Context-sensitive User Interfaces for Ambient Intelligent Environments: Design, Development and Deployment

Kris Luyten[1], Chris Vandervelpen[1], Jan Van den Bergh[1], and Karin Coninx[1]

Hasselt University – Expertise Centre for Digital Media
transnationale Universiteit Limburg, School for Information Technology
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
{kris.luyten, chris.vandervelpen, jan.vandenbergh,
karin.coninx}@uhasselt.be

Abstract. There is a growing demand for design support to create interactive systems that are deployed in ambient intelligent environments. Unlike traditional interactive systems, the wide diversity of situations these type of user interfaces need to work in require tool-support that is close to the environment of the end-user on the one hand and provide a smooth integration with the application logic on the other hand. This paper shows how the Model-Based User Interface Development methodology can be applied for ambient intelligent environments; we propose a task-centered approach towards the design of interactive systems by means of appropriate visualizations and simulations of different models. Besides the use of typical user interface models, such as the task- and presentation-model, we focus on user interfaces supporting situated task distributions and a visualization of context influences on deployed user interfaces. To enable this we introduce an environment model describing the device configuration at particular moment in time. To support the user interface designer while creating these complex interfaces for ambient intelligent environments, we discuss tool support using a visualization of the environment together with simulations of the user interface configurations. We also show how the concepts presented in the paper can be integrated within Model-Driven Engineering, hereby narrowing the gap between HCI design and software engineering.

1 Introduction

Modern middleware solutions allow mobile and embedded software components to communicate with each other while residing on heterogeneous platforms. Modern middleware also offers automatic discovery mechanisms to locate necessary software and hardware available in an ubiquitous environment. While this can be considered as a step toward the ubiquitous computing vision Mark Weiser predicted [27], there still exists a large 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 gap is caused mainly by two missing pieces: the lack

Dagstuhl Seminar Proceedings 05181
Mobile Computing and Ambient Intelligence: The Challenge of Multimedia
http://drops.dagstuhl.de/opus/volltexte/2005/377

of a task-centered user interface design approach on the one hand and the lack of support for distributable user interfaces in ambient intelligent environments on the other hand. In this paper we present our ongoing work on model-based user interface development techniques to enable the design and deployment of effective distributable user interfaces for heterogeneous environments.

Distributable user interfaces enable the user to exploit more and new possibilities of an ambient computing environment by allocating tasks to interaction resources that best support those tasks. We define an *interaction resource* as an atomic I/O channel. In this context, atomic means the I/O channel is "one way" and limited to a single modality. Examples of interaction resources are keyboards, mice, all sorts of screens, speech synthesizers, force feedback devices,.... Usually, an interaction resources is advertised in an environment through the computing device it is attached to. This computing device is called an *interaction cluster* and manages input from or output to interaction resources attached to it. The aforementioned definitions imply also a multi-modal user interface is composed of different interaction resources, not necessarily located on the same interaction cluster.

During the last couple of years many research papers have been published discussing requirements, frameworks and models for distributed user interfaces (e.g. [23, 1, 25, 11]), 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 interaction resources in an ubiquitous computing environment is a tedious task and has not yet been addressed extensively. Distributed interfaces are typical for supporting interaction in ambient intelligent environments.

In this paper we present our ongoing work on a task-centered methodology for the *design* and the *deployment* of distributable user interfaces: MoDIE (Mobile Distributable Interface Engineering). In addition, the integration of MoDIE with UML 2.0 based models is proposed. As such our approach provides the opportunities of UML-based modeling methodologies and tools whilst bridging the gap between traditional software engineering models and models from model-based user interface development. Such an integration can be a first step towards integration of model-based design within model driven engineering approaches. This work is part of the CoDAMoS project, a joint project with three other Flemish universities and several industrial partners aimed at solving a set of key challenges in the area of Ambient Intelligence (AmI), where personal devices will form an extension of each user's environment, running mobile services adapted to the user and his context.

The approach we present in this paper is based on the Model-Based User Interface Development (MBUID) methodology. MBUID is already in use to develop multi-device user interfaces [7, 3, 16], and we show it can be extended for ambient intelligent environments. In MBUID, different abstract models highlight different aspects of the user interface independent of details of the target devices. Concrete models will "fill in" more specific details towards the presentation of the interface. Even for the domain of multi-device interface design there is still work to be done to visualize the different models and the influence of model

manipulations on the final user interface. A task-centered approach offers a way to validate whether the user interface supports the goals of the user. This paper considers three concepts that are important for a task-centered approach. The first is *situated task allocations*: the execution of a task is dependent on different parameters that are not part of the software itself, such as the location of the user. The second concept is the *distribution of interface presentations* among the available interaction devices (see section 6). The third concept is the *visualization of context influences* to inform the designer of possible influences of the environment on the proposed design.

