Adaptation for Device Independent Authoring

Guido Menkhaus and Sebastian Fischmeister
Software Research Lab, University of Salzburg, Salzburg, Austria
{Menkhaus, Fischmeister}@SoftwareResearch.Net

ABSTRACT
The impact of device independent authoring on software engineering manifests itself mainly at the middleware level. Until recently middleware platforms were targeted at vertical coverage of specific scenarios. Consumer devices with integrated Internet-access are becoming more popular and their diversity grows with their market penetration and with the extension of the mobile communication infrastructure. This requires software architectures that are capable of supporting horizontal coverage of a wide range of devices and scenarios. This paper presents the Multi User Interface, Single Application project. It provides a feasable approach for multi-platform support through the introduction of an adaptable and abstract interaction-oriented user interface language.

Author Keywords
Device Independent Authoring, User Interface Adaptation

INTRODUCTION
Due to the emergence and proliferation of new classes of devices accessing services on the Web, device independent authoring became an important issue. The vision of device independence authoring is to allow services and content on the Web to be accessible by *anyone, anywhere, anytime, and anyhow* [21] by simultaneous reuse of authored source across multiple contexts and environments. Because of the variety of different UI platforms, content authors cannot afford creating and developing content and UIs that target only one out of a set of target platforms. This has a profound impact on the way UIs are build. Systems must scale up well to environments that include a wide variety of different devices that can easily and flexibly connect to application logic. The objective is to develop UIs for the same application only once and not for each particular class of computing device to avoid fragmentation of the web space into spaces that are solely accessible with specific type of devices.

There are research projects that look into new generation UIs that no longer consist of a display, but for example of wearable glasses projecting the UI onto the eye of the user [9] or wearable computers with small and semitransparent displays placed only centimeters from the user's eye in front of the line of sight [24]. These UIs, although small in physical size, will no longer have size constraints concerning the UI. However, user acceptance seems to be low [2]. We think that traditional UIs still have a strong potential for improvement and that this technology will prevail in the near future on the consumer market [11].

The dynamics of the mobile environment and the limitations of mobile computing resources make adaptation a necessary technique. Adaptation is necessary when there is a significant mismatch between the supply and demand of a resource, which is typical for a mobile and pervasive computing environments [18]. A permanent solution therefore requires models and techniques that allow UI designer to generate adaptive UIs.

This article presents the Multi User Interface, Single Application (MUSA) system. The MUSA project concentrates on multi-platform support. We argue that the introduction of an abstract interaction-oriented UI language is an essential component that eases the development of UIs for mobile computing devices. MUSA allows a Web-based service to be delivered to a variety of target platforms without additional effort. When a user requests a service, context information triggers the adaptation mechanism tailoring the event handler graph (written in EGXML) and the content of the service to a target platform specific presentation form. The event handler (EH) graph mediates between the concrete UI and the application logic. Figure 1 sketches the scenario of device independent authoring with the EH graph.

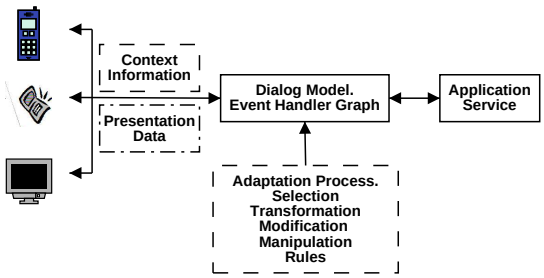

Figure 1: Scenario of device independent service authoring.

The remaining of the article is organized as follows: The following section presents a short overview of UI architecture

and related work. We then introduce the MUSA system and model. The adaptation mechanisms of the EGXML is discussed and results are presented. We close the article with concluding remarks and a short discussion about our future work.

USER INTERFACE ARCHITECTURE AND RELATED WORK

Model-based UI software development has introduced concepts and techniques that assist in the process of UI development and a large number of layered architectures have been devised [15, 20]. Myers for example, has identified four general layers: window, widget, view, and model [13]. This division corresponds roughly to the linguistic model of architecture of interactive software that distinguishes between the following three layers: The semantic layer describes the tasks users should be able to perform using the application for which the UI provides the interaction means. This layer corresponds to the model in Myer's layered architecture. The syntactic layer describes the structure and the interaction behavior of the UI. This can be mapped onto Myer's view component. The lexical layer consists of the detailed description of the visual part of the UI and corresponds to the window and the widget layer of Myer's architecture.

