Generation of Multi-Device Adaptive MultiModal Web Applications

Marco Manca, Fabio Paternò, Carmen Santoro, and Lucio Davide Spano

CNR-ISTI, HIIS Laboratory, Via G. Moruzzi 1,
56124 Pisa, Italy
{marco.manca,fabio.paterno,carmen.santoro,
lucio.davide.spano}@isti.cnr.it

Abstract. This paper presents a set of tools to support multimodal adaptive Web applications. The contributions include a novel solution for generating multimodal interactive applications, which can be executed in any browser-enabled device; and run-time support for obtaining multimodal adaptations at various granularity levels, which can be specified through a language for adaptation rules. The architecture is able to exploit model-based user interface descriptions and adaptation rules in order to achieve adaptive behaviour that can be triggered by dynamic changes in the context of use. We also report on an example application and a user test concerning adaptation rules changing dynamically its multimodality.

Keywords: Model-based design, Multimodal User Interfaces, Adaptation, Adaptation rules.

1 Introduction

Recent years have been characterized by the increasing availability of interaction platforms, differing also in modalities. One example is the vocal modality, which nowadays enjoys greater support and is used in various applications in the mass market (e.g. Google Voice, iOS SIRI). Unfortunately, although Web applications are widely used for many purposes, this richness in interaction modalities has penetrated the Web only in a limited manner. Indeed, besides the issues of addressing the heterogeneous set of available devices, Web applications are affected by the problems determined by traditional input/output interactive mechanisms that can result in rigid and unnatural interactions, which are often inconvenient especially for mobile Web users. Exploiting multimodality in Web applications is an important opportunity, since multimodal interaction can significantly increase the communication bandwidth between the user and the devices by offering flexibility and freedom. However, to date, there is a lack of approaches able to flexibly address the issues of developing multimodal Web applications able to adapt to the different contexts of use to better support mobile users. Some reasons for this can be found in the inherent complexity of developing adaptive multimodal interactive systems.

F. Daniel, G.A. Papadopoulos, P. Thiran (Eds.): MobiWIS 2013, LNCS 8093, pp. 218–232, 2013.

To this regard, the exploitation of model-based descriptions can be a promising solution for addressing the increasing complexity of current interactive applications. Furthermore, the potential of model-based approaches can be better exploited when addressing user interfaces using different modalities, but unfortunately their application has mainly focused on mono-modal interfaces or desktop-to-mobile adaptation, basically involving just the visual channel. In addition, most model-based approaches have mainly considered adaptation at design time (e.g. [7, 13]). One major challenge is thus to be able to handle effectively and dynamically adaptation of user interfaces at runtime. This seems a viable solution for providing users with the most effective user interfaces in various contexts, as it is able to address unforeseen (at design time) situations and also configure solutions for future needs.

We present a novel solution that supports multimodal (combining graphical and vocal modalities) adaptive user interfaces by exploiting a model-based language. In particular, the main contributions are: a novel generator of multimodal interactive applications that are implemented in HTML annotated with particular CSS classes, which can be easily interpreted by different clients in different devices; and an adaptation support able to change the multimodality at various granularity levels according to dynamic contextual events.

In the paper, after discussing related work, we provide some useful background information to introduce some concepts that have been further refined and exploited in this work. Next, we describe the overall architecture supporting multimodal adaptive Web interfaces, including the descriptions of some elements, such as the language for specifying adaptation rules. We provide details on the innovative solution proposed in this work that is able to generate multimodal Web pages. Then, we introduce an example application and report on an evaluation in which various multimodal adaptations have been considered. Lastly, we draw some conclusions and provide indications for future work.

2 Related Work

A model-based Framework for Adaptive Multimodal Environments (FAME) has been proposed in [5]: it includes an architecture for adaptive multimodal applications, a new manner to represent adaptation rules and a set of guidelines to assist the design process of adaptive multimodal applications. However, FAME's objective is just to guide the development of adaptive multimodal applications, but it does not provide support for automatic application development, as in our case.

Octavia et al. [9] describe an approach to design context-aware application using a model-based design process, but they do not address multimodal adaptation. The model-based approach is considered also in [12] for its flexibility. The context of use is a structured information space whose goal is to inform the adaptation process. When some changes in the context occur, a module identifies the appropriate transformation to adapt the UI to the current change. However, authors of [12] do not consider run time adaptation without regenerating the whole interface and they do not consider multimodal adaptation.

A model and an adaptation architecture for context-aware multimodal documents has been put forward [3]. However, the adaptation model described in that paper only considers static adaptation phase (at design time), and there is no runtime adaptation.

