UNIFYING CONVERSATIONAL MULTIMEDIA INTERFACES FOR ACCESSING NETWORK SERVICES ACROSS COMMUNICATION DEVICES

G. Di Fabbrizio, S. Narayanan, P. Ruscitti, C. Kamm, B. Buntschuh, J. Hubbell, J. Wright, J. Hamaker
AT&T Labs – Research
180 Park Avenue, Florham Park, NJ, 07932
{pino,shri,ruscitti,cak,bb,jhubbell,jwright}@research.att.com, janna@cs.msstate.edu

ABSTRACT

This paper investigates architecture and interface design issues in engineering conversational multimedia interfaces for accessing network-based services. In particular, we focus on the problem of providing *conversation-capable* multimodal, multimedia services that can be accessed from a variety of communication devices and environments (telephones, PDAs, desktops etc) with a "consistent" user experience. The goals are two-fold: (i) provisioning a standards-compliant architecture and (ii) designing interfaces that are adaptive to the access device characteristics. We use a directory assistance application that can be accessed from a telephone, kiosk, desktop or a PDA to illustrate architecture and interface design issues.

1. INTRODUCTION

The recent revolution in the computer and communications industry aims at providing ubiquitous multimedia communication services *anywhere* and at *anytime*. There is a rapid evolution and integration, of the public switched telephone network toward IP-based communications networks that can support voice, video and data in a reliable, affordable and seamless manner. Concurrent with these developments is the availability of a variety of terminal devices with varying levels of audio-visual capabilities and communication throughputs. For instance, if we consider the number of multimedia communication/computation devices that were introduced in the market (for example, at the recent COMDEX, Las Vegas Oct 1999), it becomes clear that the number of access options available to an end user is mind boggling. There are several scenarios wherein the user accesses the same set of services from across a variety of devices. For example, let us consider messaging: the user accessing messages (voicemail, email, fax etc.) from telephones, PDAs, and desktop workstations, all either over wired or wireless connections. It is this general problem of designing consistent and seamless interfaces to multidevice, multimedia services that we address in this paper.

Voice access as a part of the multimedia user experience is believed to add naturalness, and in some cases, efficiency to human-computer interactions. In this paper, we will specifically focus on interfaces that include the capability of spoken inter actions. A crucial part of investigating conversational multimedia interfaces is the concomitant design and implementation of interface strategies and the underlying architecture. Another goal is to evaluate and quantify performance and usability measures as a function of interface strategies and access-device types. To achieve these goals, we will employ a case study of a directory retrieval application that can be accessed from a telephone, a public kiosk, the desktop and a PDA-like device.

2. SYSTEM ARCHITECTURE

Computer Telephony (CT) and IP Telephony (IPT) standards are continuously evolving towards flexible, scalable, distributed architectures to support industry demands and new innovative technologies. Over the last two years, the Enterprise Computer Telephony Forum (ECTF) [6] has played a fundamental role defining standard interoperability agreements. The computer telephony industry has in large part endorsed the ECTF's output, including its CT Bus interface specification, Service Provider Interface (SPI) definition, Application Programming Interface (API) specification, and others. This effort has resulted in the definition of a client-server open middleware environment that supports resource-location independence, distributed-system architecture, media processing, and call control.

However, this extended CT paradigm does not provide adequate support for interactive conversational systems that require automatic speech recognition (ASR) for large vocabulary applications, spoken language understanding, and, in general, multimodal user interfaces. Although some attempts have been made in this direction, the debate is ongoing. Among the questions being discussed are the following:

- What are the essential architectural components for supporting a mixed-initiative dialogue system?
- What level of interface should be defined?
- Is the dialogue manager part of the system resources or does it operate at the application level?

