Tools and Architectural Support for Crowdsourced Adaptation of Web Interfaces

Michael Nebeling and Moira C. Norrie

Institute of Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{nebeling,norrie}@inf.ethz.ch

Abstract. There is a vast body of research dealing with the development of context-aware web applications that can adapt to different screen and user contexts. However, the range and growing diversity of devices used for web access makes it increasingly difficult for developers to provide a design and layout that adapts well to every client. To address this, we propose a crowdsourcing approach that allows developers to define a default web interface suitable for many devices and enables the crowd, i.e. other developers or even non-technical end-users, to adapt it to particular use contexts. We present an architecture for creating and sharing adaptations as well as suggesting and applying these in matching contexts. In addition, we discuss the underlying crowdsourcing principles and present a set of visual tools that facilitate the adaptation process.

1 Introduction

We are currently in a period where there is a proliferation of new devices with very different characteristics in terms of, not only screen size and resolution, but also input and output modalities. It is becoming increasingly difficult for application developers to cater for these in a responsive manner. For example, the emergence of new tablet computers such as Apple's iPad and tabletop systems such as Microsoft's Surface requires applications to adapt to much larger forms of touch screens than offered by mobile phones, but many applications on the iPad still use an interface developed for the iPhone and therefore do not take advantage of the greater screen space. When it comes to large interactive surfaces, most application developers, including web site providers, have not even considered these despite the fact that they are now becoming commonplace in offices, meeting rooms, public spaces and homes. In addition, with the means for direct touch input and support for gesture-based modalities on this new generation of multi-touch devices, the trend is also towards user interfaces that promote more natural interactions and are therefore required to scale according to a user's motor skills and potential impairments. We believe that the only way that application developers can cater for such a diverse and rapidly evolving range of use contexts is to adopt a crowdsourcing model where they provide a kernel application and other developers, or even non-technical end-users, create and share adaptations using visual tools in order to support particular devices and preferences.

In this paper, we present an architecture that enables user-driven web site evolution in terms of both the context-awareness and adaptive behaviour of the system through user participation. The approach is based on a context-aware toolkit that integrates with existing web pages so that users can create adaptations directly in the browser. These adaptations are then deployed on a server and automatically downloaded and applied the next time that the web site is accessed from a matching client. The main technical contributions are (1) an extension of the common web application architecture that facilitates crowdsourced web site adaptations, (2) a visual toolset for performing the adaptations that works with existing web sites and does not impose certain conventions on the hypertext specification and (3) a lightweight implementation that builds on only CSS and CSS media queries to support the adaptation process directly in the browser.

We begin in Sect. 2 with a discussion of the background and then provide an overview of the approach in Sect. 3. The proposed adaptation operations are presented in Sect. 4, followed by the crowdsourcing architecture in Sect. 5 and implementation details in Sect. 6. Finally, we discuss the crowdsourcing approach in Sect. 7 and give concluding remarks in Sect. 8.

2 Background

Crowdsourcing has become a popular technique for activating user communities and allowing them to contribute their experience and knowledge [1]. This is generally done by providing a simple interface to access and extend content or functionality and often by turning users into developers [2]. Two of the best known examples of crowdsourced web platforms are Facebook[1] and WordPress[2]. Many parts of these platforms have been developed by volunteers and shared with users in the form of small applications, plugins or themes that can simply be installed and hooked into the running system to extend its functionality. Our aim is to develop mechanisms whereby crowdsourcing can be integrated into existing web applications in order to collectively solve problems of design deficiencies with respect to different screen and user contexts through user participation.

There are two main approaches to supporting adaptivity in applications—*adaptive interfaces* and *adaptable interfaces*. An adaptive interface dynamically adapts the application to support the current viewing context. In contrast, an adaptable interface provides means for customisation, primarily relying on the user to employ the mechanisms to adapt the interface. The fundamental difference is who is in control of the adaptation process: adaptive interfaces are system-controlled relying on context information provided to the application, whereas adaptable interfaces are user-controlled and therefore require user intervention [3]. To cater for the proliferation of different devices used for web

[1] http://www.facebook.com
[2] http://www.wordpress.org

browsing, we propose a novel crowdsourcing approach that makes use of both techniques in that system developers can provide an adaptive interface that initially caters for the most common use contexts, but adaptive features can then evolve at runtime with the help of users who can refine the adaptations to better match their particular use context.

The adaptation of web interfaces to devices is often dealt with in web engineering as one particular aspect of context-awareness and efforts have been made to uniformly address the requirements of personalisation, internationalisation and multi-channel access. For example, for web design methodologies such as WebML and Hera, extensions have been proposed for modelling the behaviour of context-aware sites, e.g. [4,5]. In general, when a context-aware page is requested, the adaptation operations are executed in order to adapt the content, navigation and presentation according to the client context. For the specification of adaptation operations, both rule-based approaches [6] following the Event-Condition-Action paradigm and aspect-oriented techniques [7] that promote the use of aspects to achieve a systematic separation of general system functionality and context-aware adaptation are popular.

