MMIR Framework: Multimodal Mobile Interaction and Rendering

Aaron Ruß

Intelligent User Interfaces
DFKI (German Research Center for Artificial Intelligence) GmbH
Project Office, Alt-Moabit 91c
10559 Berlin
aaron.russ@dfki.de

Abstract: In this article, we present the MMIR (Multimodal Mobile Interaction and Rendering) framework that is targeted at building lightweight dialog systems. The framework is geared towards multimodal interaction development for applications where only limited resources are available, e.g. so that they can run self-contained even on mobile devices. The framework utilizes SCXML for modeling application-specific states and state transitions as well as for processing input modality events.

1 Introduction

Over the last decade, software applications that use modalities, which formerly were considered *exotic*, have now become commonplace. Especially the dissemination of smartphones has propagated the concepts of speech and gesture interactions into the wider user base. Many modern smartphones have built-in support for employing modalities aside from the classical mouse/touch and keyboard input methods as well as the standard graphical screen and sound output. Nowadays, the average smartphone features geo-location-, acceleration sensors, vibration feedback, touch screen, video camera, microphone, and loud speakers.

At the same time, smartphones bring additional restrictions to classical GUI (Graphical User Interface) interactions due to their small screen size in combination with their standard input method touch-by-finger: on the one hand, the amount of displayed information is restricted, while on the other hand, the use of finger touch-input requires control elements to have a minimal, appropriate size to allow touch interaction.

A classical argument for offering multimodal solutions is that they can help improve user experience by offering more natural methods of interaction. In the mobile context, this argument may be even more valid with regard to the restrictions of mobile devices; if multimodality cannot help completely overcoming these restrictions, well designed multimodal interactions can at least help facilitating the perceived constrictions and reduced usability that are imposed by mobile devices.

Ruß (2013) MMIR Framework: *Multimodal Mobile Interaction and Rendering*. In Horbach, Matthias (Ed.): *INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt*, Köllen Druck+Verlag GmbH, Bonn, 2013, GI Lecture Notes in Informatics (LNI) P-220, 2702-2713

Moreover, the rich set of sensors and feedback-mechanisms that are common features in most smartphones supply an excellent starting point for multimodal interaction development. But realizing multimodal applications remains a complex task, that often requires expert knowledge in multiple areas. One approach for alleviating the requirement of detailed implementation knowledge and for furthering re-usability of developed components is by usage of model-based techniques for multimodal interaction design.

In this paper, we introduce the MMIR (Multimodal Mobile Interaction and Rendering) framework. MMIR is a lightweight framework for building dialog systems on the basis of web technology. The development is work in progress. The following sections describe the goals and current status of the framework's implementation. The MMIR framework is available as Open Source project on GitHub at http://github.com/mmig/mmir-starter-kit.

Using web technology opens up multiple ways for designing and developing applications in a model-based fashion, integrating the model-based approach to various degrees. Currently the MMIR framework utilizes SCXML (State Chart XML) [W3C12b] – albeit on a rather low abstraction level – for modeling the application state and the event-triggered state changes, as well as the mapping of multimodal input events to application-specific events (*modal fusion*).

While the usage of handheld devices in dialog systems for multimodal solutions has a long tradition, they usually play the role of "thin clients". Our framework aims to run self-contained on mobile devices, without the requirement of using external web services. Implementing as much functionality with standardized APIs, as the current technology allows (e.g. HTML5 APIs), the frameworks aims to lessen the burden on users, requiring as little installation of additional external device drivers or software as possible.

Recent HTML5 specifications define access to system resources that were previously only accessible by browser specific plugins or external solutions. These new APIs allow implementation of multimodal functionality directly in HTML5-capable interpreters. While far from complete, the implementation of these new APIs by the major browser engines is noticeably progressing; for instance, APIs for accessing the users· microphone and camera are available current Chrome-based, a Firefox-based developer versions. The Apache Cordova[1] project gives access to HTML5 functionality for a multitude of mobile platforms.

For the current implementation of the MMIR framework, functionality that, as of now, is not yet available as HTML5 technology, is implemented and made available via web services or native plugins (e.g. through Cordova on the Android OS).

