DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development

Tim Clerckx, Kris Luyten, and Karin Coninx

Limburgs Universitair Centrum – Expertise Centre for Digital Media
Universitaire Campus, B-3590 Diepenbeek, Belgium
{tim.clerckx,kris.luyten,karin.coninx}@luc.ac.be

Abstract. The last few years a lot of research efforts have been spent on user interfaces for pervasive computing. This paper shows a design process and a runtime architecture, DynaMo-AID, that provide design support and a runtime architecture for context-aware user interfaces. In the process attention is focused on the specification of the tasks the user and the application will have to perform, together with other entities related to tasks, like dialog and presentation. In this paper we will show how we can model tasks, dialogs, and presentation when the designer wants to develop context-sensitive user interfaces. Besides the design process, a runtime architecture will be presented supporting context-sensitive user interfaces. Pervasive user interfaces can change during the runtime of the interactive application due to a change of context or when a service becomes available to the application. We will show that traditional models like task, environment and dialog model have to be extended to tackle these new problems. This is why we provide modeling and runtime support solutions for design and development of context-sensitive user interfaces.

keywords: model-based user interface design, pervasive user interface, context, design process, runtime architecture, task model, service.

1 Introduction

There is a continuing and growing interest in designing user interfaces for mobile computing devices and embedded systems. This evolution is driven by a very fast evolving hardware market, where mobile computing devices like Personal Digital Assitants (PDAs) and mobile phones are getting more powerful each new generation. The mobile nature of portable devices and the increasing availability of (wireless) communication with other resources require applications that can react on context changes. When we talk about context and context-aware applications, we mean applications that can adapt to environmental changes, like the change of platform, network capabilities, services that become available and disappear or even physical conditions like light intensity or temperature. In [8], Hong states there are several goals why context-aware computing is interesting to achieve. Advancing development of context-aware computing gives incentives to:

- increase the amount of input channels for a computer;
- gather implicit data;
- create more suitable models for the input;
- use the previous elements in useful ways.

To create consistent adaptive user interfaces (UI), UI developers should consider adaptivity in early design stages. When using the model-based approach in the design phase some problems can be identified: traditional models, like a task model and a dialog model are static and not suited to adapt to context changes. This paper shows how designers can take adaptability of the UI in consideration by extending these traditional models to support design of context-sensitive user interfaces.

In previous work [3] we have shown how a modified task notation can be used in order to design context-sensitive user interfaces for static context. Our former approach limited the influence of the context upon the different models in time. The context was sensed when the UI was deployed and started on the target device. From that moment on no context changes were taken into account. In this paper we extend this method to design and provide runtime support for user interfaces that can be affected by dynamic context changes. With dynamic context changes we do not only take into account the target platform, network properties and other environmental conditions. We also seek a way to consider how we can design a UI for a service. How to cope with this service when it becomes available to the application on the portable device of the user is an important issue and the main contribution of this paper.

According to [5], a **service** is "*a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications*". An example of a service is a software component, running on a particular device, offering access to some functionality it provides (e.g., a surveillance camera can "export" its output video stream, zoom and focus functions). A service offers functionality that should be used in conjunction with other application logic. Arbitrary clients can connect to this service and make use of the exported functionality.

The next section shows existing Model-Based User Interface Development approaches that support context changes in different ways. In section 3 we discuss our own design process, **DynaMo-AID** (**Dyna**mic **Mo**del-b**A**sed user **I**nterface **D**evelopment), to develop context-sensitive user interfaces that support dynamic context changes. DynaMo-AID is part of the Dygimes [4] User Interface Creation Framework. Section 4 introduces a runtime architecture to support user interfaces created with the DynaMo-AID process. Afterwards a genuine case study will be shown in section 5 to illustrate the practical use of DynaMo-AID. In this paper we show how the DynaMo-AID process is supported by the appropriate design tools. Finally the paper is concluded with a discussion of the obtained results and a description of the future work.

2 Related Work

The current literature shows a growing interest in the creation of context-sensitive user interfaces. During the last few years we see more interest in defining and exploiting context information on several levels of the UI conception. The primary goal of most initiatives is to more flexibly design user interfaces, with increasing design/code reusability resulting in user interfaces that become more usable in different contexts of use.

The different levels for introducing context information can be summarized as follows. First, the task model can be made dependent on the context, as shown in [15,21]. Next, at the dialog level navigation can be dependent on the context e.g. allowing navigation to take place in a multiple-device setting where the user can take advantage of multiple devices or settings in the same time span [3,28]. Finally at the presentation level context information can be considered to choose the most appropriate widgets, as in [27,14]. Notice we consider the user model to be part of the context information. In this work we will allow to integrate context on different levels of the user interface design and creation like shown in the next sections.

Calvary et al. [2] describe a development process to create context-sensitive user interfaces. The development process consists of four steps: creation of a task-oriented specification, creation of the abstract interface, creation of the concrete interface, and finally the creation of the context-sensitive interactive system. The focus however, lays upon a mechanism for context detection and how context information can be used to adapt the UI, captured in a three-step process: (1) recognizing the current situation (2) calculating the reaction and (3) executing the reaction. In our approach we will focus on the exposure of a complete design process using extended versions of existing models, and how context reflects on these models. Furthermore we extend context by taking into account the effects of incoming and abolished services.

Mori et al. present a process [15] to design device-independent user interfaces in a model-based approach. In this approach, a high-level task model is constructed to describe tasks that can be performed on several platforms. Afterwards, the designer has to specify which tasks of the high-level description can be performed on which device. When this is done, an abstract UI will be created followed by the UI generation. In our approach we describe the differences between target platforms in one complete task model and provide the possibility to take into account other sorts of context information than platform.

