Splitting Rules for Graceful Degradation of User Interfaces

Murielle Florins[1], Francisco Montero Simarro[1,2], Jean Vanderdonckt[1], Benjamin Michotte[1]

[1]IAG/ISYS, Université catholique de Louvain, Place des Doyens 1, B–1348 Louvain-la-Neuve (Belgium)
[2]Escuela Pol. Sup. de Albacete, Univ. de Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete (Spain)

{florins, montero, vanderdonckt, michotte}@isys.ucl.ac.be, fmontero@info-ab.uclm.es

ABSTRACT

This paper addresses the problem of the graceful degradation of user interfaces where an initial interface is transferred to a smaller platform. It presents a technique for pagination of interaction spaces (e.g., windows, dialog boxes, web pages) based on a multi-layer specification in the user interface description language UsiXML. We first describe how an interaction space can be split using information from the presentation layer (Concrete User Interface). We then show how information from higher abstraction levels (Abstract user Interface, Task model) can be used to refine the process. This technique belongs to a collection of transformation rules that have been developed to adapt a user interface to smaller, more constrained displays.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *User interfaces*. H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces*.

General Terms

Design, Human Factors, Languages.

Keywords

Design, graceful degradation, multiplatform systems, pagination, splitting rules, user interface extensible markup language.

1. INTRODUCTION

Computer-based information systems are an essential part of modern organizations. Users of these systems have often to deal with a variety of computing platforms from which they expect to have access to the same data and functionalities. Those computing platforms range from desktops and laptops to PDAs and mobile phones. Their capabilities are very different, especially in terms of screen size and resolution. Designing multiple user interfaces (UIs) for such different platforms remains a challenging and difficult task, implying perpetual trade-offs between:

- The usability of each particular version: each UI should be adapted to its particular platform;
- The cross-platform consistency.

One design method in particular, referred to as *Graceful Degradation of user interfaces* [4], addresses this trade-off between continuity and adaptation. The Graceful Degradation approach consists in specifying one source interface, designed for the least constrained platform, and to apply transformation rules to this source interface in order to produce specific interfaces targeted to more constrained platforms. These transformations rules [4] include:

1. *Splitting rules*, which split the initial UI into chunks;
2. *Interactor and image transformation rules* (e.g., widget substitution), which transform the initial widgets into smaller widgets supporting the same functionalities;
3. *Moving rules*, which are applied to reshuffle widgets to obtain a UI that consumes less screen space;
4. *Resizing rules*, which are applied to shrink widgets or to re-align them after they have been moved;
5. *Removal rules*, which are applied to delete unnecessary or less useful widgets while preserving the main purpose of the UI.

This paper focuses on the splitting problem because it is a difficult and significant step of the whole process. Automatic pagination has been partially addressed in the existing studies that are described in Section 2. Section 3 presents our reference framework and language. Splitting will be examined at two levels of abstraction: concrete UI (Section 4) and abstract UI (Section 5). Section 6 concludes the paper and suggests some avenues for future work.

2. DISCUSSION OF RELATED WORK

Pagination of web pages has been widely researched. The Covigo library of special tags for HTML [8] implements pagination of web pages at run-time, using simple heuristics such as breaking every fifth <tr> or breaking by size. RIML [10] relies on XHTML and defines additional mark-up which permits to specify paginating containers. After pagination, the sections that belong to a paginating container can be distributed over different pages, while the content of non-paginating containers will be repeated on each resulting page. Unlike the two first approaches, Watters and Zhang's [13] approach can process any pre-existing HTML form using partition indicators such as horizontal lines, nested lists and tables.

Another group of approaches relies on a generic description of the user interface in a higher level language, instead of HTML. Göbel *et al.* [6] use a language called DDL. A DDL dialog is composed of containers and elements. Containers whose elements must appear together are called atomic. Elements are assigned weights indicating their requirements in terms of memory and screen size. Fragments with similar weights are generated, while respecting the integrity of atomic containers. Navigation elements are added to permit navigation between dialog fragments. Ye & Herbert [13] apply similar heuristics to a description in XUL. PIMA [1] also relies on a UIDL, which is converted into multiple device-specific representations. This conversion includes a splitting process. Like other approaches, PIMA's algorithm uses grouping constraints as well as information on size constraints. PIMA also takes navigation into account and the possibility of applying distinct navigation policies between screens resulting from a splitting process.