MyUI [11] provides a framework for run-time adaptations while supporting accessibility. To cover the heterogeneity of users, environments and devices, MyUI relies on an extensible repository of multimodal UI design patterns expected to contain the knowledge needed to perform the described three-stage process of UI generation and adaptation. MyUI's approach, although focused on accessibility issues, shares with our work some concepts. However, its design-patterns approach can quickly become cumbersome, e.g. in the need of consistently specifying/maintaining the relations existing between the design patterns specified within the repository.

Another approach [6] proposes and investigates a semi-automatic rule-based generation of multimodal user interfaces where a discourse model (representing modality-independent interaction) is transformed to different, automatically rendered modalities. Although the model transformation process exploited in that work can be compared to the one used in our approach, tool support has not been deeply investigated in that approach.

DynaMo (Dynamic multiModality) [1] proposes an approach aimed at handling dynamic environments where devices and applications are rapidly evolving. It is based on partial interaction models, thus it combines and completes them at runtime. Differently from this work, we use UI models which are complete, and are able to cope with the evolving context by dynamically adapting the UIs at runtime according to current conditions of users, devices, and surrounding environment.

W3Touch is a tool that aims to collect user performance data in order to help identify potential design problems for touch interaction. In mobile devices it has been exploited [8] to support adaptation based on two metrics related to missed links in touch interaction and zooming issues to access Web content. However, that work has not considered multimodal UIs. While [2] introduces some initial ideas about adaptive user interfaces through the use of model-based approaches, in this paper we present a novel solution for this purpose able to build multimodal Web interactive applications, which can be executed in any browser-enabled device, without requiring developers to use any particular API in addition to standard HTML, CSS, and JavaScript.

3 Background

We considered the MARIA language [10] for describing interactive applications as a starting point of our work because it is one example of engineered model-based language, with already some tool support. It provides one language for the Abstract User Interface level (AUI), and multiple languages for platform-dependent concrete UI (CUI) descriptions. The associated tool (MARIAE) supports multiple transformations between levels, and generators for various platforms (desktop, vocal, multimodal). However, the multimodal generator provided [7] generates X+V implementations,

which was the implementation language considered also in other model-based generators [13]. X+V was an attempt to combine VoiceXML and XHTML, but it is no longer supported by current browsers. Thus, we decided to investigate novel solutions able to generate multimodal user interfaces that can be exploited with current browsers.

The MARIA concrete multimodal language uses the CARE properties to specify how the various modalities are allocated. Such properties were originally defined in [4]. We interpret them in the following way: *Complementarity*: when the considered part of the UI is partly supported by one modality and partly by another one; *Assignment*: the considered part of the UI is supported by one assigned modality; *Redundancy*: the considered part of the UI is supported by both modalities; *Equivalence*: the considered UI part is supported by either one modality or another.

To provide a flexible definition of multimodal interfaces it is possible to apply such properties to various UI concepts of the MARIA language: composition operators (groupings, repeaters, …), interaction and only-output elements. In order to have different modalities allocated even at a finer grain, interaction elements themselves can be further structured into prompt, input, and feedback with this meaning:

- *Prompt*: it represents the UI output indicating that it is ready to receive an input;
- *Input*: it represents how the user can provide the input;
- *Feedback*: it represents the response of the system after the user input.

We interpret the CARE properties from the user viewpoint. Thus, the *equivalence* property is applied only to the input part of an interaction element when multiple modalities are available, in order to indicate the possibility to insert the input indifferently through one of these modalities. We do not apply it to output elements because even if the system can choose between two different modalities to present output, in the end only one of them will be provided and perceived by the end user, and so the user will perceive a modality assignment to the output element. *Redundancy* is applied to output elements and to prompt and feedback but not to input for interaction elements since once the input is entered, it does not make sense to enter it again through another modality.

We have also developed a high-level XML-based description language for defining the transformations affecting the interactive application when something occurs at the context level or in the interactive application. This language enables the specification of adaptation in terms of rules expressed as Event, Condition, and Action(s):

- *Event*: It can be an elementary event or even a composition of events occurring in the interactive application or in the context of use. In any case, its occurrence triggers the application of the rule.
- *Condition* (optional): a Boolean condition to be satisfied to execute the associated action(s). It can be related to something happened before, or some state condition.
- *Action*: How the abstract/concrete/implementation description of the interactive application should change in order to perform the requested adaptation.

The impact of adaptation can be at different granularities: complete change of UI; change of some UI parts; change of attributes of specific UI elements.

