Managing Non-Native Widgets in Model-Based UI Engineering

Dimitrios Kotsalis
Department of Applied Information Technology & Multimedia,
Technological Education Institution of Crete, Stavromenos 71004, Heraklion, Crete, Greece
Phone: (+30) 2810 379 843 – Fax: (+30) 2810 379 767
epp665@epp.teiher.gr

ABSTRACT
This paper sets out to describe on-going research and development activities aiming to provide new insights to building advanced user interfaces by assembling diverse widgets. To this end, we draw upon the relative merits and drawbacks of the two dominant approaches for developing interactive applications, namely toolkit programming and model-based user interface engineering. We motivate the problem by considering a simple example representative of what toolkit programming may deliver and then contrast its implications on prevailing model-based UI principles and practice. Our analysis reveals the key role of widget abstraction in developing specification-based tools to manage radically different widget sets in a uniform manner. The ultimate goal of this work is to extend MBUI engineering approaches so as to enable them to take account of richer interaction vocabularies becoming increasingly available.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer-aided software engineering (CASE), User interfaces*.
D.2.12 [**Software Engineering**]: Interoperability – *Interface definition languages.*
H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User interfaces – *Graphical user interfaces (GUI).*

General Terms
Design, Human Factors.

Keywords
Model-Based UI development, toolkits, specifications, creativity

1. INTRODUCTION
Graphical user interface (GUI) development toolkits constitute a particular type of design language, widely used for constructing interactive software. GUI toolkits make provisions for reusing a standard set of implemented widgets [17], which together with supporting guidelines encourage application developers to use a common design vocabulary [23]. Throughout the years, the development of graphical toolkits has been continuous, addressing cutting-edge issues in 2D graphical interaction [8], information visualization [13], ubiquitous computing [6] and multi-user applications [15]. Recently, there has also been a rich collection of widgets for Web-based applications (e.g., Yahoo, AOL, ICQ, Google Web Toolkit and Echo2). This type of widgets, although not within the scope of the present work, constitute on their own a new category, introducing interactive elements oriented in variant domains, such as weather, mail, TV, etc.

With the advent of network-attachable devices, new input/output devices and mobile terminals the prevalent GUI design language for interacting with software has changed both in ontological scope and symbolic manifestation. As a result, alternative user interface development methods have emerged making use of abstract notations and mark-up languages – typically dialects of XML – to facilitate mapping of abstract components to platform-specific toolkit libraries by delegating the display to a platform-specific renderer [16]. This latter approach is followed by UIML [14], AUIML [4], XIML [12] and many other similar efforts.

Comparing toolkit programming against recent model-based user interface development techniques leads to the conclusion that each development method has relative merits and drawbacks, while they may also conflict at times. Some of the advantages of toolkit programming-based techniques include the capability to build improved interaction facilities to construct novel interaction object hierarchies and to utilize third-party interaction components. The disadvantage is that realizing such capabilities remains programming-intensive and demanding task. On the other hand, model-based approaches using device-independent markup languages are increasingly supported by tools, are less demanding in terms of programming skills, while they adopt some sort of abstraction-based mechanism to make a step towards 'write once, run everywhere' user interfaces [19]. As for disadvantages, these tools are still in an infant state, they support a limited set of interactions, while their multi-platform capability typically does not easily account for the improvements introduced by toolkit programming-based techniques.

2. PROBLEM DESCRIPTION
The major challenge addressed by this PhD work is devising suitable extensions to the model-based UI engineering paradigm for taking advantage of novel and creative widgets available under different platforms. Thus, far model-based UI engineering addresses native components of a target platform. This makes the target presentation vocabulary relatively stable and well defined to facilitate the transformation of abstract UI models to concrete

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EICS'09, July15–17, 2009, Pittsburgh, Pennsylvania, USA.
Copyright 2009 ACM 978-1-60558-600-7/09/07...$5.00.