A second stream of research on model-based user interfaces has spawned a number of frameworks and tools which separate out several levels of user interface abstraction to be able to adapt to different user, platform and environment contexts, e.g. CAMELEON [8]. The suggested development processes typically unfold along a four-step, top-down approach, starting with domain concepts and task modelling, followed by subsequent transformation steps from abstract to concrete and the final user interfaces. The authoring of adaptive and multimodal user interfaces has also been the subject of extensive research, e.g. [9]; however, in reality, many steps in the promoted top-down engineering process are often skipped by web developers who tend to directly produce the final interfaces through a series of rapid prototypes using rich WYSIWYG tools such as Photoshop or Flash.

As an alternative, we have proposed an approach based on a domain-specific language and a runtime platform using versioning principles and a multi-variant component model [10] to support the development of context-aware web sites. The essence of this language-based approach is to support multiple application-specific aspects of context-awareness by specifying adaptations along a combination of clearly defined context dimensions and states, and to then use a context algebra that allows for powerful matching expressions to support variations of the final interface at the different levels of content, structure and layout depending on the context.

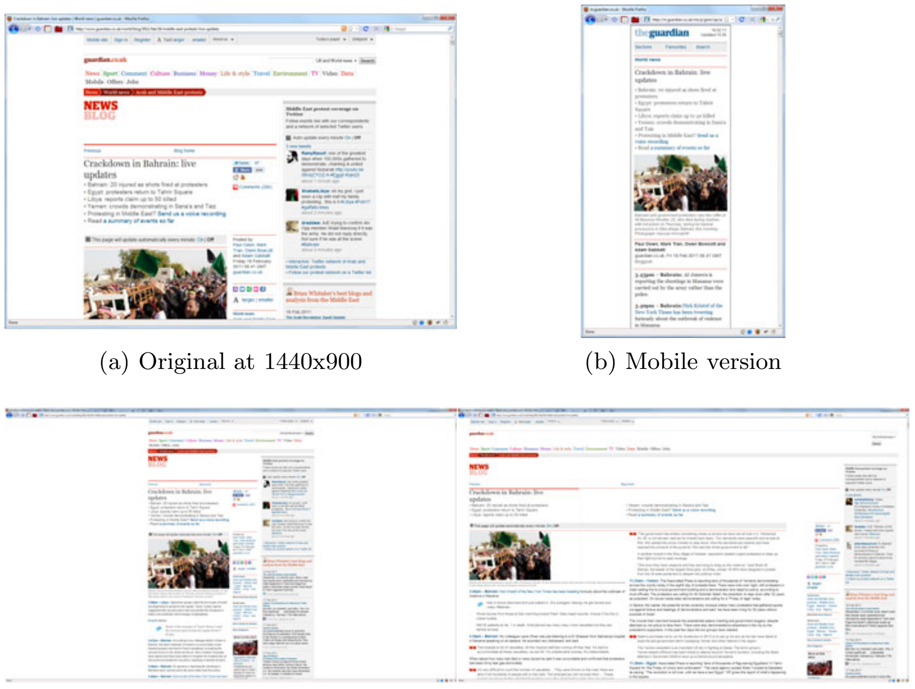
While we acknowledge that all these approaches to context-aware adaptation are very systematic and therefore good at supporting developers in the definition and deployment of adaptations, they primarily target developer-specified adaptation and come at the price of increased complexity and therefore costs. For the proposed crowdsourcing approach, we aim to abstract from underlying models and languages by providing visual tools for the adaptation of the final interface directly in the browser, which may not only increase the productivity of developers, but is potentially also more attractive to a much larger group of non-technical users.

3 Approach

The main goal of our work is to augment developer-specified web interfaces with user-contributed adaptations to cater for a much wider range of use contexts to which applications can adapt. This crowdsourcing model involves a double role of users—one that sees the user as the usual consumer that merely benefits from shared adaptations and the other that turns them into active contributors and allows them to define and deploy adaptations that better match particular client contexts. If users choose to contribute, they can adapt the web interface using our tools that providers can integrate and bundle with a web site or offer separately in the form of a browser plugin. In the first case, created adaptations are managed by the web site and directly shared with consumers, while in the second case they are deployed as part of a separate service where users can take the initiative and create new adaptations and share them even across sites. For the latter scenario, we would like to follow the popular examples of programmableweb.com and userscripts.org, where already large communities of active users maintain shared collections of web mashups and augmentations.

