Simplifying the Development of Cross-Platform Web User Interfaces by Collaborative Model-based Design

Vivian Genaro Motti[1], Dave Raggett[2], Sascha Van Cauwelaert[1], Jean Vanderdonckt[1]
[1]Université catholique de Louvain, Louvain School of Management,
Place des Doyens, 1 - B-1348 Louvain-la-Neuve, Belgium – +32 10 47{8349, 9013, 8525}
{vivian.genaromotti, sascha.vancauwelaert, jean.vanderdonckt}@uclouvain.be
[2]World Wide Web Consortium – dsr@w3.org

ABSTRACT
Ensuring responsive design of web applications requires their user interfaces to be able to adapt according to different contexts of use, which subsume the end users, the devices and platforms used to carry out the interactive tasks, and also the environment in which they occur. To address the challenges posed by responsive design, aiming to simplify their development by factoring out the common parts from the specific ones, this paper presents Quill, a web-based development environment that enables various stakeholders of a web application to collaboratively adopt a model-based design of the user interface for cross-platform deployment. The paper establishes a series of requirements for collaborative model-based design of cross-platform web user interfaces motivated by the literature, observational and situational design. It then elaborates on potential solutions that satisfy these requirements and explains the solution selected for Quill. A user survey has been conducted to determine how stakeholders appreciate model-based design user interface and how they estimate the importance of the requirements that lead to Quill.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [**Software Engineering**]: Miscellaneous – *Rapid Prototyping; reusable software*. H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces*. I.3.6 [**Computer Graphics**]: Methodology and Techniques – *Interaction techniques*.

Keywords
Collaborative development; cross-platform design; model-based design of user interfaces; user interface description language.

1. INTRODUCTION
Developing the user interface (UI) of an interactive system is notoriously recognized as a complex and powerful [24], yet open, iterative, and incomplete process [22], namely because various stakeholders (end users, developers, designers, analysts, project leaders, marketing people) are involved with different background and inputs [6]. These stakeholders currently face many challenges when dealing with various *contexts of use* [7] in which end users are carrying out their interactive task with the system. Contexts of use vary mainly in terms of [3]: *users' profile* (disabilities, user preferences, and cognitive styles), *platforms* (device types, screen sizes, resolutions and interaction techniques), and *environmental* settings (mobile vs. stationary location, light, noise, and stability).

SIGDOC'13, September 30–October 1, 2013, Greenville, North Carolina, USA.

ACM 978-1-4503-2131-0/13/09…$15.00.
http://dx.doi.org/10.1145/2507065.2507067

Application domains (e.g., e-Health, automotive industry) could also influence these contexts. Thus, the mobility, pervasiveness, and ubiquity of current computing trends also pose challenges when developing UIs with responsive design and with cross-device consistency, high usability, and great user experiences.

Given that it is neither scalable nor feasible to implement several UI versions considering specific characteristics of all contexts of use, developers must rely on approaches that are both: sufficiently generic to simplify the software development lifecycle, and flexible enough to properly accommodate requirements and constraints coming from these different contexts of use.

Model-Based Design of User Interfaces (MBUI) has maintained some attraction due to its main benefits [1,3,5,8,10,14,21]: it permits incremental development for a wide variety of technologies, it enables a common understanding of the UI specification through models by the exchange of common vocabulary, it reduces the cost of targeting multiple platforms, it facilitates changes to be applied at all points in the lifecycle, it enables developers to work top-down, bottom-up or middle-out. These benefits spring from a separation of concerns enabling designers to focus on important aspects of the development process while avoiding distractions with details that are best delegated to specialists in specific platforms. MBUI allows this to happen without incurring the high communication costs normally associated with collaboration between people with different skill sets. Many powerful tools exist today for developing Web UIs [13], the most typical being the interface builder with a visual editor for each corresponding operating system or environment. On the one hand, interface builders do not fully support adaptation to the context of use; on the other hand, there is a lack of tools (e.g., editors, design assistants and development environments) to facilitate MBUI adoption.

To tackle the aforementioned shortcomings, this paper presents Quill, a web-based development environment that enables various stakeholders of a web application to collaboratively adopt a model-based design of the user interface for cross-platform deployment, by defining and editing UI models addressing contexts. A *Systematic Literature Review* (SLR) helped to identify relevant requirements for creating and managing MBUI projects. These requirements lead to design decisions for the implementation of Quill that enables stakeholders to create and manage MBUI projects, and that designers and developers collaborate in the definition of UI models by dragging-and-dropping their components, specifying adaptation rules, and setting specific contexts of use (e.g., concerning the target delivery device).

This paper is organized as follows: Section 2 summarizes the motivations of Quill based on related work, Section 3 presents a set of requirements and their respective design decisions, Section 4 describes the tool and its main features that address these requirements, Section 5 validates the decisions with a case study, Section 6 reports on the results on an experimental study, and Section 6 concludes this paper with final remarks and future works.

2. RELATED WORK

The Cameleon Reference Framework (CRF) [3] defines the structure of UI models in four levels (Figure 1): Task and Domain, Abstract User Interface (AUI - that is independent of any interaction modality and implementation), Concrete User Interface (CUI - that is independent of any implementation for a given modality) and Final User Interface (FUI). Mappings between such levels are also specified (e.g., abstraction, reification and reflexion) and they vary according to specific contexts of use, leading consequently to specific transformations [9]. To support the development of MBUI, interaction modeling has been introduced in software engineering with three pillars: *models* that capture various UI-related aspects (e.g., task, domain, user, platform, environment) along with a *language* [10] that expresses these models, a step-wise *approach* that manipulates these models throughout the development life cycle, and *software* that supports applying the steps of this approach based on the models.

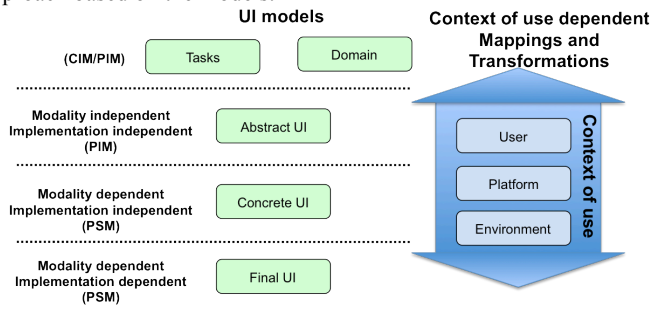

Figure 1. The Cameleon Reference Framework (CRF) [3].

Several MBUI environments have been introduced [1,2,5,8,9, 10,14,18], in particular for cross-platform [14,21], some of them being reported in the W3C MBUI Incubator group report [4]. For instance, Hera [23] provides transformations to generate web applications by relying on a 3-tier framework, integrating semantics, application and presentation aspects to generate UIs. Adaptation is also considered, but mainly focused on the users' characteristics. Mappings are defined and RDF was used to specify data transformations. Roam [5] exploits a task model for automatically generate an UI for different devices equipped with different resolutions. Gummy [14] adopts MBUI for supporting the development of cross-platform UIs in a coordinated way. Although most of these works are dedicated to tackle specific MBUI shortcomings, it is considered hard for them to follow recent advances, such as new technological standards (e.g., languages, approaches, devices, interaction modalities) that could induce a significant change in the contexts of use, regarding mobility, pervasiveness, ubiquity and context-awareness per se.