The Arch-Slinky model refines the level of abstraction with which it describes the reference model for UI management systems [23]. It proposes a structural and functional decomposition into the following five components:

1. **Functional Core.** The functional core is the creator of data that the system represents. It manipulates data and performs other domain-oriented functions.
2. **Functional Core Adapter.** The functional core uses the functional core adapter as its channel of communication to the dialog component.
3. **Dialog.** The dialog component mediates between domain specific and presentation specific data. It controls task sequencing.
4. **Logical Interaction.** The physical interaction and the dialog component interact through the logical interactor that provides corresponding interfaces and objects.
5. **Physical Interaction.** The physical interaction component consumes the presentation specific data and provides input to the functional core. It deals with input and output of data on the target device.

The Arch-Slinky model minimizes the effect of future modification to an application and its environment by isolating the dialog component from the functional core and the physical interaction component.

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 [3] to informal, structured guidelines on how to build UI software [10]. However, lots of effort has been dedicated to approaches that can be placed somewhere in the middle of the both extremes [1, 6, 17]. Common to these approaches is the application of a single XML-based description implementing the presentation and/or the dialog model, respectively. The description is adapted according to a device profile at run-time into a device adequate form. The approach supports device independent authoring so that a single description is enabled to serve a multitude of platforms. Wong presents a high-level task model description of a web application that has a tree-like structure [22]. The tree-like structure is adapted according to device-dependent information to match the target device. Adaptation on the task and concept and the UI level is presented in [8]. Rules are defined and prioritized that tailor a UI at different levels for graceful degradation. Most of the work concentrates on transformation of UI elements. Adaptation at the task and concept level focuses on deletion and insertion of tasks.

The introduction of a custom platform-independent markup language can help to solve the problem of the *Tower of Babel* in UI languages, since one obstacle to device independent authoring is that each platform with its typical browser has its own markup language and each language aims at a specific platform and is optimized for supporting it. However, the support of different platforms is a problem that can be solved at the physical interaction / UI level of abstraction. Another main obstacle to device independent authoring is the growing number of networking enabled devices with a wide variety of UI capability and device specific platforms. One of the main differences they share is different screen size. How to enable content to be adapted to various screen sizes? The same content may require varying numbers of windows to display and a different navigational structure, depending on the platform. For example, content fitting on one PC window may require three windows on a mobile phone. Yet, all these windows originate from the same, single authored UI. The device profile delivers information about the limitations and restrictions of the target platform. The adaptation process for EGXML exploits this information for adapting the content and the navigational structure. This results in a hierarchical structure of dialogs.

MUSA

The MUSA system utilizes device independent UI description in EGXML and supports the integration and composition of Web services. The objective is a reduction of development time, cost, as well as improved maintainability and flexibility.

Figure 2 illustrates the high-level architecture of MUSA. The system is conceptually split into four tiers and employs an event-driven design.

Client: The client environment represents the first tier. The client is represented by a device with a UI. No service data is installed on the client side and the client communicates via wireline or wireless Internet with the service. Typically, the client is some sort of browser, but in principle, could also be a device with no visualization capacity such as a telephone.

Request Processor and Client Gateways: The communication between the service and the client passes through the

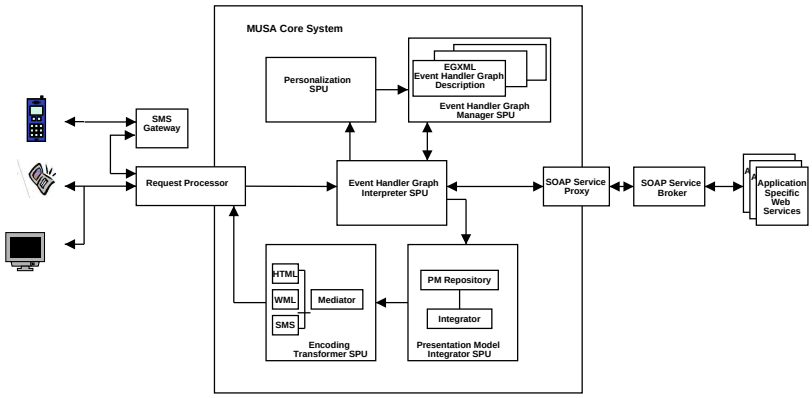

Figure 2. High-level architecture of MUSA system.

request processor. It forwards the communication stream to the MUSA core system and converts the client requests into events that are used throughout the MUSA architecture.