From a technical point of view, our approach builds on the general web architecture where the browser renders the web interface from HTML holding the content and CSS defining the format and style as returned by the server in response to client requests. The principal idea is that all adaptations are essentially represented as modifications of the CSS that can additionally be downloaded for the original web site. This makes for a lightweight adaptation technique as no additional versions of the HTML document must be maintained and also web site generation is not required contrary to many other approaches. Another benefit of using only features of CSS to represent the adaptations is that the actual adaptation process can run completely on the client-side simply by linking the adapted CSS to the web document and hence does not depend on more expensive server-side computation. Also important is the fact that CSS definitions can be cascaded with the consequence that the layout of elements can be adjusted in multiple steps by building on from previous definitions. For users this means that adaptations do not always have to be defined from scratch, but can easily be based on other users' contributions that mostly need to be refined, and therefore tends to require less effort of the end-user.

Web sites that could benefit the most from our crowdsourcing idea are those that typically attract large numbers of users and support them in carrying out a real task. Failure to adapt to the particular screen and user contexts is then perceived as especially disturbing and counter-productive to achieving the task in an efficient way. To give an example, news web sites represent one particular type of application that is typically accessed from a range of different devices. The main task is to provide up-to-date news content and support users in accessing this often text-heavy information: special attention must therefore be paid to all


(a) Original at 1440x900

(b) Mobile version

(c) Original at 2560x1600

(d) Adapted version for large displays

Fig. 1. Examples showing the Guardian's news web site in (a) its original design and (b) the mobile version; (c) demonstrates the relatively poor viewing situation on a large screen and (d) shows adaptations created using our tools to make more effective use of the screen with multi-column layout

factors concerning on-screen readability. Although there is an increasing trend to provide special, mobile versions in order to manage with limited resources and computing power, there is often still a mismatch due to the great diversity in terms of screen resolutions and input/output modalities of hand-held devices. Moreover, looking at the other end of the spectrum, the user experience on large, wide-format screens often does not improve despite the fact that a much larger screen space is available. The main reason for this is a static, fixed-width design of many web sites which let content flow primarily in the vertical direction and therefore tend to leave large parts of the screen unused and, as a result of this, impose unnecessary scrolling. In a recent study [11], we have investigated news site content layout in large-screen contexts and shown that the majority of sites do not adapt well at resolutions above 1024x768 pixels despite the obvious trend towards resolutions much higher than that.

Figure 1 shows the Guardian's online newspaper as one example of a very common news site design viewed in different settings. The site comprises the typical web page elements, such as the header containing the main navigation bar at the top, followed by the main content in three columns and the footer at the very bottom of the page. The leftmost column is the largest and contains the news article content which typically consists of a heading, a picture and the article text itself. The other columns contain a combination of advertisements, related pages and various other services. The site is best viewed at resolutions around 1024x768 as illustrated in Fig. 1a. The mobile version offered by the web site is shown in Fig. 1b at a resolution of 640x960, as an example of how it would view on an iPhone and similar devices. Already this setting could benefit from smaller adaptations in order to use the entire width. To demonstrate that the fixed design is even more problematic in a widescreen setting, Fig. 1c shows the web site in its original design viewed on a 30" screen at a resolution of 2560x1600. To make more effective use of the much larger screen real estate similar to Fig. 1d, our tools would allow a user to define new adaptations suited to the widescreen format by resizing the main content container to fill most of the space available and realigning the inside elements. Additionally, multi-column layout could be used for both the main content area and the sidebar to automatically divide the now relatively large content areas into smaller portions. As a result, paragraphs are of appropriate width rather than producing excessively long lines, also showing more content and related navigation options on the first screen and therefore reducing the amount of scrolling necessary.

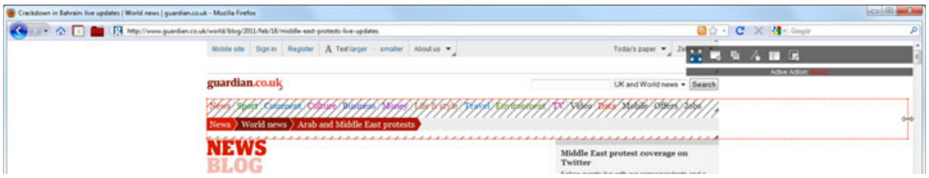
4 Adaptation Operations

To support users in carrying out the adaptation process, we build on a simple set of tools that are loaded into the browser and work directly on the visual representation of the web document. We have defined the set of adaptation operations summarised in Table 1 by looking at which adaptations are required to make more effective use of particularly small and large screens for many web sites. The examples that we considered were not constrained to news sites only, but ranged from blogs, wikis, forums to other forms of applications that are typically used by active user communities for both the consumption of content and content contribution. The defined operations, especially when used in combination, cater for a wide range of adaptations, but also reflect what is technically feasible without imposing a particular model of web site design.

