Towards a Framework for Adaptive Web Applications

Ana Isabel Sampaio and José Creissac Campos

Departamento de Informática/Universidade do Minho & HASLab/INESC TEC
Campus de Gualtar, Braga, Portugal
pg20190@alunos.uminho.pt, jose.campos@di.uminho.pt

Abstract. We have developed a framework to support adaptive elements in Web pages. In particular we focus on adaptive menus. Developers are able to define rules for menu adaptation according to the features of the device and browser in use. This paper briefly describes the selected adaptation patterns and their implementation.

Keywords: Web applications, User interfaces, Runtime adaptation

1 Introduction

The diversity of devices available in the market has changed the way we access and share information. At the same time, it has created a number of challenges for application developers. It has become more and more important to offer solid user experiences to an increasing number of contexts. Context is an all-encompassing concept that must be understood in relation to a specific purpose [2]. In this particular case, we are mainly interested in how to develop applications that adapt to the diversity of devices mentioned at the start. That is to say that the particular aspects of context we are dealing with are the input-output characteristics of the devices where the applications should run. Of particular relevance, of course, is screen size, but other aspects such as supported interaction modalities must also be considered.

The main motivation for this work stems from an ongoing project on prototyping ubiquitous environments [6]. A framework (APEX) has been developed that is based on building virtual world simulations of the environments. Users are able to interact with the environments both implicitly, through simulated sensors, and explicitly, through interactive devices (e.g., smartphones or public displays). Such devices can be simulated, or actual devices connected to the simulation framework. This, then, begs the question of how to deploy applications that are capable of adapting to different execution contexts. We base our solution on the use of Web applications, and in this paper we explore how to support the adaptation of such applications to different devices.

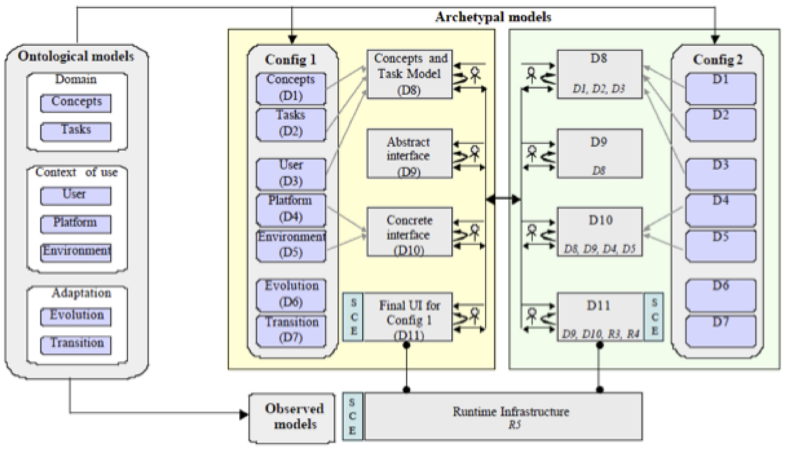

Fig. 1. Cameleon reference framework (from [1])

2 Deploying applications to multiple devices

Two basic approaches can be considered when thinking of deploying an application to a multitude of devices. One is to develop native applications for each category of device. Another is to develop Web applications, using the browser as a multi-target runtime platform. In the case of Web applications, we can further opt for performing adaptation on the server side, delivering different versions of the applications based on the platform/browser used, or performing adaptations on the client side (i.e. in the browser). While the former has been the most common approach, technological development is making the latter a viable alternative.

From a methodological perspective, the Cameleon reference framework [1] (see Figure 1) provides the conceptual foundations to reason about this adaptation. The framework proposes a number of abstraction levels from Tasks and Concepts (the Domain), down to the Final User Interface (the code of the running user interface), and posits that these should be adapted to concrete contexts of use, according to an adaptation model. The framework rest on the assumption that a runtime platform is available that supports the dynamic adaptation of the applications according to context.

At the technological side, Responsive Web Design [4] and Progressive Enhancement [3] already offer approaches to support the development of Web applications that adapt to the characteristics of the browser/device they are running on. One characteristic of these applications is the existence of a single code base that is able to provide users with increasingly rich user interfaces (evolution), based on the characteristics of the platform, but also change the structure of the user interface to adapt to different form factors (transition), in response to different platforms. Although a number of patterns and libraries exist that