MUSA Core System: The MUSA core system consists of specialized processing units (SPU), which reflect the separation of concerns of the system.

1. **Event Handler Graph Interpreter SPU.** The EH graph interpreter handles the event processing. The incoming events from the request processor are sent to individual EHs that contain the necessary information to properly respond to the input event. In response to the event processing the system generates outgoing events for each incoming event, which are further processed in the MUSA system.

2. **Event Handler Graph Manager SPU.** The EH graph manager SPU manages the EH graphs. It controls the access and the transformations of the EH graph.

3. **Personalization SPU.** The personalization SPU enables users to personalize, i.e., to modify and adapt the EH graph to their preferences. The adaptation is done via direct manipulation technique [19]. For example, EHs can be removed from the EH graph. Once an EH is removed, the removal can be undone. EHs can be given a different description and they can be preset with default values. This is helpful, when a user applies a form over and over again, and the text of only a few input fields varies. It helps reducing the time to fill a form.

4. **Encoding Transformer SPU.** The transformer SPU transforms and maps outgoing EHs of the EH graph to an appropriate presentation form. If the client is a device with a graphical UI, the EHs are mapped to those concrete UI elements, which are able to implement and trigger the specified events, which are associated to specific EHs. The SPU applies a transformation on the EH graph depending on the client's profile. Figure 2 shows three transformers: a HTML, a WML, and a SMS transformer.

5. **Presentation Model Integrator SPU.** The integrator SPU models the overall presentation layout of the EH graph, which are transmitted in the course of the current interaction between users and applications. The presentation model integrator SPU has a repository of presentation models. The presentation models are created at design time by a UI designer. This allows the EH graph and the presentation models to be developed, maintained, and modified independently. A presentation model in the repository consists of a file written in a concrete UI language enriched with special *integrator commands*, which indicate where to integrate the EHs of the EH graph. An opening command opens an integrator command and each opening command has a corresponding closing command, which delimits it, such as a XML element has an opening tag and a closing tag. Each concrete UI element, which is between the opening and the corresponding closing command, belongs to this integrator command. The insert command indicates the place where to insert the EH of the EH graph. If the associated EH is not present, the complete command and its content is removed and not transferred to the encoding transformer SPU. The simplest presentation model consists solely of integrator commands.

Service Proxy: The service logic is the body of code for which the MUSA system provides the service facade. The Web services that implement the service logic are accessible via service proxies, which connect the MUSA system to other Web services.

MUSA Model

MUSA builds on the Arch-Slinky model adopting the cardinal functional decomposition in a physical interaction, logical interaction, dialog, functional core adapter and the functional core component. The MUSA model (Figure 3) refines the Dialog component, by introducing the MUSA EH graph and the Physical interaction component, by splitting it into a global and local component. The MUSA Model tries to map the abstraction a Web designer would use while designing a Web-based service onto the vocabulary of the EH graph. Within the dialog component, we observe two kinds of information flows: the vertical traversal of the EH graph and the lateral information flow with the functional core adapter and the logical presentation component. An EH of the EH graph may be related to one or multiple objects of the func-

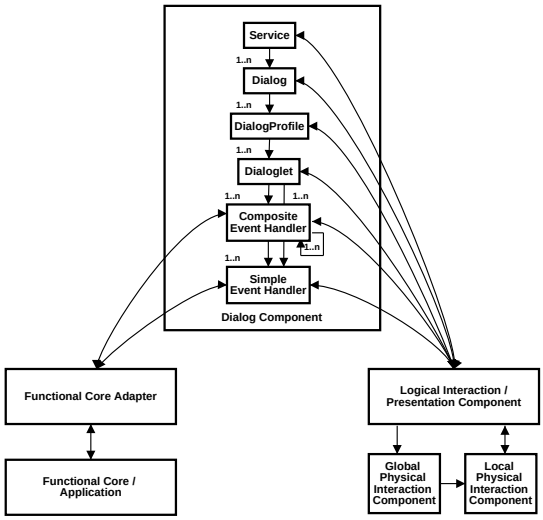

Figure 3. MUSA Model.

tional core adapter. Similarly, each part of the EH graph is connected to one or multiple presentation objects of the logical interaction component.

