AGORA: A GUI APPROACH TO MULTIMODAL USER INTERFACES

Manolis M. Tsangaris and Alexandros Potamianos

Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07074, U.S.A.

ABSTRACT

Multi-modal user interfaces bring the best of the two worlds to the user; the navigational aspect of graphical user interfaces and the declarative aspect of voice interfaces. The user can move around forms providing a concrete visual representation, while speech input allows natural form filling and form navigation. Unlike the unimodal speech interfaces, the visual interface efficiently reflects the state of the dialog as well as the speech recognition and understanding results. Agora uses a hierarchical form flow motivated from the traditional GUI design philosophy and is loosely based on VoiceXML. Agora also brings up several design issues including: ambiguity resolution, modality synchronization, focus representation, preferred modality for input and output.

1. INTRODUCTION

After several decades of speech processing research we have mature technology that is being used today in real world applications and services. More than two decades of work on graphical user interfaces has also enabled us to effectively design user friendly applications.

Mobile terminals present new interface design challenges that can be addressed by a combination of speech and visual interfaces [2, 7]. Limited screen real estate, the absence of keyboard, and their usage in unpredictable mobile environments, set new constraints. Thus, multimodal interfaces have emerged, as systems supporting speech input and output, as well as visual input and output.

A fundamental issue when designing multimodal systems is the choice, integration and appropriate mix of input and output modes in the user interface [6, 8]. Few guidelines exist for selecting the appropriate mix of modalities [2, 4], however, it is clear that a spoken language interface is not always the best choice. This is especially true for system output, where speech plays a secondary role to visual input (with the exception of hand-free applications). It is often the case when designing multimodal user interfaces, that the developer is biased either towards the voice or the visual modalities. This is especially true if the developer is voice-enabling an existing GUI-based application or building a GUI for an existing voice-only service. Our goal is to follow an approach that respects both modalities, creating an interface that is both *natural* and *efficient*. The first step towards the seamless integration of the input and output modalities is a *consistent* user interface: the GUI represents system state and possible actions (including state and actions that are specific to the voice-modality). A second important step towards a truly multimodal experience is creating a user interface that utilizes the *synergies* between the input and output modalities, e.g., using visual feedback from the recognizer (including possible ambiguities).

This paper, represents a new philosophy in building such interfaces; in our system, the visual interface offers a concise task oriented procedural style interface, while the speech part, facilitates easier input and navigation. No attempt is made to make the two modalities equivalent in terms of what they can do to the state of the system; the user can choose the appropriate modality for the given task. In this paper, we built on the extensive literature and experience on graphical user interfaces (GUIs) design and on voice user interfaces (VUIs) design. Web standards such as HTML and VoiceXML have also influenced our approach [10, 1].

The organization of the paper is as follows: First the system architecture is discussed. A modular client-server architecture is used; a thin client is run on the mobile device and the bulk of the processing is happening at the server side. In Section 3, the Agora Multimodal Service Language (AMSL) is introduced for building multimodal interfaces. AMSL defines the voice interface semantics, the visual form semantics and then binds the two semantic spaces; arbitrary TCL code can be executed before or after the form is loaded. The form flow model and the GUI approach to multimodal interface design, is described in Section 4. Next we present an example application from the travel reservation domain. Implementation issues for different parts of the application (form-filling, result presentation and navigation) are presented. Issues such as ambiguity resolution, focus changes and interface consistency between the two input modalities are also mentioned. We conclude in Section 6.

2. MULTIMODAL SYSTEM ARCHITECTURE

The multimodal interaction model defines the communication between two actors: the user and the (multimodal) terminal. The multimodal Agora terminal, is a device capable of

1. visual input (using pen, buttons and possibly keyboard),
2. speech input (via an attached or a lapel microphone),
3. visual output (using a color or black and white high resolution screen),
4. speech output (using a headset or the attached speaker).

We have implemented Agora, a prototype to study some of the multimodal user interface issues, as a three level architecture. The multimodal terminal (MMT) is a Compaq IPAQ running a thin client; it collects audio from its microphone, it plays audio, and displays a local graphical user interface using a TCL/TK engine. It connects to the Agora multimedia presentation server (MMPS) using either a wireless LAN (802.11) or a GPRS/GSM link. MMPS is a collection of Unix server processes including a GUI engine written in TCL, a Lucent ASR/TTS engine for performing speech recognition and synthesis, and an AMSL engine for fetching and interpreting multimodal scripts from the Service Back-end servers.

