Design by Example of Graphical User Interfaces adapting to available screen size

Alexandre Demeure, Jan Meskens, Kris Luyten, and Karin Coninx

Hasselt University – tUL – IBBT
Expertise Centre for Digital Media
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
{alexandre.demeure, jan.meskens, kris.luyten, karin.coninx}@uhasselt.be

Abstract. Currently, it is difficult for a designer to create user interfaces that are of high aesthetic quality for a continuously growing range of devices with varied screen sizes. Most existing approaches use abstractions that only support form based user interfaces. These user interfaces may be usable but are of low aesthetic quality. In this paper, we present a technique to design adaptive graphical user interfaces by example (i.e. user interfaces that can adapt to the target platform, the user, etc.), which can produce user interfaces of high aesthetic quality while reducing the development cost inherent to manual approaches. Designing adaptive user interfaces by example could lead to a new generation of design tools that put adaptive user interface development within reach of designers as well as developers.

Keywords: design by example, adaptation, toolglass, design space.

1 Introduction

More and more computing devices have been appearing on the market, each having very different characteristics in terms of CPU power, display size, memory, etc. This situation has led to a renewal in HCI research about User Interface (UI) adaptation, which is nowadays often referred to as plasticity. Plasticity is defined by Calvary et al. [1] as the capacity of a UI to withstand variations of contexts of use while preserving predefined usability properties. The context of use is defined in terms of user (e.g. expert, novice…), platform (e.g. memory, screen size…) and environment (e.g. noise, luminosity…). Instead of trying to cover all, we target the creation of UIs adapted to different screen sizes.

Most of the research that has been done up to now tried to achieve plasticity by relying on automatic approaches [6]. This reduces the cost to create and maintain a user interface for each platform significantly. In addition, creating different user interfaces for different screen sizes manually may lead to inconsistencies between the designs. Unfortunately, most of the automatic UI generation approaches use abstractions that lead to the "greatest common divisor" interfaces that work for all targeted platforms [9]. In practice, the greatest common divisor lead to form based UIs.

In this paper we introduce the possibility to adopt a manual approach to create aesthetic Graphical User Interfaces (GUIs) at low cost. This is achieved with design by example that gives the designer full control over the adaptation process. In this paper, we do not focus on preserving consistency between GUIs neither how to provide guidance to the designer.

There are two main advantages of design by example, as pointed out by Frank [5]. First, the designer manipulates a concrete object rather than its abstractions. Second, providing examples is less complex than programming the corresponding algorithm, which puts the control back in the hand of the designer instead of the developer.

In addition, we blur the distinction between runtime and design time by letting the designer specify examples on the running UI. Thus, the effect of an example is immediately perceivable which highly facilitates a trial and error approach.

In the remainder of this paper, we first discuss the philosophy of our approach. Next, we introduce the algorithms that we use. Afterwards, we discuss how our approach addresses some issues that are often perceived in example-based UI design tools. Finally, we discuss relevant related works, draw the conclusions and provide ideas for future work.

2 Design by example

We define a design space as a set of user interfaces that have similar behavior and goals and support the same set of interaction tasks. Each UI in this design space is appropriate for a certain range of screen sizes.

Fig. 1 illustrates the design space of a UI that supports the user task of selecting a slide number from a running presentation (e.g. PowerPoint). Different UIs can be defined to support this task and populate the design space; The UIs of examples a and b are adapted according to the available screen width, while they are not influenced according to the height. Example c is adapted according to primarily the increase in screen height, while example *d* is adapted to both an increase in width and height. This example design space shows that the presentation for the same task can differ significantly: the structure, style and layout of the user interface are tailored according the screen size when the functionality that is offered remains unchanged.

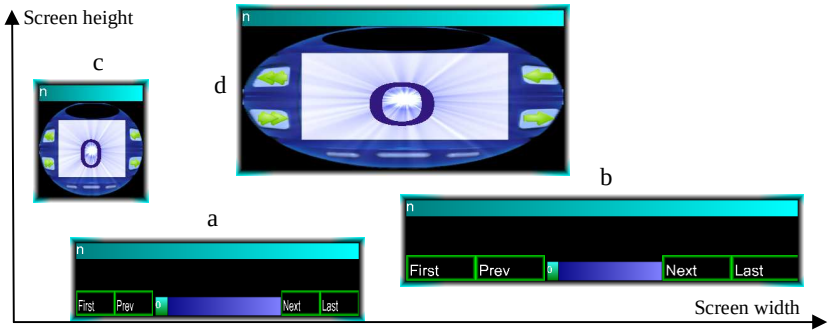

Fig. 1. Illustration of a design space of UIs that supports the user task of selecting a slide.

In our approach, the designer can create these example UI designs for different screen sizes. These UI designs are interpolated and extrapolated to generate user interfaces for all other sizes the designer did not take into account explicitly. Interpolation and extrapolation means that, given some examples for certain screen sizes, the system will propose UIs for all intermediate window sizes (*interpolation*) and even smaller or bigger ones (*extrapolation*). In this sense, interpolation can be considered as a specialization of the design space, while extrapolation extends the coverage of the design space.