Several concepts are relevant for the design of usable distributed user interfaces. In the remainder of this paper, we focus on two concepts that are supported by this approach: interface *completeness* and *continuity*. The former can be obtained by ensuring all tasks are represented in the user's environment at the required time, the latter is obtained by defining a set of transition rules to progress from one task to another. The remainder of this paper is structured as follows: section 2 gives an overview of the related work that defines the underlying concepts for the topic of this paper. Next, section 3 discusses the different aspects that need to be taken into account to support a task-centered approach to design user interfaces for ambient intelligent environments. Section 4 explains how context can have a big influence on the task execution and what needs to be done to anticipate this while modelling. Section 5 presents the design tool we are developing to support the design process, followed by a discussion of the opportunities that are available when integrating UML-based modeling. Finally, section 7 gives a conclusion.

2 Related Work

There is a growing interest in the design of interactive systems that can be deployed for ambient intelligent environments. Most research work in this area is focused on a particular subtopic. Georgantas and Issarny show a functional approach towards modelling a situation sensitive user interface in [8]. Just as in the ICrafter [21] a service framework for user interface services is created. Most of this work reflects the need for some kind of unified framework to design and develop the interactive part of a computing system that is deployed in an ambient intelligent environment. The service-oriented approach presented in these papers provide 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 these approaches. However, there is no design support to constrain the dynamic behavior of these systems so the resulting user interface is usable and still supports the envisioned tasks depending on the situation.

Heider and Kirste propose a goal-driven approach to decide which interaction resources to use in [10]. 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 enormous com-

plexity of designing a user interface that should work in an ambient intelligent environment. A task-centered 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.

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 center of interaction. In [5] an ontology for multisurface interaction is proposed by Coutaz et al. This ontology offers an unifying framework for reasoning about distributed user interfaces. Because of the complexity of the covered types of problems, these kind of ontologies can only be shown to full advantage when it is used in a HCI design tool.

Balme et al. presented the CAMELEON-RT Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces [1]. Some type of middleware is provided (the Distribution-Migration-Plasticity middleware) to allow smooth integration of user interfaces that reside on different physical locations. In [26] we show the feasibility of automatically distributing a highly interactive website over several interaction resources.

Two requirements should be fulfilled to result in a usable distributable user interface: *completeness* and *continuity*. Completeness of a user interface means that all interaction tasks necessary to reach a goal at a particular moment are made accessible to the user regardless of the devices available in the environment (including the user's personal devices). This is achieved by using a task-centered approach. On the other hand, user interface continuity ensures the user can interpret and evaluate the internal state of the system while using different input/output devices [6]. Even when the distribution of the interface parts among the different interaction resources changes at run time, this property must hold. 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 [19, 15].

3 Properties of Ambient Task Modelling

3.1 Task Notation and Dialog Derivation

We use Paternò's ConcurTaskTrees (CTT) notation [20, 17]; a notation for task modeling that provides temporal operators between tasks. This notation allows to extract 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* (ETSs) [20]. For a given task model M several of such 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 $f : M \rightarrow TS_1, TS_2, \cdots, TS_n$ that maps a task model M on the enabled task sets $TS_{i,1 \leq i \leq n}$. Several design tools exist that provide this ETS extraction functionality by means of the Con-

curTaskTrees notation and their use is described in existing literature [17, 16, 13, 24].

Each task set $TS_{i,1\leq i\leq n}$ contains a subset of tasks $t_1, t_2, \cdots, t_m$ 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. Notice a user interface distribution is defined in section 1 and specifies the combination of tasks in a dialog with the available interaction devices. Because of the temporal relations between different tasks, together with the fact there are no two ETSs that can overlap in time, a sequence of ETSs can be identified that the user(s) should execute to reach the goals at hand. Figure 1 depicts an example of such a sequence of enabled task sets (labeled with TS1, TS2,...).

3.2 Task-set Distributions

The first property we consider in our approach is *completeness*. User interface completeness indicates that all interaction tasks needed to reach a goal at a particular moment are made accessible to the user regardless of the interaction resources available in the environment (including the interaction resources exposed by the user's personal devices). The use of ETSs to guide the design process ensures this property: all tasks of the active ETS need to be allocated to interaction resources that can handle these tasks. From a given task model the number of ETSs that can be found is exactly the minimal number of logically different interfaces (or "presentation units" according to [7]) the designer should provide to allow the user to access the complete functionality of a system. Figure 1 shows how tasks in an active ETS are distributed over interaction resources in the environment. Notice ETSs can be ordered in time because of the definition given above (this ordering is also referred to as the dialog model).