While the fragmentation methods enumerated so far mostly work on a hierarchy of interface components (i.e. on elements related to the presentation of the UI), the splitting algorithm of the Roam system [3] takes as its input a tree structure combining a task model, which is only a hierarchy of tasks without temporal constraints, and a layout structure. The nodes of the tree can be annotated as splittable or not. Roam's algorithm does not attempt to find the best place to split, but merely places the extra widgets that do not fit in a page onto a new page. Navigation between the new pages is also generated, although without a lot of flexibility. To overcome the shortcomings identified above, our splitting approach should satisfy the conditions:

- Be language-independent, not tied to a given technology: we do not want to write a separate set of algorithms for HTML pages [10, 15, 17] and another one for AWT/Swing windows.
- Not introduce any additional construct, no need for yet another mark-up language specially designed to support pagination, nor any additional language constructs (unlike [10]).
- Be fully designer-controlled. In the approaches described above, no human control is envisaged after the specification stage.
- Use semantic information to help the designer determine where to split: when higher-level specifications, especially task models, are available, these specifications must be used to refine the splitting process. In particular, the temporal relationships between tasks must be used, which is not the case in [4].
- Be able to adapt the dialog (i.e., the transitions between pages or windows) in a flexible, customizable way. Most of the splitting approaches do not fulfill this requirement.

3. FRAMEWORK AND LANGUAGE

To address the requirements identified in Section 2, our splitting approach will rely on a high level description of the initial user interface. This description will be expressed in the user interface description language UsiXML [7]. The principles set out below are, however, generally applicable and other UIDLs such as XIML [11] or UIML [1] could also be used. UsiXML is a single language, but structured in four abstraction levels, following the 'CAMELEON reference framework' [3] (see Figure 1).

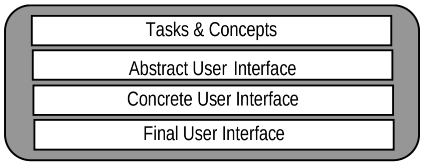

Figure 1. The four abstraction levels used in the framework.

The final user interface (FUI) refers to the actual UI (code). The concrete user interface (CUI) is a specification of the UI in terms of widgets (concrete interaction objects in UsiXML), layout, navigation and behaviour. The abstract user interface (AUI) abstracts the CUI into a definition that is independent of any interaction modality (such as graphical, vocal or haptic modes). The AUI expresses a UI in terms of interaction spaces, i.e. the grouping of tasks that have to be presented together to carry out the task. In UsiXML, the AUI is populated by abstract components and abstract containers. The tasks and concepts level describes the interactive system specifications in terms of the user tasks to be carried out and the domain objects manipulated by these tasks. In the remainder of this paper, these levels will be successively considered to demonstrate that the splitting algorithm will satisfy all the requirements identified in Section 2.

4. SPLITTING AT THE CUI LEVEL

Not all the layers listed above are mandatory in a user interface specification in UsiXML. In the simplest case, the designer may just produce a description of the concrete user interface (CUI). The CUI may be built by hand (using any XML editor or text editor) or with a graphical editor (GrafiXML, www.usixml.org). Different constructs in the CUI model of UsiXML can be used for pagination purposes:

- The layout of each container (e.g., a window, or a dialog box) is specified using embedded *boxes*. Those boxes are declared as *splittable* or *unsplittable*, which is the basic ingredient for pagination.
- Each container and each component is marked as *pageable* or *unpageable*. Pageable components can be distributed between the graphical containers created during the splitting process, while unpageable components must be present in each fragment. For example, a menu bar or a widget for logging out of the system should be considered as unpageable components because their presence in each container is required.
- *Transitions* can be specified between each pair of containers.