The cornerstone of the approach presented here is the design space which can be changed at design time as well as at run time. The designer can specify examples and view the results of interpolation or extrapolation directly on the final running UI [1], and manipulate the design space this way. The designer can use the UI the same way the end-user will use it and edit the design space if the behavior is not what is expected. In order to be really effective and usable by designers, this approach requires the underlying mechanism of interpolation and extrapolation to be easily understandable but yet remain powerful enough to also allow end-users to change the UI according to their preferences. Artistic resizing [4] demonstrates this for the graphical aspect of a user interface.

We aim at maintaining a semantic equivalence during the adaptation process. Therefore, each modification of the UI needs to preserve the semantics (i.e. the user task). That is, tools should ensure that the substitution of an interactor is only possible with one that is compatible with respect to the original semantics.

3 Example inter- and extrapolation

The algorithm we use to interpolate examples is based on the orthogonal interpolation technique used in Artistic resizing [4]. This algorithm is simple and easily understandable even by non-programmers but yet allows for interesting results. However, the algorithm focuses on the graphical representation only, and does not deal with semantical equivalence when resizing. We generalize artistic resizing by providing a common infrastructure to express dependencies between any combinations of UI variables.

Fig. 2 shows a UML schema of the data structure we use to compute interpolation and extrapolation on the base of the given examples. A zone defines a subspace in the screen size space, i.e. a set of points <window width, window height> for which a set of examples is defined. These examples will be used to compute other required user interfaces through the interpolation and extrapolation mechanism whenever the window is resized within the zone boundaries. E.g. the window stays larger than (100,100) and is smaller than (200,200).

Each example is composed of two values: a reference value (e.g. the width of the window when the example was defined) and a related value (e.g. the button position within the window when the example was defined). Every example is linked to one related variable (e.g. the related variable corresponding to the x-coordinate of the button position). A related variable contains a function for computing interpolation and extrapolation from a set of examples (e.g. linear interpolation). Each related

variable is linked to one reference variable (e.g. the reference variable corresponding to the window width). Each reference variable is linked to one or more zones.

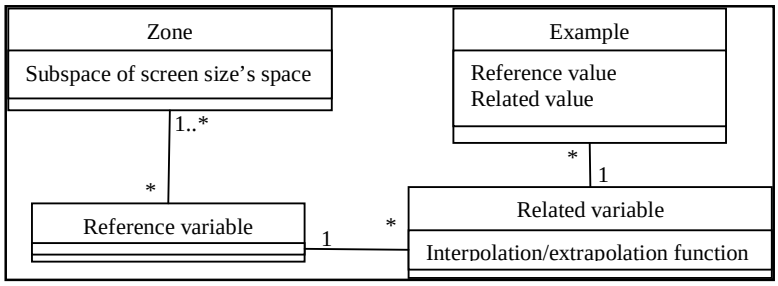

Fig. 2. UML class diagram representing the interpolation data mechanism.

The designer can define zones in which examples are or are not taken into account to compute the inter- and extrapolation. As mentioned before, zones are subspaces in the space of reference variables (usually width and height). In order to correct the behavior of the UI, the designer may define two zones as shown in Fig. 3. A subset of the examples can be associated with each zone (Fig. 3): a and b for the lower zone, c and d for the upper one.

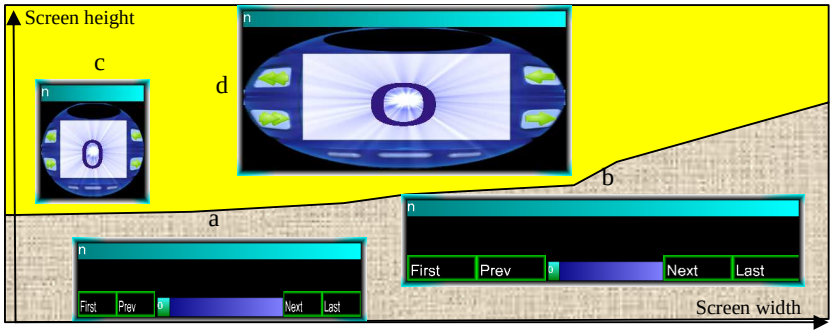

Fig. 3. Designer can define zones where only a subset of examples applies.

Using this technique, the designer has the possibility to construct fine grained UI behavior. This behavior is not only restricted to interactors' position, size or rotation, but also to its type. The key point for success is how to give the designer the right tools to edit these zones and examples. We currently explore ways to realize such tools[1].

[1] **http://iihm.imag.fr/demeure**

4 Related work

Artistic resizing [4] was a source of inspiration for this work. It allows designers to build visual variants of graphical objects by using their traditional tools (e.g. Adobe Illustrator). These variants are used as examples which are interpolated and extrapolated afterwards. Given our objectives, the two main restrictions of this work are: (1) it is not possible to substitute a graphical object by another; and (2) the designer has to switch between tools such as Adobe Illustrator and the Artistic resizing runtime environment to verify the effects.