The second property we consider is *continuity*. User interface continuity ensures the user can interpret and evaluate the internal state of the system while using different interaction resources. When the distribution of the interface parts changes at run time, this property must hold. 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 [19]. In our approach continuity is supported by constraining the possible task-distribution strategies. For example; a possible constraint to support continuity is the *fixed task constraint* which is formalized as follows: if ETS_i is enabled and $\exists t \in M : t \in ETS_i \wedge t \in ETS_j$ then t will not be re-distributed when a transition from ETS_i to ETS_j is executed. In other words: a task that reoccurs in different ETSs can be restricted to the same device when the transition to the following ETS is made. In figure 1, *task 3* is an example of the application of such a fixed task constraint for the transition from ETS_5 to ETS_3.

We can add more specific constraints depending on the properties of the devices. For $t \in M$, t can be constrained to a set of devices D_c that is a subset of all available devices D, and $\forall d \in D_c$, $\pi_c(d) = 1$. $\pi_c(d)$ is a projection of the property c over the element d; c represents a property of a device d. E.g.:

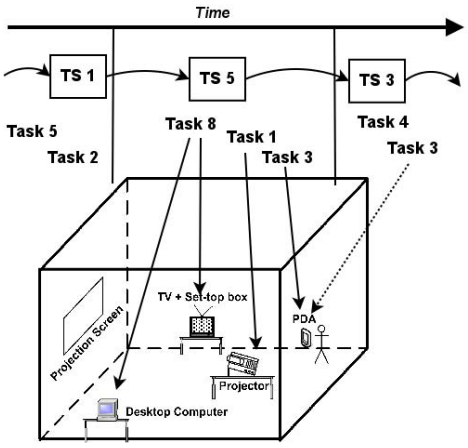


Fig. 1. Different Enabled Task Sets on a timeline with their distribution

when distributed, certain tasks can be constrained to devices that have some kind of network communication available. In this example the property value is 0 if there is no network communication available and 1 otherwise. Of course, a distribution can also be constrained according to a value of a property such as the quality of network communication that is available. This can be expressed as $b_1 \leq \pi_b(d) \leq b_2$, where b is the attribute of d representing the bandwidth available at element d, b_1 is the lower boundary of the required bandwidth and b_2 the upper boundary.

The properties such as the bandwidth should be made explicit in the design tool. This allows the designer to use these properties while modelling the interactive system. Section 5 shows how the device model and constraints are combined in an environment model and used in a tool to support task-modelling for ambient intelligent environments. The environment model can also be represented as a UML deployment diagram that encodes the available interaction resources, relations between interaction resources and properties of both resources and relations. Section 6 shows the relation of the environment model and the deployment diagram.

3.3 Task Migration Paths

The previous section discussed the distribution of tasks of each individual ETS taking into account continuity and completeness of the user interface. Once an appropriate set of task-set distributions is found for each ETS, the designer should be able to constrain the transitions from one ETS to another. A lack of continuity because of a context switch (other devices that come into play, tasks that appear and dissapear,...) can have a disastrous effect on task performance.

In traditional MBUID this is represented by a dialog model and the transitions between different dialogs. These transitions could be invoked by simple

interactions such as a window manipulation [24]. In an ambient intelligent environment things are more complicated however: the *physical location of the user interface parts* differs from one dialog to another in contrast with a single-device system where a dialog is always represented on the same device. The design of such a system should make sure the cognitive burden of making a transition is minimized while supporting the tasks and goals of the user. Denis and Karsenty describe a set of design principles to ensure inter-usability in a multi-device environment [6]: inter-device consistency, transparency and adaptability of device usage. In this paper we focus on the first principle to support task set transition continuity. Inter-device consistency is composed out of four levels: perceptual (appearance and structure), lexical (labeling), syntactical (operations) and semantic (service functionality) consistency. The former two levels, perceptual and lexical consistency, are provided by the presentation model that is used. The latter two levels, syntactical and semantic consistency, can be enforced by defining a set of constraints in the environment model as shown in the previous section. The support of these types of consistency levels inside the different models contributes to a better continuity while making the transition from one ETS to another.

3.4 Task Representations

Each interaction task from the task specification should be presented in the environment one way or another so the user can interact with it. In particular the 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 $x \in \{X_1, X_2, ..., X_n\}$ can be retrieved, the set of related user interface descriptions is referred to as the *presentation model*. Based on the findings in related research (of which an overview can be found in [12]), a user interface description is specified using an XML-based notation. Figure 1 shows how the high-level user interface descriptions of *all* tasks available in an ETS are distributed among different appropriate interaction resources available in the environment while the user continues her/his interaction with the application, from one active ETS to the other.