Figure 1 shows an example rule related to environment noise. If the noise level is greater than 50 (decibels) as specified in the condition part (rows 6-10), then this rule is triggered and the presentation is adapted in a way that only graphical interaction is supported: indeed the corresponding action updates the presentation CARE value by setting the graphical assignment value to the output attribute (row 13-16).

```
1  <rule priority="0" name="onlyGUI" id="rule4">
2      <event>
3          <simple_event event_name="onEnvironmentNoiseLevelIncreased"
4                        xPath="/user/environment/@noise_level" externalModelId="ctx"/>
5      </event>
6      <condition operator="gt">
7          <entityReference xPath="/user/environment/@noise_level"
8                           externalModelId="ctx"/>
9          <constant value="50" type="int"/>
10     </condition>
11     <action>
12         <update>
13             <entityReference xPath="/descendant::presentation[@id='agencyList_iu']/@output"
14                              externalModelId="cui"/>
15             <value>
16                 <constant value="graphical_assignment" type="string"/>
17             </value>
18         </update>
19     </action>
20 </rule>
```

Fig. 1. Rule example

4 Approach

Figure 2 shows the overall architecture of the adaptation environment that can perform different kinds of adaptations, not only the multimodal one. By using an authoring environment at design time, it is possible to obtain both the logical specification of the interactive application ("Logical UIs" in Figure 2) and the specification of the rules to be used for adaptation ("Adaptation Rules") which are both passed as input to the Adaptation Engine. The latter determines the optimal adaptation for the interactive system at hand in the current context of use, and based on the specified adaptation rules. To achieve this goal, the Adaptation Engine receives from the Context Manager information about the current context (e.g. events detected), and checks whether some adaptation rules could be triggered, by analyzing if the event/condition parts of some rules are satisfied. In the positive case, the corresponding action part of the selected rule is sent to the module in charge to perform the associated modifications. Such module can change depending on the type of adaptation defined in the triggered rule. As Figure 2 shows, there are three possibilities:

- if there is a complete change of modality (e.g. from a graphical interface to a vocal UI) the CUI of the current UI is sent to the adapter for the target modality, which has to change the structure of the application in order to make it suitable for the new modality, and then provide the new specification to the relevant generator;

- if there is a change in the structure of the current user interface (e,g. a change in the user interface layout structure), then the adaptation engine modifies the current CUI and sends the result to the generator to update the UI;
- if there is a small change in terms of UI attributes (e.g. the font sizes are increased) then for sake of efficiency the adaptation is performed directly on the current UI implementation through some scripts.

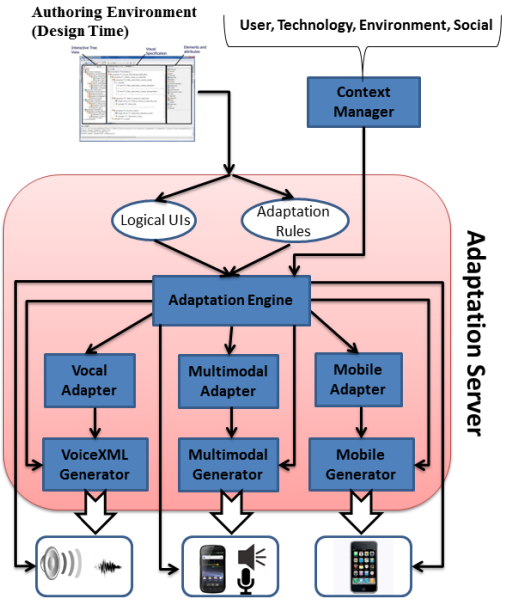

Fig. 2. Overall architecture

The adaptation engine receives events from the Context Manager, which handles relevant information regarding context (which is defined in terms of four key dimensions: user, technology, environment, social aspects). The Context Manager has a client/server architecture (its description is out of the scope of this paper for lack of space). The context events are detected by Context Delegates (small applications installed on the devices), which monitor environment parameters and send updates to the Context Manager.

5 Multimodal Generator

The goal of the new multimodal UI generator is to produce implementations in standard HTML+CSS that can be executed by any browser-enabled device. Such implementations are structured in two parts: one for the graphical UI and one for the vocal one. The vocal part is supported by JavaScript code accessing the Google APIs for vocal interaction.

In order to understand when to activate the JavaScript functions that access the vocal libraries at run-time (see Figure 3), each user interface element is annotated with a specific CSS class in the implementation generated at design time, according to the indications of the CARE properties. If it contains a vocal part, the class *tts* for the output elements and prompt part of interaction element is added, while the class *asr* is added for the input parts of interaction elements. The generated elements are marked with these classes because the included scripts use them to identify all the graphical elements having an associated vocal part.

