A Hybrid Approach to Adaptive User Interface Generation

Guido Menkhaus and Wolfgang Pree

Software Research Lab, Department of Computer Science, University of Salzburg, Austria

Due to the diversity of display capabilities and input devices, mobile computing gadgets have caused a dramatic increase in the development effort of interactive services. User interface (UI) tailoring and multi platform access represent two promising concepts for coping with this challenge. The article presents a hybrid approach to the generation of adaptive UIs based on a linking strategy of hierarchies of graphs.

Keywords: user interface adaptation, multi-platform support.

1. Introduction

The current trend of Web access and computing is drifting away from the desktop PC as the principal device to access services and information on the internet to consumer devices such as mobile phones, handheld computers and a wide spectrum of Personal Digital Assistants (PDAs). Most of the limitations that users experience will disappear in future generations of consumer devices. They will have easier to read displays, greater storage, and CPUs that are more powerful. These changes in design exclude the UI. Although the devices will have easier to read, higher contrast color displays, the actual screen size will not change, since the user demand devices that can easily be carried around and held in one hand. The objective of UI adaptation is to avoid fragmentation of the web space into spaces that are solely accessible with specific type of devices [14].

The article presents an approach to UI adaptation. It is based on an abstract UI description, which is shared among the different platforms. The adaptation technique tailors the UI description to minimize the mismatch between its presentation and the platform's capacity to present it.

The remaining of the article is organized as follows: The following section presents a short overview of UI architecture. Section 3 introduces adaptation of presentation models and related work. The hybrid adaptation technique is discussed in Section 4. We will then present results and Section 6 concludes the article with a brief talk about our future work.

2. Short Overview of User Interface Architecture

A large number of layered architectures have been devised in the context of UI software. Myers for example, has identified four general

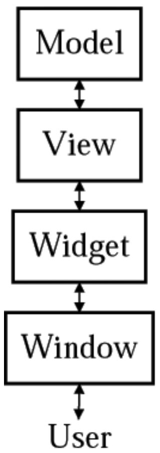


Fig. 1. Common layers for UI software.

layers: window, widget, view and model (Figure 1) [12]. Using this terminology, the view is a combination of the view and the controller of the Model-View-Controller architecture [11]. The architectural decomposition that we use in the article applies a different terminology but is similar to the one described above and was introduced in [6]. The three model components relevant to UI description in a mobile computing environment are: task model, platform model, and presentation model.

1. *Task Model.* The task model is a formal description of the service the user accesses. The task model is hierarchically organized and contains information regarding the trigger of a task, the precondition, postcondition, and the action of the task itself. The task model corresponds to the model in Myers layered architecture.
2. *Platform Model.* The platform model accounts for the different devices from which the user may access a service. The platform model corresponds to the window and widgets layer in Myers layered architecture. It contains information about the capabilities, restriction, and limitations of the target platform and maps conceptual elements of the presentation model to platform specific elements. This model is usually exploited dynamically at run-time.
3. *Presentation Model.* The mapping of the presentation model to Myers architecture is not smooth. The presentation model partly corresponds to the view layer of Myer's architecture. It is the description of how the UI is structured to support the task model. Since the task model is hierarchically organized, so is the presentation model. It usually presents windows consisting of a set of widgets and a set of transisitons, that allow navigation from one window to the next.

There are a number of approaches for designing and implementing UI software. They range from the automatic generation of the presentation model from a more or less formal task model [2] to informal, structured guidelines on how to build UI software [11]. However, most recent work has been dedicated to approaches that can be placed somewhere in the middle of both extremes [1, 3, 8]. Common to these approaches is the introduction of an abstract UI description in a custom markup language implementing the presentation model. The description comprises generic and general UI elements, which are platform-independent. These elements will be mapped to standard markup languages (like HTML or WML) or programming languages. This approach is attractive, since a single custom format serves a multitude of target platforms. The use of a custom markup language entails the following benefits [1]:

- Natural separation of UI code and non-UI code.
- Usable by non-programmers and occasional users.
- Facilitates rapid prototyping.
- Allows a family of interfaces to be created in which common features are factored out.

The introduction of a custom platform-independent markup language can help to solve the problem of the "Tower of Babel" [9] in UI languages. Each platform with its typical browser has its own markup language. Each language aims at a specific platform and is optimized for supporting it. However, the support of different platforms is only one problem that needs to be solved.