For each ETS there are different possibilities of how the user interface representing the set of tasks can be divided. In [26] we showed a method that uses XHTML as the presentation language and a set of rules and a cost function to select the "preferred" distribution configuration among all possible configurations. The XHTML document was subdivided according the tasks it supported, and the different parts were distributed among the available devices in the neighborhood of the user.

4 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. Figure 2 shows this scenario. In [2] we presented an approach to take these kind of context switches explicitly into account in the task and the dialog model. A decision task can be inserted in the task model: this type of task allows a designer to specify a set of rules that can select an alternative task set to execute according to the context of use. This approach allows us to insert a decision rule in the task specification that will select another task set when the device is out of range. How this works and what the effects on the dialog model are is described into detail in [2]

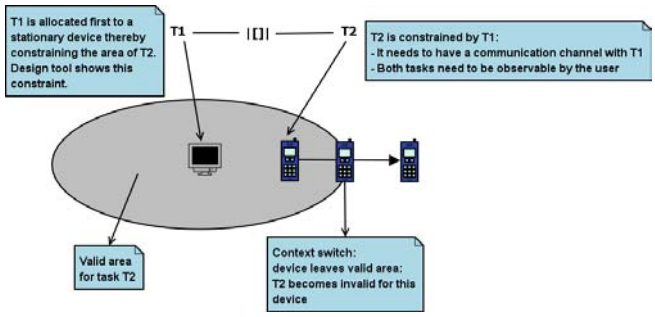


Fig. 2. Location constraint example for the task specification $T_1 \,|\, [] \,|\, T_2$.

To support this kind of reasoning for a task-centered approach we need to extend the semantics of the task specification with new constraints besides the temporal constraints and hierarchical structure. More precisely: we need to relate the context of use and the task set in terms of constraints over the task distribution behavior. Elaborating on the example of figure 2, where there are two tasks that can be executed in parallel and exchange information while performing ($T_1 \,|\, [] \,|\, T_2$,) 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 T_2 has to be located in the predefined area of the device presenting T_1. Notice T_1 and T_2 belong to the same ETS, since they can be executed during the same period of time. This example is equally valid for the construction $T_1 \,|\, [] \,|\, T_2 \,|\, [] \,|\, \cdots \,|\, [] \,|\, T_n$ ($T_1, \cdots, T_n$ belong to the same task

set), but the number of constraint checks involved to evaluate a distribution configuration for all tasks increases to $\binom{n}{2}$ in this situation. If the number of areas increases to m, the number of constraint checks increases exponentially since there are now $\binom{n}{m}$ possible combinations. The number of possibilities that a designer would have to check by hand is not feasible without any tool support. Our approach allows to visualize these constraints and automatically define valid task distribution configurations according to the task specification.

The example in the previous paragraph focused on a typical intra-ETS relation: a relation between two tasks in the same ETS. It is sufficient to take this into account for a distribution configuration for a single ETS. In other cases however, similar concerns arise. For example, the construction T_1 `[]>>` T_2 `[]>>` T_3 `[]>>` $\cdots$ `[]>>` T_n implies that every task is in a separate ETS, but still requires each task $T_{i,1\leq i,\leq n-1}$ to exchange information with its successive task $T_{i+1,1\leq i,\leq n-1}$. These types of relations have to be taken into account for the possible migration paths between task sets.

5 The MoDIE Platform

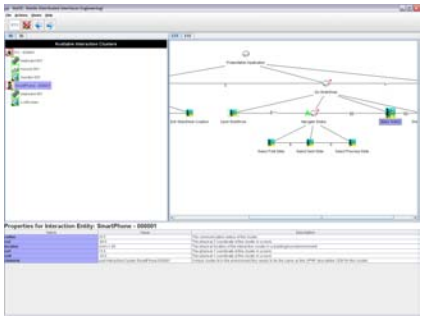
The models discussed in the previous section are all integrated by the MoDIE platform, a platform that supports a user interface design process for ambient intelligent environments. The central model is the *task model*, describing the set of tasks the (ubiquitous) application supports. Other models include the environment model that lists all available interaction resources in the environment of the user, a dialog model containing the ETSs derived from the task model, a presentation model that can be 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 in MoDIE offers direct manipulation of these different models and visualizes the relations between different models appropriately. Figure 3(c) shows an environment view combined with a task view that allows to assign tasks to interaction resources.

The environment model in MoDIE can be used in two different manners: statically and dynamically. The former implies the user interface designer defines a custom list of interaction resources, the latter implies that this list is created automatically by using off-the shelf service discovery protocols (currently MoDIE supports UPnP[1]). In both approaches the location or range of operation of a device is included in the environment model. We extended the UPnP discovery mechanism to retrieve this information if possible. With the support for a realistic environment model in place we can design, test and change a task specification for different environment models.

