Quill: A Collaborative Design Assistant for Cross Platform Web Application User Interfaces

Vivian Genaro Motti
Université catholique de Louvain
Louvain-la-Neuve - Belgium
Place des Doyens 1, 1348
vivian.genaromotti@uclouvain.be

Dave Raggett
W3C/ERCIM
2004, route des Lucioles
Sophia Antipolis - France
dsr@w3.org

ABSTRACT

Web application development teams face an increasing burden when they need to come up with a consistent user interface across different platforms with different characteristics, for example, desktop, smart phone and tablet devices. This is going to get even worse with the adoption of HTML5 on TVs and cars. This short paper describes a browser-based collaborative design assistant that does the drudge work of ensuring that the user interfaces are kept in sync across all of the target platforms and with changes to the domain data and task models. This is based upon an expert system that dynamically updates the user interface design to reflect the developer's decisions. This is implemented in terms of constraint propagation and search through the design space. An additional benefit is the ease of providing accessible user interfaces in conjunction with assistive technologies.

Categories and Subject Descriptors: D.2 SOFTWARE ENGINEERING D2.2 Design Tools and Techniques: User interfaces **Keywords:** model-based user interface design; collaborative assistant

1. INTRODUCTION

People increasingly access the Web from a wide range of devices with different characteristics. Desktop devices such as laptop computers have large high resolution displays, keyboards and a pointer control. User interaction is based upon windows, icons, menus and pointers (WIMP). By contrast smart phones have much smaller touch-sensitive displays. User interaction is driven by touch gestures, and controls need to be large enough for easy operation by the user's fingers. Tablets are similar to smart phones, but with a larger display that allows for richer user interfaces. User interaction with applications running on connected TVs is via a hand held remote, or through a coupled smart phone or tablet. Cars are emerging as the next new area for web technologies, and look likely to boost adoption of multimodal interaction as a response to the need for driver safety. Applications can run in the car's head-unit, or on a connected smart phone.

Application development teams are thus confronted with an increasing range of platforms and interaction metaphors. Businesses will want to provide access to their services across this range. This is putting pressure on developers to ensure a consistent user interface across all of the target platforms whilst containing the development cost and time budget. One approach to this has been the use of high level device independent user interface descriptions that are compiled for each target platform. This approach restricts the ability of developers to directly influence the user interface for each platform. The rest of this paper describes a new approach where the computer acts as a design assistant that collaborates with the developer to maintain consistency of the user interface across platforms as changes are made. For a survey of model-based user interface design practices, see [7].

In many cases, when starting on a new project, the first step is to establish the business requirements. A systems expert can then translate these requirements into definitions for the data interfaces and the user interaction tasks. A user interface expert can use these to work on developing the corresponding user interface. A mock up in PhotoShop may be used for initial review, but it is much better to have working prototypes on each of the target platforms. Existing user interface prototyping tools such as [2] faciliate creation of mockup/wireframe prototypes along with a means to skin the style. Such approaches focus on the concrete look and feel of a user interface, but don't assist with synchronizing the design across multiple target platforms.

In Quill, the user interface design is held as layered abstractions following the Cameleon Reference Framework [3]. The top most layer contains the domain interfaces and interaction task models. This is followed by models describing abstract user interface in a manner that is essentially independent of the target platforms and modes of interaction. Below this is the concrete user interface, which involves a commitment to particular modes of interaction on broad categories of devices. The bottom most layer is the final user interface on each of the target devices. This is generated from the concrete user interface where the detailed styling is determined by a theme or skin provided by the design team. Further information and example languages can be found in [5]

Quill embodies an expert system that generates the abstract and concrete user interfaces from the task and domain model. The human designers adjust this design, and Quill then propagates the implications of these changes to search for a consistent overall design for all of the target platforms. In essence, Quill frees designers to make changes without having to do the tedious and error prone work of synchronizing these across the entire project. In principle, Quill could be integrated as part of Web content management systems.

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

2. DESIGN KNOWLEDGE

The following sections introduce the conceptual structure and algorithms used to realize the design assistant. This is followed by a brief introduction to the authoring user interface and ideas for further work.

2.1 Rich Domain Model

This defines the data interfaces between the user interface and the application back-end. In addition to basic data types, Quill supports default values, examples, constraints and embedded documentation. Constraints include regular expressions that constrain the value of string properties; expressions indicating that a given interface, method or property is relevant based upon the values of other properties; whether a particular property is optional or must be provided by the user; and whether a particular property or interface is persistent, in the sense that the values provided by a user are preserved in between invocations of the user interface.