Another main obstacle to platform-independent content authoring is the fact that the growing number of networking enabled gadgets has a wide variety of UI capacities. One of the main differences they share is different screen size. The cardinal question that needs to be solved is: How to enable content to be adapted to various screen sizes? The platform model delivers information about the limitations and restrictions of the target platform. Is it possible to trade on this information?

The presentation model describing the flow of transition reflects the hierarchical structure of the task model. However, the internals of a window remain unstructured. A window is visualized unaltered on each platform, from a compositional point of view, although using different platform specific widgets. The challenge is to remodel the widgets of a window into a new composition of "small" windows with a reasonable flow of transitions between them (Figure 2).

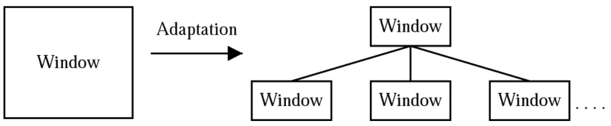


Fig. 2. Adaptation of a window into a set of "small" windows.

3. Presentation Model Adaptation

The definition of a single presentation model is still oriented at the "one device — one functionality" paradigm, but today we can access mutually any service through any device [7]. This requires an appropriate mechanism to dynamically adapt the presentation model.

We shall view a window as a two-dimensional matrix whose row and column indices identify a widget.

$$window_{P\times Q} = [widget(x, y)]_{P\times Q}$$

where $P \times Q$ is the size of the window and $widget(x, y) \in UIE$, the set of abstract UI elements, which the custom UI description language allows to use. In this article, the size of a window is discretized and the unit is a widget. Without loss of generality we consider only the case, where $Q = 1$.

Presentation model adaptation is the process of grouping a window of the presentation model into a set of non-intersecting regions of widgets, such that each region satisfies a homogeneity predicate. Non-intersecting regions of widgets mean that no widget is in two regions. Thus, no redundant information is created, once the regions will be converted to a set of smaller windows, which will substitute the single original window. We consider the case, where the current presentation model was intended to be displayed on a device like a desktop PC with a monitor and the actual device that accesses the service is a PDA with a much smaller screen. This situation is typical for mobile computing: Services target primarily desktop PC with a monitor and migrate then to a wide variety of mobile computing devices. The situation where a service targets small devices and is accessed by a desktop PC with a monitor is not further discussed here.

Formally the adaptation process of the presentation model can be defined as follows: If a *window* consists of a set of *widgets* and P is a homogeneity predicate defined on a group of connected widgets, then the adaptation of the presentation model is a partitioning of *window* into a set of connected regions $(R_1, R_2, \ldots, R_n)$, such that:

$$\begin{aligned} window &= \bigcup_{i=1}^{n} (R_i \backslash navigationWidgets(R_i)) \\ R_i \bigcap R_j &= \emptyset, i \neq j \\ P(R_i) &= \text{true}, i = 1, \ldots, n \\ P(R_i \cup R_j) &= \text{false, if } R_i \text{ is adjacent to } R_j \end{aligned}$$

Adaptation of the presentation model partitions a window into regions of non-intersecting widgets complying with a homogeneity predicate. A user accessing a service supported by a presentation model needs to navigate from one region to the next region. However, the widgets necessary to navigate are not in the originally window. Thus, they have to be integrated into the regions resulting from the adaptation process. The set of all widgets in the regions equals the widgets in the original window plus the integrated new widgets dedicated to the navigation between the regions, the $navigationWidgets(R_i)$.

Approaches exploring presentation model adaptation can broadly be divided into two categories. The first category uses non-contextual information to remodel the presentation model. The other category is contextual, task-model based.

- *Non-Contextual.* This approach groups widgets into regions regardless of any semantic dependencies between them. For example, textual information explaining the use of a button and the button itself are modeled as two distinct widgets. If these two widgets were grouped into two different regions after the adaptation process, the adapted presentation model had low usability. In the non-contextual approach, the single criterion for grouping widgets is their presentation size. I.e., in default of semantic, contextual information, widgets are remodeled into regions as long as they can be reasonably visualized on the target platform [4]. The result is a set of regions, which can be navigated in a linear, sequential way. To access the last widget of the original window, each new "small" window has to be traversed.