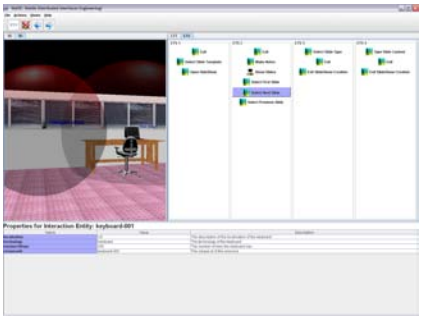
Tasks can be related with interaction resources of the environment model in two ways:

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

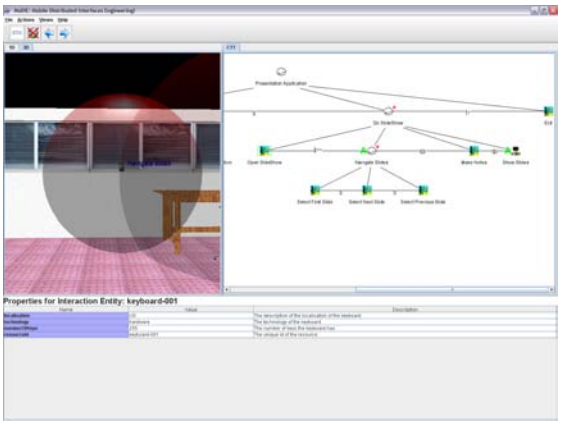
[1] http://www.upnp.org



(a) View on the Interaction Resources of the environment model



(b) Visualizing a distribution configuration for a task set



(c) Allocating tasks from a task specification to devices in an ambient intelligent environment

Fig. 3. Different views of the MoDIE tool.

2. Manually: usually, there are a number of solutions that are valid w.r.t. the constraints defined by the different models. MoDIE supports manual editing of the task allocations (which actually presents the task-environment interrelation): the designer can relate tasks with interaction resources and observe the effects of these changes.

Constraints on properties of the interaction resources that are available in the environment model can be checked by using XPath queries, since the environment model is expressed as an XML document. The XML document is based on the CoDAMoS ontology [22] and can be constructed and processed by our

MoDIE tool. A constraint check on a property of an interaction resource can be translated in an XPath query that is executed on this XML document. Part of the XPath query is just an implementation of the projection function of section 3.2 which maps an interaction resource property on a value in the domain of this property. Another part of the XPath query reflects the condition on the value of this property. Listing 1.1 shows an excerpt of the environment model. Other properties such as the physical location of an interaction resource in a room inside a building can also be included in the environment description.

Listing 1.1. An MoDIE environment model description.

```
<modie:interactionCluster
   xmlns:modie="http;//research.edm.luc.ac.be/modie/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:codamos_ont="http://edm.luc.ac.be/codamos#">
 <modie:interactionResourcesList>
   <modie:interactionResource owlClass="LcdScreen">
      <modie:propertyList>
        <modie:property type="xsd:string"
            name="resourceId" unit="false">
          <modie:propertyValue>lcd-002</modie:propertyValue>
        </modie:property>
        <modie:property type="xsd:int"
            name="width" unit="true">
          <modie:propertyValue>4</modie:propertyValue>
          <modie:propertyUnit>cm</modie:propertyUnit>
        </modie:property>
        <modie:property type="xsd:int"
            name="height" unit="true">
          <modie:propertyValue>12</modie:propertyValue>
          <modie:propertyUnit>cm</modie:propertyUnit>
        </modie:property>
        ...
    </modie:interactionResource>
    ...
  </modie:interactionResourcesList>
  ...
</modie:interactionCluster>
```

An important aspect of the MoDIE design tool will be the possibility to simulate the run-time behavior of the distributed user interface. This simulation is considered as a view on the different models that are built with the MoDIE support tool and is integrated with the other views. The simulation creates a 3D model of the environment model (using the Java3D API[2]), and uses the list of interaction resources to dynamically render the user environment. A simulation module aids in defining the appropriate Task Migration Paths. Figure 3(a) shows the MoDIE combined view of the environment model and the dialog model (expressed as a set of task sets). By moving the mobile device away from the

[2] http://java.sun.com/products/java-media/3D/

desktop computer the designer can see what kind of transitions are invoked and how the design fits in the simulated situation.

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.

6 Integration with UML-based Software Engineering

This section details a mapping of the previously introducted concepts to UML 2.0 [18] and how this mapping can be used to combine the approach with model-driven engineering. We made the choice to use UML 2.0 in order to have a rigorous description of the different aspects of interactive software based on a proven technology. The fact that is has better facilities for model-driven development which can bring a boost to design methodologies based on a diverse set of models is promising.