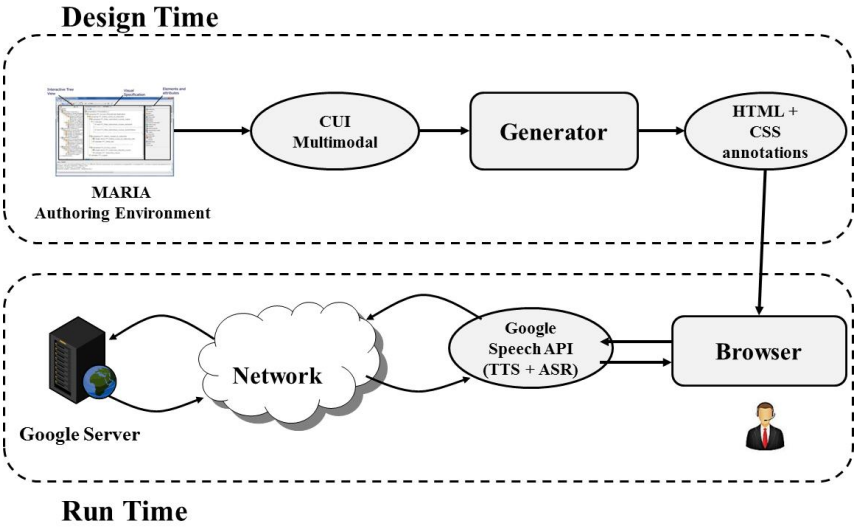

Fig. 3. The generation process and the Multimodal support

It is worth pointing out that the way in which JavaScript code accesses the vocal libraries depends on the platform type (desktop or mobile). More specifically, for desktop applications we exploit the Chrome extension mechanism. Chrome extensions are small software programs that can modify and enhance the functionality of the Chrome browser. They can be written using Web technologies such as HTML, JavaScript, and CSS. We have implemented an extension that is able to exploit the Chrome APIs for automatic speech recognition (ASR) and text-to-speech synthesis (TTS). The basic idea is that such extension, by analysing the DOM of the multimodal web page can identify the elements that need vocal support and call the corresponding vocal Google libraries (depending on the CSS class).

Since the mobile version of Chrome is currently not able to support extensions, for mobile devices we cannot access the Google APIs for ASR and TTS through JavaScript. Thus, for the mobile Android platform we developed a Multimodal Browser Application that exploit a Java component called WebView, which is able to show Web pages using the WebKit rendering engine. In this case the DOM of the Web pages is accessed through a JavaScript that is inserted in the page by the WebView

component created for this purpose. The difference with respect to the desktop solution is that the calls to the vocal synthesizer and recognizer are not carried out through JavaScript but using Java functions. Thus, the mobile version still uses JavaScript to access the DOM and find the vocal elements but then call Java functionalities for implementing the vocal part. This is obtained thanks to the fact that Android supports the possibility to invoke methods contained into a Java class through a JavaScript interface in a WebView component. In the Java class we can define the methods we want to expose to JavaScript, such as the methods to synthetize a text or to recognize a vocal input. In this way it is possible to call from the JavaScript executed in the WebView the Java methods defined in the JavaScript interface. Besides these technical differences in the implementation, both solutions can interpret the same HTML descriptions of the multimodal UIs, which are structured in the same way in both mobile and desktop versions.

Figure 4 shows a code example for multimodal interaction generated from a concrete user interface (CUI) and concerning the definition of a *select* element with both a graphical and vocal part.

As you can see, even the vocal part (which starts from row 11) is obtained through HTML (through *span* and *p* tags), and it is hidden (see row 11: the *style* attribute value is "display:none") but it is always accessible from the DOM.

The label element (row 2) is associated with the *tts* class because the prompt part is associated with the redundancy property. The corresponding vocal prompt has the same id plus the suffix '_tts' (row 11). It contains the specification of two prompts (by using the p element annotated with tts_speech class, each prompt has different and sequential count attribute, which is used to indicate in which order the prompt will be synthesized) and a value for the break length (p element annotated with tts_break class) that represents a pause after synthesizing the prompt and before recording the vocal input. For each output vocal element it is possible to define a number of properties (e.g. row 12) such as pitch, rate, gender, volume that permit to control the Text-to-Speech Engine.

The *select* element (and obviously all the input elements) is annotated with ASR class (row 5) because its input part value is "Equivalence", the corresponding vocal element has the same id plus a suffix that depends on the element type (row 22: suffix '_select'). This element contains some parameters: listValues (row 23): when true after the prompt all the possible input values will be synthesized; askConfirmation (row 24): when true the multimodal support asks users to confirm the vocal input recorded; vocal Feedback (row 25): it indicates that the multimodal support will synthesize the text indicated in the element and the input recorded; vocal Event (row 29) such as 'no input' and 'no match': this element indicates the behavior of the multimodal support when a user does not enter an input or enter an unrecognizable input, it is possible to synthesize a message or synthesize the next prompt (the prompt identified by the next value in the count attribute).