In the next sections we integrate several solutions to build context-sensitive user interfaces into one process with appropriate tool support for this process. To our knowledge there is no other initiative trying to combine context-information on the different levels of model-based user interface development. The distinct parts of this process will be presented separately.

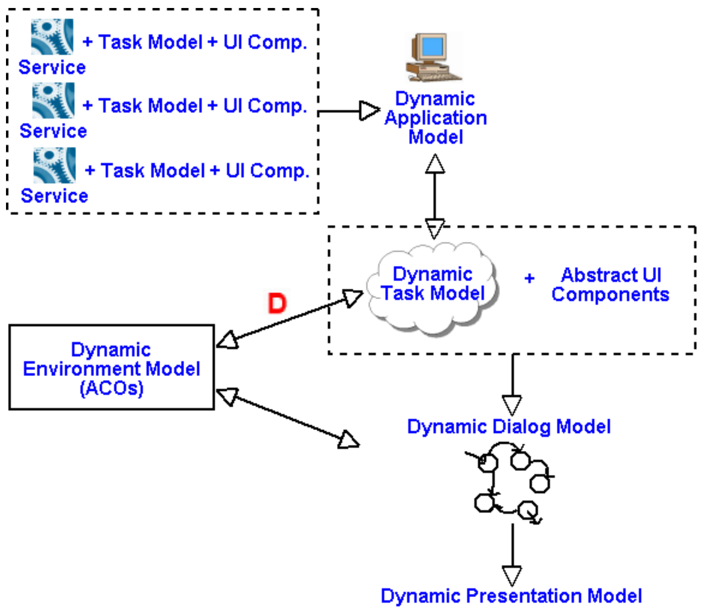


Fig. 1. The DynaMo-AID Design Process.

3 The DynaMo-AID Design Process

The main goal is to create a process that enables the user interface designer to create user interfaces for pervasive systems. Since pervasive interfaces have a strong link to the information provided by their direct environment, these interfaces should be capable to evolve according to the context changes initiated in their environment. Figure 1 gives an overview of the DynaMo-AID Design Process. In this process the designer can specify the interaction by constructing and manipulating abstract models because at design time it may be unknown for which environments (available hardware and software services, physical environment, target user,…) the UI will be rendered.

The models used in our process try to enhance the ones commonly used in Model-Based User Interface Design [20]. This is why extra attention is payed on the representation and semantics of these models: we will investigate how expressive traditional existing models are, and where they need to be extended for pervasive systems. For this purpose a "meta" model is introduced: the *Dynamic Model* is a model that can change at runtime in a way that the model can be merged with another model from the same type (e.g. attaching subtrees to an existing tree) or parts of the model can be pruned. This way the *Dynamic Model* can be seen as a dynamic extension of *Interface Model*, as introduced in [22]. The Interface Model exists out of the set of relevant abstract models (task, dialog, domain, user,…) necessary to describe the interface of a system.

In the DynaMo-AID Design Process there is a difference between the main application, for example running on a PDA or a cell phone, and services (applications that provide a service and an interface) that can be encountered during the runtime of the interactive application. Services have to be modelled separately from the design of the main application.

In summary, the DynaMo-AID Design Process consists of the following steps:

1. constructing the Dynamic Task Model for the main application (section 3.1).
2. attaching abstract descriptions to the unit tasks[5] of the Dynamic Task Model. Platform-independent high-level user interface components are connected with these leaf tasks similar as we have shown in previous work [4,13,3].
3. calculation of the *ConcurTaskTrees Forest*. This is the collection of ConcurTaskTrees describing the tasks to be performed for each common occurence of context during the runtime of the main application. For uncommon occurences of context, these tasks have to be specified as a service.
4. automatic extraction of the dialog model for each ConcurTaskTree in the ConcurTaskTree Forest.
5. construction of the atomic dialog model by the designer. This dialog model consists of the subatomic dialog models created in the previous step and contains all transitions that may occur during the runtime of the main application, triggered by an action of the user, the application or even a change of context (section 3.2).
6. linking context information to the task and dialog model through abstract context objects (section 3.3).
7. modeling the services: accomodate each service with a task tree describing the tasks user and application can perform when they are able to use the service (can be done anywhere in the process and services can be used by different applications)

This process enables us to design context-sensitive user interfaces and supports fast prototyping. It enables us to create a prototype presentation using the methodology we introduced in [4]. This will be further explained in section 3.5. This design process demands further explanation. This is why the Dynamic Models will be separately discussed in the following subsections.

3.1 Dynamic Task Model

To specify tasks we use a modified version of the ConcurTaskTree notation, introduced by Fabio Paterno [17]. This notation offers a graphical syntax, an hierarchical structure and a notation to specify the temporal relations between tasks. Four types of tasks are supported in the CTT 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

[5] A unit task that can not be devided in subtasks any further. In a ConcurTaskTree specification these are the leaf tasks [21]

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 independency (|=|).The support for concurrent tasks is very valuable because of our envisioned target: pervasive systems where users can transparently interact with the (embedded) computing devices in their environment. Some tasks can be supported by multiple devices, thus concurrent usage of these different resources should be supported in the task design notation. In the remainder of this paper we will make extensive use of "Enabled Task Sets" (ETS). An ETS is defined in [17] as:

> *a set of tasks that are logically enabled to start their performance during the same period of time.*

To link abstract information about how a task can be performed by an actor (user or application), we attach platform-independent high-level user interface components to these leaf tasks [13,3]. This way all possible user interfaces are covered by a complete annotation of the task specification.