An integration with the UML modeling languages is important for several reasons: first of all it makes aids in bridging the gap between the HCI designer and the software developer. In particular for the complex domain described in this paper (ambient intelligent environments) there is currently no support to integrate the design of application logic with the interaction design. Besides the fact UML is widely accepted, it also offers a more formal way to describe the functionality of a system and it provides the tools to relate a user interface with the functionality that is represented by this user interface.

6.1 Mapping to UML 2.0

To represent the architectural aspects of a distributed user interface, we propose the use of UML 2.0 deployment diagrams. The deployment diagrams can be used to describe some features in more detail, which cannot be seen easily in the 3D view. One part of the architecture is the communication channels that are available between the different interaction clusters. Another is the composition of an interaction cluster; which interaction resources are contained in the cluster. Traditionally, the deployment diagram is a static diagram, but in the current setting this diagram depends on the context-of-use. Section 5 introduced the dynamic discovery of available interaction clusters: the result of such a discovery can be visualized as a deployment diagram. 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 UML 2.0, where the former can contain other Devices (interaction resources) and

the latter cannot, due to the definition of an interaction resource (see section 1). A part of a user interface can be allocated to a certain device by specifying the deployment of an Artifact to a specific node. Specific stereotypes can be used to define the kind of Artifact that is used. In the early stages of design, these stereotypes can be used to denote the function of the user interface part (*input*, *output*, or *action*; triggering of functionality).

In figure 4 the scene and allocations from figure 1 are represented using the UML deployment diagram. In this diagram, interaction clusters and interaction resources are represented by Nodes, while the physical representation of the task, the user interface through which the task can be performed, is depicted as an Artifact. An integration of this approach into MoDIE allows the designer to get adapted representations of the deployment diagram for the different possible situations. The use of stereotypes and the associated tagged values can allow adapted representations in standard UML modeling tools, should MoDIE support saving configurations manually or semi-automatically selected by the designer into XMI [9]. The advantages and disadvantages of both approaches are currently under investigation.

The earlier mentioned Artifacts can be stereotyped to denote the kind of user interface parts they represent. Three types of parts are identified: *input*, *output* and *action*. An *input* part can also contain labels, drawings or sound that is necessary or aids in the understanding of what information should be put into the system. Selection from a non-empty set of options is also considered input. An output part is a part of the user interface that shows information provided by the application core, including all relevant labels etc. An *action* is a part of the user interface that is responsible for triggering functionality in the application core. Note that *action* and *input* are not mutually exclusive.

Artifacts can optionally be linked to the components or classes that are used to realize them, using the standard UML 2.0 *manifest* relationship to define the implementation of the user interface. We thus propose that the physical (and logical) structure of the user interface is defined using Artifacts, while the structure of the implementation is defined using the well-known UML structures as class diagrams or component diagrams. When the user interface is rendered based on the related presentation specifications, the realisation of the user interface can be done based on the Artifacts structure.

6.2 Model-driven engineering

Abstract presentations can be derived from the tasks/task sets. These can be converted to concrete, albeit high-level description based on marks (stereotypes) made to the abstract models and context information (e.g. user profiles). The resulting concrete representation can be similar to the notation used for Canonical Abstract Prototypes [4], possibly including an allocation to interaction resources.

The use of 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. The use of model-driven development starting

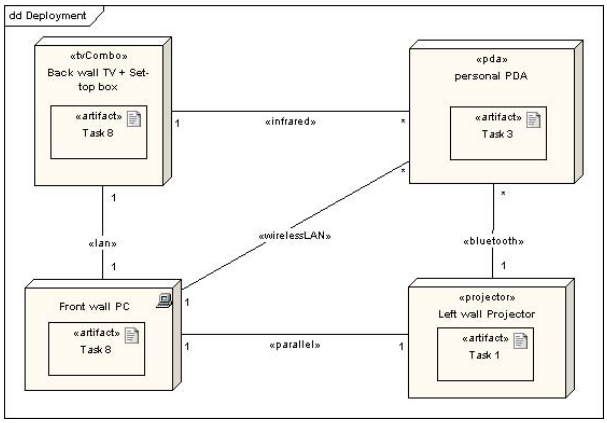


Fig. 4. Deployment diagram describing the task allocation of figure 1

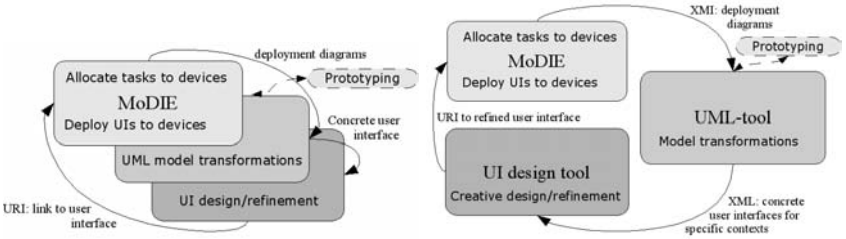
from a high-level (platform independent) model that is refined through one, or possible multiple, transformations into a concrete user interface to drive or guide the design of a user interface can offer many benefits.