[1] http://cordova.apache.org/

2 Related Work

Most existing dialog systems for multimodal interactions follow a client-server architecture; the implementations of the server component have a high demand on resources (disc space, processing power/time, working memory) so that they cannot be run on current mobile devices. The clients are usually very "thin" in that they basically only stream the input-events to the server, and display/*sound out* the results; the heavy lifting of processing, the multimodal fusion and fission is done on the server side. For instance, ODP (Ontology-based Dialogue Platform, e.g. [SNSH09]) and Olympus/RavenClaw [BR09] are prominent examples of these kinds of systems.

The MINT (Multimodal INTeraction) framework [FP11] is also server-based, and employs SCXML for its multimodal interaction processing. In MINT the SCXML documents are used to formally specify the states and functionality of input-/output-interactors at different abstraction levels as well as UI-elements; the interactors and UI-elements are then connected by mappings defined by a custom XML schema. As an example: an SCXML definition specifies the interactor "mouse", modeling its states and functions; for instance, if its buttons are pressed or not, or if it is moving or not. This model is then connected with a mapping-XML to a UI-list-element, by specifying that the left mouse button corresponds to the function-call "next-element" of the UI-list-element.

Less server-centric than the previously described approaches is the WAMI (Web-Accessible Multimodal Interfaces) framework [GMB08]: it is still server-based, but most of the framework is designed to run on the client side. Similarly to our framework, WAMI is based on Web Technology for presenting the system's output to the user, relying mostly on HTML and Javascript for realizing the UI (User Interface) and XML (eXtendible Markup Language) for message exchange; JSGF (Java Speech Grammar Format) is used for specifying the speech grammar used in the server-based speech recognizer component of the framework.

While multimodal applications on mobile devices have a comparatively long tradition in research systems, up to now, there are very few commercial applications on mobile devices that are "truly" multimodal. The commercial search application Speak4it [JE11] allows multimodal input by touch (e.g. drawing a line-path on a map) and speech for initiating location-based queries. The application uses a server-based architecture for processing its multimodal input and output.

Cutuagno et al. [CLMR12] present a dialog system for multimodal interactions that runs on the client side. A reduced set of SMUIML [DLI10] features are used to specify the multimodal interactions. The protype implementation offers similar functionality the Speak4it application and is implemented as a native App on the Android OS (Operating System). Some processing- and data-intensive parts, mainly the database for location-based queries, are realized as server-based resources.

3 The MMIR Framework

The MMIR framework targets development of lightweight dialog systems. The goal is to implement a framework with web technologies, so that it allows applications to run on any HTML5-capabale interpreter (e.g. a browser on a smartphone), without explicitly requiring an external server-component. At the current state, the support of HTML5 by the major browser vendors is less than complete, but many relevant features for implementing multimodal interactions are supported by at least by one engine, e.g. the *getUserMedia* API implemented by recent versions of the Chrome browser offers access to the microphone resource – the WAMI framework in [GMB08] still required a Java applet for integrating this input resource.

The main idea for the MMIR framework is, to provide components that help building lightweight multimodal applications, without introducing too much additional complexity – in comparison to classical GUI applications – into the development process.

While the MMIR framework is based on the W3C's Multimodal Interaction Framework [W3C03], it does not strictly adhere to the W3C specification. Specifically, the MMIR framework is, at least in its current state, not targeted to support distributed device scenarios (see e.g. chapter 8 in [W3C03]).

As we are describing work in progress, only few web standards are fully supported yet at the current state of the framework's development. For the future, we plan to use and implement additional web standards where feasible. For instance, we do not plan to use EMMA, but a similar, albeit simplified JSON-format instead. Since we want to keep the framework executable on mobile platforms we need to strike a balance between large standardized interfaces, and small efficient solutions. In this regard, we plan to further explore, to which degree we can implement and/or reuse existing standards and solutions for web-based multimodal interaction systems.

The current implementation for this framework focuses on integration on the Android platform (as mobile/smartphone environment) and Google's Chrome browser (as desktop/browser environment); the Chrome browser was selected as it is currently the only browser that allows access to the microphone via the HTML5 getUserMedia API [W3C12a]. For the Android environment, Cordova is used, which allows standardized access to a wide variety of modal resources on many mobile platforms. The implementation for the browser environment requires additional use of web services for speech recognition and synthesis.