2.2 Task Models as Basis for Dialog Design

Task models describe user interaction at an abstract level, e.g. which tasks can be carried out *concurrently*, which tasks pass information *enabling* other tasks, and which tasks represent a *choice* in the sense that one of a given set of tasks must be performed. Some tasks are performed by the user whilst others are performed by the system. The task model can be used to determine what parts of the user interface to present in parallel and what parts must be presented sequentially. On a small display, it may be appropriate to break the user interface into a sequence of simple dialogs, while on a larger display, these could be presented together, or split across separate panes in a tab control. Tasks can be annotated to indicate whether they are common or rare. The latter may be handled as part of an *advanced* user dialog. For a diagrammatic notation for task models, see [9].

2.3 Layout Expertise

Layout expertise is needed to generate candidate designs for the concrete user interface. This involves platform specific knowledge, e.g. the difference for touch based controls on a smart phone from those driven by a mouse pointer on a laptop. The design is influenced by rough estimates of the size of each control, based upon information in the domain model, including examples of expected user input. Quill deliberately uses a simple model of layout, e.g. vertical, horizontal and grid[1] layout managers. This is enriched and mapped into CSS when skinning the final user interface generated from the concrete user interface models.

2.4 Design Rules

The design knowledge for the design assistant is expressed as rules. These fall into three categories:

- Rules that propose designs, and which embody design preferences for particular platforms.
- Rules that determine which relationships hold in a given context of use. These rules typically apply across different levels of abstraction.
- Rules that critique designs, for example, looking for color contrasts that would create problems for people who have one of several forms of color blindness.

[1]For a proposal for extending CSS to support rich grid layouts, see [8]

2.5 Abduction and Constraint Satisfaction

One approach to propagating the effects of changes across a design is to use *event-condition-action* rules. When the event occurs the action is triggered if the associated condition hold true. This approach requires every change to be matched with a rule, resulting in the need for large numbers of rules, that then become hard to maintain. A contrasting approach is to express logical relationships across different levels of abstraction. If you know certain facts and also that certain relationships hold true, then it is possible to infer additional facts that must be true if the relationship is to hold. This is generally referred to as abductive reasoning. For simple conjunctive relationships, this can be cast as an extension of relational table joins, using logical variables for values shared across tables. A proof of concept was prepared as an interactive web page [12]. This makes use of a two pass algorithm for performing the logical joins and abducing facts. The demo allows you to enable and disable abduction to see the effects on the results.

Abduction can also be considered as constraint satisfaction. The design space can be exhaustively scanned to find solutions that fulfil the constraints. This can be time consuming depending on the size of the space. If no such solution is found, an explanation can be generated to guide the user (the designer) as to why. The demo cited in the previous paragraph uses an exhaustive search which in the worst case is the cross product of the number of records (facts) for all of the tables in the relationship. If the relationship shares variables across the tables in the relationship, the search space diminishes in size. Each table prunes the size of the search space for the next table to consider. This is an example of constraint propagation which can be used to reduce the size of the search space, but in most cases, you will still have some search left to carry out. A refinement is to use dependency directed backtracking for this search, as this provides a natural basis to generate explanations [10]. A survey of algorithms for constraint satisfaction problems can be found in [6].

3. AUTHORING USER INTERFACE

Quill is being developed as a browser based authoring tool, where the models are held on a web server, and targeting devices that support HTML5, such as desktop computers, smart phones, tablets and connected TVs. A direct manipulation interface is being explored for the task, abstract and concrete user interface models, along with a tabbed metaphor for switching between the different levels of the Cameleon Reference Framework. Figure 1 is a screenshot showing selection of target platforms for the current project.

The concrete UI (Figure 2) supports a palette of components that are dragged onto the canvas. The task and abstract UI (Figure 3) are rendered graphically using automatic force directed layout that treats each node as a charged particle and each link as spring. The layout process occurs dynamically using animation of the HTML5 2D Canvas. The domain model is realized as a syntax colored text editor based upon CodeMirror [4]. A concise text format is used for loading and saving models to the server.