The physical interaction component is split into a global and a local part. This allows the both parts to vary independently. This avoids a permanent binding of the encoding of the objects of the logical interactor and the global layout of the logical interactor to each other.

The MUSA Event Handler Graph (EGXML)

The EH graph is at the core of the MUSA model. The introduction of the EH graph follows the idea of the reactive constraint graph described in [4]. It is an abstract description of a service logic, which is available for service access to a wide range of clients. The basic building blocks of the EH graph are represented by specific EHs. The EHs receive events from the client dispatched by the logical interaction component and emit events in response to the event processing. In case of a client with a UI, outgoing events are assigned and eventually mapped in the physical interaction component to concrete UI elements that are able to trigger the corresponding EHs. The UI elements trigger the event either on display of the UI elements or in response to user interaction.

The objective of the concept of the EH graph is to structure the service design by using the abstractions Service, Dialog, Dialoglet, Composite and Simple Event Handler. Each of these plays an important role in service UI design in practice. By providing the EH graph for describing these abstractions, the vocabulary of the designers informal design practices is matched. This makes it easy for the designer to map its vocabulary to the abstractions, both in terms of formalizing an informal specification and communicating the results to other stakeholders.

The EH graph as an implementation of the dialog component runs inside an EH graph interpreter and contains the description of the service logic in EH graph XML (EGXML). It handles the event sequencing and processing. The following hierarchical structures help the designer to organize the service logic into a dialog model.

1. **Simple Event Handler.** An EH is an abstract interaction object. It contains the necessary information on how to handle an event coming from a UI and to delegate the processing of the event and its associated data. It is a concrete UI object's target and represents it from a behavioral point of view.
2. **Composite Event Handler.** An EH is composite, if it is composed of other EHs.
3. **Dialoglet.** A dialoglet consists of a number of EH, which belong to one group – logically and semantically.
4. **Dialog Profile.** A Dialog Profile consists of a device profile and one or more dialoglets.
5. **Dialog.** A dialog is designed to represent a task or a sub task of a specific Web-based service. A dialog contains one or more dialog profiles. A dialog profile represents the dialog through the filter of a specific device profile. A dialog is composed of an initial dialog, from which other dialogs are chained.
6. **Service.** A service is composed of a sequence of dialogs.

During the design of the EH graph, the difficulty consisted of the support of a wide variety of possible UIs for the access of web-based interactive services. The least sophisticated format determines the features of the EH graph. This effect is also known as the least denominator problem. The EH graph format incorporates elements that can be transformed to equivalent elements in all target formats. For example, the graphical UI of a service intended for a desktop computer may be quite different to a UI that is appropriate for a mobile telephone with a very small display. However, although the concrete UI elements are quite different for each target device and the layout mechanisms vary, the interaction mechanisms are similar.

EGXML ADAPTATION

The definition of a single dialog model is still oriented at the "one device - one functionality" paradigm, but today we can access mutually any service through any device [5]. This requires an appropriate mechanism to dynamically adapt the dialog model. In this article a dialoglet of the dialog model will be represented by a two-dimensional discrete function $e(x, y)$, which is digitized both in spatial coordinates and feature value: $dialoglet = [e(x, y)]_{P \times Q}$ where $P \times Q$ is the size of the dialog, (x, y) denotes the spatial coordinate and $e(x, y) \in EH$ the type of EH of the EH graph from the set of available EHs EH. Without loss of generality we consider only the case, where $Q = 1$.

Clustering EHs that implement the dialog model into a hierarchical structure of dialoglets is the essential step in our adaptation process that leads to device independent authoring (Figure 4). For this, the dialog model adaptation process partitions the EHs implementing the dialog model into

non-intersecting dialoglets such that each dialoglet satisfies a homogeneity predicate. We consider the case, where the current dialog 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 or mobile telephone with a much smaller screen. This situation is typical for mobile computing: Services target primarily desktop PC with a monitor and latter are ported to a wide variety of mobile computing devices. The situation in which a service targets small devices and is accessed by a desktop PC with a monitor is not further discussed here.