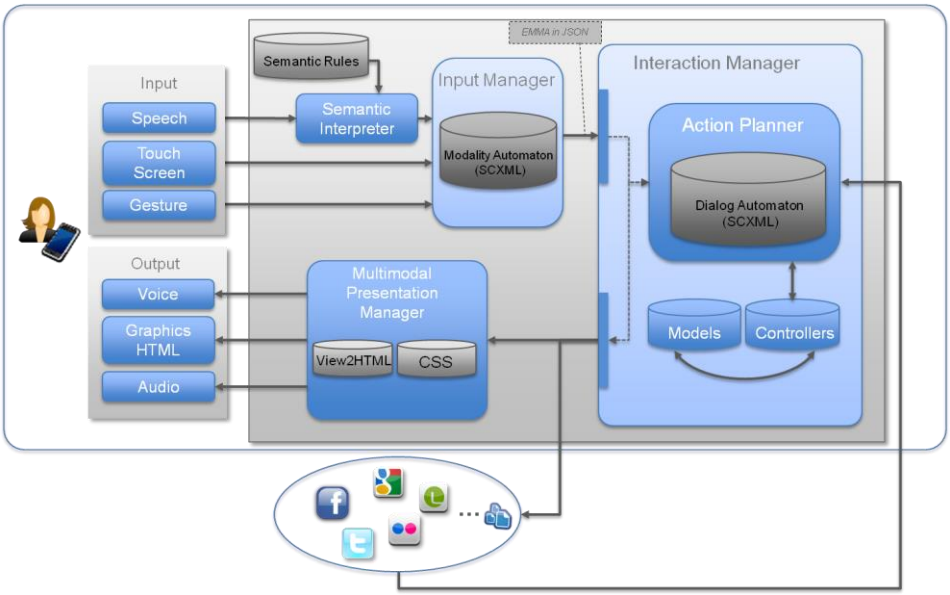

Figure 1: Overview of current state for the MMIR architecture

3.1 Dialog System

At the heart of the MMIR dialog system lie the SCXML-based Interaction Manager and Input Manager. SCXML-based in this context means that the application-specific behavior is defined by SCXML documents, in which *states* and *state transitions* are specified; transitions can have optional conditions and are triggered by events.

The Input Manager is responsible for gathering the "raw" events of different modalities and, if possible, merging and translating them to meaningful "application events"; for instance, a SPEECH_INPUT event "record_audio_comment" and a TOUCH_INPUT event "click_poi_hdk" could be (possibly based on an additional time constraint) merged into the event "record_audio_comment_poi_hdk", ie. *"record an audio comment for the point of interest HDK"*.

The Interaction Manager is responsible for keeping track of the application's current state and, on receiving events, determining appropriate actions (e.g. output) and state transitions.

Following the idea of a lightweight framework, the MMIR framework also allows to omit use of the Input and/or Interaction Manager. For instance, if the framework is used for a classical GUI application, one could omit the SCXML specification of the Input Manger and only supply a simplified SCXML specification that processes transitions based on "raw" touch input events.

Figure 2 visualizes a (simplified) SCXML document for modeling login/registration of a touch-base application: ellipses represent the states, the labels on the edges (arrows) specify the events and/or conditions that trigger a state change, and the additional annotations describe actions that are executed when the application changes into the corresponding state (*onentry*).