Implementing splitting rules starting from the CUI is straightforward: the splittable attribute tells us where to split, and the pageable attribute indicates which elements will be duplicated. Each execution of our splitting rules is fully controllable and configurable by the designer. The parameters of the algorithm are:

- The number of interactive spaces at output.
- The content of the n interactive spaces at output.
- The names assigned to each interactive space at output, which will be used as windows titles and for widgets pointing to these interactive spaces.
- The type of transitions generated between the new interaction spaces generated by the splitting algorithm. Four types of transitions are proposed: linear navigation (e.g., through 'next-previous' links or buttons), indexed navigation (creation of a new page, the index, which links to the other interaction spaces), mixed navigation (combination of linear and indexed navigation) and fully-connected (typically rendered as a tabbed panel).

Default values are provided for each parameter. The splitting algorithm has been integrated into a plug-in for the GrafiXML environment. This plug-in implements a collection of transformation rules (see section 1), to be applied to a source CUI in order to produce specific interfaces targeted to more constrained platforms.

5. SPLITTING AT THE AUI LEVEL

In the previous section we discussed how the designer can develop the pagination process from an existing CUI. This scenario can be expanded by considering the task model and the AUI corresponding to the CUI. With this information, the pagination process can produce results that are more meaningful from the user's point of view, since the task model drives the pagination. The task model we are using is more than a hierarchy of tasks: temporal operators connect sibling tasks and the task model is not simply mirroring the hierarchy of presentation components (unlike Roam [4]). Our splitting algorithm has been integrated into the IdealXML environment [8]. IdealXML is an interface development environment which allows designers to specify user interfaces in UsiXML at different abstraction levels: task model, AUI model, and mappings between these levels. As explained above, the AUI level is composed of abstract containers and abstract individual components. If the designer decides that a container contains too many components, he or she may choose to split this container into smaller units. The designer selects the abstract container. The tool retrieves the set of leaf tasks linked to the components inside the container. The splitting algorithm produces two subsets of tasks to be integrated into two separate subsets of containers. The original container in the AUI is replaced by two new containers, each containing appropriate components. The mapping between leaf tasks and components remains constant; only the mappings between higher level tasks and containers are updated (see Figure 2).

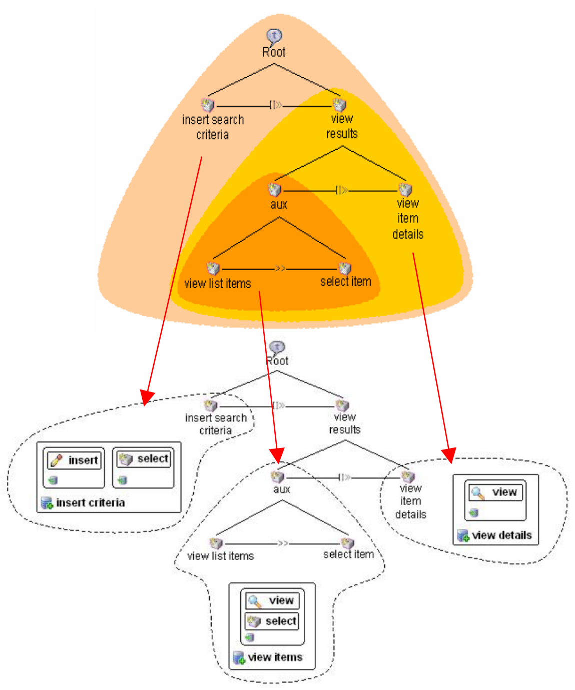

Figure 2. Different paginations and corresponding AUIs.

6. CONCLUSION AND FUTURE WORK

We have described a pagination technique, which relies on a high level description of the UI using the UsiXML UIDL. When applied at the CUI level, the proposed algorithm is quite classical, but goes further than state-of-the art approaches by satisfying the requirements outlined in Section 2. It is language-independent: once splitting has been applied at the CUI level, code can be generated in Java or HTML (thanks to GrafiXML). It does not require any additional constructs but relies on the pre-existing structures of UsiXML. It can be applied automatically, using default parameters, but it can also be fully controlled by the designer, who can choose the number of pages of output, the type of dialog generated and the content of the pages. It suggests a large range of dialog styles, where other approaches often only generate one single result (typically, a sequential navigation). However, the originality of the proposed technique is that it involves UI description at several levels of abstraction. As far as we know, there has been no similar attempt.