An example can be the realization of a wizard-based user interface on a kiosk system for American market versus a single form on a wall-sized screen operated by his smart phone for the Chinese market when both user interfaces could be used to perform the same tasks. At the highest level, both interfaces are represented using the same set of user interface components. Using multiple transformations, this high-level model is gradually translated into concrete models. This can be done by applying HCI design patterns or best practices based upon contextual information.

We envision two possible tool configurations for integration of MoDIE with model driven engineering to create a complete environment for the design of distributed user interfaces. In both configurations MoDIE provides the task-based distribution facilities based upon the discovery of interaction devices and resources in the neighbourhood and designer input. MoDIE works at the task-level and relies on the linking of tasks to (declarative) user interface descriptions (using URI's) 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, as can be seen in figure 5(a). In this configuration, the distribution created in the MoDIE tool is passed to a separate part of the tool that works with UML and starts from a deployment specification as discussed in the previous section. Starting from this model, several transformations will be made that gradually transform the abstract model into different concrete models, specifying concrete user interface configurations for certain hardware configurations. These concrete models can then be translated into XML-based user interface descriptions of which the URI's can be delivered back into the MoDIE tool to create an actual deployment.

The second configuration splits the total functionality over three parts (see figure 5(b)): MoDIE (1) for task distribution, once this task distribution is created, it is delivered in XMI-format to an UML-tool (2) supporting model driven engineering (MDE). After one or more transformations, the abstract representation that was given to the UML-tool is translated into one or more concrete models (of user interfaces) and is delivered to a user interface modeling tool that provides the look-and-feel to the user interfaces through the use of constraints and styles.



(a) Integrated environment (b) Separate tools

Fig. 5. MoDIE tool and model driven engineering.

7 Conclusions and Future Work

In this paper we investigated the requirements to support task-centered user interface design for ambient intelligent environments, where the user interface suppporting the tasks of the user is distributed among the available devices. We introduced MoDIE, a system that uses a task-based approach to design user interfaces for ambient intelligent environments. MoDIE allows a designer to combine a presentation and task model smoothly with an environment model. It visualizes task allocations in an environment and supports the design process by visualization and simulation techniques. User interface completeness (is the required functionality to reach the user's goals accessible?) and continuity (can we create a usable user interface for a dynamic environment) are the two main properties that are considered here. Both the visualization of the task allocations in the environment and the simulation of the execution of a task specification are the primary tools to ensure completeness and continuity.

Although there are several theoretical frameworks for determining the influence of device switching on the usability of a system, there is no support for a designer to apply these frameworks while designing a multi-device user interface. Further research is necessary to use these frameworks in tools that can

visualize the effects of certain design decisions on the usability of the system. It is clear there are an overwhelming number of aspects that need to be taken into account to use MBUID for ambient intelligent environments. Traditional MBUID approaches do not take dynamic environments with different devices that can be used in parallel into account. This work contributes to a solution for this problem by investigating the issues that are specific for the design of user interfaces for these type of environments and structuring them so they can be incorporated in design tools and user interface generators.

8 Acknowledgments

The authors would like to thank Geert Houben, Frederik Winters and Tim Clerckx for co-developing the software supporting the ideas of this paper.

Part of the research at EDM is funded by EFRO (European Fund for Regional Development), the Flemish Government and the Flemish Interdisciplinary institute for Broadband technology (IBBT). The CoDAMoS (Context-Driven Adaptation of Mobile Services) project (IWT 030320) is directly funded by the IWT (Flemish subsidy organization).