Several approaches that use the ConcurTaskTrees Notation [17] exist for modelling context-sensitive human-computer interaction. In [18], Paternò and Santoro show how ConcurTaskTrees can be used to model user interfaces suitable for different platforms. Pribeanu et al. [21,26] proposed several approaches to integrate a context structure in ConcurTaskTrees task models. The main difference in our approach is the support for runtime context-sensitivity introduced in the different models.

In order to make a connection with the dynamic environment model we choose the approach described in [3] where *decision nodes*, denoted by **D**, collect distinct subtrees from which one of them will be selected at runtime according to the current context of use. To link the dynamic task model with the dynamic environment model and to gather information about a suitable presentation of the UI, decision nodes are coupled to Abstract Context Objects (section 3.3). We can summarize it here as follows. The decision nodes notation enables to specify task models that describe the tasks (1) a user may have to perform in different contexts of use and (2) where tasks that are enabled by new incoming services will find there place in the task model. To obtain this, services are accompanied by a task description as a formal description for the goals that can be accomplished through their use. Figure 5 shows a decision tree where "Use ImogI" is a decision node where a distinction in tasks is made between the use of a mobile application inside or outside a certain domain.

3.2 Dynamic Dialog Model

A dialog model describes the transitions that are possible between user interface states. Although transitions usually are invoked by a user action or a call from the application core, in this case the current context is also an actor that can perform a transition.

To specify a dialog model, several notations are used: State Transition Networks [29], Dialogue Graphs [25], Window Transitions [28], Petri Nets [19],… The State Transition Network (STN) notation describes the dialog between user and application by defining states (including a start-state and possibly several finish states) of the UI and transitions between these states.

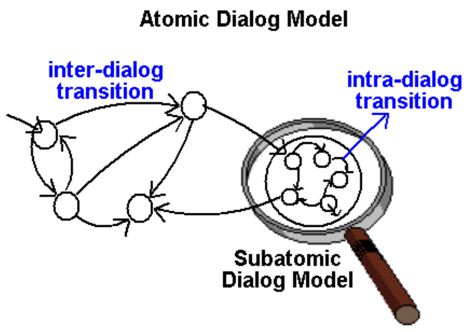


Fig. 2. Dynamic Dialog Model.

Puerta and Eisenstein [23] introduced the *mapping problem*: the problem of mapping abstract models (domain/task/data model) in model-based user interface design to more concrete models (dialog/presentation model). Limbourg, Vanderdonckt et al. [12,28] proposed several rules to derive dialog information from constrained ConcurTaskTrees task models (a parent task has exactly one child task). In [13] we have already shown it is possible to extract a dialog model automatically from a task model. We made use of the ConcurTaskTrees Notation to represent a task specification and the dialog model is structured as a STN. In this method, the states in a STN are extracted from the task specification by calculating the *enabled task sets* [17].

Because the context may change during the execution of the application, the dialog model becomes more complex. First, the dialog models can be extracted automatically from each possible ConcurTaskTree that may occur. Afterwards the designer can draw transitions, that can only be invoked by a context switch, between the dialog models. This way a dynamic dialog model is created. To express this approach, we introduce following definitions:

Definition 1 *An **intra-dialog transition** is a transition in a STN caused by the completion of a task through user interaction or by the application. Intra-dialog transitions connect enabled task sets from the same ConcurTaskTree. Transitions are triggered by the execution of a task, either by the user or by the application, and can be denoted by:* $(CTT_i, ETS_k) \xrightarrow{task} (CTT_i, ETS_l)$

Definition 2 *An **inter-dialog transition** is a transition in a STN caused by a context switch. Inter-dialog transitions connect enabled task sets from different ConcurTaskTrees of the same ConcurTaskTrees Forest and are triggered by a positive evaluation of a context condition. Inter-dialog transitions can be denoted by:* $(CTT_i, ETS_k) \xrightarrow{condition} (CTT_j, ETS_l)$

Definition 3 *A **subatomic dialog mode** is a STN containing the states and transitions from the same ConcurTaskTree. This means a subatomic dialog model is a regular STN, extracted from one ConcurTaskTree.*

Definition 4 *An **atomic dialog model** is a STN where the states are subatomic dialog models and the transitions are inter-dialog transitions between states of different subatomic dialog models.*

Figure 2 illustrates the definitions of *subatomic* and *atomic* dialog model. The subatomic dialog model is the classical dialog model where actions of user or system imply the transition to another state. When a context change occurs, this dialog model can become obsolete. As a result a transition to another subatomic dialog model takes place and an updated UI comes into play. Note that a context change can also invoke a system function instead of performing an inter-dialog transition (e.g. turning on the backlight of a PDA when entering a dark room). This explains the *invocation* arrow in figure 4 that connects dialog and application.

3.3 Dynamic Environment Model

Despite several efforts to describe context information and using it for interactive applications [2,7,24,11], it still is a challenging issue due to the lack of a standard and practical implementations.

Calvary et al. [1,2] introduce an *environment model* to be specified by designers for defining the current context of use together with the platform model. Furthermore the *evolution model* describes when a context switch takes place and defines the appropriate reaction.

Coutaz and Rey [7] define the *contextor*, a software abstraction of context data that interprets sensed information or information provided by other contextors. In this way a chain of contextors can be created to produce one logical component.

Salber et al. [24] describe a widget-based toolkit, the Context Toolkit, containing abstract widgets in order to:

- encapsulate rough context details to abstract context from implementation details (like the proxy design pattern);
- reuse widgets in different applications.