In the future, we plan to implement a larger collection of transformation rules, in order to demonstrate how higher level information can be used to improve the transformations. This multilevel approach is quite new. Existing model-based tools which generate several UI versions for multiple platforms adopt a totally different approach: either they generate code starting from a description at the tasks and concepts level [9] (which offers little or no control over the layout and structure of the final interface), or they require a distinct CUI to be specified for each target platform or for each family of target platforms [1] (which demands more work from the designer and offers no guarantee of consistency between the different UI versions). In contrast, our approach requires only one specification, which can be given at any level of detail desired while taking advantage of the information specified at higher levels of abstraction if this is available.

7. ACKNOWLEDGMENTS

We gratefully thank the Salamandre Project, funded by the ‘Initiatives III’ program of the Ministry of Walloon Region under contract No. 001/4511 and the SIMILAR network of excellence, supported by the 6th Framework Program of the European Commission, under contract FP6-IST1-2003-507609.

8. REFERENCES

[1] Ali M.F., Pérez-Quiñones M.A. and Abrams M. Building multi-platform user interfaces with UIML. In *Multiple User Interfaces: Engineering and Application Framework.* John Wiley and Sons, Chichester, UK, 2004, 95–118.

[2] Banavar, G., Bergman, L.D., Gaeremynck, Y., Soroker, D. and Sussman, J. Tooling and system support for authoring multi-device applications. *Journal of Systems and Software 69, 3* (2004), 227–242.

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers 15, 3* (June 2003), 289–308.

[4] Chu, H., Song, H., Wong, C., Kurakake, S. and Katagiri, M. ROAM, a seamless application framework. *Journal of System and Software 69, 3* (2004), 209–226.

[5] Florins, M. and Vanderdonckt, J. Graceful degradation of user interfaces as a design method for multiplatform systems. In *Proc. of ACM Conf. on Int. UIs IUI’04* (Funchal, Jan. 13–16, 2004). ACM Press, New York, NY, 2004, 140–147.

[6] Göbel, S., Buchholz, S., Ziegert, T. and Schill, A. Device independent representation of web-based dialogs and contents. In *Proc. of the IEEE YUFORIC ´01* (Valencia, Spain, November 2001). IEEE Computer Society Press, Los Alamitos, 2001.

[7] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. and Lopez, V. UsiXML: a language supporting multi-path development of user interfaces. In *Proc. of EHCI-DSV-IS’2004* (Hamburg, July 11–13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, 200–220.

[8] Mandyam, S., Vedati, K., Kuo, C. and Wang, W., User interface adaptations: indispensable for single authoring. In *W3C Workshop on Device Independent Authoring Techniques (DIA’2003)* (St. Leon-Rot, 15–26 September 2002).

[9] Montero, F., López-Jaquero V., Vanderdonckt J., Gonzalez P., Lozano, M. D. and Limbourg, Q. Solving the mapping problem in user interface design by seamless integration in IdealXML In *Proc. of DSV-IS’05* (Newcastle upon Tyne, UK, July 13–15, 2005). Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2005.

[10] Paternò, F., Mori, G. and Santoro, C. Tool support for designing nomadic applications. In *Proc. of ACM Conf. on Intelligent User Interfaces (IUI’03)* (Miami, January 12–15, 2003). ACM Press, New York, NY, 2003, 141–148.

[11] Puerta, A.R. and Eisenstein, J. XIML: a common representation for interaction data. In *Proc. of IUI’2002* (San Francisco, Jan. 13-16, 2002). ACM Press, New York, 2002, 214–215.

[12] Spriestersbach, A., Ziegert, T., Grassel, G., Wasmund, M. and Dermler, G. Flexible pagination and layouting for device independent authoring. In *WWW2003 Emerging Applications for Wireless and Mobile Access Workshop* (unpublished).

[13] Watters, C., and Zhang, R. PDA access to internet content: focus on forms. In *Proc. of HICSS’03* (Big Island, Hawaii, USA, January 2003). IEEE Computer Society Press, Los Alamitos, 2003, 105–113.

[14] Ye, J. and Herbert, J. User interface tailoring for mobile computing devices. In *Proc. of UI4All’2004* (Vienna, Austria, 28–29 June 2004). Lecture Notes in Computer Science, Vol. 3196, Springer-Verlag, Berlin, 2004, 175–184.