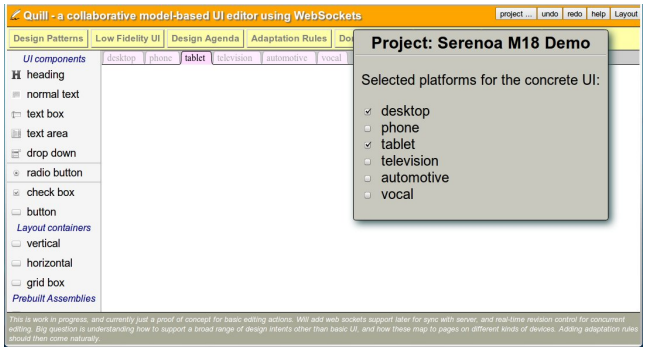

Figure 1: Quill: Target Platform Selection

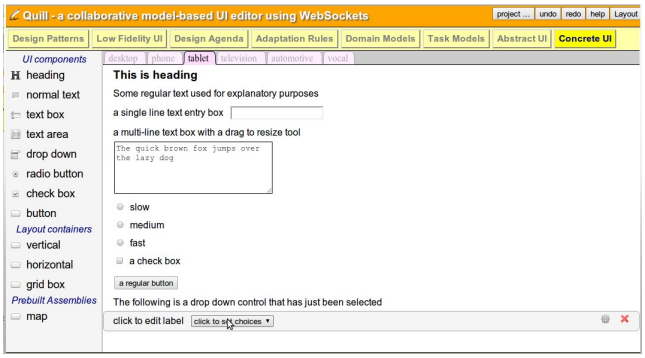

Figure 2: Quill: Concrete UI

Further work is planned to allow for live review of edits by distributed teams of developers. This will be based upon a system where one of the browsers in the editing session is dynamically selected to be the primary editor with the role of automatically reviewing edits proposed by other browsers in near real-time. This involves a 3-way merge based upon analysing changes since a common ancestor version of the model. Proposed changes with respect to that version are mapped to the latest version of the model by propagating them through the effects of accepted changes. The local undo/redo stack for each client operates on changes made by just that client's user and is adjusted to reflect the changes accepted by the primary editor. The algorithms involved operate on hierarchical descriptions of models and sequences of model mutations, and have been tested in an experimental markup editor.

4. NEXT STEPS

Quill is still a work in progress, and will be released as an open source project under the Apache2 license. The idea of a collaborative design assistant based upon constraint satisfaction is still relatively new, and further work is needed to realize the full potential as a means to reduce the cost and time for developing consistent user interfaces across multiple platforms. Of particular interest are the challenges raised by multiscreen applications, e.g. using a smart phone together with a connected TV, or for multimodal applications in cars, where hands free operation is essential. For the relationship to responsive design, see [11].

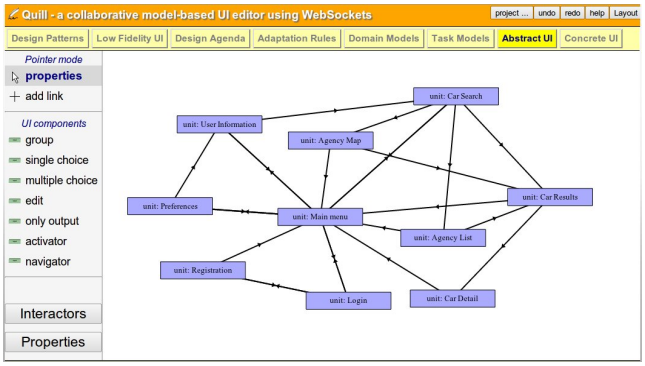

Figure 3: Quill: Abstract UI

5. ACKNOWLEDGEMENTS

This work was supported by funding for the Serenoa Project [1] from the European Commission's Seventh Framework Program under grant agreement number 258030 (FP7-ICT-2009-5).

6. REFERENCES

[1] SERENOA Project. 2012. http://www.serenoa-fp7.eu/.
[2] EaSynth Solution Inc. Ltd. Foreui. 2012. http://www.foreui.com/.
[3] G. Calvary et al. The cameleon reference framework, cameleon project. 2002. http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON20D1.1RefFramework.pdf.
[4] M. Haverbeke. Codemirror – a code editor in the browser. 2013. http://codemirror.net/.
[5] José M. C. Fonseca et al. W3C Model-Based UI Incubator Group Final Report. 2010. http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/.
[6] V. Kumar. Algorithms for constraint satisfaction problems. *AI Magazine*, 1992. http://www.cs.cinvestav.mx/~constraint/papers/kumar-survey.pdf.
[7] G. Meixner, F. Paternò, and J. Vanderdonckt. Past, present, and future of model-based user interface development. *i-com 10(3): 2-11*, 2011. http://giove.isti.cnr.it/attachments/publications/icom%202011%200026%20-%20model-based.pdf.
[8] A. Mogilevsky, P. Cupp, M. Mielke, and D. Glazman. CSS Grid Layout. 2012. http://www.w3.org/TR/2012/WD-css3-grid-layout-20121106/.
[9] F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *INTERACT '97 Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction, Pages 362-369*, 1997.
[10] C. J. Petrie Jr. New algorithms for dependency-directed backtracking. *AI TR86-33*, 1986. ftp://ftp.cs.utexas.edu/pub/AI-Lab/tech-reports/UT-AI-TR-86-33.pdf.
[11] D. Raggett. Responsive Design. 2013. http://www.w3.org/2013/Talks/responsive-design.pdf.
[12] D. Raggett. Testbed for abduction over relationships. 2013. http://www.w3.org/2013/01/abduction/.