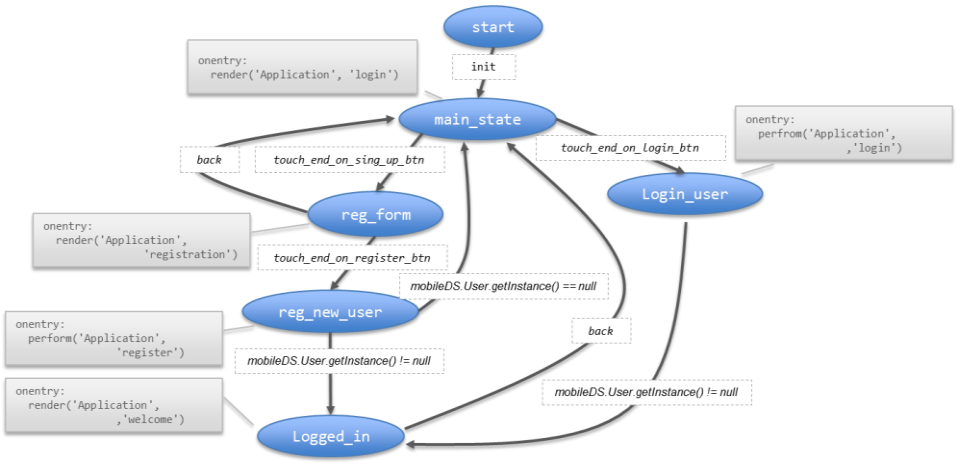

Figure 2: Graphical example for an application's state chart

The example in Figure 2 models the *login* procedure of the *Application* controller. After some general initialization (e.g. setting up components, network resources; this is not depicted in the state chart) the *init* event is raised and triggers the transistion from the (un-initilized) *start* state into the *main_state*. On entering the *main_state*, the view *login* of the controller *Application* is rendered (as specified by the *onentry* annotation). Not depicted in the state chart representation, this view would offer buttons to the user for either logging in (*touch_end_on_login_btn*) or registering as a new user (*touch_end_on_sign_up_btn*); pressing a button then triggers the corresponding event and subsequently the transition in the state machine. Some transitions are conditional, e.g. outgoing transitions for the state *reg_new_user* become active depending on the instance of the user-model object (*mobileDS.User*). The example specifies a function call in the *reg_new_user* state's *onentry* annotation; the annotation states, that a function call *register* on the controller *Application* will be performed when entering this state. The concrete function would be specified in the controller implementation and is again not depicted in Figure 2; for this example, we simply assume that the function call somehow modifies the user-model's instance (*mobileDS.User.getInstance()*) and depending on this, the next state transition is triggered. Note, that this is not a complete example for modeling the login, as, for instance, the state *login_user* does not treat the case, that erroneous login data was provided (e.g. by an additional transition that handles the condition *mobileDS.User.getInstance() == null*).

3.2 Graphic UI: View Templates

The GUI presentation layer is design similar to MVC-aware server-side frameworks, as for example ASP.net MVC and RubyOnRails. The MMIR framework provides a general separation mechanism for controllers, models, and views. Controllers and models are implemented via JavaScript.[2] Views can be specified via an extended HTML syntax (*templates*) that offers additional syntax for realizing modularization and integrating *dynamic* content; very similar to mechanisms in JSP, ASP, RubyOnRails etc.

For instance, the template syntax allows to specify sections with *@yield(name)* expressions; the content for sections via *@contentFor(name){…}@*); text-references with regard to internationalization tables/dictionaries using *@localize(name)*; and control constructs such as IF (*@if(condition){ … }@*) and FOR (*@for(loop-expression){ … }@*). Currently, the framework parses template definitions on application startup using a JavaScript-based parser that is generated with ANTlr[3]; for the future we plan further tool support for pre-compiling templates and syntax highlighting.

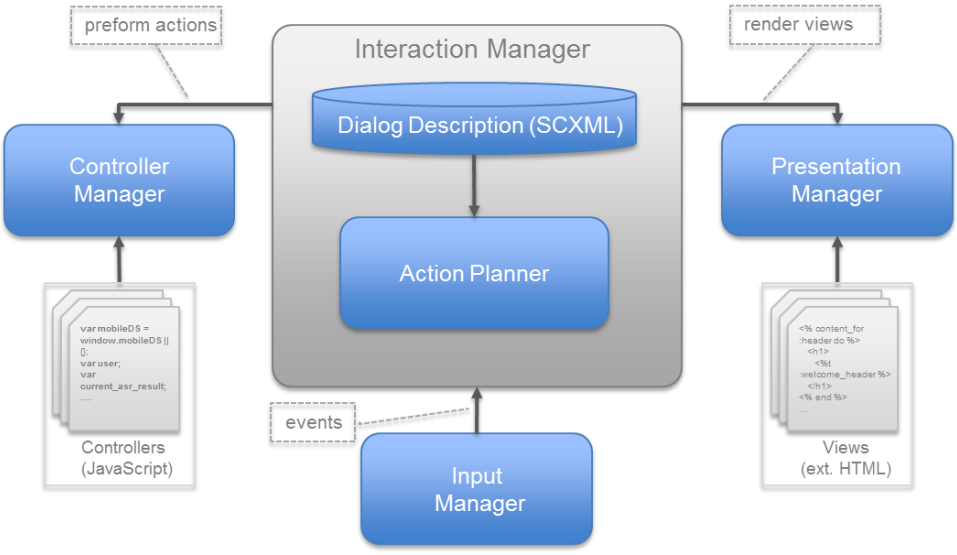

Figure 3: Overview for relationship between Interaction Manager, Controllers, and Views

Each controller implementation can have (i.e. *control*) one or more views. For modularization, three different types of views can be used: *layouts* provide a general skeleton that is reused for all rendered views of the same controller. Normal *views* are

[2] Here *models* mainly mean „MVC models“ in the sense of *data models*. As of yet, the MMIR framework does not impose any special restrictions on these kinds of models.
[3] ANTlr (version 3): ANother Tool for Language Recognition, see also http://www.antlr3.org/

rendered into the controller's *layout* definition (see Figure 4), while *partial views* are rendered "as-is", that is without the enveloping *layout* template. For instance, a *view* may use a *partial view* to render a *view*-specific header component; while the parts of the header that are static for all *views* of the controller are defined in the controller's *layout* definition.