UI elements of a target toolkit. The limitation is that model-based UI engineering cannot be exploited for the design and development of creative and novel UIs. Instead, the UIs built will always be bound to the interaction affordances of a designated platform. Although this may be sufficient to serve engineering goals such as automatic or semi-automatic generation of UI code or multi platform UI development, it clearly does not suffice to foster creative design or novel UIs.

The rationale for raising this issue as first class research challenge is evident from the pathway followed by model-based UI engineering methods over the past few years. Specifically, since the early efforts aiming to replace the prevailing toolkit programming model for implementing UIs by articulating models, model-based development has shifted its focus to designing UIs. Consequently, if the primary focus remains designing as opposed to implementing UIs, model-based engineering should make inroads to addressing novel design challenges. One such challenge may be served by devising methods to exploit in useful causes available as well as new interaction widgets irrespective of how they are engineered, packaged and made available. It is believed that successful methods aimed at this challenge should focus on 'extending' the model-based UI engineering paradigm so as to make it capable of accommodating alternative concrete user interface models.

3. ANALYSIS

To gain insight to the challenge outlined above it is perhaps worth analyzing some of the primitive concepts involved in manipulating presentation vocabularies.

3.1 Presentation Vocabularies

3.1.1 GUI Native Widgets

As native widgets we define those interactive elements which are offered as part of a predefined widget-set of a particular GUI-toolkit (i.e. Swing, AWT in java or Qt-GUI toolkit as its name implies in QT). These interactive elements/widgets are, in principle, manifested so as to supply programmers with general-purpose constructs facilitating most common graphical user interaction development needs. This has significant impact on reducing development time alleviating the gap between development effort and financial cost. Characteristic examples of native widgets found in the basic widget-set of almost every GUI-toolkit include buttons, lists, trees, tables, to name a few.

3.1.2 Custom Widgets

One popular technique for expanding or extending the capabilities of a toolkit is custom widget construction. This trend has brought about customized widgets of various types as the Toggle switches [10], the AlphaSlider [1], the Fisheye menu [7], the SpiraClock[1], the CrossY system [2], the OrthoZoom system [4], etc. All the above are just a few representative examples of using custom widget construction in order to address UI design goals such as: usability, plasticity, creativity, etc.

3.1.3 Novel Widget Platforms

Originally, the goal of widgets was to simply deliver a miniaturized version of a specific piece of content (e.g., weather forecasts, photo albums, time keeping, etc) outside of the sourcing platform, denoting the couple constituted by the web platform and author. Today, interest in such widgets remains very high and there are few players pushing forward their sets of widgets (Google[2], Opera[3], AOL widgets[4], etc). W3C has responded to this trend by publishing in 2006 a draft of the first widget specification. The goal of this effort is to standardize how widgets are scripted, digitally signed, secured, packaged and deployed in a way that is device independent. A key theme in the specification is widget interoperability aiming towards making widgets as interoperable as possible with existing market-leading user agents on which widgets are run.

The above examples are only indicative of what is widely available today in terms of widgets. Unfortunately, these widgets exhibit a variety of mismatches resulting from the different implementation and run-time environments, different programming models followed and the lack of interoperability. Consequently, making them available as first class interaction elements under one programming-oriented UI development framework seems an impossible task. An alternative is to promote specification-oriented and model-based approaches to declare the capabilities of these widgets through models and to hide platform-specific complexity. In this context, it is possible to envisage a solution, which exploits model-based UI engineering concepts such as abstraction and transformation.