The Dynamic Environment Model (figure 3) represents context changes, and provides us with a model to react on these changes in an appropriate way. In contrast with other approaches, a service is also part of the environment in our model. Since a service offers (previously unknown) functionality that can integrate with the whole of the application, a more dynamic approach is neccessary here. This means calculated changes in the navigation through the interface should be supported. To explain the effect of the Dynamic Environment Model, some definitions are introduced here:

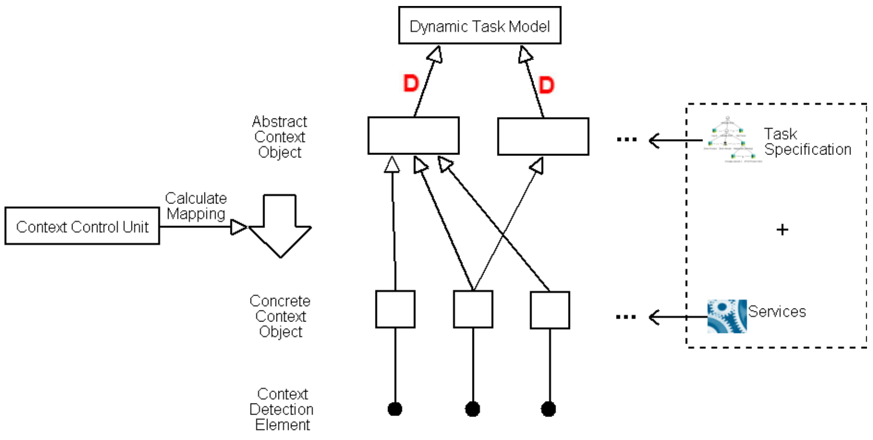


Fig. 3. Dynamic Environment Model.

Definition 5 *A* ***Concrete Context Object (CCO)*** *is an object that encapsulates entities (like low level sensors) that represent one sort of context.*

Definition 6 *An* ***Abstract Context Object (ACO)*** *is an object that can be queried about the context it represents.*

Different from the approach in [24] we separate the abstraction and encapsulation functions of a context widget. This is necessary because due to context changes, the number of available widgets can change on the abstract and concrete level. Moreover this separation allows to support context-sensitive user interfaces on the design level. First, a new service may introduce new abstract widgets (ACOs), linked to the accompanying task specification. Furthermore, a change of platform resources (e.g. moving into the reach of a wireless LAN may imply connection to a server and a printer) can give or take away access to CCOs. As a result, the mapping of an ACO to CCOs has to be repeated when the collection of ACOs or available CCOs changes.

This can be taken care of by defining mapping rules in order to select the appropriate CCOs currently available for each ACO used by the interactive application. The mapping function can be implemented by dividing CCOs into categories, and specify for each ACO the appropriate CCOs relevant to the abstract widget. The detection of context changes and the call to repeat the mapping is handled by the **Context Control Unit (CCU)** that is part of the runtime architecture (section 4).

To link the environment model to the task and dialog model, ACOs are attached to the decision nodes (section 3.1). For each subtree, a *query* is provided to denote which conditions have to be fulfilled by an ACO to select the subtree. In this way, when the atomic dialog model is constructed, the transitions can be marked with the correct ACOs and belonging queries.

Remark the analogy with abstract interaction objects (AIOs) and concrete interaction objects (CIOs) [27] used to describe user interface components in a platform independent way.

3.4 Dynamic Application Model

The functional core of the application does change when a service (dis)appears: this change influences the models. As stated before, services are accompanied with a task specification they support to provide a high-level description of the interaction that should be enabled when the service becomes available. When the designer wants the main application to update the UI at the time an unknown service becomes available, he/she has to reserve a decision node to specify where in the interaction a place is provided to interact directly with the service (e.g. the "Service"-task in figure 5).

When the service becomes available, the dialog and environment model also have to be updated. The atomic dialog model has to be extended with the new subatomic dialog models, provided by the task model attached to the service. Next, the environment model needs to be changed on two levels: (1) the new task model can provide new decision nodes. As a result new ACOs can be introduced, and these have to be mapped on the available CCOs. (2) the service can provide access new CCOs. In this case the CCU will also have to recalculate the mappings.

3.5 Presentation Model enabled for Fast Prototyping

During the design of the different models we support direct prototyping of the UI. Our system supports the automatic generation of the UI from the different models that are specified. For this purpose we start with calculating the ETSs from the annotated task model: each ETS is a node in the dialog model. One such node represents all UI building blocks that have to be presented to complete the current ETS (section 3 showed that UI building blocks were attached to unit tasks).

The designers (and future users) can try the resulting interface during the design process. Important aspects of the UI can be tackled in the design phase: improving navigation, consistency, layout and usability in general are done in an early stage. Tool support is implemented and presented in section 6. There is only limited support for styling the UI; enhancing the graphical "aesthetic" presentation is currently not supported in our tool.

4 The DynaMo-AID Runtime Architecture

To put a designed UI into practice, a runtime architecture must exist to support the results of the design process. [6] gives an overview of several software architectures to implement interactive software. Architectures based on SEEHEIM, ARCH, SLINKY and PAC make use of a dialog controller, to control the interaction flow between the presentation of the UI and the functional core of the interactive application. Because we present a runtime architecture where tasks and environment can change during the execution of the application (sections 3.3 and 3.4), the dialog controller is assisted in making decisions about dialog changes by the task controller and the Context Control Unit.

Figure 4 shows the DynaMo-AID runtime architecture. When the application is started, first the current context will be detected, and the applicable task model will be chosen before the UI will be deployed. Then the subatomic dialog model belonging to this task model will be set active and the start state of this model will be the first dialog to be rendered in the concrete UI. The context will be sensed by scanning the information provided by posing the queries in the ACOs.