AMSL is a simple form language we have developed that can express all service logic (forms and spoken I/O) declaratively (using an XML like syntax) and procedurally (using TCL scripts).

3. AMSL: THE INTERFACE DEFINITION LANGUAGE

AMSL stands for the Agora Multimodal Service Language, and it is currently under development. AMSL is the UI engine part of the classic three tier service architecture. Like VoiceXML for voice [1], AMSL is a multimodal service definition language. But more like JavaSpeech and unlike VoiceXML, AMSL does not rely on external logic in order to interact with the user and handle visual/voice input/output. As it happens with graphical user interfaces, a service specific computational back-end provides the user interface engine with the proper data services, in order for the entire service to be implemented.

AMSL is partly based on VoiceXML, a standard followed by much of the spoken dialogue services community. AMSL adds multimodal support, inheritance, and full-scripting capabilities. We feel that the multimodal interface language needs to include both declarative syntactic structure (that could be expressed using XML) as well as procedural behavioral constructs. In the development of our system we heavily borrow concepts and practices from the WEB and (Voice)XML worlds, and similarities are not accidental.

Next, we illustrate Agora in a step by step process through our standard air travel reservation example.

The Agora user needs to download and install the Agora browser (AMUI) on the mobile terminal. When accessed for the first time, the user will be automatically registered to the Agora server; any appropriate transport (IP over WLAN or IP over GPRS) could be utilized, provided that it can support the necessary Quality of Service (i.e. adequate bit rate and delay).

The service author writes the AMSL code and stores it on one or more AMSL code pages on a standard HTTP server. For example:

```
http://air-travel.com/reserve.amsl
```

When the above URL is registered with the the Agora server it will appear next time a user logs in to the Agora server as new services to try. A general service discovery method has been suggested, but it is out of the scope of this paper. The user may decide to subscribe to the service, and in that case the corresponding icon will be available at the top level menu.

The file reserve.amsl contains TCL code defining the form(s) needed by this application:

```
form toplevel -title "Trips" {
  grammar {
    city    -> "athens" | "rome" | "milan" : {T $1}
    sfcity  -> "from"      <city> : {city $city.T}
    stcity  -> "to"        <city> : {city $city.T}
    sdepart -> "departing" <date> : {date $date.T}
  }
  fields {
    departure.city      "From" city
    arrival.city          "To" city
    departure.date "Departing" date
  }
  bind {
    sfcity  -> departure.city
    stcity  -> arrival.city
    sdepart -> departure.date
  }
  # -- insert user specific TCL code here
  code {

  }
```

The form encapsulates grammars, fields, event bindings, and TCL code:

- The *grammar* is expressed as BNF definitions with tag extensions, as it is the case within the speech community. In the example, a city is simply defined as either "athens" or "rome" or "milan", and once parsed, it will generate an event tagged as `city` and the name of the city. For example, when the user says: "from athens" the following tagged event record will be generated:
  `event grammar sfcity city "athens"`
- The form may contain *fields*, which is currently a list of slots with tag, name, and type. Fields will be intelligently laid on screen automatically, but that could be overridden by the user.
- *Binding* enables speech events to be associated to form actions. For example, if the user says "from athens" the generated event (`sfcity`) will be bound to the "city" field and the value of city will be assigned to the field "departure.city". If the user says "from athens to rome" two events will be generated:
  `event grammar sfcity city "athens"`
  `event grammar stcity city "rome"`
  causing two assignments to the corresponding from and to fields. Any events that do not match the binding rules are currently rejected.
- Arbitrary TCL *code* can be executed before or after a form is loaded, and event handling could be passed to the code as well.

A major design decision was to use a flat name space for forms, and to make URLs have nothing to do with naming. As a result, an application or a service can be conveniently stored on one or more AMSL pages and it will be incrementally loaded as needed using the standard HTTP protocol. This was mostly motivated by the Java Applets architecture [REFERENCE?].

On the client side, each different service will be given its own "thread of control", and will run in its own protected namespace; there is no direct possibility to interact with other services that happen to be active on the user terminal.