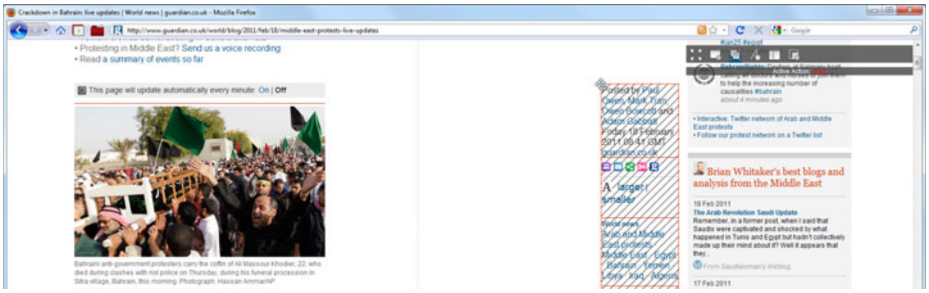
The adaptations carried out by users on different components of a web page, such as the header, navigation, content or footer, need to be reflected at the hypertext level where these are typically represented as a hierarchy of HTML elements or nodes associated with format and style defined in CSS. The two core operations supported by the toolkit allow web page elements to be repositioned and resized using drag-n-drop interaction as showcased in Fig. 2. This is technically supported by adjusting the CSS attributes related to the offset and width/height of elements while being dragged. When dropped, target elements can be positioned either relative to other elements or freely in the web page, which can be helpful when a nested design needs to be broken temporarily in the case that a number of grouped elements need to be realigned. Additionally, the margin between elements can be adjusted by adding new spacer elements to

Table 1. List of the adaptation operations supported by our toolkit with a focus on spatial factors of web site layout

Operation	Description
Move	Changes the position of target elements using drag-n-drop interaction. Moved elements can be docked at the left/top/right/bottom of the drop target, as well as float or be fixed at an absolute position in the web page.
Resize	Resizes target elements at their current position in the web page. When hovered, elements show a size grip in the lower right corner that can be dragged to change the width and/or height. Alternatively, the right side can be dragged to adjust only in the horizontal direction, or the bottom for vertical changes.
Spacer	Adds a new spacer element that will be docked to target elements and can be subject to other operations such as resize to increase the horizontal and/or vertical spacing between elements.
Hide	Hides target elements, or restores elements that were previously hidden using this operation.
Collapse	Substitutes target elements with a placeholder link that can be clicked to expand the original content.
Grow/Shrink Font	Increases or decreases the font size of target elements. Also line spacing can be controlled relative to the change of the font size, or using absolute values.
Single/Multi-column Layout	Controls the number of columns used for an element's layout so that content can be distributed horizontally and flow from one column to another in a flexible way.


Crackdown in Bahrain: live updates | World news | guardian.co.uk - Mozilla Firefox
guardian.co.uk
News ⟩ World news ⟩ Arab and Middle East protests
NEWS BLOG
Middle East protest coverage on Twitter

(a) Resize operation for making appropriate use of the available width


Crackdown in Bahrain: live updates | World news | guardian.co.uk - Mozilla Firefox
This page will update automatically every minute: On | Off
Brian Whitaker's best blogs and analysis from the Middle East

(b) Move operation for realigning elements to fit the new dimensions

Fig. 2. Screenshots showing the adaptation toolkit and two example operations, namely (a) *resize* and (b) *move*, being performed directly in the browser.

improve on the layout where required. Users can also hide elements, or restore hidden ones. Alternatively, elements can be collapsed and replaced by a placeholder link that users can activate to unfold the substituted content. This feature is an example inspired by the mobile version of Wikipedia that automatically does this for sections of an article to first show only their headings. We believe that this kind of adaptation should be made available to a wider range of applications where it cannot be automated so easily because the content may not follow the wikitext conventions, but this could still be achieved manually with the help of users. Finally, the font size of text elements can be adjusted together with the line spacing and additional columns can be used for content layout, the number and size of which can be controlled using the single or multi-column layout operation.

The first set of operations in Table 1 primarily concern spatial factors of the design such as the size and positioning of content in the browser window. Users are provided with additional methods to adjust and balance the spacing between elements, which can be important to save some of the very limited space on mobile devices, or to make effective use of greater amounts of screen real estate on large displays. The means to collapse or even hide certain elements allows for simplifying complex layouts during the adaptation process at design-time, but also provides technical tools to optimise navigation and presentation of content at run-time according to the screen context. It is important to note that all operations can be combined with each other, except when elements are hidden, and generally apply to all web page elements. The last two have a specific focus on controlling text style and flow. This can be important to alleviate the problem of text appearing too small on large displays, or if the original font is too large on a small-screen device. Moreover, multiple columns can lead to a more effective use of the screen space in the horizontal direction, which can be essential for controlling the line length of paragraph elements that may otherwise get excessively long, especially in widescreen environments.

In two related projects, we have proposed a set of layout metrics [11] that can guide users in performing the adaptation as well as experimented with different adaptation techniques specifically in large-screen contexts [12]. The latter also expands more on the idea of using multi-column layout in wide-format viewing situations and explores ways of combining the resulting horizontal alignment of content with the dominant vertical scroll model used by the majority of web sites today.