The first column of Table 1 lists other UI editors currently available for modeling and diagraming the UI of interactive systems. In the first column are listed software products for model creation, editing, and exploitation, but these tools do not support UI models. The second column of Figure 1 lists UI prototyping and sketching tools, but they do not support UI modeling neither. Concerning the UI design, sketching, and prototyping, there are several tools available in the market. Among them is MAQETTA [12], a visual authoring tool for designing HTML5-based UIs adopting an approach where UI elements are dragged from a palette and dropped onto a working area. Because MAQETTA [12] is itself a web-based application, no plug-ins, no add-ons, no download of any piece of software (e.g., a Java application) are required. Although it permits the design of UI mockups, the model-based approach is not integrated [12].

Table 1. Graphical Editors for Modeling, Diagraming, Sketching and Prototyping UIs [Source: Wiki (http://en.wikipedia.org/wiki/List_of_UML_tools) and Tools (http://c2.com/cgi/wiki?GuiPrototypingTools)]

Models and Diagrams	UI Sketching and Prototyping
ArgoUML: supports UML modeling runs as a Java platform being distributed under Eclipse license (http://argouml.tigris.org/)	**Balsamiq:** for sketching interfaces rapidly, and communicate design ideas (www.balsamiq.com)
Dia: for drawing diagrams for generic purposes, under GPL license (https://live.gnome.org/Dia)	**JustInMind:** a platform for defining prototypes for web and mobile applications (www.justinmind.com)
Visio: commercial editor of Microsoft for creating and sharing diagrams, enables collaborative features (http://visio.microsoft.com/)	**MAQETTA:** a visual authoring tool of HTML5 user interfaces in the browser, open source, WYSIWYG (maqetta.org)
Visual Paradigm: graphical tool for UML modeling (free for non-commercial use) (http://www.visual-paradigm.com/)	**SketchFlow:** a UI prototyping tool to create interactive prototypes (http://www.microsoft.com/ expression/products/ SketchFlow_Overview.aspx)

Besides MAQETTA, a series of alternative graphical editors have been issued to support activities for sketching and prototyping UIs with various levels of fidelity [6], some representative examples are briefly described in the second column of Table 1. Further relevant references and literature analysis can be retrieved from [4].

Although several environments have been developed to support UI specifications, most of them do not support the heterogeneity of contexts of use, e.g. regarding the fragmented device market that results from the quick and continuous technological evolution. To tackle this issue, tools like screenqueri.es, emulate the rendering in a set of pre-defined devices based on screen resolution mainly. Although an analysis of the (lack of) adaptation of UI can be achieved, users must manually submit the URLs of web pages, select the target delivery device(s), and then analyze potential adaptations required by these devices. Such tools are relevant, but do not simplify the ever increasing development complexity. These tools are rarely integrated within an Integrated Development Environment (IDE), which further delays the development.

It could be concluded from Table 1 that modeling tools are appropriate to capture one or many conceptual models of an interactive system in general, or a web application in general, but they do not cope with UI models specifically. When they do, they are restricted to one context of use, without taking into account the multiple contexts. UI sketching and prototyping tools are more fine grained to capture UI variations depending on the context of use, but it is the responsibility of designers and developers to properly address the peculiarities of these contexts through designs that are adapted, while considering the various viewpoints expressed by the stakeholders involved in a development lifecycle.

Therefore, a need arises for a collaborative web-based editor for creating UIs to web applications adopting a MBUI approach covering task, domain, abstract UI, concrete UI, and adaptation explicitly, which is Quill made for.

3. DEVELOPMENT OF QUILL

As previously discussed, an authoring tool needs to manage models for each level of abstraction as the user works on the applica-

tion. Users must be able to select UI controls (e.g., edit fields, radio buttons, check boxes, list boxes, push buttons), drag them onto a canvas, adjust their properties, and link them for obtaining their behavior. In a persistent approach, when the user adds a group of radio buttons at the concrete level, an interaction unit of type "selection" must also be added in the abstract UI level, and automatically connected up to the application domain model [18], thus synchronizing the different levels. Further adjustments can be manually executed later, if necessary.

Authoring tools must be flexible, i.e. allowing users to choose their preferred work approach: top-down (e.g., starting from a selection that gives rise to a list box), bottom-up (e.g., vice-versa) or middle-out (e.g., an interaction unit of type "selection" is mapped onto a corresponding task in the task model and to a list box at the concrete level). Since transformations between these levels are not always transitive, the synchronization across levels remains a challenge. This feature can be ensured through a combination of techniques including direct manipulation of graphs, property lists, and browsing mechanisms. An agenda is also useful to guide developers on outstanding design tasks [18].

The next section describes the main shortcomings identified in the literature review that motivated the development of Quill. The presentation of the shortcomings is followed by their respective requirements, i.e. what Quill is expected to address. These requirements are consequently followed by the description of their respective design decisions, i.e. how Quill addresses these development challenges.

3.1 Shortcomings

The main MBUI shortcoming is still the lack of design environments and interactive systems that support all development stages (in a powerful, robust, and complete manner). By the analysis of the related works and tools, as presented and described in the previous section, a set of specific shortcomings could be identified:

S1. Inflexible Approaches. Stakeholders play different roles in the UI development life cycle (e.g., creating, editing, updating, validating a model), and have also different preferences, expertise levels and domains for ensuring these roles [4]. Although these roles may imply several levels at once, stakeholders are forced to work at one level at a time. The same applies for UI design, when starting from only one level of fidelity [6] is usually considered.

S2. Device's Incompatibilities. The outcomes generated with several tools are often not interoperable with different operating systems and/or devices [14]. Java-based applications, for instance, often require a set of libraries, pre-installation of plugins, or extensions and run just in specific environments (e.g., Eclipse [2]).

S3. Partial Consideration of Contexts of Use. Most tools address one specific contextual dimension at a time [21] (i.e., either the user or the platform or the environment, but not all at once), usually the platform constraints (e.g., screen resolutions, interaction techniques). Characteristics from users' profile, interaction modalities, or specific environmental constraints are often ignored. Because context-aware adaptation rules are not supported, a third-party simulation must be used for analyzing UI rendering.

S4. Little Design Guidance. There is a wide body of design usability knowledge (e.g., usability guidelines, style guides, UI patterns, design rules) that must be applied for each context [7], particularly a platform, but it is only partially incorporated in tools.

S5. Limited scalability. Since the CRF (Fig. 1) involves several levels of abstraction [3], either individually or concurrently, models located at these levels should be synchronized, thus affecting performance depending on model complexity [21].

S6. Inconsistencies. Current tools were not designed to run and perform equally in distinct platforms. This may lead to inconsistent [20] rendering or behaviors of the editors when they are used from different platforms [14,15].

S7. Limited Persistency. The transformations between the various CRF levels or fidelity levels [6] are rarely available in current editors [13]. Because the transformation across levels is not considered, they are either badly synchronized or not synchronized at all. As such, users must manually define and apply them, what can lead to inconsistent approaches and results, and also requires extra efforts of the end user to achieve persistency.

S8. Limited Scope. Current editors support either only modeling or prototyping activities [6,12], thus forcing users to use different software for modeling and designing activities [21]. Since such activities are tightly connected, MBUI design would benefit from a single integrated environment [18].