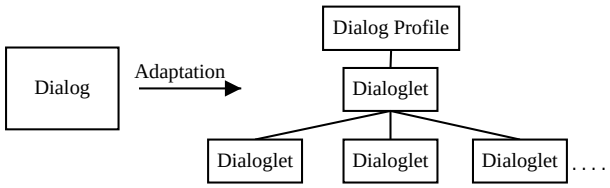

Figure 4. Dialog adaptation.

Formally, the process of adaptation of the dialog model can be defined as follows: If a dialog model consists of a set of EHs and P is a homogeneity predicate, then the adaptation of the dialog model is a partitioning of EHs into a set of connected dialoglets $(d_1, d_2, \ldots, d_n)$, which will eventually be converted into a hierarchical navigationable structure of dialoglets, such that:

$$\begin{aligned} dialoglet &= \cup_{i=1}^{n}(d_i \backslash e_{navigation}(d_i)) \\ d_i \cap d_j &= \emptyset, i \neq j \\ d_i &\text{ is a connected set of event handlers} \\ P(d_i) &= \text{true}, i = 1, \ldots, n \\ P(d_i \cup d_j) &= \text{false, if } d_i \text{ is adjacent to } d_j \end{aligned}$$

A user accessing a service supported by a dialog model needs to navigate from one dialoglet to the next dialoglet. However, not all navigation elements are in the original dialog. Thus, they have to be integrated into the dialoglets resulting from the adaptation process. The set of all UI elements in the dialoglets equals the EHs in the original dialog plus the integrated new EHs dedicated to the navigation between the dialoglets, the $e_{navigation}(d_i)$.

EGXML Adaptation using Event Handler Clustering

The above definition of the process of adaptation is very similar to image segmentation as defined in [16]. Analogous to segmentation and clustering processes, the more context and domain information is known beforehand and integrated into the process, the better the process' results.

Approaches exploring dialog model adaptation can broadly be divided into two categories. Processes of the first category do not consider context knowledge such as screen size during design time. They work bottom-up and rely uniquely on dynamic adaptation of the dialog model. The other category explicitly uses top-level domain and task model knowledge during design time. The processes are configured with a priori known target contexts. The quality of the latter approach depends on the configuration and the type of content that is presented. The former approach has the drawback of working only on syntactic information. We propose a hybrid approach that combines the advantages of both approaches, fast design, no need to produce sophistic configuration data and integration of semantic information.

The two main challenges of the hybrid approach to dialog model adaptation are: How to incorporate low-level semantic information into the dialog model? How to adapt the dialog model respecting the semantic information? Our adaptation technique is based on a linking strategy of two hierarchies of graphs [12, 14]. It allows remodeling a dialog of the dialog model into dialoglets of connected EH and the use of low-level task model information.

The elements of the dialog model are placed as EHs into a stack of regular grids, as illustrated in Figure 5. In the lowest level of the stack, each cell of the grid corresponds to a single EH. Each cell of level $i + 1$ represents a group of cells of level i. The adaptation algorithm always forms linear structures 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.

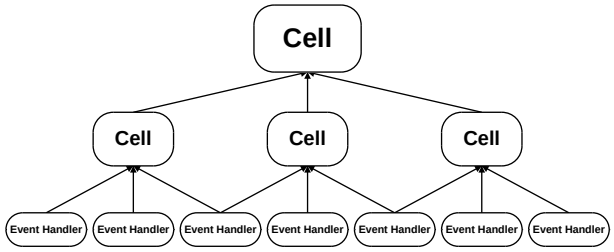

Figure 5: Stack of a regular grid of cells that places a structure on a set of EHs. Three EHs form a cell on the lowest level. Cells on a lower level are candidates for cells on a higher level.

The clustering of a set of EHs into a set of dialoglets is done within the boundaries of the induced stack of cells and is of primary interest. To come to the final set of dialoglets, we dynamically build up a stack of EH-regions. An EH corresponds to a EH-region on the lowest level. Adaptation of a dialog is performed by clustering EH-regions of level i into EH-regions of level $i + 1$. However, EH-regions can only be grouped within the boundaries of a cell in which they reside, as illustrated in Figure 6, and if they satisfy the homogeneity predicate. This guarantees that we cluster only connected and adjacent EH-regions. Complete EH-regions, i.e. regions that cannot further be clustered, result in dialoglets.