Looking at the example in Figure 4: on rendering, the *yield*-sections in the layout-definition are replaced by the corresponding *contentFor*-definitions from the view; the layout in Figure 4 specifies three yield-sections (*header*, *content*, *footer*) while the view-definition supplies content for two of them (*contentFor("header")*, and *contentFor("content")*). The *@localize*-expressions refer to "dictionaries": for each supported language a dictionary-file must be supplied that contains variable-value pairs. Then, depending on the currently set language, the argument of the *@localize*-expression is looked up as variable in the corresponding dictionary, and replaced by its value.

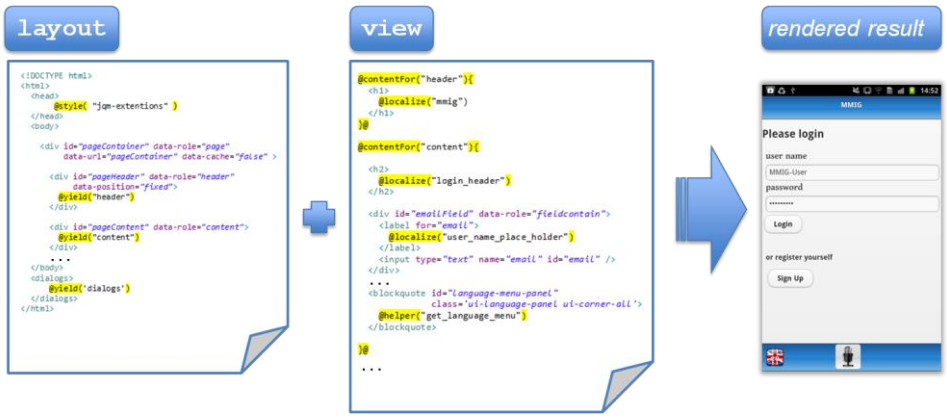

Figure 4: Example for rendering a layout and view template (result displayed on an Android smartphone)

Figure 5 shows a partial view that could be used a selection menu for setting a language. The example code for the partial view creates a list (<*ul*>) for all languages that are known to the application (*mobileDS.LanguageManager.getInstance().getLanguages()*). The list entries (<*li*>) are generated using a *@for*-expression. The current language setting influences the rendered result, e.g. the concrete value for the *@localize*-expression (the heading <*h2*>) is selected depending on the current language.

This partial view definition could then be used and re-used in multiple views for displaying a language selection dialog; the partial view is integrated into other views by inserting the expression *@render('Application,'languagemenu')*, assuming that the partial view is named *languagemenu* and specified for the controller *Application*.

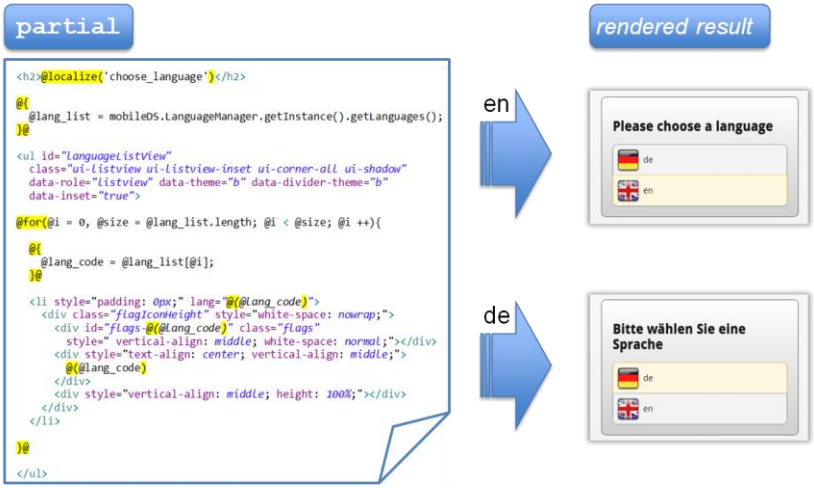

Figure 5: Example for a partial view that generates a list-view for language selection. The rendered result depends on the currently selected language (note that for brevity, the codes is slightly simplified; e.g. the code for highlighting the currently selected language is omitted).

