delivered to a user interface modeling tool. The advantage of this was that existing tools could be used for manipulating the models.

Discussion, conclusions and future work

For the moment, the ambient intelligence in the system is mainly about the interactions between the human user and the system. Internal functioning of the software is not fully considered deeply and the software is not connected to the environment. The systems described here do consider user experience but more work is required to make the ambient intelligence more user-centred so that the user is placed nearer the centre of the design activity taking place. Tests to improve the design (or even co-create the design together with the designer) could be introduced.

The distributed interface tools could be made more dynamic. Communication could be seamless and interoperability could be improved. The approaches described enabled the team to design and test user interfaces that could be distributed among different interaction resources and verified the results in (Luyten, 2006).

User interface completeness (the required functionality to reach a user's goals) and continuity (can a usable user interface be created for a dynamic environment?) were the two main properties considered in this paper. Both the visualization of the task allocations in the environment and the simulation of the execution of a task specification were the primary tools to ensure completeness and continuity.

There are many aspects to be considered when using a model-based approach for designing user interfaces that in ambient intelligent environments. Traditional model-based user interface development approaches do not consider dynamic environments with different devices that can be used simultaneously. This paper ahs described some prototyping that can contribute to a solution to this problem.

References

Balme L, Demeure A, Barralon N, Coutaz J, Calvary G. (2004). CAMELEON-RT: a software architecture reference model for distributed, migratable, and plastic user interfaces. In: Markopoulos P, Eggen B, Aarts EHL, Crowley JL, editors. EUSAI, Lecture notes in computer science 3295. Berlin: Springer; pp 291–302.

Bergasa-Suso J, Sanders DA and Tewkesbury, GE (2005). Intelligent browser-based systems to assist Internet users. IEEE Trans on Ed' 48 (4), pp 580-585.

Campos PF, Nunes NJ. Canonsketch (2005). A user-centred tool for canonical abstract prototyping. In Bastide R, Palanque P, Roth J, editors. *Engineering human computer interaction and interactive systems: joint working conferences EHCI-DSVIS 2004*. Lecture notes in computer science 3425. Berlin: Springer; pp 146–63.

Constantine LL (2003). Canonical abstract prototypes for abstract visual and interaction design. In Proceedings of DSV-IS 2003. Lecture notes in computer science, vol. 2844. Funchal, Madeira Island, Portugal. Berlin: Springer, pp. 1–15.

Chester S, Tewkesbury GE, Sanders DA *et al* (2006). New electronic multi-media assessment system. Web Information systems and technologies, pp 320-324.

Clerckx T, Luyten K, Coninx K (20004). Generating context-sensitive multiple device interfaces from design. In Proceedings of the 4th Int Conf on computer-aided design of user interfaces CADUI'2004, 13–16 January 2004, Funchal, Isle of Madeira, Portugal.

Coutaz J, Lachenal C, Dupuy-Chessa S (2003). Ontology for multi-surface interaction. In Rauterberg M, Menozzi M, Wesson J, editors. Human–computer interaction INTERACT '03: IFIP TC13 international conference on human-computer interaction.

da Silva PP and Paton NW (2003). User interface modelling in umli, *IEEE Software* **20** (4), pp 62–69.

Dey AK. (2001). Understanding and Using Context. Personal Ubiquitous Computing 5 (1), pp 4–7.

Ducatel K, Bogdanowicz M, Scapolo F, Leijten and JC Burgelman (2001) Scenarios for Ambient Intelligence in 2010, Technical Report 10, ISTAG, February 2001.

Ducatel K, Bogdanowicz M, Scapolo F, Leijten and JC Burgelman (2003) Ambient Intelligence: from Vision to Reality, Technical Report, ISTAG.

Eisenstein J, Vanderdonckt J, Puerta AR (2001). Applying model-based techniques to the development of uis for mobile computers. In Intelligent user interfaces, pp 69–76.

Faconti P and Massink M (2000). Continuity in human computer interaction. In CHI 2000 Workshop report. 〈http://www.acm.org/sigchi/bulletin/2000.4〉.

Haag et al (2006). Management Information Systems for the Information Age, 3rd edition, page 331.

Heider T, Kirste T (2002). Supporting goal-based interaction with dynamic intelligent environments. In van Harmelen F, ed. ECAI; IOS Press, pp 596–600.

Larsson A, Berglund E (2004). Programming ubiquitous software applications: requirements for distributed user interface. In: Maurer F, Ruhe G, ed. SEKE, pp 246–51.