S9. Centralized and Local Development. Although UIs and their models are created and edited by stakeholders that are distributed in time and space [22], centralized and local development imposed by editors prevent them from collaborating, synchronously or asynchronously, remotely or together, in the same project.

S10. Accessibility Concerns. The usage of multiple modalities is absent or often just partially covered, causing accessibility issues [2] and incompatibility among the resulting models or UIs [20].

The 10 aforementioned shortcomings, identified during a systematic literature review, summarize critical factors for the success of the MBUI design. Editors currently available only partially support several features that are quite relevant for implementing MBUI. The CRF's 4 abstraction levels are usually not integrated and considered in the design process, thus making it difficult for stakeholders to appropriately implement UIs for different contexts of use. Therefore, specific context of use constraints are not considered and often a *one-size-fits-all* approach is adopted.

3.2 Requirements Elicitation for Quill

Based on the 10 shortcomings discussed in the previous section, corresponding requirements were elicited. They guided the definition of the design decisions and the consequent implementation of features that are available in Quill. These generic requirements cover both functional and non-functional aspects. Requirements were then analyzed, prioritized and associated with appropriate validation criteria to enable Quill's assessment. While the priorities were set as "must have", "should have" or "could have" (according to the MoSCoW method), the validation criteria were labeled as implemented, partially implemented or not (yet) implemented (Table 2).

R1. Flexibility. The development of MBUIs requires a flexible approach, in which stakeholders, depending on their profiles, preferences, interests or needs, must be able to follow top-to-bottom, bottom-to-top or middle-out approach, starting from the level of interest, and then being able to decide which level to tackle next. *Top-down* approach starts from a more abstract model and then specializes it (e.g. from a common AUI, one or several CUIs can be derived, according to contexts of use). *Bottom-up* approach bases in a concrete definition, generate a more abstracted one (e.g., from a CUI model specific for a context, by reverse engineering or retargeting [11], a common AUI can be achieved. In a *middle-out* approach, both directions can be followed.

Table 2. Association between functional and non-functional requirements and respective Design Decisions taken.

Req's	Design Decisions
R1: Flexibility	DD1 (must have): To offer choices for different users with different preferences or needs, Quill provides top-to-bottom, bottom-to-top or middle-out approaches as starting point for implementation
R2: Portability	DD2 (must have): HTML5, web-based application, no pre-installation of plugins or add-ons required
R3: Context-awareness	DD3 (must have): Browser-based, option to select specific platforms (or other contexts of use) for defining the CUI models (e.g. phone, tablet, vocal)
R4: Usability	DD4 (should have): GUI, automatic layout changes, animation to represent forced directed layout of graphs, design patterns and adaptation rules combined, manual transformations suggested and triggered by the system (organized in an agenda), HTML5 elements, such as canvas, facilitate the user interaction enabling also features like drag-and-drop items to the model
R5: Scalability	DD5.1 (must have): Voronoï diagrams, properties for the concepts handled (e.g. tasks) are presented in pop-ups dialog, overview+detail paradigm DD5.2 (must have): Regarding the rule engines, to provide inference and reasoning, first JavaScript tests will be run, and then file Node.js accordingly updated
R6: Consistency	DD6 (must have): The tool must perform consistently across different devices, besides being portable
R7: Persistence	DD7 (should have): To assure persistence across models of different abstraction levels, all transformations between them must be implemented (since not always the definitions are straightforward, a semi-automatic approach is adopted)
R8: Functionality	DD8 (must have): the application covers both model editing and UI design in a joint approach
R9: Collaboration	DD9 (must have): browser-based approach, models are stored in the cloud, users have associated roles and corresponding permissions, the editing is distributed and conflicts are solved with control system and conflict resolution mechanisms
R10: Efficiency	DD10 (could have): Varied complexity levels must be supported for users with different expertise levels, and also multi-formats and modalities must be considered to assure also interoperability

R2. Portability. Traditional UI development only enables reusability at the code level, while MBUI itself facilitates this consideration, because one single task model for instance, can derive several FUI models according to various contexts of use. Portability is then ensured by partially or fully reusing the models involved.

R3. Context-Awareness. Since the interaction nowadays takes place from different contexts of use, thus imposing constraints from the user (e.g. the user profile), the platform (e.g., interaction modality), and environment (e.g., location, light), MBUI should support context-awareness in the UI development life cycle.

R4. Usability. To ensure UI usability, relevant knowledge should be explicitly incorporated in the IDE to exploit it whenever needed, and not afterwards. For instance, a catalog of design patterns by means of associated adaptation rules orient the UI development towards good usability levels. Given that not all stakeholders are experts in quality domains, an editor should provide them with guidance on *how to* and *when* apply this usability knowledge, e.g., by selecting, retrieving, applying a pattern [21].

R5. Scalability. When a large set of model elements must be handled, it may be not scalable to visualize them, and also to manipulate them. Aiming to support tasks in this complex scenario, the editor must handle large-scale models without significantly degrading the performance and responsiveness of the application.

R6. Consistency. The application must also assure that its rendering (appearance) and behavior across devices, is consistent [20].

R7. Persistence. Any model creation or update must be reflected in the other levels by a set of transformations in order to consistently synchronize models of different abstraction levels.

R8. Scope. Both UI design and MBUI design must be covered in a joint approach.

R9. Collaboration. Stakeholders must be able to collaboratively interact while working for a common project.

R10. Efficiency. Models concerning interactive systems of varied complexity levels, i.e. ranging from simple to largely complex applications, must be supported by the editor, without significant performance decay, visualization or interaction problems. Moreover, the formats adopted should consider interoperability and exchange across different standards.

3.3 Quill Design Decisions

To address the requirements elicited in the previous sub-section, a set of solutions and possible implementations were analyzed and lead to the definitions of the corresponding design decisions, which were organized in a 2-layered architectural approach for both external and internal aspects, i.e. in a client-server approach.

DD1. Flexibility of interaction for creating the models. Given that the relationships between models follow a logical association, e.g. for an AUI Select the corresponding element can be the CUI Radio Group, the corresponding transformations are bi-directional holding the transformation from any levels and for both directions. Once the changes can be synchronized across different abstraction levels, i.e. the models are transformed according to editing made ensuring persistence, it is feasible that users can start their editing from any level, and then follow the work as needed.

DD2. Capability to render in different platforms. By using HTML5, AJAX and WebSockets, Quill is able to be executed in multi-platform environments and presents a consistent behavior, performing equally across devices. Because HTML5 is a standard technology, and platform-independent, in principle, any operating system must be supported in an attempt to assure portability.

DD3. Context-Awareness Support. Design Preferences Rules are considered and applied to fill gaps that are caused due to missing information. This process is analogous to W3C's Cascading Style Sheets (CSS), in which the conditions, when fulfilled, lead to changes in properties of elements. The rules indicate the actions for each given context of use, i.e. the contexts of use, once identified, are taken into account to adjust the UI model.

DD4. Usability Guidance. Rules as critics allow known problems to be detected, e.g. the use of specific color contrasts for users with color blindness. Such rules are relevant for assessing compliance with corporate guidelines. To provide such guidance in Quill a tab of Design Patterns provides relevant information from this domain of knowledge. Design patterns guide developers in implementing them. As such, the design assistant could note that the current design could present problems for people who are color blind, however, it would then be up to the designer to introduce additional color palettes and associate their use with different contexts of use.