- *Contextual.* Global techniques depend very much on the detailed specification of a task model. Here, adaptation is the process of mapping the task model onto different presentation models, under consideration of the platform model. However, by its very nature, a detailed task model might be as complex to produce as the actual implementation of a set of presentation models. Any simplification or abstraction may hide details that are critical to successful adaptation. Another obstacle is that in most commercial software systems, there is no detailed or formal specification of a task model. However, Eisenstein and Rich have developed first promising techniques [5].

The disadvantage of the contextual approach results from the fact that building the task model is an inherently difficult process. This might also be the reason why this technology has not yet been widely accepted. The non-contextual approach has the drawback of working only on local information. We propose a hybrid approach that combines advantages of the non-contextual approach (fast, no need to produce a task model) and of the contextual approach (integration of task model information).

4. Hybrid Approach to Presentation Model Adaptation

The two main challenges of the hybrid approach to presentation model adaptation are:

1. How to incorporate low-level task model information into the presentation model?
2. How to adapt the presentation model respecting task model information?

No current markup language supports the feature of integrating low-level task model information that could guide the adaptation process. We have developed Event Graph XML. A markup language that allows adding information to each element stating the semantic relation to its neighboring elements. For more information on Event-Graph XML, please refer to [8]. In the next section, we introduce the presentation model adaptation process.

4.1. Presentation Model Adaptation

The adaptation technique is based on a linking strategy of two hierarchies of graphs [13, 10]. The approach allows remodeling a window of the presentation model into regions of connected widgets and the use of low-level task model information.

The set of widgets of a window is placed into a stack of regular grids, as illustrated in Figure 3. In the lowest level of the stack, each cell of the grid corresponds to a single widget. Each cell of level $i + 1$ represents a group of cells of level i. The cells form a linear structure of 3×1 cells. The cells overlap in such a way that the outer cells on level i belong to two cells of level $i + 1$. The cells in a group of level i, represented by a cell of level $i + 1$, are called the subcells or the children of this cell. The representing cell is called the parent of its children.

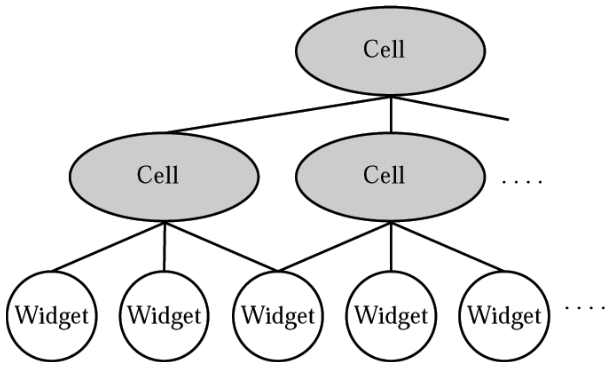


Fig. 3. Stack of a regular grid of cells that places a structure on a window and its widgets. Three widgets form a cell on the lowest level. Cells on a lower level are candidates for cells on a higher level.

The grouping of widgets of a window into a set of regions is done within the boundaries of the induced stack of cells. To come to the set of regions we dynamically build up a stack of regions. A widget corresponds to a region on the lowest level. Adaptation of a window is performed by grouping regions of level i into regions of level $i + 1$. However, regions can only be grouped within the boundaries of a cell in which they reside, as illustrated in Figure 4, and if they satisfy the homogeneity predicate.

The framework to describe the adaptation technique is the description as a hierarchy of graphs. A graph is denoted as $G = (V, E)$, where V is the set of vertices of the graph and E is the set of edges. The stack of cells can be described as follows: A hierarchy of cells is a sequence of graphs $G_i = (C_i, E_i), i = 0, \ldots, n$ and a sequence of mappings:

$$\begin{aligned} (\pi_1, \ldots, \pi_{n-1}),\quad & \pi_i && : C_i \rightarrow \mathcal{P}(C_{i+1}) \\ (\kappa_2, \ldots, \kappa_n),\quad & \kappa_i && : C_i \rightarrow \mathcal{P}(C_{i-1}) \\ (\rho_1, \ldots, \rho_n),\quad & \rho_i && : C_i \rightarrow \mathcal{P}(R_i) \end{aligned}$$

$\mathcal{P}(\cdot)$ is the Powerset. $\mathcal{P}(R_i)$ is a set of regions of level i. The description of the hierarchy of regions is given shortly. Each graph of the hierarchy $G_i = (C_i, E_i)$ corresponds to a grid, which is considered as a linear graph, where each vertice is connected with a predecessor and a successor (Figure 3).