The framework to describe the adaptation technique is the description as a hierarchy of graphs. The first hierarchy of graphs forms a syntactic based and static structure that guarantees that the resulting dialoglets are connected. The second hierarchy is dynamically built up respecting the low-level semantic information integrated into the dialog model

at design time. The two hierarchies of graphs implement the dialog model adaptation process. The process consists of the following four phases:

- **Bottom-up Clustering.** EH-regions of level i are grouped into EH-regions of level $i + 1$ within the boundaries of their cell and satisfying a predicate P.
- **Top-down Separation.** EH-regions that fail to group on level i are separated recursively down to level 0.
- **Horizontal Separation.** Large-sized EH-regions of level i, especially when they contain a single EH, are split.
- **Relinking.** The user should be able to navigate from one dialoglet to the next dialoglet. To ensure usability, EH-regions are relinked by integrating additional navigation EHs.

Bottom-up Clustering: The clustering process determines the set of connected EH-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 are determined. Each subcell has a set of regions associated 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 they have a common subregion: u_{i-1}. Regions of the lowest level are connected with their neighboring regions. The overlapping structure of the stack of cells guarantees that the clustering process considers only those regions, which are connected or have a path of connected regions on the lowest level. The clustering process is illustrated in Figure 6.

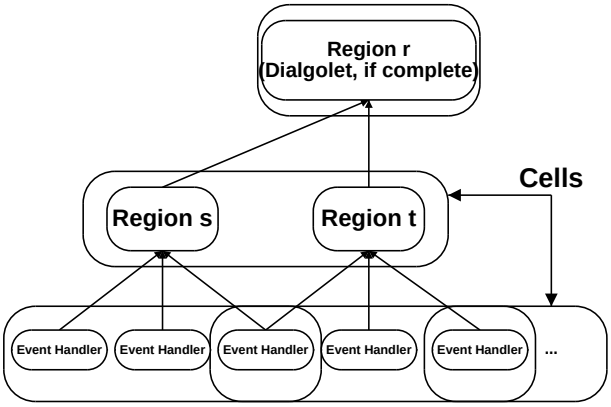

Figure 6: Clustering process. Regions are grouped within the boundary of a cell.

The homogeneity predicate decides, if regions will be clustered or not. The predicate consists of two parts, which both need to evaluate to true; $P(r) = Size(r) \wedge Context(r), r \in R_i$.

- **Size.** On different devices a dialoglet is displayed with a varying number and size of EHs. If the size of a region and its parent region is lower than a predefined threshold (e.g., three times of the screen size) the regions are clustered, otherwise they are separated, either horizontally or top-down. The size of a region is device dependent.
- **Context.** The designer of the original dialog model integrates in it semantic information. The information deals with the semantic relation of an EH with its neighboring EHs. A region s_i and its tentative parent region r_{i+1} will be grouped, if their semantic intent does not exceed a predefined threshold $d(\sigma(s_i), \sigma(r_{i+1})) < \Theta$. In the current version of the adaptation process, we simply assign integer values to EHs, to indicate semantic similarity. $d(\cdot, \cdot)$ is a distance measure like the Euclidian distance.

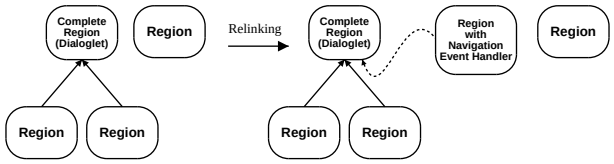

Figure 7: A region containing a single navigation EH will replace a *complete* region. The new region takes part in the bottom-up clustering phase on behalf of the *complete* region.

Top-down Separation: If the grouping process fails, because a region s_i does not satisfy the homogeneity predicate P, the region need to be separated from its connected region t_i. The region need to be separated since they have a common subregion u_{i-1}, which needs to be assigned to a single parent region (Figure 6). The separation process assigns the common subregion to the region, whose semantic value is the most similar. The process is recursively applied down to the lowest level. For level $i - 1$ in Figure 6 it would be applied to u_{i-1}, the common subregion of s_i and t_i, and to those subregions of regions of level i, which have a common subregion with u_{i-1}.