DD5.1. Scalable Model Visualization. To provide a scalable visualization that performs consistently regardless of complexity levels, adequate rendering and manipulation mechanisms must be

correctly implemented. By adopting Voronoï diagrams [16], large and complex UI models can still be visualized in a more scalable and accessible fashion, i.e. a large project can be completely accessible by navigating among the links and nodes of the diagrams, in an overview+detail paradigm that enables the view of the complete model (zoomed out) or a partial visualization of the model in more details (zoomed in). Graphical representations of models make it easier to view and interact with models compared with purely textual representations. A force-directed layout algorithm works for graphs, but fails to separate sub-branches of trees.

DD5.2. Scalable Rule Engine. From the server side perspective, complex models require powerful inference mechanisms. In Quill, a Forward Chaining Rule (FCR) engine permits dealing with the rules in a structure that connects events, conditions and actions. In principle, a change in model could be able to fire events, however this approach turned out to not be feasible due to scalability issues (the amount of rules can become exponential), and as such the abduction approach was proposed. For Complex Inferences and Reasoning: the current focus of Quill concerns the rule engines, while the prototyping occurs in a client application running JavaScript, a Pratt parser is used for high-level text syntax (facilitating to transform high level rule syntax into JavaScript objects) and a RETE algorithm [9] combined with the Abduction engine can be used for performing inferences for the models. The rule engines are then executed on the server via node.js.

DD6. The application must be compatible with and consistent in different systems. To perform equally in any platform, and to reach a large number of users, avoiding familiarity issues, the development in a web-based approach was chosen. This approach enables it to be used in any browser and online, which assures not only compatibility, but also a more "lightweight", simpler and faster interaction (since no previous installation, or specific plugins are initially required).

DD7. Persistence across different abstraction levels. The Abduction (ability to infer an explanation, given an observation and its corresponding theory) is applied in MBUI for relating AUI and CUI. An extension of relational table joins is considered, in which models can be represented as relational tables, which hold undefined values, and infer missing models. Conditions permit to perform inference and reasoning considering and acting upon several contexts of use and adaptation techniques to generate CUIs.

DD8. Large scope (concerning features available). In one single graphical editor, users have both features available: the model editing and the UI design integrated. Users are able to drag and drop components available in the menu, link them, specify their properties, having a complete and unified view of the UI design models.

DD9. Collaborative Interaction. The collaborative features, in regards to the revision control mechanism, permit several stakeholders to be concurrently involved participating in a common work project with live updates. Designers, developers, programmers, software architects and engineers can design, access and edit UI models for interactive systems within a single project, in a distributed fashion. Each user has a specific role associated (e.g., junior or senior), which provides also access to specific features of the application. One of the clients is appointed as a senior editor with the responsibility for committing changes to the models. The changes provided by other (junior) clients are passed to the senior editor for review. While a senior editor can review and accept changes, a regular editor specifies and proposes changes of the models for the senior editor. To solve conflicts resulting from concurrent editing, a nearly real time revision control system is adopted, with a 4-way conflict resolution mechanism. This happens automatically, based upon algorithms for serializing changes, and for rolling back and rolling forward sequences of changes. Each client keeps its own local undo history which is automatically updated to reflect changes committed by the senior client.

DD10. Efficiency. The graphic modality summed with a WYSIWYG approach intends at a didactic and intuitive interaction manner. Quill aims at covering different expertise levels of users to facilitate the interaction and although the visualization and editing is graphically performed, it can also be exported in an interchange format for other purposes of use, aiming a good accessibility. Moreover, by applying adaptation rules targeted at constrained contexts, users can identify good design decisions and adaptation techniques that are applicable in a given context.

3.4 Design Knowledge

The **Rich Domain Model** defines the data interfaces between the user interface and the application back-end. Besides basic data types, Quill supports default values, examples, constraints and embedded documentation. Constraints as regular expressions, constrain the value of string properties; indicate that a given interface, method or property is relevant based upon the values of other properties, is optional or must be provided by the user, and is persistent, i.e. that the values provided by a user are preserved in between invocations of the user interface.

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. Some tasks are performed by the user whilst others by the system. The task model can be used to determine what parts of the user interface to present in parallel or sequentially. On a small display, it may be appropriate to break the user interface into a sequence of simple dialogues, while on a larger display, these could be presented jointly or split across separate panes in a tab control.

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 layout managers. This is enriched and mapped into CSS when skinning the final user interface generated from the concrete user interface models.

The **design rules** express knowledge for the design assistant. For instance: (i) Rules that propose designs, and which embody design preferences for particular platforms; (ii) Rules that determine which relationships hold in a given context of use; (iii) Rules that propose changes in response to events signaling changes in the context; (iv) Rules that critique designs, e.g., searching color contrasts that would create problems for color blind people.

Propagating changes across a design with event condition action rules requires every change to be matched with a rule, resulting in various rules that are hard to maintain. Another solution is to express logical relationships across different abstraction levels. If certain facts and certain relationships hold true, then it is possible to infer additional facts that must be true if the relationship is to hold. This is 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 is available as an interactive web page [9], it uses a 2-pass algorithm for the logical joins and abducing facts and allows enabling and disabling abduction to see the results.

4. A RUNNING EXAMPLE ON QUILL

Quill's architecture (Fig. 2) is organized in a client-server approach. The client application is implemented in HTML5 and provides the features as previously mentioned. The back-end of Quill is a cloud provisioned authoring server that implements an asynchronous messaging system, generating responses according to the changes made by the user and streamed to the server. The server, besides also streaming changes back to the UI (in HTML5), persists the UI models across the four CRF levels. This server is deployed as a Tomcat web application. In order to propagate changes across levels, a rule engine is adopted, simulating the effects of the changes according to specific contexts of use.

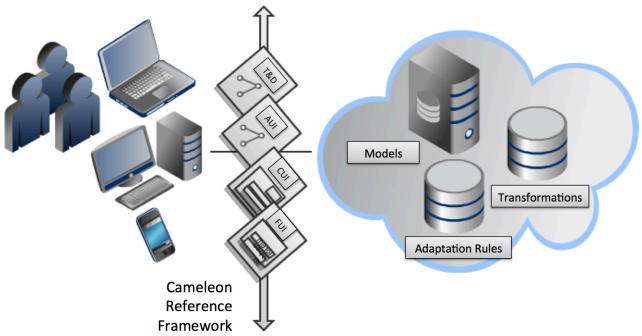

Figure 2. Architectural organization of Quill.

We exemplify Quill's functionalities and features, illustrating them with an actual application scenario as a proof-of-concept. This case study defined consists in a car rental application example in which users are able to rent a car (this is the official case study considered in the W3C MBUI group: http://www.w3.org/wiki/Model-Based_User_Interfaces). Thus, by interacting with the application, they can select the car of interest to rent, set the period for the rental (begin and end dates, in hour, day, month, and year), specify details about the car (preferences, requirements, constraints, etc.). For this application, a common domain model (Fig. 3) and a task model (Fig. 4) were defined, serving as a ground for generating the models for the other levels: abstract UI (Fig. 5), concrete UI (Fig. 6), and final UIs (Fig. 7).