From now on interaction can take place and the state of the UI can change due to three actors: the user, the application and the Context Control Unit (CCU).

The user interacts with the target device to manipulate the presentation. As a result, the dialog controller will perform an intra-dialog transition and update the presentation of the UI. The second actor is the application. The application core can also manipulate the UI (e.g. displaying the results of a query after processing). Also, an incoming service extends the application core and can carry a task model containing abstract user interface components. This is why the task controller will be notified with an update to modify the dialog model. It is obvious that an abolished service also implies an update of the task as well as the dialog model. The last actor that is able to change the state of the UI is the CCU, introduced in section 3.3.

The tasks of the CCU are:

1. **detection** of context changes: a context change will be detected by the CCU when an ACO throws an event.
2. **recalculation** of mappings from CCO to ACO: a service can also be a provider of context information and this is why, in that case, the service must be reachable for the CCU to recalculate ACO to CCO mappings. When the service is abolished, the CCU will also apply the recalculation.
3. **selection** of the current context-specific task model: the CCU will inform the Task Controller of the changed ACO and the Task Controller will return the current valid context-specific task model.
4. **execution** of inter-dialog transition (together with the dialog controler): using the appropriate context-specific task model, the dialog controller will be informed to perform an inter-dialog transition.

The next section will show how the runtime architecture and the design process can be of practical use.

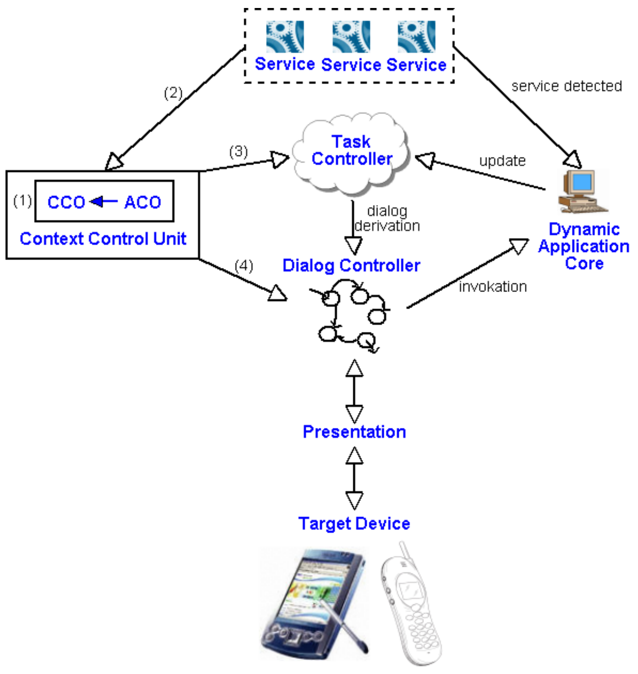


Fig. 4. The DynaMo-AID Architecture.

5 A Case Study

Within a few kilometres from our research department there is an open-air museum of 550 ha large. It contains a large collection of old Flemish houses and farms of the late 18th century, and allows the visitors to experience how life was in those days. Currently we are developing a mobile tourist guide "ImogI" for this museum, and use the methodology discussed above to create a usable context-sensitive interface for this application. The hardware setup is as follows: the visitor has a PDA with a GPS module as a touristic guidance system and museum artefacts are annotated with "virtual information" that can be sent to the guide once the tourist enters the artefacts range. The mobile guide contains a map of the museum and some information about the whereabouts of the artefacts; more detailled information is sent by the artefacts themselves (through a built-in system using bluetooth communication) to the mobile guide. This makes sure new artefacts can be placed at an arbitrary place in the museum without the guidance system becoming obsolete. The system depicted on the mobile guide is always up-to-date.

Figure 5 shows a simple ImogI task specification. On the first level of the task specification there are two context-dependencies expressed as decision nodes: the first one determines whether the user is inside or outside the domain. When the user is situated outside the museum premises, the application will act like a normal GPS navigation system. When the user moves into the open air museum, the application transforms into a mobile guide and vice versa. The other decision node allows to attach new services that become available in the direct surroundings of the PDA. The former context information is obtained by a GPS module on the PDA. We are currently implementing the latter with Bluetooth. The task specification in figure 5 can anticipate visitors leaving the actual museum boundaries to explore the facilities outside the premises. Figure 6 shows how the resulting dialog specification supporting automatic detection of the context change looks like. The dashed arrows $(CTT_1, ETS_3) \xrightarrow{OutsidePremises} (CTT_2, ETS_4)$ and $(CTT_2, ETS_4) \xrightarrow{InsidePremises} (CTT_1, ETS_3)$ specifiy the transition between the different dialog models. An important remark is the designer must specify between witch ETSs of the different ConcurTaskTrees inter-dialog transitions can occur. This way the designer can preserve usability when the user is performing a task existing of several subtasks. For example, the user can be confused if the user interface suddenly changes when he or she is scrolling through a map or performing some other critical task. Notice the two dialog models are the result out of two different enabled task sets. A context change influences the task groupings, and by consequence influences the navigational properties of the interface. For this reason dialog specifications are considered separately for each context change. In our example, the ETS *E(CTT$_1$)* is followed by *E(CTT$_2$)*.

$$E(CTT_1) = \begin{cases} ETS_1 = \{Start\ ImogI\} \\ ETS_2 = \{Select\ Show\ Map, Select\ Find, Service, Back, Close\ ImogI\} \\ ETS_3 = \{Show\ Map, Service, Back, Close\ ImogI\} \\ ETS_4 = \{Search\ Sight, Service, Back, Close\ ImogI\} \\ ETS_5 = \{Show\ Sight\ Location, Service, Back, Close, ImogI\} \end{cases}$$

$$E(CTT_2) = \begin{cases} ETS_1 = \{Start\ ImogI\} \\ ETS_2 = \{Select\ Destination, Service, Close\ ImogI\} \\ ETS_3 = \{Calculate\ Path, Service, Close\ ImogI\} \\ ETS_4 = \{Show\ ImogI, Quit\ Map\ View, Service, Close\ ImogI\} \end{cases}$$