This section addresses these questions through a case study of an application framework based on CT and IPT standards. We "expand" the current paradigm, defining new architectural elements for information access (e.g. messaging, directory look-up), alerting, and multimodal input/output integration, in order to facilitate the design, development and deployment of advanced voice-enabled services. Next, we illustrate how to separate the application logic from low-level resource management, and discuss the steps needed to integrate a generic dialogue manager in such an environment. We then present a generic multimodal interface for telephones, kiosks/desktops and handheld devices that extends user inputs to mouse clicks, speech, touch, and typed text. Output modes include speech, text, graphics, gestures and animations. Finally, we describe a speech-enabled IP telephony directory query application with multimodal access. This, we hope, will set the stage for investigating the interplay between device-dependent dialog strategy selection and usability measures to learn about optimal interface design strategies.

2.1 ARCHITECTURAL COMPONENTS

The application framework described in this paper was designed to satisfy the following goals:(1) providing access to the service from different devices; (2) providing a set of user interfaces that maximize the efficiency and usability of the modalities offered by various devices (e.g., telephones, desktop PCs, PDAs) while providing consistent functionality; (3) effectively synchronizing and coordinating the system resources. Solicited and unsolicited events may be generated from both the system resource and the user's inputs. All the events need to be time-stamped and hierarchically prioritized to allow the dialogue manager to take the appropriate action; (4) facilitating the integration of the dialogue manager with system resources by providing a platform independent declarative language to represent the dialogue manager's actions; (5) adopting standards-based solutions, where possible, for CT, IPT, messaging, and directory lookup.

These goals influence the architectural design of the application framework, which is shown in Figure 1. The grayed area is a high level representation of an ECTF compliant CT server. It defines several standard resource classes for performing media operations such as Text-To-Speech Synthesis (TTS), Automatic

Speech Recognition (ASR), Signal Detector/Generator (SD/SG), prompt Player and Recorder.

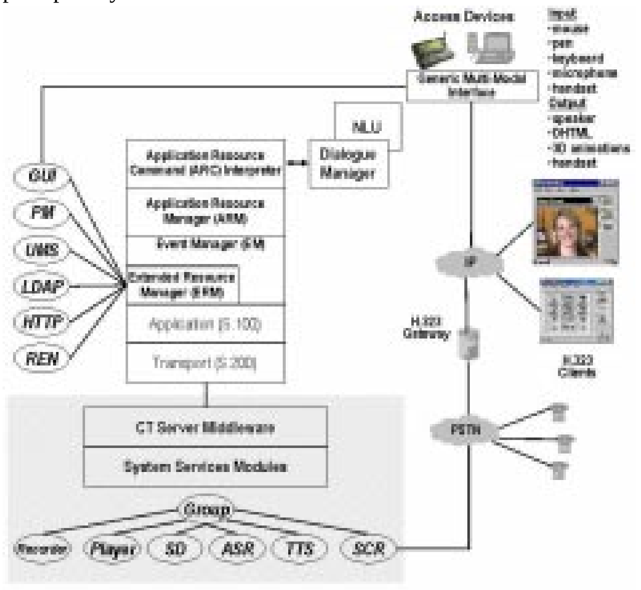


Figure 1. System Architecture

The System Call Router (SCR) represents a unified interface for call control and resources management. The upper layers (S.100 and above in Figure 1) are the client side of the application. The application operates by requesting a specific group of resources from the CT server in order to perform media processing on a call. Once the server grants the resources, the application can answer or make phone calls over either the traditional circuit-switched network or the IP-based packet network through ITU-H.323 endpoints [7]. We adopted a commercial ECTF compliant CT server as the service platform prototype and we integrated AT&T's Watson speech technology for ASR and TTS [3].

The ECTF S.200 layer shown in Figure 1 is the network protocol specification used to transmit ECTF S.100 commands to the server and responses or events back to the client. The ECTF S.100 standard defines a C Application Programming Interface (API) for media control between the application and the CT server. The S.100 API specification has been extended to make most of Watson's advanced features available to the application programmer.

A client application may use either a *synchronous* or *asynchronous* programming model. We use the asynchronous model to guarantee maximum efficiency and to capture all user interface interactions with minimum latency.

The Event Manager (EM) module receives solicited and unsolicited events from both the CT server telephony middleware and the Extended Resource Manager (ERM). It also registers the callback functions for processing the dispatched events.