For this case study, specific scenarios for contexts of use, mainly concerning the platform type, were selected, namely: a Desktop PC, a Tablet PC and a Smartphone. Their specifications guided the definitions of the models and also their transformations by means of appropriate adaptation rules. For further examples of various contexts of use, a *model voyager* enables the user to navigate within various levels of abstraction for the same case study and see the transformations (http://sites.uclouvain.be/mbui/). Fig. 8 reproduces the tree browsing of the car rental study that reaches to a context of use in which smartphones are used.

Fig. 4 shows the *task model* of the application, visualized as a Voronoï diagram, in which each (sub-)task has its properties specified or modified by the end user, the tasks can be also accessed and visualized in details. The task format is compatible with the CTT (Concur Task Trees) following its formal specification [17]. Fig. 5 illustrates an example of the *Abstract UI model* (AUI), while Fig. 6 reproduces an example of the *Concrete UI (CUI) model* for the car rental example. It shows possible fields that compose an entry form for the end users. In this phase, no layout specification is set. Selected adaptation rules appropriate to the target context will be responsible for defining specificities of the layout according to the context of use. In the example the UI model is being specified for a tablet device (delivery target).

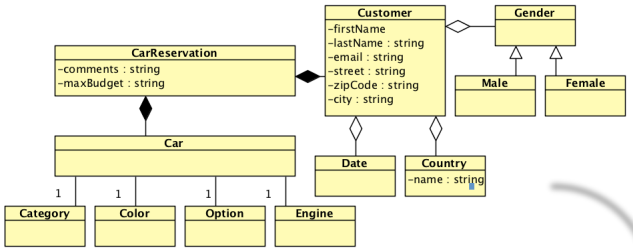

Figure 3. Domain Model for the Car Rental case study.

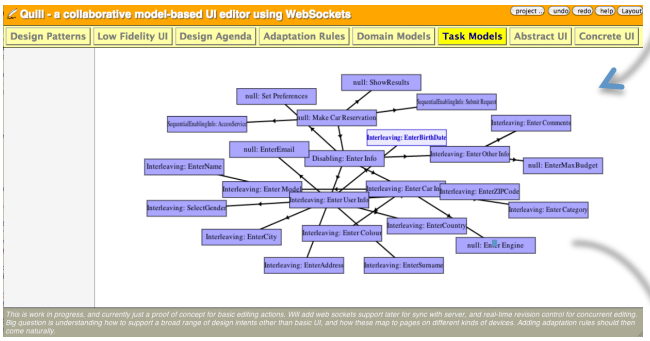

Figure 4. Task Model for the Car Rental case study.

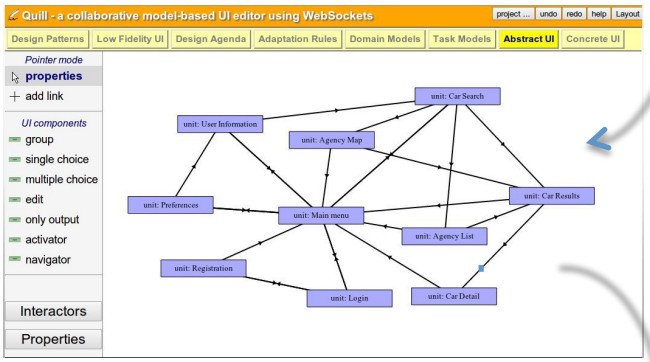

Figure 5. Abstract UI Model for the Car Rental case study.

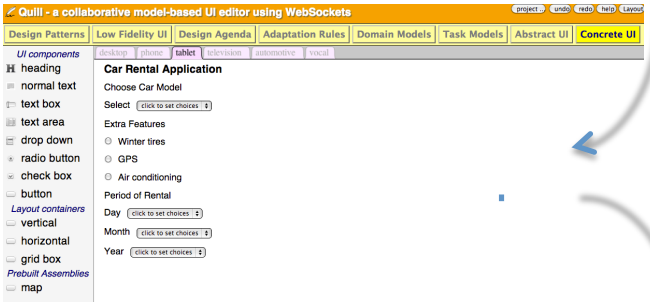

Figure 6. Concrete UI Model for the Car Rental case study.

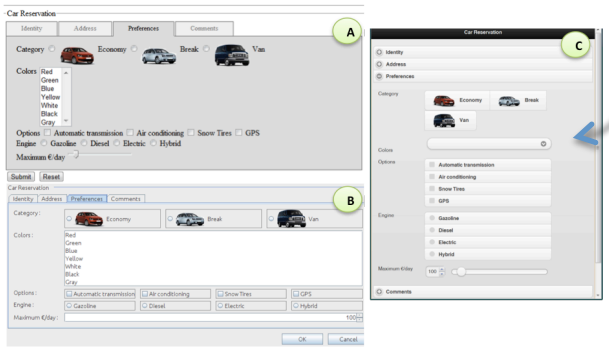

Figure 7. Final UIs for the Car Rental case study.

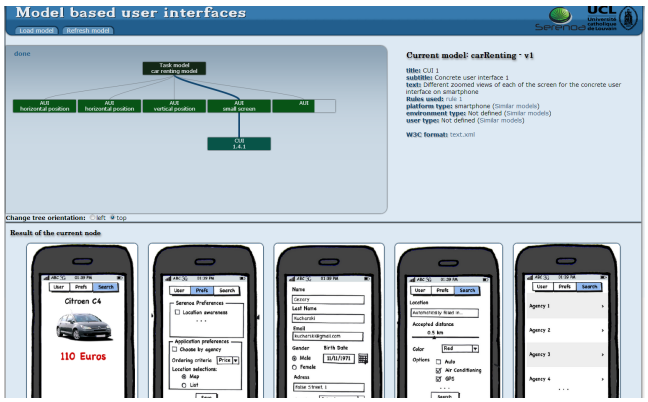

Figure 8. Model Voyager for the Car Rental case study.

Figure 9. Quill Main Interface.

Quill supports features that are both inherent to the project management options (i.e. defining project, undo/re-do actions, help, layout specification – right side of Fig. 9) and related to the editing functions, that are organized in horizontal menus with selectable components (buttons and tabs), and a vertical menu with draggable components (buttons and icons). The tabs items support switching to different modes in the following tabs.

The *Design Patterns* tab (Fig. 9) provides a catalog of design patterns that guide users in the development of their applications. In low fidelity UI users can depict the UI enabling end user evaluation in the early stages of the development process. The *Design Agenda* tab enables users and system to collaboratively define and control tasks and milestones that "have already been" or "must still be" achieved. The tasks of this agenda are either triggered by the application or defined by the user. The *Adaptation Rules* tab gathers the transformations and actions triggered given specific conditions fulfilled by the context of use. Rules consider context belonging to the user, platform and environments, and application resources including navigation, presentation and contents (regardless of their given format). The *Domain Model* tab permits users to describe application domain properties, actions and notifications for the domain (Fig. 3). The *Task Model* tab enables users to access and edit task models that were previously created and that are associated with a given project, or to create new models for new projects. For the task models, properties like temporal relationship (e.g., order, sequences, and associations), name, task type. The *Abstract User Interface model tab* considers several containers and components that can be included, edited and specified. For the *Concrete User Interface model* tab, either FUIs suffer reverse engineering processes to be abstracted or AUI models are transformed (reified) into CUIs. User can also create their own CUI models, by selecting and organizing their components.