5 Architecture

So far we have sketched the general idea of user-driven web site adaptations and presented a set of visual tools that facilitate the adaptation process. In this section, we present and discuss an architecture that can be integrated with existing web sites and applications with only minimal effort.

Figure 3 shows an extended version of the general client/server architecture behind a typical web application as well as processes (1) and (2) initiated by

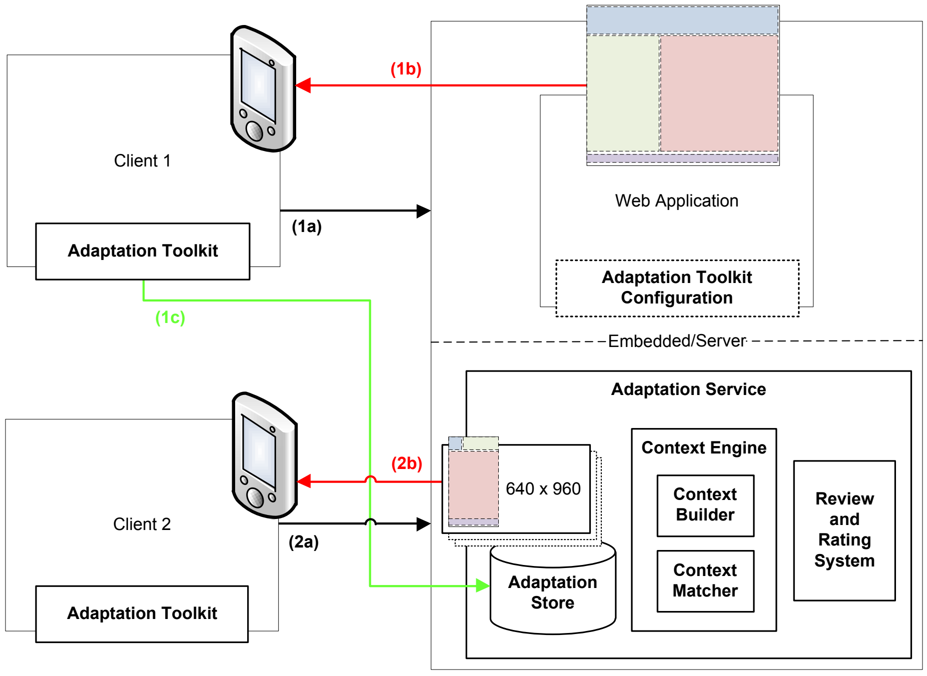


Fig. 3. Conceptual model of crowdsourcing architecture also showing the definition and deployment processes carried out by one client to the benefit of others

to separate clients. As for the extensions, both clients have the adaptation toolkit from the previous section installed and running in the browser. The web application can optionally define a configuration for the client toolkit in order for developers to control which and how web site elements can be adapted by users. Also located on the server-side is the adaptation service that comprises the following three components: (i) the adaptation store which is essentially an interface to an underlying database designed for the storage and retrieval of web site adaptations, (ii) the context engine which functions as a recommender system to rank and select the best-matching adaptations using the context builder and matcher, and (iii) the review and rating system for users to take influence on the recommendations made by the system. The dashed line labelled 'Embedded/Server' indicates that the architecture can vary at this point so that the adaptation service is either managed and provided as part of the particular web site, or administered by an independent service provider. Good examples of the latter scenario are the programmableweb.com or userscripts.org platforms mentioned earlier.

To give a concrete example of how the proposed architecture enables crowdsourced web adaptation, we show two clients using a mobile phone for accessing an example site that has initially been created with a standard resolution of 1024x768 in mind. When the first client accesses the application (1a), it will receive the content in its default layout (1b). This is however too large for the small screen of the client device and therefore difficult to operate. As a countermeasure, client 1 uses our toolkit for adapting the web site to better fit the device resolution of 640x960. The example adaptations include shrinking of the header, repositioning and resizing of the navigation that was originally shown in the sidebar, maximising the main content area and adjusting the footer—all of which are supported by the proposed toolset. The client toolkit automatically synchronises with the adaptation store where it maintains a record with the defined adaptations for the current client context (1c), here simply represented by the screen resolution. Now when client 2 accesses the web site also using our toolkit (2a), it will download recommended adaptations and apply them (2b) so that the end-user automatically sees the mobile version that was created and shared by client 1.