Our starting-point here is the support for dynamic extensible models to have better support for designing context-sensitive user interfaces. The case study here shows their use: the open-air museum can change the location of their information kiosks or add other artefacts without constantly updating the mobile guide. Information kiosks can communicate with the mobile guide and offer all kinds of services (photo publishing, extra information, covered wagon reservations,…). Figure 7 shows the task specification for the kiosk. This task specification will be integrated within the context-sensitive task specification. The transitions between the different dialog specifications are done similar with the previous example.

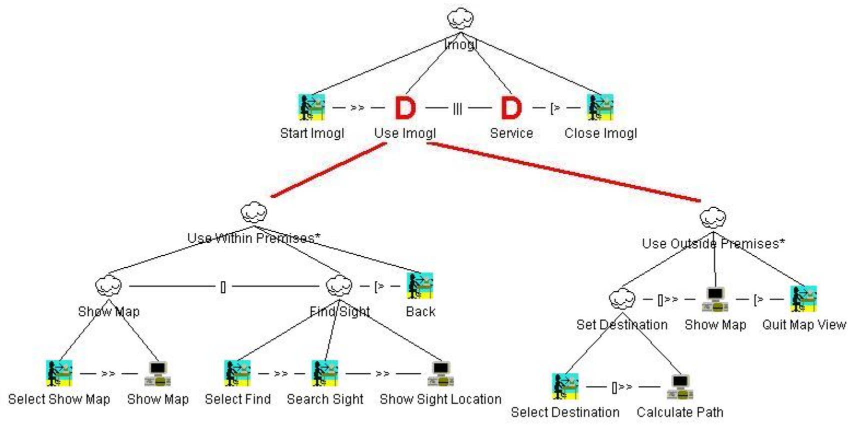


Fig. 5. ImogI Decision Tree

6 Tool Support

To test our approach we have implemented a limited prototype of the DynaMo-AID design process and runtime architecture using the Dygimes rendering engine. The DynaMo-AID tool (figure 8) *aids* to construct a context-sensitive task model [3], to attach abstract presentation information, and to construct atomic dialog models. The construction of the atomic dialog model by the designer supports automatic extraction of the subatomic dialog models belonging to all ConcurTaskTrees in de ConcurTaskTrees Forest.

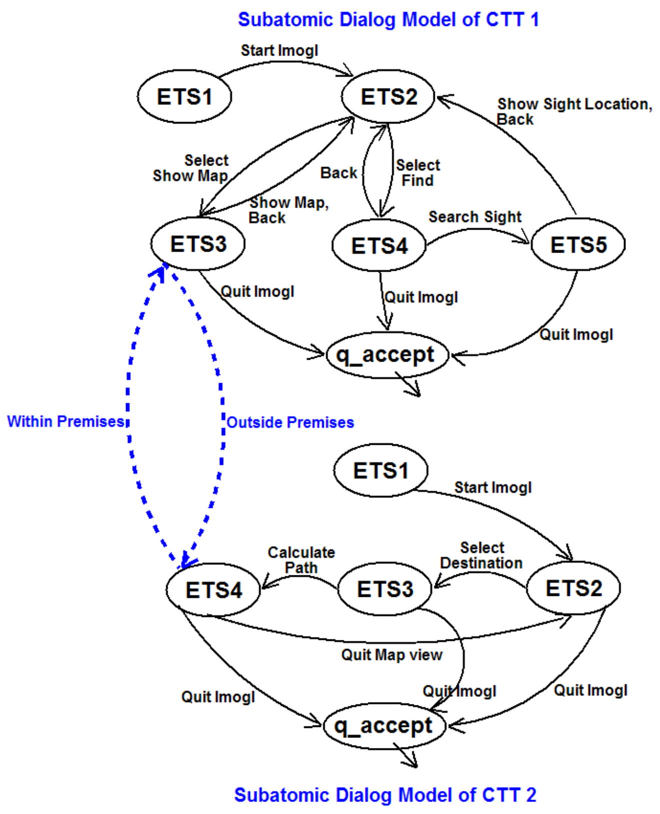


Fig. 6. ImogI Atomic Dialog Model.

After the modeling phase, a context-sensitive user interface prototype can be rendered. When the prototype is deployed, a control panel is shown where the user interface designer can manipulate context parameters. The designer can then see how a change of context reflects on the prototype.