Quill enables prototyping of UIs in various levels of fidelity and abstraction by dragging components from the left menu and by dropping and arranging them in the central canvas. Their properties can be then refined. The components of the left menu vary according to the tab selected by the user. When the AUI choice is selected, two pointer modes are available: properties or add link. Seven UI components can be selected (group, single or multiple choice, edit, only output, activator and navigator). When the CUI choice is selected, users can choose among: 8 UI components (heading, normal text, text box, text area, drop down, radio button, check box, and button), 3 Layout Containers (vertical, horizontal, and grid box), or 1 Prebuilt Assembly (map).

The CUI level also enables users to choose the platform of interest among the delivery targets available, i.e. a Desktop PC, Mobile Phone, Tablet PC, Television, Automotive, or Vocal. The central canvas is associated with the respective constraints imposed by such devices, providing users only features that are available in these specific cases. CUIs for the other targets are (semi) automatically synchronized and updated according to any model change.

Fig. 7 illustrates three Final UIs implemented, the selected tab shows the preferences of cars that are available for the end user, for instance regarding categories of cars, colors, options, engine and maximum cost can be specified. This UI is illustrative, since one specific device type (i.e. tablet PC) was considered, however other platforms, like a Desktop PC, or mobile ones, like a smartphone can also be selected as target delivery device. Fig. 10 demonstrates that the HTML5 code generated exhibits adaptation capabilities to the platform: the UI layout changes according to the screen resolution by, for instance, repositioning labels justified to the left or positioned on top of related edit fields.

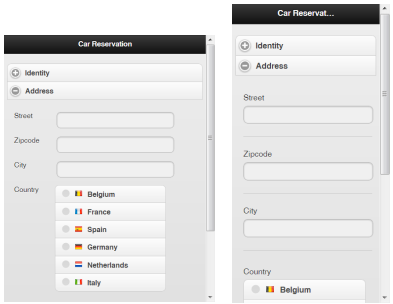

Figure 10. Final UI with adaptation capability.

Concerning the technical perspective, the technology adopted to implement Quill is HTML5, in a portable and lightweight client application. The main page is coded in 126 lines of code, and accommodates 8 specific functions implemented in JavaScript. The style sheets were specified using CSS (Cascade Style Sheet).

The first JavaScript function called WebSocket is responsible for connecting the application with the server. Then vector is a function that computes the graphical layout of the models. Force directed is also responsible for calculating the layout of the models, i.e. re-arranging nodes and links in a force directed graph layout approach, in this sense damping, repulsion and field are set, and an animation is applied to illustrate the movement transition. Quill launches the main UI, including the menus and canvas and it also connects to the server to access and load current projects on user demand. Abstract loads features that correspond to the respective UI level, composing the appropriate menu and rendering the model in the canvas if previously selected by the user. Concrete composes the horizontal sub-menu, with device options, load the workspace and respective features.

ASFE-DL [17] consists of a specific adaptation language that generates the AUI model. Appropriate interactors are charged and loaded and their respective properties are presented for editing. The Task function loads the workspace with the current task model, if available and previously selected and calls the functions that calculate the behavior, re-arrangement and animation of the task tree model diagram. Quill adopts an Apache2 open source license.

5. USER SURVEY

5.1 Method

While for the scientific community there are clear benefits of using model-based approaches and context-aware adaptation, for the industry, it may be no so evident whether the benefits actually compensate for the costs involved. To investigate this, a survey based on two main hypotheses has been defined:

H1) Stakeholders are aware of the importance and the benefits of: context-sensitivity, model-based approaches and adaptation.

H2) Stakeholders do not fully incorporate into their daily work practices: context-sensitivity, model-based approaches and adaptation.

The target respondents of this survey are practitioners working for Information Technology companies, with different expertise, background and roles (e.g. software engineers and architects, developers and designers). They live in different countries, (e.g. Belgium, Brazil, France, Germany, U.K., Spain) and work for different companies (e.g. Yahoo, Sony, BNP Paribas – Fortis). The questions focus on: (i) practitioner profile (years of experience, main role, company size); (ii) context (dimensions and information considered, perceived relevance, methods employed, and usage); (iii) adaptation (how techniques are identified, applied and presented); and (iv) MBUI approach and its perceived relevance (pros and cons). The survey has been defined and published online using Google docs, a message sent via email, invited participants to voluntarily collaborate.

5.2 Results

30 practitioners working for I.T. companies or as independent consultants replied to the survey. Concerning their *profile*, and their years of experience, 43% of the participants work from 5 to 10 years in the I.T. domain, 40% have been working for more than 10 years and only 17% for less than 5 years (Fig. 11 left). Concerning the company size, 46% of the participants work for large companies, 20% for small companies, 17% for medium-sized companies, 10% for micro-entities and 7% work independently (Fig. 11 right). Concerning their main roles, 40% of the participants are developers, 27% software engineers, 20% project managers, 7% software architects, 3% system analysts, and 3% support team leader (Fig. 12).

Concerning the *context*, in absolute numbers, out of the 30 participants, 25 stated to consider the users, 24 the platform, 12 the application domain and 9 the environment. Concerning the perceived relevance of context and its actual usage (Fig. 13) the user is classified as the most relevant dimension for most of the participants, followed by the platform and the application domain, while the environment is considered as the least relevant dimension. These results concern the participants' perception of the context relevance. When compared with the actual usage, again the user and platform are considered as the most relevant dimensions, while in practice also the application domain and environment are the least considered dimensions. However, although users are perceived as the most relevant dimension of context, in practice their information is not always used. The platform is more considered in practice than perceived as relevant. The environment is perceived as relevant and considered in practice, and the application domain is more considered as relevant as actually used in practice. Fig. 14 illustrates how many participants consider context by relevance level (left) versus actual usage (right). These graphics show that users are perceived as the most relevant dimension (by almost half of the participants), followed by the platform, application

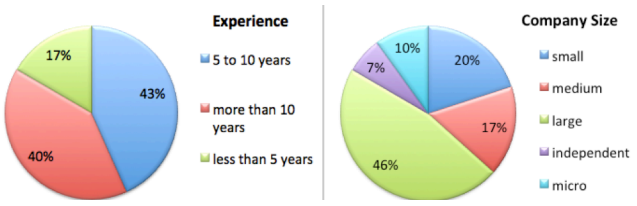

Figure 11. Participants profile: experience and company size.

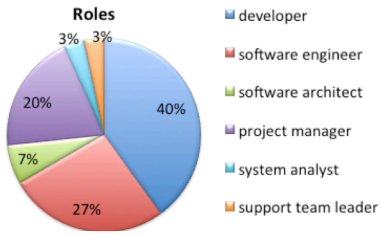

Figure 12. Participants profile: main roles.

domain and environment. The same trend is observed for the practical use of the dimensions (although with less significant differences). The environment was the dimension considered the least relevant by more participants and the least frequently used in practice. In practice, all participants consider to some extent: platform, application domain, and user. Fig. 13 shows the context dimensions that the participants use while developing systems, out of 30 participants, 25 consider the user, 24 the platform, 14 the application domain and 9 the environment. Only 4 out of 30 participants actually use all 4 dimensions together (user, platform, environment and application domain), also 8 use 3 dimensions. Most of them (18 out of 30) use just 2 (14) or 1 dimension (4).