4. FORM FLOW

As a GUI approach to multimodal interfaces, AMSL is heavily based on the GUI form flow model. Forms structure the user interaction, making each step simpler and more effective. At each step, only one form is active; visual and speech events are mostly processed by the active form. The user navigates the form hierarchy, visually or via speech, and the visual context concisely represents the form hierarchy.

Forms structure the user interaction, making each step simpler and more effective. Each form represents a logical step of the user-machine interaction, used for gathering information (input fields/form filling), for displaying information (output fields/results), for controlling other forms, or for a combination of the three. In the example above, the first one is a control form, the subsequent forms are mostly input forms, and the last one is an output only form.

A form may define and activate one or more sub-forms; at each step, only one form is active, visual and speech events are mostly processed by the active form. The child form is given the visual and speech focus, which remains there until the form explicitly finishes. During this time, the user cannot access any other form visually or via speech. This restriction allows the service designer to structure the user interaction as a sequence or a hierarchy of simpler steps.

As the user navigates the form hierarchy, visually or via speech, the visual context concisely represents the form hierarchy. When the form is completed, it is removed from the display and its parent form receives the result; that is, a standard (TCL) exception, or a set of tagged values (exactly as the grammar does when parsing succeeds). Those values can be used by the parent of the form, thus, making the form a unit of modularity and re-usability.

Forms are object oriented, and as such, their behavior is a affected by inheritance. All forms inherit by default from the `baseform`, which defines many standard aspects of the form behavior. For example, form cancellation or confirmation are properties of the `baseform`. Form objects also encapsulate the procedural behavior, which can be hidden or exposed at the interface level.

5. AN EXAMPLE: AIR TRAVEL RESERVATION

Next, we will illustrate the proposed model, using a sequence of examples from the classic application of air travel reservation. The sequence of forms is shown in Fig. 1.

5.1. Simple Forms

As the user enters the service (say by dialing the service or clicking a button, or even automatically as the terminal is switched on), a top level screen is displayed. In our case, the screen is click sensitive, so "Trips", "Weather" or "Taxis" can be selected visually; a reasonable spoken command is also accepted.

Notice here one of the most important aspects of multimodal interfaces. The visual part can serve as a cue card reminding the user of possible choices as well as voice commands to make those choices. The visual part will also highlight the choice before an action is taken, thus serving as a precise feedback mechanism.

5.2. Forms with fields

As soon as the "Trips" choice is made, the user is transferred to the reservation screen. That screen also exploits the cue-card philosophy, inviting the user to explore it both phonetically and visually.

The user can select the departing and arriving cities in a variety of ways. Sentences like: "from athens", "from athens to rome", "from athens greece to rome italy" are all accepted. As soon as a choice is spoken and it is accurately recognized, the form will be updated to indicate the choice made.

5.3. Subordinate Forms

Since the departure time was not specified precisely, the system brought up a calendar form in order to help the user make a selection easier. The calendar can accept visual choices of specific dates, it can "scroll" through the year, and can accept speech commands like: "august sixth", "next month please", "september" etc. In response to those commands, the calendar reacts, and when appropriate, it highlights the user's choice. The form will not go away, until the choice is committed visually or via speech.

The form is cascaded (not shown in Fig. 1), as a standard way to indicate context. If the user touches the parent form, or if "back" is said, the calendar choice will be canceled and the parent form will become active again.

5.4. Rendering the Results

Once the input form is filled a query is made to the database back-end which may return flight results. In our case, the user specified flight schedule produced three different ticketing possibilities. As the page is being displayed, a brief TTS synthesized sentence informs us about the aggregated result and also reconfirms our query: "there are three flights from athens to rome". We will not attempt to read the entire page to the user. The spoken sentence is just a feedback mechanism and a summary of the result. The flight query results form can be further rendered by the user via speech or pen input.