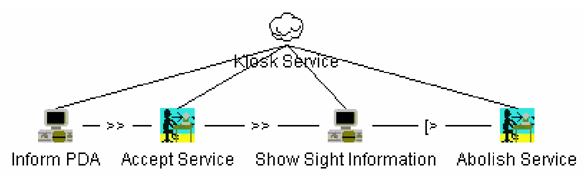
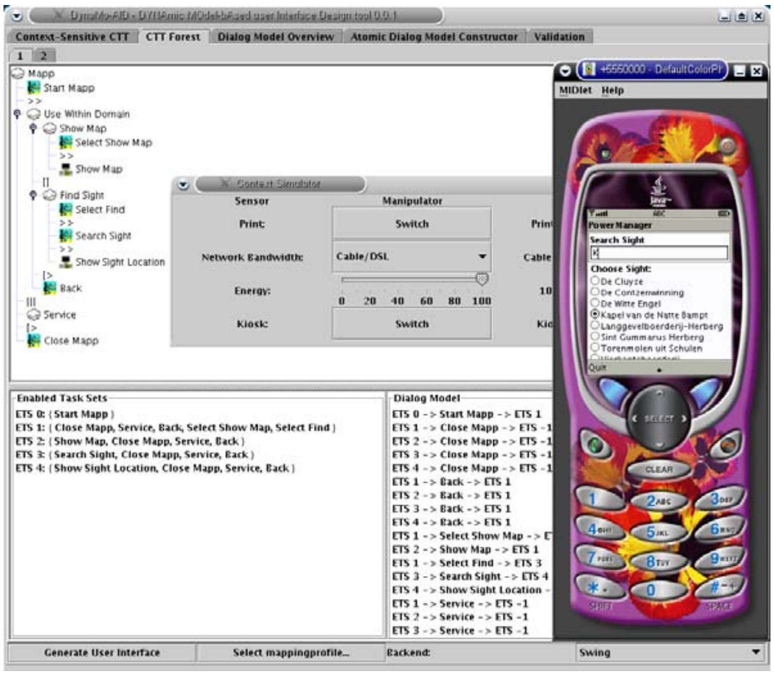


Fig. 7. Task Model attached to the Kiosk Service.



Enabled Task Sets

```
ETS 0: { Start Mapp }
ETS 1: { Close Mapp, Service, Back, Select Show Map, Select Find }
ETS 2: { Show Map, Close Mapp, Service, Back }
ETS 3: { Search Sight, Close Mapp, Service, Back }
ETS 4: { Show Sight Location, Close Mapp, Service, Back }
```

Dialog Model

```
ETS 0 -> Start Mapp -> ETS 1
ETS 1 -> Close Mapp -> ETS -1
ETS 2 -> Close Mapp -> ETS -1
ETS 3 -> Close Mapp -> ETS -1
ETS 4 -> Close Mapp -> ETS -1
ETS 1 -> Back -> ETS 1
ETS 2 -> Back -> ETS 1
ETS 3 -> Back -> ETS 1
ETS 4 -> Back -> ETS 1
ETS 1 -> Select Show Map -> E
ETS 2 -> Show Map -> ETS 1
ETS 1 -> Select Find -> ETS 3
ETS 3 -> Search Sight -> ETS 4
ETS 4 -> Show Sight Location -
ETS 1 -> Service -> ETS -1
ETS 2 -> Service -> ETS -1
ETS 3 -> Service -> ETS -1
```

Fig. 8. The DynaMo-AID Tool.

7 Conclusions and Future Work

We have presented both a design process and a runtime architecture to support the creation of context-sensitive user interfaces. We believe this work can be an incentive for reconsidering the model-based user interface development approaches to enable the design of user interfaces for pervasive computing applications.

The next step is to integrate more general context specifications. At the moment our applications consider a fixed set of Abstract Context Widgets, but there is work in progress within the CoDAMoS[6] project to construct a more general context specification and integrate it in our system. Another extra feature could be to support propagating the effect of new services to the UI prototype of the main application. Another issue we whish to tackle is usability. At the moment usability is to a large extent the responsibility of the user interface designer when he/she draws the inter-dialog transitions. In this way context switches can only affect the UI where the designer wants the UI to change. To bring a change of context to the user's attention,

[6] `http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/CoDAMoS/`

changes with the previous dialog could be marked with colors, or a recognizable sound could tell the user a context-switch has occured.

8 Acknowledgements

Our research is partly funded by the Flemish government and European Fund for Regional Development. The CoDAMoS[2] (Context-Driven Adaptation of Mobile Services) project IWT 030320 is directly funded by the IWT (Flemish subsidy organization).

References

1. Gaëlle Calvary, Joëlle Coutaz, and David Thevenin. Embedding Plasticity in the development process of interactive systems. In *6th ERCIM Workshop "User Interfaces for All". Also in HUC (Handheld and Ubiquitous Computing) First Workshop on Resource Sensitive Mobile HCI, Conference on Handheld and Ubiquitous Computing, HU2K,* Bristol, 2000.
2. Gaëlle Calvary, Joëlle Coutaz, and David Thevenin. Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism. In *Joint Proceedings of HCI 2001 and IHM 2001. Lille, France*, pages 349-364, 2001.
3. Tim Clerckx, Kris Luyten, and Karin Coninx. Generating Context-Sensitive Multiple Device Interfaces from Design. In *Pre-Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces CADUI'2004, 13-16 January 2004, Funchal, Isle of Madeira, Portugal*, pages 288-301, 2004.
4. 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 *Human-Computer Interaction with Mobile Devices and Services, 5th International Symposium, Mobile HCI 2003*, pages 256-270, Udine, Italy, September 8-11 2003. Springer.
5. George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: concepts and design, Third Edition*. Addison-Wesley, ISBN: 0201-61918-0, 2001.
6. Joëlle Coutaz, Software architecture modeling for user interfaces. In *Encyclopedia of Software Engineering. Wiley and sons*, 1993.
7. Joëlle Coutaz and Gaëtan Rey. Foundation for a Theory of Contextors. In Kolski and Vanderdonckt [10], pages 13-33. Invited Talk.
8. Jason I. Hong. The Context Fabric: An infrastructure for context-aware computing. In *CHI'02 extended abstracts on Human factors in computer systems, Minneapolis, Minnesota, USA*, pages 554-555, ACM Press, 2002.
9. Chris Johnson, editor. *Interactive Systems: Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, Glasgow, Scotland, UK, June 13-15, 2001, Revised Papers*, volume 2220 of *Lecture Notes in Computer Science*. Springer, 2001.
10. Christophe Kolski and Jean Vanderdonckt, editors. *Computer-Aided Design of User Interfaces III*, volume 3. Kluwer Academic, 2002.
11. Panu Korpipää, Jani Mätyjärvi, Juha Kela, Heikki Keränen, and Esko-Juhani Malm. Managing context information in mobile devices. *IEEE Pervasive Computing, Mobile and Ubiquitous Systems*, 2(3):42-51, July-September 2003.