We introduce this type of annotations because the multimodality support automatically attaches an event handler triggered by the *onload* event that identifies all the elements associated with the *tts* and *asr* classes, and sets a handler related to the focus event for each of them. This additional event handler identifies the corresponding

element in the vocal part (same id plus a suffix, see an example in Figure 4 where in row 5 there is the graphical element and in row 22 the corresponding vocal element) and activates it in order to obtain multimodality when the corresponding graphical element receives the focus. In case of equivalent input when the UI element receives the focus, the vocal prompt is rendered (if the prompt is associated with vocal modality as well); in any case the vocal input recording is then started.

If the input is entered graphically, the application generates an *onchange* event (all the input elements have a handler for this), then the vocal input recording is stopped; if the feedback is associated with the vocal modality, it is rendered accordingly. In addition, all the graphical elements with an associated vocal part have a *tabIndex* attribute with consecutive values: if the application parameter named "continuous reading" is set to true, then after having synthesized a text or recorded an input, the focus is automatically moved to the element with the next *tabIndex* value; otherwise the application waits for the user to set the focus manually.

```
1  <!-- GRAPHICAL PART -->
2  <label for="departure_city" id="departure_city_label" class="tts" tabIndex="46">
3      Choose the departure city
4  </label>
5  <select id="departure_city" name="departure_city" class="asr" tabIndex="47">
6      <option id="departure_city_1" value="milan">milan</option>
7      <option id="departure_city_2" value="rome">rome</option>
8      <option id="departure_city_3" value="florence">florence</option>
9  </select>
10 <!--VOCAL PART -->
11 <span style="display:none" id="departure_city_label_tts">
12     <p class="tts_speech" title="{'count' : '1', 'timeout' : '10s', 'pitch' : 'medium',
13                                  'rate' : 'medium', 'gender' : 'female'}">
14         Choose your departure city
15     </p>
16     <p class="tts_break">1s</p>
17     <p class="tts_speech" title="{'count' : '2', 'timeout' : '10s', 'pitch' : 'medium',
18                                  'rate' : 'medium', 'gender' : 'female'}">
19         Choose the departure city, try to say Rome
20     </p>
21 </span>
22 <span style="display:none" id="departure_city_select">
23     <p class="listValues">true</p>
24     <p class="askConfirmation">true</p>
25     <span class="vocalFeedback" title="{'count' : '1', 'timeout' : '3s'}">
26         The departure city is
27     </span>
28     <p class="tts_break">1s</p>
29     <span class="vocalEvent">
30         <p class="noInput" title="{'reprompt' : 'true'}">No input</p>
31         <p class="noMatch" title="{'reprompt' : 'true'}">
32             You choose a city not available, sorry
33         </p>
34     </span>
35 </span>
```

Fig. 4. Multimodal generated code

The management of the two modalities through the JavaScript event handlers is completely automated by the multimodal support. In the Web application markup code there is only the need to indicate the graphical and the vocal parts through

annotation of the CSS classes. This is performed automatically by the multimodal generator that builds final UIs from MARIA specifications. In addition, the differences in the implementation of the desktop and mobile multimodal support do not have impact on the HTML + CSS syntax used for specifying the UI, which is valid for both platforms.

6 Adaptation

Figure 2 shows the architecture of the environment, in order to receive the activated adaptation rules, the Adaptive Multimodal Application has to subscribe to the Adaptation Engine sending the application name and optionally the name of the current user (if logged-in). Consequently, when the Adaptation Engine module receives the application subscription it has to subscribe itself to the Context Manager in order to be notified of the occurrence of the contextual events expressed in the adaptation rules associated to the subscribed application.

As previously mentioned, the adaptation rules are expressed as Event, Condition and Action. The action part can be expressed by the following elements: create, read, update, delete and invoke functions. However, for supporting multimodal adaptation the update element, which changes the CARE values to modify graphical/vocal interaction, is sufficient. This change can be sent either to the multimodal generator (if it is defined in terms of the concrete user interface language) or directly to the implementation. In the former case, first the CUI is updated and then a new UI is generated accordingly.

If the rule actions are sent by the adaptation engine directly to the multimodal support of the application implementation, then it is interpreted in the following way:

- *Prompt* and *Output* part: if the new CARE value is "graphical assignment", the adaptation script (in the multimodal support) removes the TTS class so that the application does not activate the vocal part; otherwise, the script adds the TTS class. If the new CARE value is "vocal assignment" then the graphical part is hidden;
- *Input* part: if the new CARE value is "graphical assignment" then the adaptation script removes the ASR class, otherwise the script adds it. If the new CARE value is "vocal assignment" then the graphical part is visible but the input can be entered only vocally, thus the input element is read only;
- *Feedback* part: it is implemented with a *span* with *vocalFeedback* class inside the vocal element (see Figure 4 row 27); if the CARE value is "graphical assignment" the adaptation script changes the value of the *span* class from *vocalFeedback* to *noVocalFeedback* so that the application does not synthesize the feedback.

Thus, changing the CSS class of the elements according to the CARE properties is sufficient to obtain the multimodal adaptation because the support will activate or not the vocal part accordingly.