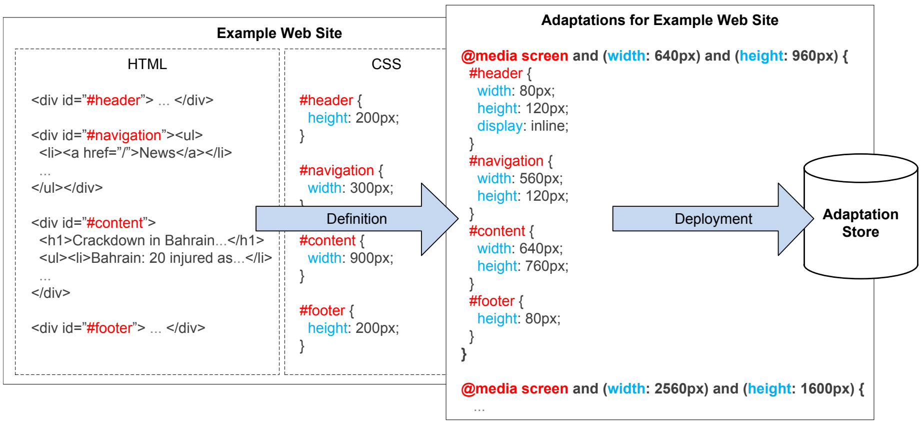
6 Implementation

This section provides more detail on the enabling technologies for the adaptation toolkit and explains supported configuration options. Special attention will be given to the definition and deployment of web site adaptations as well as the quality control system. More documentation and the complete source code are available from our web site.[3]

Adaptation Toolkit. For the implementation of our toolkit, we use client-side JavaScript based on the popular jQuery framework[4] to augment the target web site with a toolbar for invoking the different adaptation actions. If the script is not coupled with and distributed by the web application itself, as in the embedded mode of the architecture, we have also developed an extended version that additionally builds on Greasemonkey[5]—a browser extension for managing user scripts that can make on-the-fly changes to selected web pages as they are loaded. In this separate server setting, we do not constrain the domain and use the `@include http://*` statement to enable the adaptation toolkit for all web sites, but this default setting can be easily changed by users to work only with specific sites using the corresponding Greasemonkey features.

The adaptation operations offered by the toolkit are implemented as follows. The *move* and *resize* functions translate the corresponding `left` and `top` coordinates and adjust the `width` and `height` CSS properties. The *spacer* operation changes the CSS `padding` and `margin` properties of target elements to leave the indicated space blank. The *hide* function toggles the visibility of target elements by setting `display` to `none`. The *collapse* operation is realised using a combination of `display: none` for the original content and a CSS `:before` pseudo element in combination with the CSS `content` property to insert the placeholder link. The *grow/shrink font* operation directly translates to the corresponding CSS `font-size` and `line-height` parameters. Finally, *multi-column layout* is based on the

[3] http://dev.globis.ethz.ch/crowdadapt
[4] http://jquery.com
[5] http://www.greasespot.net/



```
<div id="#header"> ... </div>

<div id="#navigation"><ul>
  <li><a href="/">News</a></li>
  ...
</ul></div>

<div id="#content">
  <h1>Crackdown in Bahrain...</h1>
  <ul><li>Bahrain: 20 injured as...</li>
  ...
</div>

<div id="#footer"> ... </div>
```

```
#header {
  height: 200px;
}

#navigation {
  width: 300px;
}

#content {
  width: 900px;
}

#footer {
  height: 200px;
}
```

```
@media screen and (width: 640px) and (height: 960px) {
 #header {
   width: 80px;
   height: 120px;
   display: inline;
 }
 #navigation {
   width: 560px;
   height: 120px;
 }
 #content {
   width: 640px;
   height: 760px;
 }
 #footer {
   height: 80px;
 }
}

@media screen and (width: 2560px) and (height: 1600px) {
  ...
```

Fig. 4. Illustration of the adaptation process where adaptations are represented in CSS using media queries and stored for the example web site

new CSS3 multi-column module[6] and therefore requires a modern browser that interprets modifications of the `column-count` and `column-width` properties.

The toolkit can further be configured to automatically disable links, text selection and embedded objects such as Flash animations or videos when performing the adaptation, not to interfere with the associated click handlers and other events that could get triggered accidentally. Also, target web sites can tell the toolkit to exclude certain web page elements from the adaptation. We support this by providing special CSS marker classes that can be added to HTML elements to prevent them from being moved, resized or hidden, for example. By default, adaptations can be performed on all non-restricted elements that have an `id`, `name` or `class`, as these can be easily accessed using jQuery and because it is the only feasible way to track the adaptations if not by the node index in the HTML document hierarchy. This also makes sense considering the fact that only important web page components are typically labelled for easy programmatic access or formatting using style blocks or separate CSS resources. Moreover, in case an element is adapted that also specifies a class, users are given the choice to apply the changes to all element instances of the selected class so as to learn from the adaptation of one page element as an example to also let it affect similar elements on the web page.