12. Quentin Limbourg, Jean Vanderdonckt, and Nathalie Souchon. The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results. In Palanque and Paternò [16], pages 227-246.
13. Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In Joaquim A. Jorge, Nuno Jardim Nunes, and João Falcão e Cunha, editors, *Interactive Systems: Design, Specification, and Verification*, volume 2844 of *Lectures Notes in Computer Science*, pages 191-205. Springer, 2003.
14. Kris Luyten and Karin Coninx. An XML-based runtime user interface description language for mobile computing devices. In Johnson [9], pages 17-29.
15. Giulio Mori, Fabio Paternò, and Carmen Santoro. Tool Support for Designing Nomadic Applications. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces, January 12-15, 2003, Miami, FL, USA*, pages 141-148, 2003.
16. Philippe A. Palanque and Fabio Paternò, editors. *Interactive Systems: Design, Specification, and Verification, 7th International Workshop DSV-IS, Limerick, Ireland, June 5-6, 2000, Proceedings*, volume 1946 of *Lecture Notes in Computer Science*. Springer, 2000.
17. Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer Verlag, ISBN: 1-85233-155-0, 1999.
18. Fabio Paternò and Carmen Santoro. One model, many interfaces. In Kolski and Vanderdonckt [10], pages 143-154.
19. Carl Adam Petri. Kommunikation mit automaten, second edition. *New York: Griffiss Air Force Base, Technical Report RADC-TR-65-377, Vol.1*, 1966.
20. Paulo Pinheiro da Silva. User interface declarative models and development environments: A survey. In Palanque and Paternò [16], pages 207-226.
21. Costin Pribeanu, Quentin Limbourg, and Jean Vanderdonckt. Task Modelling for Context-Sensitive User Interfaces. In Johnson [9], pages 60-76.
22. Angel Puerta. A Model-Based Interface Development Environment. In *IEEE Software*, pages 40-47, 1997.
23. Angel Puerta and Jacob Eisenstein. Towards a general computational framework for model-based interface development systems. In *Proceedings of the 1999 International Conference on Intelligent User Interfaces, Los Angeles, CA, USA*, pages 171-178, 1999.
24. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, May 15-20*, pages 434-441, 1999.
25. Egbert Schlungbaum and Thomas Elwert. Dialogue Graphs – a formal and visual specification technique for dialogue modelling. In *Formal Aspects of the Human Computer Interface*, 1996.
26. Nathalie Souchon, Quentin Limbourg, and Jean Vanderdonckt. Task Modelling in Multiple contexts of Use. In Peter Forbrig, Quentin Limbourg, Bodo Urban, and Jean Vanderdonckt, editors, *Interactive Systems: Design, Specification, and Verification*, volume 2545 of *Lecture Notes in Computer Science*, pages 60-76. Springer, 2002.
27. Jean Vanderdonckt and François Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In *ACM Conference on Human Aspects in Computing Systems InterCHI'93*, pages 424-429. Addison Wesley, 1993.
28. Jean Vanderdonckt, Quentin Limbourg, and Murielle Florins. Deriving the navigational structure of a user interface. In *Proceedings of the 9th IFIP TC 13 Int. Conference on Human-Computer Interaction Interact2003 Zürich 1-5 september 2003*, pages 455-462, 2003.
29. Anthony Wasserman. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Transactions on Software Engineering*, 11:699-713, 1985.

Discussion

[Willem-Paul Brinkman] How do you approach the problem that the user may be confused if the interface changes because of the context? Users may not be aware that the device is able to sense the environment.

> [Tim Clerckx] This is an important issue in context-aware computing. We have tried to put this responsibility in the hands of the UI designer, to make the UI user aware. The designer can then know when a change is happening and can do something about it.

[Willem-Paul Brinkman] Do you provide any guidance to the designer as to what to do?

> [Tim Clerckx] This is difficult to do in general.

[Juergen Ziegler] I like the approach to provide different levels of abstraction. What is the range of factors that you consider: location, temporal, etc. Is there any limitation? Also, you showed that several concrete context factors can be handled in an abstract object. How do you deal with the potential combinatorial explosion of factors?

> [Tim Clerckx] Regarding the first question, we have done experiments with the hardware sensors and GPS coordinates and we can easily define other context objects. For the second question, we handle the complexity in the abstract context objects. At the moment these are ad hoc implementations to interpret the information.

[Michael Harrison] In a different context you may absorb information in a different way. It isn't clear to me how your approach would capture this kind of information.

> [Tim Clerckx] In each layer we abstract a bit of information. So these context changes can be captured.

[Michael Harrison] Yes, but in different contexts you may have different information flows. This is critical in some contextual interfaces. Is this embedded in the actions?

> [Tim Clerckx] You could encapsulate user input with a concrete context object and this could be interpreted by an abstract object.

[Bonnie John] What if the user wants to override the default task context, e.g. the user is in a museum but wants to discuss where to go for lunch. How do you reprent this in your tool?

> [Tim Clerckx] If you want to do that it must be included at the task design time, where the designer explicitly allows the user to override the context and provides some user interaction for this purpose. The concrete contetx object would be a button press. The abstract context object would say to change the context and not change it back because of sensors until the user is done.