Collignon et al. [2] propose an intelligent editor for multi-presentation UIs. This editor allows specifying several versions of the UI, each adapted to a certain screen size. To build each UI version, another editor is used (GraphiXML in this case). The selection of the right version is done at runtime, depending on the available screen size. However, the designer cannot edit the behavior of the UI when it is already in operation. Therefore, with respect to our objectives, [2] only applies at design time. Moreover, the different versions of the UI are built separately which makes it difficult to keep them consistent. On the contrary, in our approach, all examples are related to the others since they are just a different view of the same UI. Therefore, every example is semantically equivalent to the others by construction.

Most of the works done in programming by example focus on inferring the mappings between application data and its visual representation [8]. However, Li et al. [7] propose a system to create UI prototypes by graphical demonstration. This system does not aim to design plastic user interfaces but instead provides interesting insights on algorithms that could be used to construct examples. In particular, the authors demonstrate how to detect pivot points and make use of movement paths. This confirms that it is possible to specify more complex behaviors using design by example. We will explore this in the future.

Finally, Stuerzlinger [10] demonstrates the possibility to dynamically substitute interactors of legacy UIs. The philosophy is clearly close to the one that underlies our work: giving the designer (or even end-user) the ability to fully control the adaptation process by directly manipulating the UI. However, “Façades” are quite static; it is currently not possible to resize them. Moreover, it is not possible to build different versions of the UI that will be dynamically chosen while resizing.

5 Conclusion

In this paper, we presented a design by example approach to create adaptive graphical user interfaces. Using this approach, the designer can achieve plastic user interfaces of high aesthetic quality while keeping full control over the design process. This design process consists of three main steps: providing examples, editing the examples in the design space and viewing the behavior of the adaptive user interface immediately and continuously. We provided some tools based on the toolglass metaphor to support this approach directly on the running user interface. This blurs the distinction between design time and runtime, which encourages trial and error and thus lowers the threshold for developing adaptive user interfaces by example.

In future work, we will explore the possibility to dynamically switch – depending on the available screen space – between complex layout algorithms, such as treemaps, flow layout, etc. This would only require small modifications to the toolglass. In addition, we will explore ways to give the designer the possibility to generalize the behavior of an element to a set of elements. We will also explore ways to trigger graphical transitions when changing zone. Those transitions should help the user to understand how the UI was reconfigured (e.g. by using morphing). Finally, we will conduct evaluations in cooperation with designers.

Videos of the system can be found at: **http://iihm.imag.fr/demeure**

Acknowledgments. Part of the research at EDM is funded by ERDF (European Regional Development Fund). The AMASS++ (Advanced Multimedia Alignment and Structured Summarization) project IWT 060051 is directly funded by the IWT (Flemish subsidy organization).

6 References

1. Calvary G., Coutaz J., Thevenin D., Limbourg Q., Souchon N., Bouillon L. and Vanderdonckt J.: Plasticity of User Interfaces : A revised reference framework. Tamodia 2002, Bucarest.
2. Collignon B., Vanderdonkt J., Calvary G.: An Intelligent Editor for Multi-Presentation User Interfaces, In 23ème ACM Symposium on Applied Computing, SAC'2008.
3. Demeure A., Calvary G., Coutaz J., Vanderdonckt J.: The COMETs Inspector: Towards Run Time Plasticity Control Based on a Semantic Network. TAMODIA'2006. Hasselt, Belgium, 23-24 october, 2006.
4. Dragicevic, P., Chatty, S., Thevenin, D., and Vinot, J. 2005. Artistic resizing: a technique for rich scale-sensitive vector graphics. In *Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology* (Seattle, WA, USA, October 23 - 26, 2005). UIST '05.
5. Frank M.. Model-based user interface design by demonstration and by interview. PhD thesis, Georgia Institute of Technology, College of Computing, Atlanta, Georgia 30332-0280, 275 pages, December 1995.
6. Gajos, K. and Weld, D. S. 2004. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international Conference on intelligent User interfaces* (Funchal, Madeira, Portugal, January 13 - 16, 2004). IUI '04.
7. Li, Y. and Landay, J.: A. 2006. Informal prototyping of continuous graphical interactions by demonstration. In *ACM SIGGRAPH 2006 Sketches* (Boston, Massachusetts, July 30 - August 03, 2006). SIGGRAPH '06. ACM, New York, NY, 7.
8. Lieberman H., *Your Wish is My Command*, Morgan Kaufmann, 2001, ISBN 0262140535.
9. Nilsson E.: Combining Compound Conceptual User Interface Components with Modelling Patterns - A Promising Direction for Model-Based Cross-Platform User Interface Development, DSVIS 2002.
10. Stuerzlinger, W., Chapuis, O., Phillips, D., and Roussel, N. 2006. User interface façades: towards fully adaptable user interfaces. In *Proceedings of the 19th Annual ACM Symposium on User interface Software and Technology* (Montreux, Switzerland, October 15 - 18, 2006). UIST '06.