Adaptation Service. The adaptation store manages alterations of the layout in the form of CSS and CSS media queries as a result of the definition and deployment processes. To illustrate this, Fig. 4 shows a visualisation of the HTML and CSS of an example news article on the left and the adapted CSS on the right. In the original version of the web page, the header component specifies a fixed height and fills the full width of the viewport by default. Part of the adaptations performed by the first client from the previous scenario is the alteration of the header to require less space in both directions so that the navigation can

[6] http://www.w3.org/TR/css3-multicol/

be aligned next to it. The results of the respective *move* and *resize* operations applied to the header and navigation components are decoded in separate CSS media queries. The styles defined here only apply if the device matches the indicated screen resolution. We therefore denote the resolution of the client device on which the adaptations were created originally. For the deployment, we support two modes so that adaptations can either be stored as a new, alternative *version* for the current client context, or as a *revision* to overwrite and replace a previous version. The underlying database model is fairly simple and just stores the adapted CSS together with the collected context parameters. In the case that the plugin was used, the URL of the original site is additionally tracked and stored in the database.

The context engine plays a bigger role for the retrieval of web site adaptations, but is also used for storage when the context builder creates a model from the context of the client that is performing the adaptations to associate the collected context information with user-defined adaptations. Our implementation evaluates the navigator and screen objects in JavaScript to collect the context information related to the browser and device, such as the user agent string, screen size, orientation and resolution. It also allows users to review and complement this information with aspects that cannot always be queried programmatically, such as the device type, model or name and supported input methods, e.g. mouse and keyboard, pen and/or touch. Clients can also specify user-related attributes for the context builder, e.g. their current location, the language preference and whether they are right or left-handed, although these values are not interpreted by default, but this is a way for users to hint at other context facets that might also be relevant for a particular use case and help to evolve both the context model and the adaptive behaviour of the system. The collected context information is then associated with the corresponding adaptations and stored in the database. At retrieval, the context matcher is responsible for scoring user-contributed adaptations and determining the best match when the site is accessed via the toolkit. The method used to evaluate similarity between the client contexts associated with adaptations in the database and the current client context is a combination of SQL and CSS media queries. SQL is used to filter stored adaptations by target web site and mostly by user-related context aspects not supported by media queries. The device-related matching process is however delegated to the client-side where it is performed directly in the browser using the returned CSS media queries.

Finally, the review and rating system complements the recommender system with user-driven means to control the scoring applied as part of the context matching process. We allow clients to preview and choose from a set of user contributions that achieved a reasonably high score computed by the context engine. We then build on the simple idea of ranking adaptations higher the more clients actually use them, for which we keep a server-side record of the number of client accesses together with the contexts of selected adaptations. The ratio of active to matching sets of adaptations then builds an additional factor for the matching process: it controls the order in which media queries are cascaded by the toolkit so that only the adaptations with the highest scores are the ones that determine the final user interface.

7 Discussion

One of the key issues for any crowdsourcing model is the "cold-start problem" and so there has been a lot of research on motivational aspects. Results suggest that for users the willingness to contribute can be improved, for example, by using principles of social psychology [13] or by exploiting social connectedness [14], or indirectly, and for the user implicitly, through games [15]. Our approach attempts to primarily draw on the effect that users can see the interface improve directly through their actions. Similar to [16], user motivation can further be raised by letting them know how many other users with the same or similar devices can also benefit. This we can estimate by comparing the context against the history of client contexts used as part of our implementation.

Two aspects that often tend to be ignored in crowdsourcing approaches are security and privacy. As far as security is concerned, it is important to see that our approach focuses on crowdsourced *adaptation* rather than *augmentation* of web sites. Hence, it deals with adjusting the presentation rather than adding new content or functionality, which could be of potential concern, and therefore requires significantly less control compared to web augmentation [17]. Also privacy is not an issue since our focus is on the contributions of users rather than the users themselves. As a consequence, users can contribute anonymously while the sharing and targeting of user-defined adaptations is primarily based on stereotypical rather than personal aspects.

Another important aspect that we have not discussed so extensively in the paper is scalability when a crowd of users would really contribute with web site adaptations. This can have an impact on both the quality of adaptations and performance of the system. The suggested review and rating system could help maintain a high level of quality which could further be improved by giving more control to selected web site users, so-called trusted users, so that their approval or rejection has significant impact on the particular score and thus the overall ranking of user-defined adaptations. Another way to control quality is to limit which and how adaptations can be defined. The proposed adaptation operations therefore concern primarily those aspects of the design that are directly related to the viewing context, such as the size and position of elements, rather than style and colour. As for the performance, we argue that this is more a question of the implementation rather than the approach. Our experiments with the current implementation based on CSS and CSS media queries have shown a fairly high performance even when several hundred adaptations created for different client contexts were applied to an example web site. Even on mobile devices, modern browsers are very fast at parsing and executing the hypertext definitions and we can directly leverage this performance in our approach. In particular in a multi-user, multi-device scenario, we are convinced that the delegation of the adaptation process to the client-side will pay off as less of the more expensive server-side computation is involved. The lightweight implementation also has some drawbacks. For example, we know it is better to remove hidden elements from the HTML before they are transferred to the client and to let the move operation also affect the hypertext specification, not to break the structure and flow of the document by relying on relative or absolute positioning via CSS independent of the document hierarchy. This means that docking a dragged element to a target should result in moving the corresponding element before or after the drop target's position in the HTML document. These are examples of adaptations that could be additionally implemented on the server-side to complement the pure CSS-based adaptation techniques with HTML pre-processing before the content is delivered to the client.