7 Example Application

An adaptive multimodal application supporting interaction with a Car Rental service was obtained from a model-based specification using the MARIA language, through automatic transformations. Through this application users can search for cars by specifying some parameters such as location and maximum distance allowed from it.

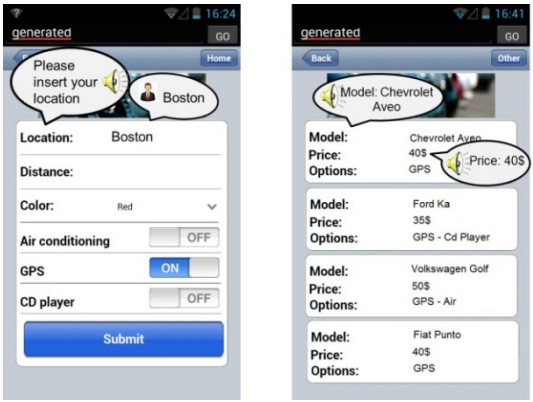

Fig. 5. The UI of the Car Rental application

Furthermore, the user can specify additional options regarding the car, i.e., the on-board availability of air conditioning, CD player and GPS. The implemented application exploits existing REST Web services described through the WADL language to support the expected functionalities. The application generated contains forms, navigation items, access to remote Web services, pages with dynamic content. Thus, it has all the main features that characterize Web applications. Through this Web service it is possible to obtain some information listing e.g. the agencies and their related information, cars available in an agency, etc. For instance, after the user specifies the search parameters through the application, a list of agencies that fulfill the search criteria is provided: for each of them the available cars along with their description, price and extra features are listed. Figure 5 shows example user interfaces of the Car Rental application obtained. The first one is the multimodal search page through which users can provide the search parameters (also using the vocal modality): for all interaction elements the prompt and the feedback part are redundant, while the input part is equivalent. The second page is the search result, rendered in a redundant way: all the output elements are rendered both graphically and vocally.

8 Evaluation

We carried out a user test on the multimodal car rental application. It was useful to gather some preliminary empirical feedback on various types of adaptations affecting a multimodal UI, and how they were perceived by users, in terms of their appropriateness and impact on user experience.

We identified five types of multimodal adaptations, described below. The rationale followed for selecting them was driven by the willingness of covering various types of adaptation of the multimodal UI considered, by varying the way in which the two modalities (graphical and vocal) were allocated. This led to selecting three 'extreme' cases (namely: *Adaptation 1, 2 and 3*) where the adaptations changed the number of modalities involved. For instance in Adaptation1, the UI moved from a multimodal (graphical + vocal) UI, to a mono-modal (only graphical UI). In addition, two cases (*Adaptation 4 and 5)* covered "intermediate" situations (i.e. the UI is multimodal before and after adaptation, but the allocation of graphical/vocal modalities to the various UI elements changes). We considered the following five adaptation scenarios.

- *Adaptation 1*: the user started with a multimodal UI where system output was redundantly provided and user's input was provided only graphically. A context change was triggered (simulating a noisy environment) and the UI became a (mono-modal) graphical UI. This is because when user enters a noisy environment, audio is less effective, while graphical modality is more appropriate.
- *Adaptation 2*: the initial UI was the same as in 1st scenario. Then, the user started walking quite fast, and the UI became a (mono-modal) vocal UI. This is because when the user walks fast, s/he cannot look often at the device display, so the audio channel is more appropriate.
- *Adaptation 3*: the initial UI was the same as in 1st scenario. The user started walking slowly, and the Adapted UI was a multimodal UI in which: i)UI output elements were rendered redundantly; ii) Selection/edit elements equivalently supported user input (either vocally or graphically), and redundantly rendered the prompt and the associated feedback; iii)for activation/navigation elements: user could graphically provide input; prompt was graphically provided, the feedback was redundantly given. This was done because when user walks slowly, there are some opportunities (more than the previous case) for the user to look at the screen. So, in the adapted UI, more flexibility is allowed (e.g. by setting Equivalence to the input of selection/edit elements instead of Graphical Assignment).
- *Adaptation 4*: at first the UI was the same as in the 1st scenario; in addition, since the user is elderly, in the adapted UI the elements providing output were redundantly supported. As for user interaction, input is vocal, prompt and feedback are redundantly rendered. This was done to offer more interaction flexibility to users.
- *Adaptation 5*: the initial UI was a mono-modal (vocal) UI; then the environment became noisy, and the adapted UI was a multimodal one in which: i)the system output is graphically provided; ii) for interaction elements, the user input and the associated prompt was provided graphically, while the feedback was provided in a redundant manner (both graphically and vocally).