The mapping π_i assigns to each cell of level i two parent cells at level $i + 1$. To be precise, the outer cell has two parent cells whereas the inner cell of the 3×1 grid has a single parent: $\pi_i(c) = \{a, b\}, c \in C_i$ and $a, b \in C_{i+1}$. The mapping κ_i assigns to each cell $c \in C_i$ its children $\{a \in C_{i-1} | c \in \pi_{i-1}(a)\}$. The mapping ρ_i assigns to each cell those regions, which have been formed in the adaptation process within its boundaries.

The cells form a rigid stack of grids, which is built independent of the underlying content of the window. It serves to control the grouping process by forcing it into a hierarchical structure. The hierarchy of graphs that helps describing the grouping of widgets into regions is of primary interest. A hierarchy of regions is a sequence of graphs $G_i = (R_i, E_i), i = 0, \ldots, n$ and a sequence of mappings:

$$\begin{aligned} (\pi_1, \ldots, \pi_{n-1}),\quad & \pi_i && : R_i \rightarrow \mathcal{P}(R_{i+1}) \\ (\kappa_2, \ldots, \kappa_n),\quad & \kappa_i && : R_i \rightarrow \mathcal{P}(R_{i-1}) \\ (\zeta_1, \ldots, \zeta_n),\quad & \zeta_i && : R_i \rightarrow \mathcal{P}(C_i) \end{aligned}$$

The mapping π_i assigns to each region at most two parent regions. The parent region is in the parent cell of the cell where the child regions reside. Regions can have two parent regions, since there are regions belonging to two neighboring and overlapping cells. A hierarchy of regions is built up by applying a grouping process to each cell while moving up the stack of cells. The grouping process stops at the boundary of each cell and therefore some regions take part in two cell-based groupings. The mapping κ_i assigns to each regions $r \in R_i$ its children $\{s \in R_{i-1} | r \in \pi_{i-1}(s)\}$. The link between the hierarchy of regions and the hierarchy of cells is established by the mapping ζ_i. ζ_i assigns to each region the cell(s) where it is located. The receptive field RF of a region $r \in R_i$ is defined as the set of all regions of the lowest level, which form the region r. I.e., $RF(r) = \kappa_2 \cdots \kappa_i(r)$.

The adaptation process adapts dynamically the presentation model by composing and decomposing widgets of a window into a set of regions, which result into a hierarchical structure of linked windows. The technique consists of the building and the transformation phase.

Building Phase: This phase is also called the "Window to Region" phase. The window from the original presentation model is grouped into a set of regions. It consists of four processes:

- *Grouping.* Regions of level i are grouped into regions of level $i+1$ within the boundaries of their cell and satisfying a predicate P.
- *Separating.* Regions that fail to group are separated.
- *Splitting.* Large-sized regions, especially when they contain a single widget, are split.
- *Relinking.* The user should be able to navigate from one region to the next region. To ensure usability, regions are relinked by integrating additional navigation widgets.

Transformation Phase: This phase is also called the "Region to Windows" phase. The resulting regions are transformed into a set of "small" windows.

4.2. Grouping

The grouping process determines the set of connected regions of level i of a specific cell and groups them. In order to form a new region r_{i+1}

(the subscript indicates the level) in a cell c_{i+1}, the set of subcells SC of c_{i+1} is determined: $SC = \kappa_{i+1}(c_{i+1}) = \{a \in C_i | c_{i+1} \in \pi_i(a)\}$. Each subcell has a set of regions associated $S = \zeta_i(a)$, $a \in SC$ that are candidates for grouping into r_{i+1}. A region s_i groups into the region r_{i+1} if it satisfies the homogeneity predicate $P(r_{i+1} \cup s_i) =$ true.

Two regions s_i, t_i are connected, if there is an edge $(s_i, t_i) \in E_i$ between them, i.e., they have a common subregion $u_{i-1} = \kappa_i(s_i) \cap \kappa_i(t_i)$, s_i, $t_i \in S$ and $u_{i-1} \in R_{i-1}$. Regions of the lowest level are connected with their neighboring regions. The overlapping structure of the stack of cells guarantees that the grouping process considers only those regions, which are connected or have a path of connected regions on the lowest level, the widget level. The grouping process is illustrated in Figure 4.