3.3 Voice UI: Speech Recognition

A necessary requirement for speech input is the ability to capture audio input. HTML5 brings a general API for accessing the user's video/microphone (getUserMedia API). Currently, only recent versions of the Chrome browser offer an implementation for this API; but development on supporting this API for other major browsers is under work, as e.g. in the development of Firefox.

For the actual speech recognizer in the browser environment, we still need to use a web-service, as there is no implementation for such components purely written in JavaScript/HTML5. The situation for speech synthesis (TTS: text to speech) is similar in that currently existing implementations in JavaScript lack quality.

On mobile devices, the Cordova framework offers access to microphone resources on many mobile platforms. However, for speech recognizers and synthesis we also need to employ non-HTML5 resources, that is native plugins (on Android) or web service (in the generic browser environment). For our prototype implementation, we used the Google's speech recognition web service (using its undocumented, unofficial API) in combination with proxy web-service that performs necessary format conversations. For speech synthesis (text-to-speech) we used the web service of the open source project MARY (Modular Architecture for Research on speech sYnthesis) [ST03].

3.4 Voice UI: Grammar

After the ASR (Automatic Speech Recognition), the resulting textual representation of the speech input needs to be "interpreted". For this, the MMIR framework allows the

definition of an application-specific grammar. The grammar specifies how to parse a (transcribed) speech input, and then returns an interpretation of what it means in context of the application. For example, the sentence "turn the radio on" may be transformed into an attribute-value-pair "radio=on".

The grammar mechanism, as currently implemented, takes a grammar definition in a BNF-like JSON-format and generates an executable JavaScript grammar.

The JSON-grammar-format allows specifying (1) a stop-word list, (2) tokens, and (3) "sentence-rules", resulting in context free grammars. The stop word list specifies "noise words" that will be removed before an input-sentence will be parsed. The tokens represent a "word dictionary" where each token represents one or multiple "words". Finally, the sentence rules of the grammar (a) specify sequences of tokens and references to other sentence rules and (b) what they mean by generating data objects.

For example, a sentence rule with tokens IMP RADIO STATE may generate the JSON object *{radio: $state[0]}*, that is an object with a field "radio" and a value that is identified by the parsing result of the token STATE (e.g. "on", "off").

The executable JavaScript grammar is generated from the JSON-grammar definition using the JS/CC library[4]. JS/CC is itself implemented in JavaScript which allows dynamic grammar generation during runtime of an application.

4 Example Application

We used the MMIR framework in context of the Voice2Social project[5] for implementing a multimodal application with Android as target platform. As a basic concept of the Voice2Social project, information from social networks is enriched with audio/speech content. The implemented Voice2Social App allows browsing audio-annotated information on POIs (Points Of Interest; e.g. restaurants, bars, etc.) through a map interface. Users can listen to – as well as record – audio comments for POIs; in addition, users can listen to location-based audio streams that are generated based on recorded and annotated audio snippets. The MMIR framework was used to realize the GUI, as well as voice command-and-control, and ASR for transcribing the recorded audio comments; as example for gesture input, the lift-2-talk gesture was implemented (see [TKS+09]) that can be used instead of the GUI's push-to-talk button for activating voice commands. Access to external data sources, to social networks etc., and speech processing were implemented as native Android plugins, interfacing through the Cordova framework.

[4] http://jscc.phorward-software.com/
[5] http://voice2social.dfki.de/

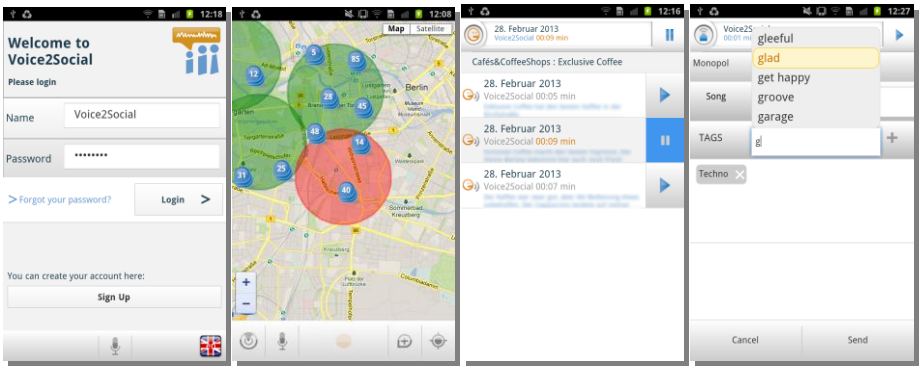

Figure 6: Screenshots of the MMIR-based Voice2Social App for annotating/accessing social network audio information