3.2 Model-Based UI Engineering

Model-based UI engineering makes use of abstract notations, typically XML dialects, to facilitate UI specification at multiple levels of abstraction so as to foster multi-platform (i.e.: Teresa [9]), multimodal (i.e.: UsiXML [21]) and ubiquitous UI development (i.e.: GUIDE-ME [22]). All these approaches are based upon the philosophy that UI development can be separated in discrete layers of abstraction. Each layer is responsible for modeling different aspects and behavior of a UI at progressively finer level of detail. In this context, transformational mappings of elements of one layer to another are used to accomplish the refinement of the UI's specification. Recent works, although making substantial progress towards addressing the multiplatform problem, thus contributing to "write once run everywhere" UIs, they exhibit several limitations with regards to the target presentation vocabularies supported. One limitation is the lack of mechanisms accounting for custom widgets or novel widgets and the associated platforms. This is due to the fact that model-based approaches and UI scripting languages target only native widgets of popular GUIs. On the other hand, supporting widget mashups requires much more than mere extensions in the descriptive power of AUI and CUI models. It requires 'boundary bridges' across target vocabularies or widget abstraction as well as mechanisms to enhancing interoperability. The interoperability challenge is not new for user interface toolkits. It was not sufficiently addressed with popular GUI toolkits and it is not being addressed by novel widgets and widget platforms. Devising solutions to support widget abstraction and interoperability will (eventually) lead to assembling plastic-UIs [11] from radically different widget sets.

[1] http://www.emn.fr/x-info/spiraclock/

[2] http://code.google.com/webtoolkit/

[3] http://widgets.opera.com/

[4] http://widgets.aol.com/

4. SKETCHING AN APPROACH

Our current thinking on these issues evolves around three dimensions. The first is concerned with extending elements of the MBUI engineering approach to accommodate descriptions of custom and novel widgets. The second is oriented towards devising abstraction layers for hiding platform-specific details and delegating these concerns to platform-specific pluggable components. Finally, the third dimension is targeted on introducing an abstraction-based workflow capable for decomposing and describing each component irrespectively from its context and target-platform orientation. We intend to approach these challenges/dimensions by utilizing a popular MBUI engineering framework, namely UsiXML [21].

In order to frame the problem we will focus on a particular example to illustrate current limitations in MBUI engineering methods with regards to their capability to manage custom widgets. Take for instance a simple case where a user must navigate through hierarchical data.

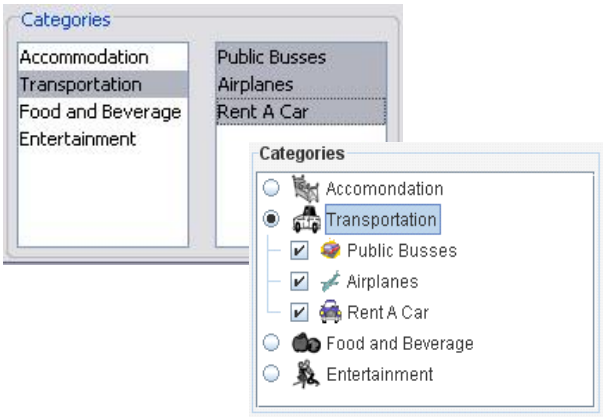

Figure 1: Basic versus creativity-based widget

Figure 1 (upper left) depicts a basic widget combination which could be described by a UIDL. On the other hand, the same scenario could be facilitated by utilizing the RadioCheckTree [2] as shown in the bottom right hand side UI in the figure. The particular component is an augmented version of the Swing's JTree and it is useful in cases where nested selection (i.e. pre-selection followed by multiple object selection) is required. With RadioCheckTree, navigation of large hierarchical data sets is made easier and more intuitive. RadioCheckTree, incorporating toolkit based programming strategies as described in [2], is comprised by a number of extensions to the predefined Swing's tree component. In general there were introduced four major classes facilitating enriched rendering capabilities while being able capturing model's possible states in terms of MVC. Nevertheless, present UIDLs cannot be used to generate a suitable model specification or transformation, which would make use of such a component.

In order to address such limitations, extensions will be required in the models and transformational mappings employed by UsiXML to arrive at a CUI. Specifically, a more abstract approach to promoting language extensibility is based on introducing inline elements to facilitate richer specifications of semantic properties of custom widgets. Then, appropriate transformation rules should map specifications to CUI elements capable of linking to the implemented RadioCheckTree component.