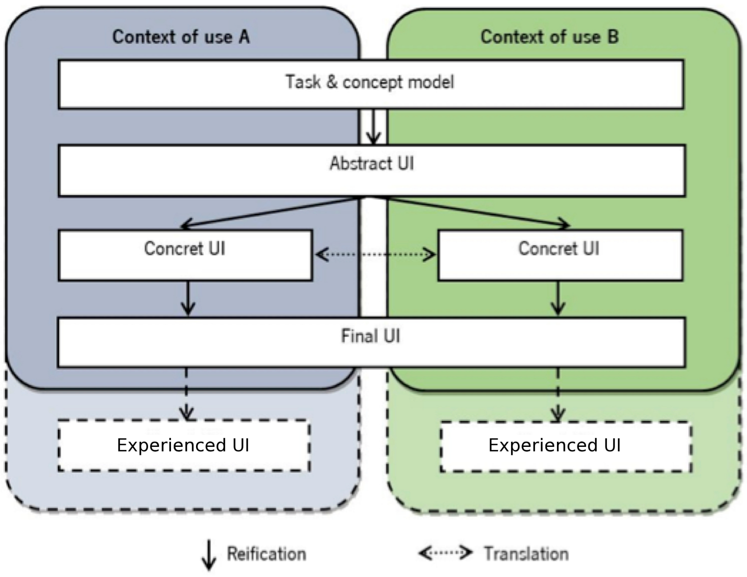

Fig. 2. Multiple concrete models, one final user interface

support the implementation of this adaptation, it is still up to developers to implement the adaptation capabilities. The browser, rather than constitute the needed platform for dynamic adaptation, acts as a mostly passive infrastructure that is queried by the application in order to determine information about context.

3 A framework for adaptive Web applications

Mapping the development of a typical responsive Web application to Cameleon, the impact of context is seen mostly at the level of the concrete interface that should be presented. Designers typically draw different mockups depending on the devices/screen size. Using a unified code base means that the different concrete interfaces are merged in a single final user interface (see Figure 2). The logic that governs adaptation, i.e. the adaptation model, is encoded in Cascading Style Sheets (CSS) and JavaScript.

An approach to better support the development of these interfaces is to push as much adaptation as possible into the final interface, and do this in an as automated as possible manner. This can be achieved by introducing adaptation capabilities into the widgets used to create the final interface. Hence, an abstract control will be mapped into a concrete widget, and this widget mapped into an adaptive control that will react to context according to some adaptation rules, to create the User Interface (UI) experienced by the User.

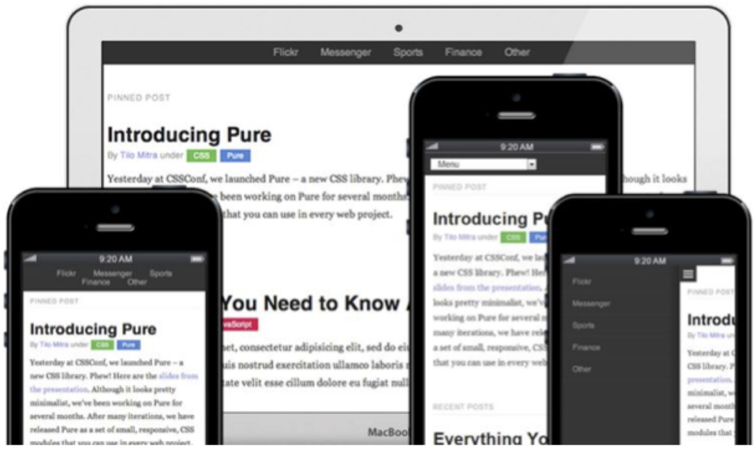

Fig. 3. Menus (Menu bar: top and left; Select menu: middle right; Off-canvas: right)

In order to test these ideas we have developed a framework to support adaptive menus. Developers are able to define rules for menu adaptation according to the context of execution (see Figure 3). This can be seen a introducing a further level in the reference framework, detailing how a final user interface adapts to the platform to generate the UI actually provided to the users (what we call Experienced UI in Figure 2).

3.1 Determining device features

Detecting the context of execution, in this case, amounts to detecting features of the device and browser the application is running in. This can be done, either on the server, or directly in the browser. Server side detection explores the user agent string sent by the browser in the HTTP requests. This supports obtaining detailed information about the browser and the device being used. However, it depends on the browser sending the correct user agent string, and (typically) on third party services to obtain the characteristics of the different devices.

Client side detection is performed directly in the browser, and a number of Javascript libraries are available to support it. Modernizr[1] is one such library. Using it, it is possible to query more than 40 features, such as support for HTML5 or CSS3 in the browser, touch or geo-referencing capabilities in the device. Although more limited than server side detection, client side detection it easier to implement (as it does not rely on external services), and does not rely on the browser sending the correct information. Considering this, it was decided to use client side detection, and Modernizr in particular, to support the implementation of the different adaptation patterns described below.