Concerning the *user* and its information considered: most of the participants consider user preferences (20 out of 30), followed by demographics (14 out of 30) and interests (15 out of 30). 10 out of 30 participants though consider only impairments. Usually a combination of 2 dimensions is used (13 out of 30 participants), e.g. impairments and preferences (4), or interests and demographics (3 out of 30). Only 3 participants consider simultaneously all 4 dimensions. Regarding the methods for gathering user information: while 17 out of 30 participants rely on observation, 14 on guidelines, 12 on interviews, 5 on surveys and 6 just try to guess information. Two participants collect and monitor real usage data. Ten

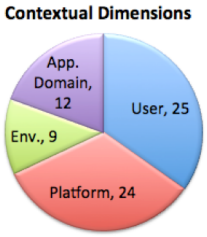

Figure 13. Context information: absolute numbers.

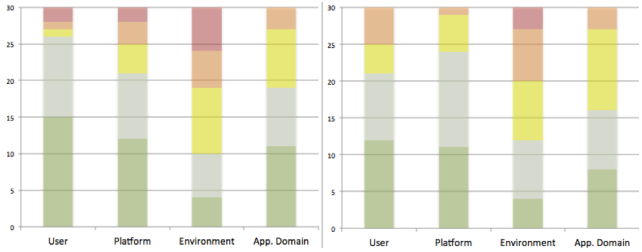

Figure 14. Context: perceived relevance and adoption.

participants use just one method, while 11 adopt 2, 6 use 3 methods and only 1 combine all 4 methods (guidelines, interviews, observations and surveys).

Concerning the *platform*, most of the participants (25 out of 30) consider the device and 23 out of 30 consider the technology, 23 consider the connections, and just 4 take the accessories into account. Just 5 out of 30 participants do not consider the device per se, but they (2 participants) consider the connections or (3 participants) the connections and the technologies available. To gather information about the platform, 16 out of 30 participants use a default specification; among which 6 also perform automatic tests, of which 2 also observe the context of use and 1 also tracks the user interaction. Two participants perform automatic tests, 4 observe users and just 1 participant interviews users. Just 1 participant tracks the user interaction (but combined with 3 other methods). Regarding the amount of methods, while the majority (16) employs just 1 technique at a time, the remaining participants (14) combine more than one technique. Three participants combine 3 methods and 11 combine 2 of them.

Concerning the *environment*, most of the participants (17 out of 30) do not consider any information. Among the remaining participants (13), 8 consider the light level, 5 the stability level, 4 the noise level, and 4 considered other information, as the user location (via GPS), temperature, and the 3G coverage. Concerning the methods adopted, observation sessions, user interviews, and surveys are applied. Just 1 participant informed to use sensors.

To search for *adaptation*, the participants use: pattern libraries (11 of 30), public guidelines (9 out 30), embedded features (8 out of 30), online repositories (7 out of 30). However, approximately half of the participants (16 out 30 participants) do not provide adaptation. Only 1 participant combines 4 information sources, while 5 combine 3, 9 combine 2, and 15 use only 1 or no source. For *adaptation strategies*, 6 out of 30 participants use UI graceful degradation [8], 10 use progressive enhancement, and 4 combine both strategies. Most of the stakeholders though (17 out of 30) do not use any of these, and just 1 participant uses animation to smoothly present to users the transition between original and adapted UIs.

Concerning the adoption of *models*, almost half of the participants (16 out of 30) informed to not use them, 6 participants use MDE, 11 use UML diagrams among which 3 use them combined with MDE. The participants of the survey remarked four main *benefits* of adopting models during the development process: (i) provide a common language and standards; (ii) facilitate reuse; (iii) generate systems that are more complete and have more qualities and (iv) aid communication, discussion and analysis. As *disadvantages* of adopting models, four remarks were noted, models: (i) are hard to customize, to adapt, and to maintain; (ii) lack support (or have incomplete support); (iii) are hard or slow to synchronize changes and (iv) require more expertise, efforts and time.

One aspect has been classified as both positive and negative for different participants: the optimization of the development phases. While some participants believe that fewer efforts are needed, others stated that more efforts are required, e.g. for expertise and time. Another aspect of disagreement is achieving a working prototype, while some participants consider it easier to do with models, others think it is actually harder. The same applies for the complexity of the projects, while one participant stated that models are not suitable for simple projects, other participants stated that models are not suitable for highly complex projects.

5.3 Discussion

The survey reached a variety of stakeholders with different roles, experience, and from different companies and countries.

Regarding the *context*, it is clear that mainly the user and platform are considered, while application domain and environment are not always so used. Actually it is possible that stakeholders were confused with such definitions, as some participants commented after replying the survey. Sometimes the concept of environment was misunderstood, being interpreted as the editor per se, and not the situation where the interaction takes place and its circumstances. The term application domain also raised some discussion, being misunderstood with cultural aspects of the user. Even by providing a short description about these concepts and some examples, not all participants could successfully comprehend such definitions. It may be that the user and the platform are more considered because when ignored or omitted the user interaction may be prevented. However, to complement such results, it is necessary to investigate to which extent the contextual information is actually covered. Concerning the H1, which states that stakeholders are aware of the importance of the concepts, it holds for context aspects, at least regarding user, platform and application domain. Environmental aspects are not considered as so important, or maybe it may be not clear for stakeholders what environment states for and how it can be effectively useful. Concerning H2, most of the participants stated to use context, at least to some extent, for their projects.

Adaptation is not used by most of the participants, since 16 out of 30 stated to not provide adaptation and to consider instead a standard scenario. This may be a result of previous work practices in software development, in which a conventional context of use was common (i.e. an able-bodied user, a Desktop PC, and a stable environment). Besides this, it is possible that stakeholders are not aware of which information to consider and how to do it. The participants are aware of the importance of adaptation, since they stated to consider context-awareness while developing their applications, which validates to some extent H1. However, concerning H2, it is remarkable that adaptation is not largely employed, which may result in static applications that are not suitable for dynamic and varied contexts of use.

The perspective of the participants about *models* shows that while they can perceive many benefits, they are still skeptical about their adoption; mainly because of the lack of support to adopt models or incomplete solutions. Without more complete frameworks, the use of models may be limited to academia or to specific activities. Concerning H1, it is clear that most participants are able to recognize the importance of models, however, concerning H2, we note that models are not widely used. Being useful to support certain activities, but not fully adopted. By analyzing the commentaries provided we believe that only by having more mature support, frameworks, standards and tools, stakeholders could see more benefits in using models, less costs, and then actually incorporate them into their daily work practices. The lack of consensus regarding the advantages and disadvantages of models may be justified by the fact that these assumptions are project-dependent, so while in certain cases more resources are indeed needed, in other ones the development is automatically optimized. Regarding complexity issues, there is a range in which models are suitable, however further investigations are needed to precise, identify criteria and to measure the complexity levels of projects, and also the costs of applying models, so that to effectively identify when it is suitable to actually adopt model-based approaches.

6. FINAL REMARKS AND CONCLUSION

This paper presented Quill, a web-based development environment that enables various stakeholders of a web application to collaboratively adopt a model-based design of the user interface for cross-platform deployment. Based on shortcomings identified in the literature and by observation, a set of 10 requirements was elicited that gave rise to design decisions on which the Quill development has been motivated and decided.