Taking this further, one could imagine the same scenario extended to utilize instead of RadioCheckTree, appropriate visualizations of the hierarchical data set. Figure 2 presents the same hierarchical data set using JGraph[5] in an alternative prototypical implementation. Again, similar challenges are raised with regards to how UsiXML could cope with such a UI instance.

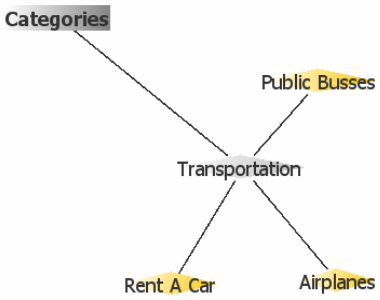

Figure 2: Graphical representation of hierarchal data

Finally, it is also worth mentioning that there may be design problems requiring the concurrent realization in the same UI of a component such as the RadioCheckTree (which would serve for orientation) and a visualization library such as JGraph (which would serve for navigation). This raises a more challenging issue, namely widget interoperability, which has proven to be difficult to handle. Our current thinking approaches this challenge by introducing the concept of 'unifying programming models', briefly described below.

The examples presented above represent programming model-compliant designs, as both the RadioCheckTree and JGraph are made available under the same platform. There are cases however that this needed not be so. If this is the case, then extensions in the model specifications and the transformational mappings from one model to another may not be sufficient to support the management and interoperability of radically different widget sets. Consequently, some sort of abstraction is needed to unify the way widgets are specified. A critical issue in this context is decoupling semantic properties of widgets that are independent from any implementation from their platform-specific instantiation, thus alleviating the gap between targeted and ubiquitous interactive vocabularies. Whereas such abstraction layers turned out to be problematic with conventional GUI toolkits available under different platforms [20], model-based techniques could improve upon this due to the more intuitive means (i.e., models) used to support abstraction.

A possible approach could be to define a universal workflow for describing complex and/or non-conventional widgets using well-established HCI models such as Nielsen's interaction model[18]. According to Nielsen, a UI specification is divided in seven levels of abstraction (namely: Goal, Pragmatic, Semantic, Syntactic, Lexical, Alphabetic, Physical), stripping off in each what is considered as redundant. Table 1 summarizes the RadioCheckTree example by highlighting four of Nielsen's seven layers. As shown at an abstract level the RadioCheckTree is an

[5] http://www.jgraph.com/

abstraction of two constituents namely the pre-selection (modeled in terms of optionslist and currently selected item) and nested selection, together with pre- and post-conditions related to the way the widget is interactively manifested on a target platform.

Table 1: RadioCheckTree analysis in respect to Nielsen's model

Abstraction –level	Example (RadioCheckTree)
Goal	Navigation in hierarchical data
Semantic	1. Pre-selector (Optionslist, CurrentSelect) 2. Nested-Selector (CurrentSelect, Sub-Options, UltimateSelect) 3. Preconditions 4. Post conditions
Syntactic	MVC dialogue model
Lexical	Platform-specific details of interaction elements

5. CONCLUSION & FUTURE WORK

In this paper, we have attempted to describe briefly early steps towards a PhD program of study aiming to revisit model-based UI engineering from a perspective which takes account of current and emerging trends in user interface software and technologies. Specifically, our intention has been to raise the issue of managing creative UI designs and how they can be accommodated in the model-based tradition of UI engineering. Early insights to the problem reveal that model-based techniques require several extensions if they are to address the design of richer interaction scenarios.