[1] http://modernizr.com (last accessed, March 7 2014).

3.2 Adaptation patterns

A total of five patterns for adaptive menus were identified and implemented. The assumption is that the menu will initially be represented by an unordered list of links. Using CSS (more specifically CSS3) it is then possible to render that menu differently according to the features of the browser. In this case, we are interested in CSS3 support, touch support, and the size of the browser window.

The simplest solution is to create a **Menu bar** at the top of the page, and reflow the options as the page size changes. This is achieved with CSS only, without need for Javascript, but works well for small menus only. For small screen or larger menus, it tends to take to much vertical space, going against the content-first, nav-second principle [7].

A pattern that better supports that principle is the **Footer anchor** menu pattern, In this case, a link (the anchor) is placed at the top of the page. This link points to the menu, which is placed in the footer of the page. Javascript is used in the implementation of this pattern to perform a number of changes in the page: move the menu to the footer; add the anchor at the top; and add a *Back to top* option in the menu to help users navigate the page.

Another option to avoid using too much space with the menu at the top of the page is to render the menu in a **Select** (drop down) component. Navigation is kept at the top of the page and each browser will render the component appropriately. This is particularly useful with complex menus. The pattern is implemented by replacing the list of links with a select element containing the links originally in the list.

A pattern that also supports hiding the menu, but without the need of adding a new component to the page, is the **Toggle menu** pattern. In this case, an icon is used to allow the user to show/hide (i.e. toggle on/off) the menu. The toggling of the menu is achieved through Javascript and CSS, by dynamically adding appropriate classes to the list and its elements. Depending on the class the elements are assigned they will be shown or hidden, as defined in the CSS.

The last implemented pattern is the **Off-canvas menu** pattern. This pattern uses a floating lateral column (typically placed on the left) where the menu is presented. As with the Toggle menu, the Off-canvas menu is accessed through an icon. However, in this case the menu appears from the left (or right) and takes all the hight of the page. How much of the horizontal space is used depends on the window's width. This pattern is commonly used in native mobile applications, so it has the advantage of providing a similar look and feel.

The Off-canvas menu pattern features the more complex implementation of the five patterns, as it requires changing properties, not only of the menu, but also of other elements in the page. This happens because the elements in the page must be moved left (right) when the menu appears. To achieve this, a wrapper is introduced that contains all elements in the page. It is this wrapper that is then slid when the menu appears/disappears. The sliding effect was achieved using CSS3 transitions.

As described, all but the first pattern resort to Javascript in order to carry out runtime changes to the Web page. This is unlike [5] which statically creates all alternative menus in the page, using CSS to select the appropriate menu from the available alternatives by hiding or showing them depending on the viewport. The advantage of runtime adaptation is that the original Web page becomes simpler, featuring the basic version of the menu only (avoiding duplicated menu codification). The drawback is that Javascript is required for the adaptation to work, which adds a delay when the page loads (even if barely perceptible).

4 Conclusions

To support the diversity of devices currently available, two main solutions are possible. Develop different version of the same application for the different deployment contexts, or include adaptations capabilities in a single solution that enables it to react to its deployment context in order to provide the best user experience possible.

In this work we have taken the latter approach and explored how, in a model based user interface design context, we can create adaptable components to support runtime adaptation of the user interface (as opposed to development time translation between contexts). To this end, we have developed a framework consisting of a number of adaptation patterns for menus. Namely: Menu bar, Footer anchor menu, Select menu, Toggle menu, and Off-canvas menu. The framework implements these menus in a mix of JavaScript (using Modernizr to determine browser and device capabilities) and CSS3. The implementation of the patterns was briefly discussed.

Acknowledgments This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundao para a Cincia e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-015095.

References

1. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
2. James L. Crowley, Joëlle Coutaz, Gaëtan Rey, and Patrick Reignier. Perceptual components for context aware computing. In *Ubicomp*, volume 2498 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2002.
3. A Gustafson. *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement.* Easy Readers, 2011.
4. Ethan Markotte. *Responsive Web Design.* A List Apart, 2010.
5. Tim Pietrusky. Responsive menu concepts. *Appliness*, 9, December 2012.
6. J. L. Silva, O. R. Ribeiro, J. M. Fernandes, J. C. Campos, and M. D. Harrison. The apex framework: prototyping of ubiquitous environments based on petri nets. In *Human-Centred Software Engineering*, volume 6409 of *Lecture Notes in Computer Science*, pages 6–21. Springer, 2010.
7. Luke Wroblewski. *Mobile First.* A List Apart, 2011.