There are still many open questions to be discussed in MBUI, such as, but not limited to: the compatibility among heterogeneous contexts of use, the accessibility issues due to usage of a graphical representation in comparison with a textual description, the interoperability of applications regarding a technology and decisions that are both platform and technology independent, the consideration of context-awareness and all its consequent specificities, the decision of a web-based application concerning its drawback of online usage only and (un)availability problems, the definition of synchronous vs. asynchronous collaboration and consequent need of managing conflicts and defining user roles (permissions). Now, in Quill users can collaborate within the same project, however both asynchronous and concurrent editing must be supported, the usage of HTML5 which requires browser that are (really) HTML5-compliant. Not all the browsers available are actually HTML-5 compliant, which may cause differences of rendering and features supported. There are still many challenges that must be discussed, decided, and overcome to consolidate and release the proposed editor, among which we can highlight, for instance the definition and application of transformations between models across different abstraction levels, not always is straightforward, requiring also manual interference, in this sense a design agenda was adopted to remind users about the manual changes that must be done to appropriately synchronize models. Other solutions could be also thought, the design agenda can be enhanced in future efforts to accommodate also agile decisions, or user-centered design activities. Quill does not fix completely all the issues encountered in MBUI as discussed. However, it aims at pointing and addressing the most challenging issues.

Quill's main benefits are: it is a browser-based application, whose models are hosted on the cloud, it enables users to adopt flexible approaches, concerning the level from which they start to work, it enables collaboration among stakeholders of different expertise levels and domains, it supports also specific roles with corresponding permissions, MBUI is compliant with CRF, it is more straightforward to extend its functionalities with additional plugins and add-ons, (e.g., to calculate specific metrics), web services could augment Quill's functionalities. So far, a design pattern tab provides information concerning this specific domain, usability guidelines, adaptation techniques could also be incorporated for users that are not experts in these fields.

Quill will be extended with: (i) an adaptation rules editor that enables stakeholders to express adaptation rules in a controlled subset of natural language that can be easily mapped into any specific standard rule format for further processing, (ii) a Final UI emulator in which users visualize in a separate browser window the UI design, (iii) syntax coloring features.

Acknowledgments

The authors gratefully acknowledge the support of the FP7 Serenoa project, funded by the European Union through under reference FP7-ICT-258030. The authors also thank Raphaël Schramme for implementing the model voyager, and the various anonymous participants to the user survey.

7. REFERENCES

[1] Aquino, N., Vanderdonckt, J., Panach, I., and Pastor, O. Conceptual Modelling of Interaction. In *Handbook of Conceptual Modelling: Theory, Practice, and Research Challenges*, Embley, D. Thalheim, B. (eds.), Chapter 3. Springer-Verlag, Berlin (2011), 335–358.

[2] Assisi, R. V4all GUI designer for eclipse manual v.1.0. 2004. http://v4all.sourceforge.net/index_start.html.

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Inter. with Comp.* 15, 3 (2003), 289–308.

[4] Cantera Fonseca, J.M., González Calleros, J.M., Meixner, G., Paternò, F., Pullmann, J., Raggett, D., Schwabe, D., and Vanderdonckt, J. Model-Based User Interface Incubator Group, 4 May 2010. http://www.w3.org/2005/ Incubator/model-based-ui/XGR-mbui/.

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

[6] Coyette, A., Kieffer, S., and Vanderdonckt, J. Multi-Fidelity Prototyping of User Interfaces. In *Proc. of IFIP INTERACT'2007*. LNCS, Vol. 4662, Springer-Verlag, Berlin (2007), 149–162.

[7] Coutaz, J., Crowley, J.L., Dobson, S., and Garlan, D. Context is key. *Communications of the ACM* 48, 3 (March 2005), 49–53.

[8] Florins, M., Montero, F., Vanderdonckt, J., and Michotte, B. Splitting Rules for Graceful Degradation of User Interfaces. In *Proc. of AVI'2006*. ACM Press, New York (2006), 59–66.

[9] Heinrich, M., Lehmann, F., Springer, T., and Gaedke, M. Exploiting single-user web applications for shared editing: a generic transformation approach. In *Proc. of WWW'12*. ACM Press, 1057–1066.

[10] Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., and Vanderdonckt, J. Human-Centered Engineering with the User Interface Markup Language. In *Human-Centered Software Engineering*, Springer, London (2009), 141–173.

[11] Kumar, R., Talton, J.O., Ahmad, S., and Klemmer, S.R. Bricolage: Example-Based Retargeting for Web Design. In *Proc. of CHI'2011*. ACM Press, New York (2011), 2197–2206.

[12] Maqetta, Visual Authoring of HTML5 User Interfaces in the browser. Dojo Foundation. http://dojofoundation.org/projects/ maqetta.

[13] McKirdy, J. Choosing the UI tool which best suits your needs. In *Proc. of 7th IFIP Int. Conf. on Human-Computer Interaction Interact'99*. IOS Press (1999).

[14] Meskens, J., Vermeulen, J., Luyten, K., and Coninx, K. Gummy for Multi-Platform User Interface Designs: Shape me, Multiply me, Fix me, Use me. In *Proc. of AVI'2008*. ACM Press, NY (2008), 233-240.

[15] Mourouzis, A., Leonidis, A., Foukarakis, M., Antona, M., and Maglaveras, N. A novel design approach for multi-device adaptable user interfaces: concepts, methods and examples. In *Proc. of UAHCHI'2011*. Vol. 1. Springer, Berlin (2011), 400–409.

[16] Nocaj, A. and Brandes, U. Computing Voronoi Treemaps Faster, Simpler, and Resolution-independent. In Proc. of *Eurographics Conf. on Visualization EuroVis'2012*. S. Bruckner, S. Miksch, and H. Pfister (eds.). *Computer Graphics Forum* 31, 3 (2012).

[17] Paterno, F., Santoro, C., and Spano, L.D. Deliverable D3.2.2. ASFE-DL: Semantics, Syntaxes and Stylistics (R2). http://www.serenoa-fp7.eu/wp-content/uploads/2012/10/SERENOA_D3.2.2.pdf

[18] Raggett, D. The web of things: Extending the web into the real world. In *Proc. of SOFSEM 2010: Theory and Practice of Computer Science*. (2010), 96–107.

[19] D. Raggett. Testbed for abduction over relationships. 2013. At: http://www.w3.org/2013/01/abduction/

[20] Richter, K., Nichols, J., Gajos, K., and Seffah, A. The many faces of consistency in cross-platform design. In Proc. of Extended Abstracts of CHI'2006. ACM Press, New York (2006) 1639–1642.

[21] Seffah, A. and Javahery, H. Multiple User Interfaces: Multi-Devices, Cross-Platform and Context-Awareness. John Wiley & Sons (2003).

[22] Sumner, T., Bonnardel, N., and Kallak, B.H. The Cognitive Ergonomics of Knowledge-Based Design Support Systems. In *Proc. of CHI'97*. ACM Press, New York (1997), 83–90.

[23] Vdovjak, R., Frasincar, F., Houben, G.-J., and Barna, P. Engineering semantic web information systems in Hera. *Journal of Web Engineering* 2, 1/2 (2003), 3–26.

[24] Wegner, P. Why Interaction is more Powerful than Algorithms. *Communications of the ACM* 40, 5 (May 1997), 80–91.