5 Outlook

In the previous sections, we describe the current state of the development of the MMIR framework. It is clear, that this is only a first step. Future work will need to explore the question, which existing standards for multimodal interaction development can be used for improving the interoperability of our framework.

Since our framework is targeted at mobile devices which usually are rather constricted with respect to processing power and memory resources, future work will need to explore the question, at which point – or which parts of – standard-conformance should be sacrificed in favor of efficiency.

Another interesting area is further integration of existing solutions for model-based development. For instance, the use of SMUIML analogous to [CLMR12] for specifying multimodal interactions more abstractly and more generically as is currently done.

For integrating model-driven methods to a higher degree, the MINT framework or parts of the framework could be used. For example, an interesting question would be, if the MMIR framework could function as a runtime environment for the MINT framework instead of its standard server-based one. Another question would concern the reuse of parts of the MINT framework, for instance, if the model-based specifications of the MINT framework for interactors, UI-elements and their mappings could be used as generalized fusion component in the MMIR framework.

6 References

[BR09] Dan Bohus and Alexander I Rudnicky. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23(3):332–361, 2009.

[CLMR12] Francesco Cutugno, VincenzaAnna Leano, Gianluca Mignini, and Roberto Rinaldi. Speech and Multimodal Interaction in Mobile GIS Search: A Case of Study. In Sergio Martino, Adriano Peron, and Taro Tezuka, editors, *Web and Wireless Geographical Information Systems*, volume 7236 of *Lecture Notes in Computer Science*, pages 27–32. Springer Berlin Heidelberg, 2012.

[DLI10] Bruno Dumas, Denis Lalanne, and Rolf Ingold. Description languages for multimodal interaction: a set of guidelines and its illustration with SMUIML. *Journal on multimodal user interfaces*, 3(3):237–247, 2010.

[FP11] Sebastian Feuerstack and Ednaldo Pizzolato. Building multimodal interfaces out of executable, model-based interactors and mappings. In *Human-Computer Interaction. Design and Development Approaches*, pages 221–228. Springer, 2011.

[GMB08] Alexander Gruenstein, Ian McGraw, and Ibrahim Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proceedings of the 10th international conference on Multimodal interfaces*, pages 141–148. ACM, 2008.

[JE11] Michael Johnston and Patrick Ehlen. Speak4it and the Multimodal Semantic Interpretation System. *Proceedings of Interspeech 2011*, pages 3333–3334, 2011.

[SNSH09] Daniel Sonntag, Robert Nesselrath, Gerhard Sonnenberg, and Gerd Herzog. Supporting a Rapid Dialogue System Engineering Process. *Proceedings of the 1st IWSDS*, 2009.

[ST03] Marc Schröder and Jürgen Trouvain. The German text-to-speech synthesis system MARY: A tool for research, development and teaching. *International Journal of Speech Technology*, 6(4):365–377, 2003.

[TKS+09] Markku Turunen, Aleksi Kallinen, Iván Sánchez, Jukka Riekki, Juho Hella, Thomas Olsson, Aleksi Melto, Juha-Pekka Rajaniemi, Jaakko Hakulinen, Erno Mäkinen, et al. Multimodal interaction with speech and physical touch interface in a media center application. In *Proceedings of the International Conference on Advances in Computer Enterntainment Technology*, pages 19–26. ACM, 2009.

[W3C03] W3C. W3C Multimodal Interaction Framework. http://www.w3.org/TR/mmi-framework/, May 2003. accessed 22-April-2013.

[W3C12a] W3C. Media Capture and Streams. http://www.w3.org/TR/mediacapture-streams/, 2012. accessed 22-April-2013.

[W3C12b] W3C. State Chart XML (SCXML): State Machine Notation for Control Abstraction. http://www.w3.org/TR/scxml/, 2012. accessed 22-April-2013.