5.5. Navigating forms

If the user now wants to return to the reservation form, it can readily do so visually by clicking on the cascaded par-

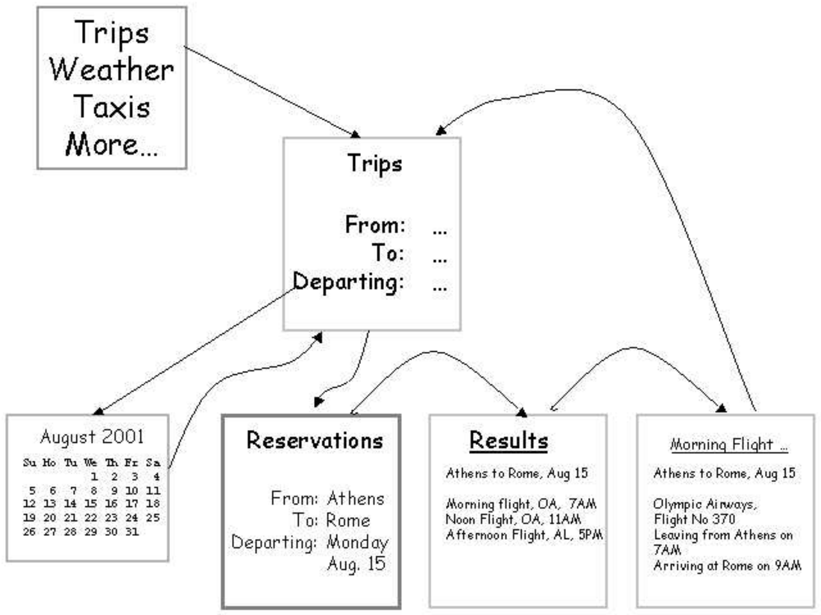

Figure 1: Form flow for the air travel reservation example.

ent form or by simply saying: “Reservations again please” or simply “Reservations”. A spoken sentence confirms the return and the form is now ready to be changed. Indeed, the user now could say: “to Milan” thus, changing the destination city. The form is updated to reflect the correction.

5.6. Handling Ambiguity

Ambiguity is probably the major difference between GUIs and VUIs (Voice Interfaces) and Agora does not encourage semantic ambiguities.

In general, GUIs have precise and clean semantics, and there are no major ambiguity issues. User actions are very specific to the graphical widget, leaving no room for further interpretation; very often, this is achieved at the cost of increased user effort. For example, a typical form filling application requires different physical fields, one for each input item, like destination and departure city, and departure date. It is very rare to allow a free text form interface, accepting a sentence like:

“travel from Athens to Rome tomorrow night”

In the case of VUI ambiguity could result from recognition errors or from loose semantic context or imprecise user phrase; saying ”Athens” in the above example raises the ambiguity of which field Athens was intended to.

Some multimodal systems have proposed specific “click to speak” gesture to resolve those issues; click to speak on a destination field could make obvious that Athens is a destination city for example. We are not in favor of such capability, simply because the user effort required to perform the correct click to speak gesture is simply too much for the achieved gain (of not saying departing from .. athens). We believe that the visible form and the context it implies, in addition to the names of the fields provide adequate context for a natural dialogue.

Some forms of ambiguity could be handled on the event mapping phase; in anticipation of such ambiguity the form author could write a new binding to much ”city” only phrases, and then post a choice subdialog so the user can resolve the ambiguity. Currently, the default action is that the phrase will be ignored with a brief audio gesture (a feedback rejection tone).

5.7. Preferred input modalities

As we have indicated previously, we do not want to go the extra mile to make both modalities fully equivalent to what they can express. This will create a cumbersome interface which will combine too many complexities from both modalities.

The issue is more evident on the calendar form posted when a date needs to be selected; we make no attempt not interpret all possible voice commands to express a date, but rather the ones to navigate the calendar and select a currently visible date. We hope that the GUI will offer enough visual cues to guide the user on the style of language accepted as input.

As a result, the user can say “1”, “13” or any number to select a day; “February”, “March”, or the name of a month to go to any month; and “Next month”, “Last Month” to move relative to the currently displayed month. Although it is possible to click on the arrows, or to click on the given date, real estate limitations make it quite tricky. In this case as in many other situations, voice is the preferred in-