References

1. Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz, and Gaelle Calvary. CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In Markopoulos et al. [14], pages 291–302.
2. Tim Clerckx, Kris Luyten, and Karin Coninx. DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In *The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems*, 2004.
3. Karin Coninx, Kris Luyten, Chris Vandervelpen, Jan Van den Bergh, and Bert Creemers. Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In Luca Chittaro, editor, *Mobile HCI*, volume 2795 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2003.
4. Larry L. Constantine. Canonical abstract prototypes for abstract visual and interaction design. In *Proceedings of DSV-IS 2003*, number 2844 in LNCS, pages 1 – 15, Funchal, Madeira Island, Portugal, June 11-13 2003. Springer.
5. Joëlle Coutaz, Christophe Lachenal, and Sophie Dupuy-Chessa. Ontology for Multi-surface Interaction. In Matthias Rauterberg, Marino Menozzi, and Janet Wesson, editors, *Human-Computer Interaction INTERACT '03: IFIP TC13 International Conference on Human-Computer Interaction*.
6. Charles Denis and Laurent Karsenty. *Inter-Usability of Multi-Device Systems – A Conceptual Framework*, pages 373–385. Wiley, 2004.
7. Jacob Eisenstein, Jean Vanderdonckt, and Angel R. Puerta. Applying model-based techniques to the development of uis for mobile computers. In *Intelligent User Interfaces*, pages 69–76, 2001.

8. Nikolaos Georgantas and Valérie Issarny. User activity synthesis in ambient intelligence environments. In Markopoulos et al. [14], pages 45–50.
9. Object Management Group. Omg xml metadata interchange. Object Management Group, WWW, http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf, 2002.
10. Thomas Heider and Thomas Kirste. Supporting Goal-Based Interaction with Dynamic Intelligent Environments. In Frank van Harmelen, editor, *ECAI*, pages 596–600. IOS Press, 2002.
11. Anders Larsson and Erik Berglund. Programming ubiquitous software applications: requirements for distributed user interface. In Frank Maurer and Günther Ruhe, editors, *SEKE*, pages 246–251, 2004.
12. Kris Luyten, Marc Abrams, Quentin Limbourg, and Jean Vanderdonckt, editors. *Developing User Interfaces with XML: Advances on User Interface Description Languages*, 2004.
13. Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt. Derivation of a Dialog Model for a Task Model by Activity Chain Extraction. In J. A. Jorge, N. J. Nunes, and J. F. e Cunha, editors, *DSV-IS*, volume 2844 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2003.
14. Panos Markopoulos, Berry Eggen, Emile H. L. Aarts, and James L. Crowley, editors. *Ambient Intelligence: Second European Symposium, EUSAI 2004, Eindhoven, The Netherlands, November 8-11, 2004. Proceedings*, volume 3295 of *Lecture Notes in Computer Science*. Springer, 2004.
15. Mieke Massink and Giorgio P. Faconti. A reference framework for continuous interaction. *Universal Access in the Information Society*, 1(4):237–251, 2002.
16. Giulio Mori, Fabie Paternò, and Carmen Santoro. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Software Eng.*, 30(8):507–520, 2004.
17. Giulio Mori, Fabio Paternò, and Carmen Santoro. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Trans. Softw. Eng.*, 28(8):797–813, 2002.
18. Object Management Group. *UML 2.0 Superstructure Specification*, October 8 2004.
19. Giorgio P. Faconti and Mieke Massink. Continuity in Human Computer Interaction. In *CHI 2000 Workshop report*. `http://www.acm.org/sigchi/bulletin/2000.4`, 2000.
20. Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.
21. Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Ubicomp 2001: Ubiquitous Computing, Third International Conference Atlanta, Georgia, USA, September 30 - October 2, 2001, Proceedings*, Lecture Notes in Computer Science, pages 56–75. Springer, 2001.
22. Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koenraad De Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In Panos Markopoulos, Berry Eggen, Emile H. L. Aarts, and James L. Crowley, editors, *Ambient Intelligence: Second European Symposium, EUSAI 2004, Eindhoven, The Netherlands, November 8-11, 2004. Proceedings*, pages 148–159, 2004.
23. Anthony Savidis, Napoleon Maou, I. Pachoulakis, and Constantine Stephanidis. Continuity of interaction in nomadic interfaces through migration and dynamic

utilization of I/O resources. *Universal Access in the Information Society*, 4(1):274–287, 2002.
24. Jean Vanderdonckt, Quentin Limbourg, and Murielle Florins. Deriving the Navigational Structure of a User Interface. In M. Rauterberg and J. Wesson, editors, *Proceedings of 9th IFIP Conf. on Human-Computer Interaction Interact'2003 (Zrich, 1-5 September 2003)*, pages 455–462, 2003.
25. Chris Vandervelpen and Karin Coninx. Towards model-based design support for distributed user interfaces. In *Proceedings of the third Nordic Conference on Human-Computer Interaction*, pages 61–70. ACM Press, 2004.
26. Chris Vandervelpen, Geert Vanderhulst, Kris Luyten, and Karin Coninx. Lightweight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In *5th International Conference on Web Engineering (ICWE'2005)*, 2005. `http://research.edm.luc.ac.be/cvandervelpen/research/icwe2005/`.
27. M. Weiser. The Computer for the 21st Century. In *Scientific American*, 1991.