Horizontal Separation: If the size of a region r_i prevents it from clustering with other regions, although it could from the homogeneity predicate's point of view, it is split into a sequence of n smaller, mutually linked regions $r_{0,i}, r_{1,i}, \ldots, r_{n,i}$. E.g., a lengthy text message is split into a sequence of regions or EHs containing each a part of the text message. Only the head of the sequence continues to take part in the grouping process.

Relinking: A region that cannot further be clustered with other regions into a region of a higher level is called *complete* and results in a dialoglet after the adaptation process. A *complete* region that has reached the threshold of the maximal allowed size or that cannot further be clustered from a semantic context point of view does not drop out of the grouping process. Instead, a new region is created containing a single navigation EH 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 7. 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 EHs of the original dialog. The intermediate nodes of the tree-structure are regions including the navigation EHs that have been created in the relinking process.

The set of complete regions resulting from the adaptation process are transformed into a set of dialoglets and eventually into a concrete UI applying the Presentation Integrator SPU and the Encoding Transformer SPU of MUSA.

RESULTS

To illustrate the adaptation technique of a dialog model, we have implemented a message board service build with software agents for a location-based systems [7]. 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 the geographical position. Different users use different devices to access the message board such as laptops, PDAs, or mobile phones. The dialog model

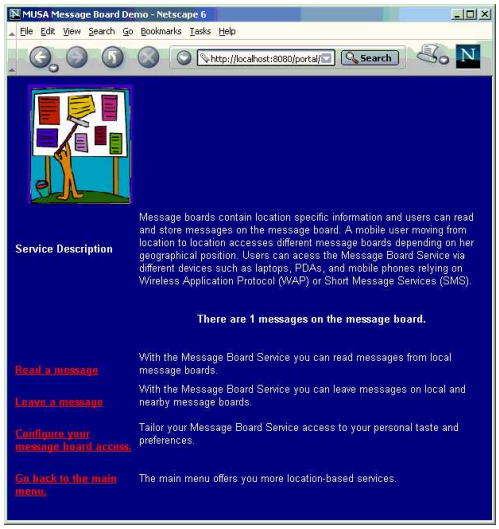

Figure 8: HTML browser showing the Message Board ”Main Menu”.

that results in the graphical UI on a HTML browser is shown in Figure 8. It shows the UI of the message board service. This browser is a powerful tool, so that there is no need to perform any adaptation of the dialog model. Additionally to the dialog a global presentation model is defined, which is responsible for the layout of the Web-page. The two aspects that guide the adaptation process are size and context. The context information is inserted into the dialog model at design time. Size, however, or the screen space that is available for presentation of the UI, is device dependent. The adaptation process needs size information of the device’s UI that accesses the service. This information is delivered in device profiles.

Figure 9 shows the same dialog model that results in the UI of a HTML browser in Figure 8, but this time adapted to the small screen of a mobile telephone. There are two things to note. First, the menu is hierarchically structured into a

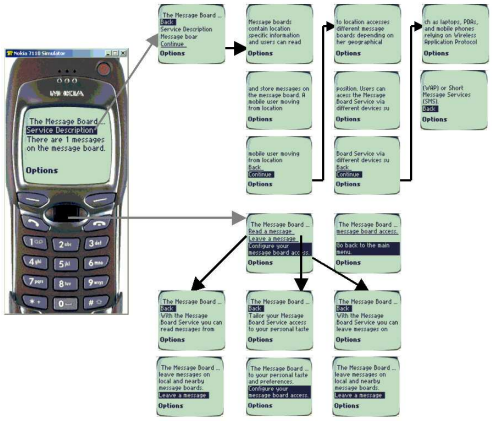

Figure 9: WML-Browser showing the Message Board ”Main Menu” on a mobile telephone.

two level menu, with a main menu containing links to each menu item, which are presented on their distinct screen. The main menu is created during the relinking process of the adaptation and is not present in the original dialog model. The clustering process groups the newly created navigation UI elements together, which results in the main menu. Second, the service description, which is a lengthy text, is split into a series of screens, which are linked with each other. The user navigates with the ”Continue” and ”Back” links from one screen containing part of the description to the next screen. The size threshold of the homogeneity predicate for this adaptation process is set to three device screen sizes.