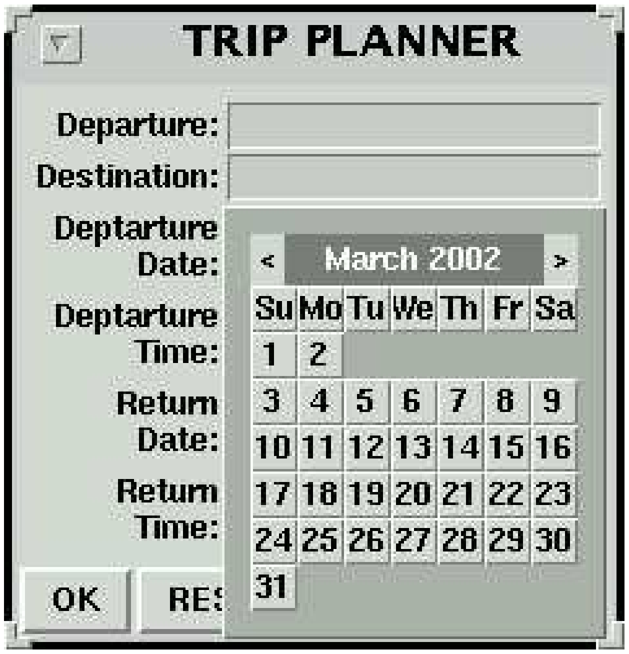

Figure 2: Calendar navigation form.

put modality, whereas the display is definitely the preferred output modality.

The exact level of priority between the two modalities is subject of further research and more experimentation.

5.8. Handling Focus

Agora forms are mostly visual and the focus semantics have been adopted to the philosophy of GUIs and PDAs.

When it comes to interactions, only one form at the time "talks" to the user. Other forms are partially visible, just to remind us the context and the tasks that have been suspended temporarily, until the current forms have been filled.

Unlike windows systems, and much like the PDAs, Agora does not allow multiple services to share the screen at the same time. Once the user switches to a different service, the former service forms disappear from the screen. As a result, inter-service focus switching is very easy.

The only exceptions is that some voice commands allow the focus to move to a different service temporarily suspending the current service; or to return to a previously posted toplevel form, thus, canceling all subordinate forms. In the case of the travel reservations example, the user could say "go to taxis" at any point during the navigation of travel forms to cancel any reservation work and explore the "taxis" options.

The Agora system is fully capable of alternative focus policies, but this is also subject of further research.

6. CONCLUSIONS

We have presented Agora, a simple framework for defining and building multimodal applications, where the visual part dictates the flow and speech is mostly used as a convenient mechanism for input. A simple interface creation language (AMSL) is defined that explicitly defines the voice interface and visual interface semantics, along with possible binding between the two. This approach has been demonstrated in a working multimodal travel reservation prototype. Further work is required to evaluate this approach and compare it with existing descriptive and procedural multimodal interface languages. We have tried to define a simple, consistent, and intuitive paradigm, which once accepted by the user it will generate the same amount of familiarity and productivity, GUI constructs like pull-down menus have achieved.

7. REFERENCES

[1] R. Beasley et al, *Voice Application with VoiceXML*, Sams, 2001.

[2] N.O. Bernsen and L. Dybkjaer, "Is speech the right thing for your application?," in *Internat. Conf. Speech Language Processing*, (Australia), Oct. 1998.

[3] P. Cohen, M. Johnston, D. M. and S. Oviatt, J. Clow, and J. Smith, "The efficiency of multimodal interaction: A case study," in *Proc. ICSLP 98*, (Sydney, Australia), 1998.

[4] V. Bilici, E. Krahmer, S. te Riele, R. Veldhuis, "Preferred Modalities in Dialogue Systems," in *Proc. ICSLP'2000*, (Beijing, China), Oct. 2000.

[5] X. Huang et al,"MIPAD: a next generation PDA prototype," in *Proc. ICSLP'2000*, (Beijing, China), Oct. 2000.

[6] S. Narayanan, A. Potamianos, and H. Wang, "Multimodal systems for children: Building a prototype," in *Proc. European Conf. on Speech Communication and Technology*, (Budapest, Hungary), Sept. 1999.

[7] S.L. Oviatt et al, "Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions," Human Computer Interaction, 2000, vol. 15, no. 4, 263-322.

[8] A. Potamianos et al, "Design principles and tools for multimodal dialog systems," in *Proc. ESCA Workshop Interact. Dialog. Multi-Modal Syst.*, (Kloster Irsee, Germany), June 1999.

[9] R. Sharma, V. Pavlovic, and T. Huang, "Toward multimodal human computer interface," *Proceedings of the IEEE*, vol. 86, no. 5, pp. 853–869, 1998.

[10] XHTML, *http://www.w3c.org/MarkUp/*