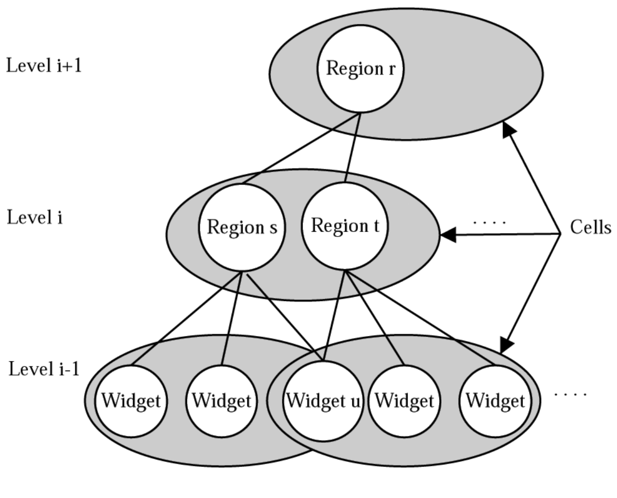


Fig. 4. Grouping process. Regions are grouped within the boundary of a cell.

4.3. Separating

If the grouping process fails, because a region s_i does not satisfy the homogeneity predicate P, the region needs to be separated from its connected region t_i. They are separated since they have a common subregion u_{i-1}, which needs to be assigned to a single parent region. The separation process assigns the common subregion to the region, which is the smaller parent region, from a size point of view. That means that u_{i-1} is removed from the set of subregions of the larger region $\kappa(v_i) = \{w \in R_{i-1} | v_i \in \pi_{i-1}(w)\} \setminus u_{i-1}$, with $v_i = max(size(RF(s_i)), size(RF(t_i)))$. The process is recursively applied down to the lowest level. For level $i-1$ it would be applied to u_{i-1}, the common subregion of s_i and t_i, and to those subregions of v_i, which have a common subregion with u_{i-1}, since $\emptyset \neq \kappa_{i-1}(u_{i-1}) \cap \kappa_{i-1}(\kappa_i(v_i))$.

4.4. Splitting

If a region's size dominates a window and there are other regions, which could, from the task's model point of view, group with that region, it can be split into a sequence of smaller, linked regions. E.g., a lengthy text message is split into a sequence of regions containing each a part of the text message. Only the head of the sequence continues to take part in the grouping process.

4.5. Relinking

A region that cannot be further grouped with other regions into a region of a higher level is called *complete*. A *complete* region that has reached the threshold of maximal allowed size does not drop out of the grouping process. Instead, a new region is created containing a single navigation widget pointing to the *complete* region. The new region takes the place of the *complete* region and continues the grouping process on behalf of it. The process is illustrated in Figure 5.

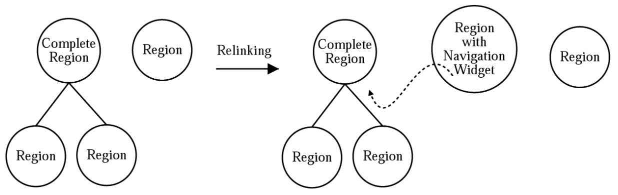


Fig. 5. A region containing a single navigation widget will replace a *complete* region. The new region takes part in the building phase on behalf of the *complete* region.

The effect of the relinking phase is that the adaptation process creates a linked tree-structure. The regions representing the leaves of that tree-structure contain the widgets of the original window. The intermediate nodes of the tree-structure are regions including the navigation widgets that have been created in the relinking process.

4.6. Transformation

After the building phase has stopped at the top of the stack of the cells, the *complete* regions are transformed into "small" windows and integrated into the presentation model. The last created region forms the entry window of the new part of the hierarchical presentation model. The new set of windows minimizes the mismatch between the presentation model and the platform model.

4.7. Homogeneity Predicate

The homogeneity predicate is employed to decide if regions can be grouped together or not. The predicate consists of two parts, which both have to evaluate to true; $P(r) = Size(r) \wedge Context(r), r \in R_i$.

- *Size.* A window will be displayed on different platforms with varying features, where screen size is of crucial importance. If the size of two regions and their parent region is lower than a predefined threshold (e.g., three times of the screen size) the regions are grouped, otherwise they are separated. The size of a region is the sum of the sizes of the widgets of its receptive field.