Figure 10 shows the results for the device profile with the size threshold set to two device screens. The adaptation process has added another level of indirection. The main menu has a hierarchical structure of depth three to cope with the small screen size. The figure shows only part of the collection of UI screens. It illustrates the different hierarchical menu structure in comparison to Figure 9.

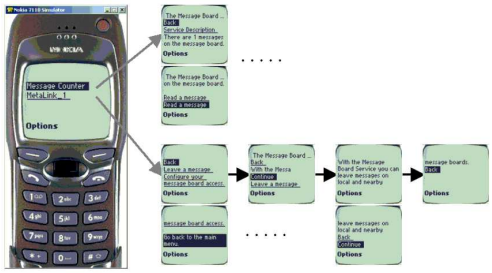

Figure 10: WML-Browser showing the Message Board ”Main Menu” on a mobile telephone with size threshold of two device screens.

CONCLUDING REMARKS

The article has presented the MUSA project. It is a novel approach to device independent authoring. Adaptation of a dialog model represented by the EH graph in EGXML is based on bottom-up clustering and top-down separation us-

ing low-level semantic context information. It results in a hierarchical structure of dialoglets by clustering, separating, and relinking regions and EH. The process is guided by low-level semantic information that is provided by the designer of the dialog model at design time. The adaptation process remodels dynamically a presentation of the dialog model to better fit it to the current device.

The presented experiments with the dialog 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.

In our future work, we will elaborate the adaptation algorithm to include user specific settings such as window size of the running application or user-preferred font size. We conduct experiments with more complex dialog models. The integration of task model related information into the dialog model is somehow simple. Exploration of more powerful but equally simple methods needs to be carried out. However, simplicity for the designer is an important objective to encourage 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. Mark Alpert. Machine Chic. *Sci.Am*, August 2002.

3. 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.

4. 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.

5. Christian Elting, Jan Zwickel, and Rainer Malaka. Device-Dependant Modality Selection for User Interfaces – An Emprical Study. In *ACM IUI*, San Fransisco, California, USA, January 2002.

6. Mir Farooq and Marc Abrams. Simplifying Construction of Multi-Platform User Interface Using UIML. In *UIML Europe Conference*, "March" 2001.

7. Sebastian Fischmeister. Mobile software agents for location-based systems. In *Agents and Software Engineering*, volume 2592 of *LNCS*, pages 226 – 239. Springer Verlag Heidelberg, 2003.

8. M. Florins and J. Vanderdonckt. Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. In *Conference on Intelligent User Interfaces (IUI04)*, pages 140 – 147, Funchal, Portugal, 2004.

9. Futuremind. The Next Generation in Light and Sound Technology Transforms your PC into the Ultimate Mind Machine, 2002.

10. Glenn E. Krasner and Stephen 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.

11. Aaron Marcus and Eugene Chen. Designing the PDA of the Future. *Interactions*, 9(1):34–44, 2002.

12. Guido Menkhaus and Sebastian Fischmeister. Dialog Model Clustering for User Interface Adaptation. In *Web Engineering, Proceedings of ICWE 03*, pages 194 – 203. 2003.

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

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

15. Paulo Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. In *Proceedings of DSV-IS2000*, pages 207–226, Limerick, Ireland, June 2000.

16. Nikhil R.Pal and Sankar K.Pal. A Review on Image Segmentation Techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

17. Subhasis Saha, Mark Jamtgaard, and John Villasenor. Bringing the Wireless Internet to Mobile Devices. *IEEE Computer*, 34(6):54–58, 2001.

18. M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Persoanl Communications*, pages 10–17, August 2001.

19. Ben Shneiderman. Direct Manipulation for Comprehensible, Predictable and Controllable User Interfaces. In *Intelligent User Interfaces*, pages 33–39, 1997.

20. P. Szekely. Retrospective and Challenges for Model-Based Interface Development. In F. Bodart and J. Vanderdonckt, editors, *Design, Specification and Verification of Interactive Systems '96*, pages 1–27, Wien, 1996. Springer-Verlag.

21. W3C. Mobility Access Activity Statement, 2001.

22. C. Wong, H.H. Chu, and Katagiri M. A. Single-Authoring Technique for Building Device-Independent Presentations. In *W3C Workshop on Device Independent Authoring Techniques*, 2002.

23. UIMS Tool Developers Workshop. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 24(1):32–37, 1994.

24. Xybernaut. poma, 2002.