Look G, Peters S, Shrobe H (2003). Plan-driven ubiquitous computing. In Artificial intelligence in mobile system.

Luyten K, Abrams M, Limbourg Q, Vanderdonckt J (2004 – Editors). D eveloping user interfaces with XML: advances on user interface description languages.

Luyten C (2002). One model, many interfaces. In Kolski C, Vanderdoncke J, editors. Computer-aided design of user interfaces III CADUI, vol. 3. Dordrecht: Kluwer Academic, pp 143–54.

Luyten K, den Bergh JV, Vandervelpen C and Coninx K (2006). Designing distributed user interfaces for ambient intelligent environments using models and simulations Computers & Graphics 30 (5), pp 702-713.

Mori G, Paternò F and Santoro C (2002). CTTE: support for developing and analyzing task models for interactive system design, *IEEE Transactions of Software Engineering* **28** (8), pp. 797–813.

Mori G, Paternò F and Santoro C (2004). Design and development of multidevice user interfaces through multiple logical descriptions, *IEEE Transactions of Software Engineering* **30** (8), pp. 507–520.

Nunes NJ, editor. Towards a UML profile for interactive systems development (TUPIS 2000); 2000. online math.uma.pt/tupis00/.

Nunes NJ, e Cunha JF (2000). Towards a uml profile for interaction design: the wisdom approach. In Evans A, Kent S, Selic B, editors. UML 2000—The unified modeling language. Advancing the standard. Lecture notes in computer science, vol. 1939. Berlin: Springer, pp 101–16.

Oxygen (2006). http://www.oxygen.lcs.mit.edu

Paternò F (2000). Model-based design and evaluation of interactive applications, Springer, Berlin.

Ponnekanti S, Lee B, Fox A, Hanrahan P, Winograd T. ICrafter (2001). A service framework for ubiquitous computing environments. In Ubicomp 2001: ubiquitous computing, third international conference Atlanta, Georgia, USA, September 30–October 2, Proc, Lecture notes in computer science. Berlin: Springer, pp 56–75.

Preuveneers D, den Bergh JV, Wagelaar D, Georges A, Rigole P, Clerckx T, Berbers Y, Coninx K, Jonckers V, Bosschere KD (2004). Towards an extensible context ontology for ambient intelligence. In Markopoulos P, Eggen B, Aarts EHL, Crowley JL, editors. Ambient intelligence: second European symposium, EUSAI 2004, Eindhoven, Nov 8–11. Proc pp 148–159.

Sanders DA and Rasol, Z (2001). An automatic system for simple spot welding tasks. Total vehicle technology: Challenging current thinking, pp 263-272.

Sanders DA and Baldwin A (2001). X-by-wire technology. Total vehicle technology: challenging current thinking, pp 3-12.

Sanders DA, Urwin-Wright SD, Tewkesbury GE, et al. (2005). Pointer device for thin-film transistor and cathode ray tube computer screens. Electronics Letters 41 (16), pp 894-896.

Sanders, D. (2007). Force sensing. Industiral robot – an international journal 34 (4), pp 268-268.

Savidis A, Maou N, Pachoulakis I and Stephanidis C (2002). Continuity of interaction in nomadic interfaces through migration and dynamic utilization of I/O resources, *Universal Access in the Information Society* **4** (1), pp. 274–287.

Schilit B, Adams N, and Want R (1994a). Context-aware computing applications. IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94), Santa Cruz, CA, US, pp 89-101.

Schilit BN and Theimer MM (1994b). Disseminating Active Map Information to Mobile Hosts. IEEE Network 8 (5), pp 22–32.

Country Code 21.

Stott IJ, Sanders DA and Goodwin M (1995). Improvements in real time high-level micro computer control of a wheelchair using sensor systems. IEEE Proc' of "Euromicro, pp 335-340.

Vandervelpen C, Coninx K (2004). Towards model-based design support for distributed user interfaces. In Proceedings of the third nordic conference on human–computer interaction. New York: ACM Press, pp 61–70.

Vandervelpen C, Vanderhulst G, Luyten K, Coninx K (2005). Light-weight distributed web interfaces: preparing the web for heterogeneous environments. In: Fifth international conference on web engineering (ICWE'2005)〈http://research.edm.luc.ac.be/cvandervelpen/research/icwe2005/〉.

Weiser M (1991). The computer for the 21st century. Scientific American 265(3), pp 94–104.