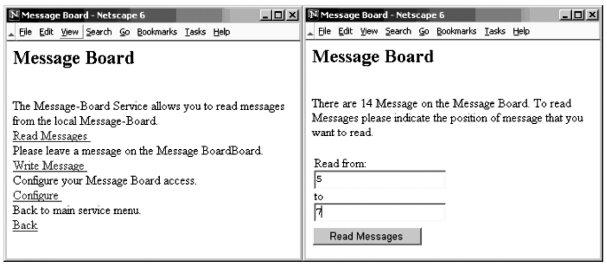
- *Context.* The designer of the original presentation model integrates in it task-model-related information (context information). The information deals with the semantic relation of a widget with its neighboring widgets.

  A region s_i, having different semantic intent than its tentative parent region r_{i+1}, is not grouped. If its semantic intent is similar or the difference does not exceed a predefined threshold $d(s_i, r_{i+1}) < \Theta$, it can be grouped. In the first version of the adaptation process, we simply assign integer values to widgets, to indicate semantic similarity. $d(\cdot, \cdot)$ is a distance measure like the Euclidian distance. The context information value of a region is the average value of its receptive field.

5. Results

To illustrate the adaptation technique of a presentation model we have implemented a location-based message board (LBMB) [8]. The message board contains location specific information and users can read and store messages on the message board. A mobile user moving from location to location accesses different message boards depending on its geographical position. Different users use different devices to access the message board such as laptops, PDAs, and mobile phones relying on Wireless Application Protocol (WAP) or Short Message Services (SMS).

Figure 6a shows the UI of the LBMB on a HTML browser. This browser is a powerful tool, so that the presentation model does not perform any adaptation. The same application logic, this time with a UI adapted to the small


Message Board - Netscape 6
File Edit View Search Go Bookmarks Tasks Help

Message Board

The Message-Board Service allows you to read messages from the local Message-Board.
Read Messages
Please leave a message on the Message BoardBoard.
Write Message
Configure your Message Board access.
Configure
Back to main service menu.
Back

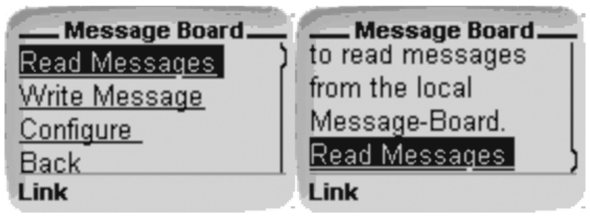
Message Board - Netscape 6
File Edit View Search Go Bookmarks Tasks Help

Message Board

There are 14 Message on the Message Board. To read Messages please indicate the position of message that you want to read.

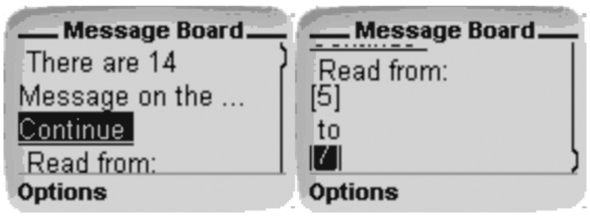
Read from:
5
to
7
Read Messages

Fig. 6. (a) HTML-Browser showing the LBMB "Main Menu" and (b) the "Read Message UI".


Message Board
Read Messages
Write Message
Configure
Back
Link

Message Board
to read messages
from the local
Message-Board.
Read Messages
Link

Fig. 7. WAP/WML browser showing the "Main Menu UI".


Message Board
There are 14
Message on the ...
Continue
Read from:
Options

Message Board
Read from:
[5]
to
[7]
Options

Fig. 8. WAP/WML browser showing the "Read Messages UI".

screen of a mobile phone, can be seen in Figure 7. The menu is divided into a two level menu, with a main menu containing links to each menu item, which are presented on their distinct screen. The main menu is inserted during the relinking process of the adaptation and is not present in the original presentation model. Figure 6b shows the UI, where users specify the number of message they want to read. The same UI on a mobile phone is divided into the input fields and the description, where the first part of the description is visible. Navigation to the second part is done using the "continue" link to another screen (Figure 8).