Finally, it must be mentioned that the adaptation toolkit itself also needs to be adaptive to support the adaptation process under any given device constraints. For example, while there is sufficient extra space to perform adaptations when working on a large device, the toolkit operations will also have to be feasible on small form-factor devices. We have started to support this by making the adaptation toolkit the subject of its own principles in that the crowd can also adapt the design and functionality of the toolkit. For the visual representation, we provide a special edit mode so that adaptation operations can also concern the toolkit components, while the source code is available for more experienced users to add new operations or refine existing behaviour programmatically.

8 Conclusions

We have presented a crowdsourcing approach to web site adaptation based on a lightweight extension of the common web application architecture and visual tools that facilitate the adaptation process. The main idea is to support developers in specifying web interfaces that can adapt to the range and increased diversity of web-enabled devices available today by also allowing users to create and share adaptations so that other owners of the same device and with similar preferences can directly benefit. The paper discussed the technical and design challenges for adopting this crowdsourcing model, in particular when it comes to quality control, and described a concrete implementation of the proposed concepts and mechanisms using adaptation techniques based on only CSS and CSS media queries. Although our experiments using the toolkit and architecture for adapting existing, real-world web sites are promising, more detailed evaluations of the visual toolkit from a user perspective and testing the implementation on a larger scale with many different user contributions are planned for the future. This will include carrying out user studies and collecting feedback when using the toolkit for access and adaptation on a range of different devices as well as experimenting with different sharing and ranking modes to improve user support. It will also be interesting to mine the data created through user participation as it is likely that user-driven adaptations generate valuable data for developers. The collected data can then inform the general design of a web site with respect to different use contexts so that patterns can be recognised and hopefully design deficiencies prevented before they get replicated.

Acknowledgements

We would like to thank Michael Grossniklaus and Stefania Leone for their valuable feedback on earlier drafts of this paper. This work was supported by the SNF under research grant 200021_121847.

References

1. Howe, J.: The Rise of Crowdsourcing. Wired 14(6) (2006)
2. Kazman, R., Chen, H.M.: The Metropolis Model: A New Logic for Development of Crowdsourced Systems. CACM, vol. 52(7) (2009)
3. Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. UMUAI 6(2-3) (1996)
4. Ceri, S., Daniel, F., Matera, M., Facca, F.M.: Model-driven Development of Context-Aware Web Applications. TOIT 7(1) (2007)
5. Frăsincar, F., Houben, G.-J., Barna, P.: Hypermedia presentation generation in Hera. IS 35(1) (2010)
6. Daniel, F., Matera, M., Pozzi, G.: Managing Runtime Adaptivity through Active Rules: the Bellerofonte Framework. JWE 7(3) (2008)
7. Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.J., Meißner, K.: Harnessing the Power of Semantics-Based, Aspect-Oriented Adaptation for amacont. In: Proc. ICWE (2009)
8. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi- Target User Interfaces. In: IWC, vol. 15 (2003)
9. Paternò, F., Santoro, C., Mäntyjärvi, J., Mori, G., Sansone, S.: Authoring pervasive multimodal user interfaces. IJWET 4(2) (2008)
10. Nebeling, M., Grossniklaus, M., Leone, S., Norrie, M.C.: Domain-specific language for context-aware web applications. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 471–479. Springer, Heidelberg (2010)
11. Nebeling, M., Matulic, F., Norrie, M.C.: Metrics for the Evaluation of News Site Content Layout in Large-Screen Contexts. In: Proc. CHI (2011)
12. Streit, L.: Investigating Web Site Adaptation to Large Screens. Master's thesis, ETH Zurich, doi: 10.3929/ethz-a-006250434 (2010)
13. Harper, F.M., Li, S.X., Chen, Y., Konstan, J.A.: Social Comparisons to Motivate Contributions to an Online Community. In: de Kort, Y.A.W., IJsselsteijn, W.A., Midden, C., Eggen, B., Fogg, B.J. (eds.) PERSUASIVE 2007. LNCS, vol. 4744, pp. 148–159. Springer, Heidelberg (2007)
14. Bernstein, M.S., Tan, D.S., Smith, G., Czerwinski, M., Horvitz, E.: Personalization via Friendsourcing. TOCHI 17(2) (2010)
15. von Ahn, L., Dabbish, L.: Labeling Images with a Computer Game. In: Proc. CHI (2004)
16. Rashid, A.M., Ling, K.S., Tassone, R.D., Resnick, P., Kraut, R.E., Riedl, J.: Motivating Participation by Displaying the Value of Contribution. In: Proc. CHI (2006)
17. Arellano, C., Díaz, O., Iturrioz, J.: Crowdsourced Web Augmentation: A Security Model. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 294–307. Springer, Heidelberg (2010)