Ten volunteers (4 females) aged 20 to 44 (M = 31.2, SD = 6.5) participated in the study. 7 out of 10 had never used a multimodal (graphical + vocal) UI before, while the others used it in Web environments (e.g. Opera or NetFront), or tried multimodal UIs in some interactive installations. As for participants' educational level: 4 held a PhD, 2 a Master Degree, 3 a Bachelor and 1 a High school degree.

As for the tasks of the test, users had to search for a car using the Car Rental application presented before. Then, upon receiving the result of their search, they were

instructed to act as if they were unsatisfied with it and then they had to modify the search. Between the first and the second search the context changed, and the UI was adapted accordingly. Therefore, the second search was carried out by users by using an adapted UI.

The test started with a short presentation in which general information about the test was given to each subject by the moderator. After, each participant answered to general questions about their education, age and experience in using multimodal UIs. Most users were novices with multimodal UIs. Then, in order to get familiar with such UIs, each subject performed a set of practice tasks equivalent to those that would be presented to them in the main part of the experiment, e.g. fill in a form and then submit it by using a multimodal (graphical and vocal) user interface (different from the one used for the test). After this, each participant actually started the test. After experiencing each adaptation, users had to rank it according to a set of relevant criteria identified after reviewing the state of art on adaptive UIs: *Appropriateness*, whether the system selected an appropriate adaptation strategy; *Continuity*, to what extent it was easy for the user to continue the interaction after adaptation; *Transition*, to what extent the adaptation process allowed users to realise what was happening during adaptation; *Impact in decreasing interaction complexity*, to what extent a decrease in the interaction complexity was achieved; *Impact in increasing user satisfaction*, to what extent adaptation increased user satisfaction. Each criterion was ranked according to a 1-5 scale (1: the worst rating, 5: the best one).

All the users successfully completed the expected tasks. In terms of *appropriateness*, the best one was Adaptation 1 (M = 4.5, SD =0.7), the worst one was Adaptation 4 (M = 3.6, SD = 1.2). For Adaptation 2, users appreciated that the UI could switch to the only-vocal modality, given the supposed contextual situation of a user starting to walk fast. Nevertheless, considering the search task proposed in the test, participants appreciated the possibility of further analysing the search list (e.g. by comparing different results) also graphically, as it could be challenging doing that by just using the vocal modality. Thus, some users preferred to maintain the modality that better suites the task rather than the current context. Furthermore, after applying Adaptation 4, the user had to graphically select the field to complete, while the associated input was vocally expected. Even if users were fully aware of what Adaptation 4 would imply (since during the adaptation transition a message was given to users informing them of the changes occurring in the UI), that solution was not natural for the users: the moderator observed that after clicking the input field, the users started typing the value. A similar situation occurred for the vocal modality: if fields were automatically read by the TTS, the user replied vocally to the prompt. As for *continuity*, the best rate was for Adaptation 1 (M = 4.9, SD = 0.3), the worst one was for Adaptation 4 (M = 4, SD = 0.9). The latter one can be explained as before: mixing the modalities for selecting and completing a field was perceived as a factor interrupting the natural flow in which the users interact with the application before and after the adaptation. Regarding *transition*, Adaptation 1, Adaptation 2 and Adaptation 3 (M = 4.6, SD = 0.5) were rated the best, the other 2 were rated the worst, although they received quite positive scores: Adaptation 4 (M = 4.4, SD =0.5); Adaptation 5 (M=4.4, SD=1). In any case, users appreciated that the application showed a message before adapting its interactive behavior, to explain how the UI was going to change. When switching between modalities (e.g. from graphical to vocal), some users

complained that, if walking fast (as in Adaptation 2), they should also have received a vocal notification. An *overall rating* was calculated from the ratings received by each adaptation rule on the various dimensions. As a result, the best adaptation was Adaptation 5 (M = 4.4, SD = 0.7), the worst one was Adaptation 4 (M = 3.8, SD = 0.9). Users appreciated Adaptation 5 as it moved from an unfamiliar only-vocal UI to a more flexible, multimodal UI. Alongside, the only-vocal input provided by Adaptation 4 was found a bit limiting, especially when the vocal recognition support did not work satisfactorily. Overall, users appreciated adaptations. This was positive because the majority of them had never used a graphical + vocal UI before the test.

9 Lessons Learnt

From the evaluation test we derived some more general findings about how to handle multimodality in adaptive applications and in particular how to select appropriate modality allocations when the UI adapts to a context change. One of the lessons learnt was that the selection of a particular modality should not be automatically fixed depending on e.g. the current context of use, but users should still have control over the adaptation process. In our test the users, while acknowledging the appropriateness of using particular modalities in specific contexts (i.e., in noisy environments the graphical modality is more effective than the vocal one), also highlighted the importance to maintain the control over the adaptation process to some extent (e.g. by selecting a "preferred" interaction modality and/or by directly controlling the adaptation strategy). Therefore, flexible environments enabling users to have some control over the performed adaptations are advisable. Moreover, it came out that when adapting the UI, the selection of the most appropriate modalities should also take into account the tasks to support, beyond the current context of use. As an example, the result of a search should also be graphically rendered in order to have an overall picture of the gathered data and then enable the user to perform on those data further refinements, filtering and comparisons in an effective manner. Rendering the result of a search only in a vocal manner could not support the same tasks in the same effective way.