6. Conclusion and Future Work

The article has presented a new approach to dynamic UI adaptation. The presentation model adaptation process is based on a linking strategy of two hierarchies of graphs. The adaptation process consists of the building and the transformation phases. The building phase forms a hierarchical structure of windows by grouping, separating, splitting and relinking regions. The building phase is guided by low-level task-model-related information provided by the designer of the presentation model at design time. The construction phase transforms the resulting *complete* regions into windows. The adaptation process remodels dynamically a presentation model to better fit it to the current platform model.

The first experiments with the presentation model adaptation technique are promising and show that the concept is sound. The use of the hierarchy of graph has been proven flexible and is a viable concept for future UI development. Future work will focus on conducting experiments with more complex presentation models. The integration of task-model-related information into the presentation model is somehow simple. Exploration of more powerful, but equally simple methods needs to be carried out. Simplicity is an important objective to encourage the use of this design technique.

References

[1] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster. UIML: An Appliance-Independent XML User Interface Language. *WWW8 / Computer Networks*, 31(11–16):1695–1708, 1999.

[2] D. Atkins, T. Ball, G. Bruns, and K. Cox. Mawl: A domain specific language for form-based services. *IEEE Transaction on Software Engineering*, 25(3):334–346, 1999.

[3] T. Ball, P. Danielson, L. Jagadeesan, R. Jagadeesan, K. Laeufer, P. Mataga, and K. Rehor. Sisl: Several Interfaces, Single Logic. *"International Journal of Speech Technology"*, 3:93–108, June 2000.

[4] K.H. Britton, R.Case, A.Citron, R. Floyed, Y. Li, C. Seekamp, B. Topol, and K. Tracey. Transcoding. Extending e-business to new environments. *IBM Systems Journal*, 40(1):153–178, 2001.

[5] J. Eisenstein and Charles Rich. Agents and GUI from Task Modles. In *ACM IUI*, San Fransisco, California, USA, January 2002.

[6] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying Model-Based Techniques to the Development of UIs for Mobile Computers. In *ACM IUI*, Santa Fee, New Mexico, USA, January 2001.

[7] C. Elting, J. Zwickel, and R. Malaka. Device-Dependant Modality Selection for User Interfaces – An Emprical Study. In *ACM IUI*, San Fransisco, California, USA, January 2002.

[8] S. Fischmeister, G. Menkhaus, and W. Pree. MUSA-Shadow: A Location-Based Service Supporting Multiple Devices. In *Proceedings of Pacific TOOLS*, 2002.

[9] K. M. Goeschka and R. Smeikal. Interaction Markup Language. In *Proceedings of the 34th Hawaii International Conference on Systems Sciences*. IEEE Computer Society, 2001.

[10] G. Hartmann. Recognition of Hierarchically Encoded Images by Technical and Biological Systems. *Biological Cybernetics*, 57:73–84, 1987.

[11] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *"Journal of Object-Oriented Programming"*, 1(3):26–49, August/September 1988.

[12] B. Myers. User Interface Software Tools. *ACM Transaction on Computer-Human Interaction*, 2(1):64–103, March 1995.

[13] P. Nacken. Image Segmentation By Connectivity Preserving Relinking in Hierarchical Graph Structures. *Pattern Recognition*, 28(6):907–920, 1995.

[14] W3C. Device Independence Working Group Charter, 2001.

Received: June, 2002
Accepted: September, 2002

Contact address:

Guido Menkhaus
Software Research Lab
Department of Computer Science
University of Salzburg
A-5020 Salzburg, Austria
e-mail: `menkhaus@softwareresearc|h.net`

Wolfgang Pree
Software Research Lab
Department of Computer Science
University of Salzburg
A-5020 Salzburg, Austria
e-mail: `pree@acm.org`

GUIDO MENKHAUS has received his PhD from the University of Salzburg, Austria in 2002. His MSc is from the University of Bielefeld, Germany. His research interest focuses on component architecture and architectures for mobile computing, dialog systems and adaptive user interface generation in mobile computing environments.

WOLFGANG PREE is professor of computer science at the University of Salzburg, Austria. He has dealt with various aspects of software construction, in particular software design and implementation, software reuse and composition, and programming methodology. His current focus is on Embedded (Control) Systems. He is the author of Design Patterns for Object-Oriented Software Development (Addison-Wesley/ACM Press, 1995).