6. REFERENCES

1. Ahlberg, C. and Shneiderman, B. The alphaslider: a compact and rapid selector. Proc. of SIGCHI'94, pp. 365-371.
2. Akoumianakis, D., Milolidakis G., Kotsalis D., Vellis G. Generic strategies for manipulating graphical interaction objects: Augmenting, expanding, and integrating components, Proc. of ICEIS'2008, 12 - 16 June, Barcelona - Spain.
3. Apitz G, Guimbretiere F., CrossY: A crossing-based drawing application. UIST'04, pp. 3–12.
4. Appert C, Fekete JD., OrthoZoom scroller: 1D multi-scale navigation. CHI '06: Proc. of the SIGCHI, pp. 21–30.
5. Azevedo, P., Merrick, R., and Roberts D., OVID to AUIML - User-Oriented Interface. Modelling, in N.J. Nunes (ed.), Proc. of the UML2000 Workshop Workshop, http://math.uma.pt/tupis00.
6. Ballagas, R., Ringel, M., Stone, M. and Borchers, J. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments. In Proc. of the ACM CHI 2003, pp. 537–544.
7. Bederson, B. B., Fisheye Menus, Proc. ACM UIST 2000, pp. 217-226.
8. Bederson, B. B., J. Grosjean, & J. Meyer. Toolkit Design for Interactive Structured Graphics, IEEE Trans. on Software Engineering, 30 (8), 2004, pp. 535-546.
9. Berti, S., Correani, F., Mori, G., Paternó, F., Santoro, C.: TERESA: A Transformation-Based Environment for Designing Multi-Device Interactive Applications. In: Proc. of CHI 2004, pp. 793–794.
10. Catherine Plaisant, Daniel Wallace: Touchscreen Toggle Design. CHI 1992, pp. 667-668
11. Coutaz, J., Calvary, G.: HCI and Software Engineering: Designing for User Interface Plasticity. The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications. Human Factor and Ergonomics series, Taylor & Francis CRC Press (2008), pp. 1107–1125
12. Eisenstein, J., Vanderdonckt, J., Puerta, A., Model-Based User-Interface Development Techniques for Mobile Computing, Proc. of IUI'2001, pp. 69-76.
13. Heer, J., Card, S., Landay, J. prefuse: a toolkit for interactive information visualization, Proc. of CHI 2005.
14. Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonckt, J., Human-Centered Engineering with the User Interface Markup Language, in Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), "Human-Centered Software Engineering", Chapter 7, HCI Series, Springer, pp. 141-173.
15. Kohlert, D., Rodham, K., Olsen, D. 1993. Implementing a graphical multi-user interface toolkit. (SPE) 23, 9 (Sept.), pp. 981-999.
16. Lee, C., Helal, S., Lee W. Universal Interactions with Smart Spaces, IEEE Pervasing Computing, 2006, pp. 16-21.
17. Myers, B. A. 1995. User interface software tools. ACM Trans. CHI, pp. 64–103.
18. Nielsen, J.: A Virtual Protocol Model for Computer-Human Interaction. International Journal of Man-Machine Studies 24,3 (1986), pp. 301–312.
19. Perry, M., O'hara, K., Sellen, A., Brown, B., Harper,R. Dealing with Mobility:Understanding Access Anytime Anywhere, ACM Trans. CHI'2001, pp.323–347.
20. Savidis A., Stephanidis C., & Akoumianakis, D. (1997) Unifying toolkit programming layers: a multi-purpose toolkit integration module. In: Proc. of DSV-IS'97, pp. 177–192.
21. Vanderdonckt, J., A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of CAiSE'05, pp. 16-31.
22. Viala, J., Dubois, E., Gray, P.: GUIDE-ME: Environement Graphique de Manipulation de la Notation ASUR. In: Canals, G., Giboin, A., Nigay, L., Pinna, A.-M., Tigli, J.-Y. (eds.) ACM Proceedings of the French conference: Mobilite et Ubiquite. 2004, Nice, France, pp. 74–78.
23. Winograd, T., Editor (1995): Bringing Design to Software (New York: Addison Wesley).