Another lesson was that users tend to see the level of single UI elements as the finest UI granularity for assigning different interaction modalities. Attempts to go further in depth with finer adaptations might cause confusion. In our experiment we noticed that mixing modalities at the granularity of *parts* of single UI elements was not always appropriate. The case of a text field graphically selected and vocally filled in was illustrative: after graphically selecting one UI element, users expected to use the same (graphical) modality to interact with that UI element, and they were a bit puzzled when this did not happen.

10 Conclusions and Future Work

In this paper we have presented a novel solution for providing adaptive multimodal applications in Web environments, and report on an example application and a first user test. The new generator of multimodal Web applications is able to exploit current browsers (Chrome) and will soon be publicly available.

We have also reported on a first user test that provided some useful feedback about how users perceive the multimodal adaptation. We plan further empirical validation for better investigating these aspects and how they impact the user experience.

Future work will be dedicated to exploiting more effective and sophisticated mechanisms to increase intelligence in processing adaptation rules. In addition, we plan to carry out further assessment of our approach also involving designers/developers.

Acknowledgements. This work has been partly supported by the EU SERENOA STREP project (EU ICT FP7-ICT N.258030).

References

1. Avouac, P.-A., Lalanda, P., Nigay, L.: Autonomic management of multimodal interaction: DynaMo in action. In: Proceedings EICS 2012, Copenaghen, pp. 35–44. ACM Press (June 2012)
2. Bongartz, S., Jin, Y., Paternò, F., Rett, J., Santoro, C., Spano, L.D.: Adaptive User Interfaces for Smart Environments with the Support of Model-Based Languages. In: Paternò, F., de Ruyter, B., Markopoulos, P., Santoro, C., van Loenen, E., Luyten, K. (eds.) AmI 2012. LNCS, vol. 7683, pp. 33–48. Springer, Heidelberg (2012)
3. Celentano, A., Gaggi, O.: Context-aware design of adaptable multimodal documents. Multimedia Tools Appl. 29(1), 7–28 (2006)
4. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R.: Four Easy Pieces for Assessing the Usability of Multimodal Interaction: the CARE Properties. In: Proceedings INTERACT 1995, pp. 115–120 (1995)
5. Duarte, C., Carriço, L.: A conceptual framework for developing adaptive multimodal applications. In: Proceedings of IUI 2006, pp. 132–139. ACM Press (2006)
6. Ertl, D.: Semi-automatic multimodal user interface generation. In: Proceedings EICS 2009, pp. 321–324. ACM Press (July 2009)
7. Manca, M., Paternò, F.: Supporting Multimodality in Service-Oriented Model-Based Development Environments. In: Forbrig, P. (ed.) HCSE 2010. LNCS, vol. 6409, pp. 135–148. Springer, Heidelberg (2010)
8. Nebeling, M., Speicher, M., Norrie, M.C.: W3Touch: Metrics-based Web Content Adaptation for Touch. To Appear in Proceedings CHI 2013, Paris (2013)
9. Octavia, J.R., Vanacken, L., Raymaekers, C., Coninx, K., Flerackers, E.: Facilitating adaptation in virtual environments using a context-aware model-based design process. In: England, D., Palanque, P., Vanderdonckt, J., Wild, P.J. (eds.) TAMODIA 2009. LNCS, vol. 5963, pp. 58–71. Springer, Heidelberg (2010)
10. Paternò, Santoro, C., Spano, L.D.: MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment. ACM Transactions on Computer-Human Interaction 16(4), 19:1-19:30 (2009)
11. Peissner, M., Häbe, D., Janssen, D., Sellner, T.: MyUI: generating accessible user interfaces from multimodal design patterns. In: Proceedings EICS 2012, pp. 81–90. ACM Press (2012)
12. Sottet, J.-S., Ganneau, V., Calvary, G., Demeure, A., Favre, J.-M., Demumieux, R.: Model-driven adaptation for plastic user interfaces. In: Baranauskas, C., Abascal, J., Barbosa, S.D.J. (eds.) INTERACT 2007. LNCS, vol. 4662, pp. 397–410. Springer, Heidelberg (2007)
13. Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., Simarro, F.: A transformational approach for multimodal web user interfaces based on UsiXML. In: ICMI 2005, pp. 259–266 (2005)