The ERM module augments services to the standard pool of media resources. Each service is executed in a different thread of control and communicates with the main thread of execution via the session event queue. The completion event for a specific command executed in a thread is placed on the event queue and evaluated by the EM event handler just like any other system message. This approach is consistent with the *asynchronous* model described above and transparent to the application, which handles new resources similar to the other middleware components. In order to terminate a command correctly, event priorities are implicitly assigned by a finite state machine implemented at the ERM layer. Only the relevant information will be propagated to the upper layers of the dialogue manager, so that the execution of a compound command looks like serialized threads of execution. ERM includes the following services:

- Graphic User Interface (GUI). In addition to presenting the requested information and system messages to the user, the GUI service collects keyboard, mouse, and gesture (e.g. touches on the screen and pen taps) inputs and dispatches them to the dialogue manager.
- Prompt Manager (PM). The PM service allows the application to concatenate pre-recorded and text-to-speech synthesized utterances in a single prompt. It also expands special keywords in the actual prompts mapping. For example `PLAYER {$prompt = RoomNumber, $spelling = D105}` will be translated into a sequence of five prompts chosen to produce accurate prosody for the utterance *"The room number is D one oh five"*.
- Unified Messaging System (UMS). The UMS service is a client for accessing remote message stores kept on email servers using SMTP, POP3, or IMAP4 protocols.
- Lightweight Directory Access Protocol (LDAP). The LDAP module is a service to access on line directory services. LDAP servers are the *de facto* standard for corporate databases.
- Hyper Text Transfer Protocol (HTTP). This service implements a lightweight HTTP server. It can be used to push dynamically created web pages over a generic web client. In order to synchronize the user interaction, a completion event is generated after each page is downloaded.
- Remote Event Notification (REN). REN provides alerting services via TCP/IP. A client can dispatch a customized event to a running application to notify the application that something must be communicated to a specific connected user.

The Application Resource Manager (ARM) and the Application Resource Command (ARC) interpreter translate the dialogue manager directives to internal function calls and synchronize the thread of execution. For example, to play a prompt to the user, activate the ASR, and display a 3D-animated avatar at the same time, the dialogue manager generates the content and sends the command:PLAYER{$prompt=welcome}ASR{$grammar=main} AGENT{$genie=smile}.

In this case the ARM module converts the command to the low level S.100 API calls and sets the appropriate Run-Time Control (RTC) conditions to synchronize the prompt *welcome* with the activation of the speech recognizer. If the user is connected through a graphical device, a smiling 3D-animated character will be displayed on screen; otherwise the AGENT command will be ignored. These commands can be executed sequentially simply by changing the command syntax by inserting a "|" character between successive commands.

```
      PLAYER{$prompt=call,$name=di_fabbrizio}
ASR{cancel} SD{pattern=* * 9} SCR{bridge=973360[redacted]}
```

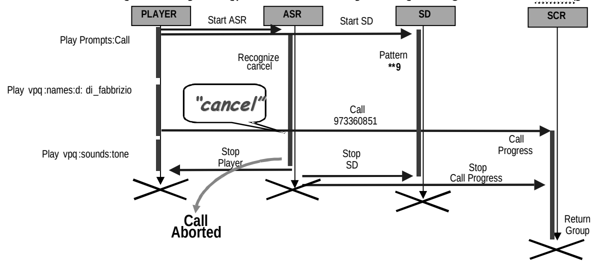


Figure 2. ARM Command Execution Example

Figure 2 depicts a Message Sequence Chart [8] for a typical situation where the Dialogue Manager (DM) asks ARM to make a phone call and to bridge the caller and the callee. The request is interpreted and executed by ARM by spawning four different commands: 1) it plays the prompt "calling Di Fabbrizio", and at the same time 2) starts the recognizer with a grammar called "cancel", 3) enables the signal detector for detecting the pattern `**9`, and finally 4) makes the outgoing call to the number 973 360 8517. If the call completes successfully, ARM bridges the two parties. Figure 2 shows also how the user can stop the call during set up by saying *"cancel"*. In that case, ARM aborts the ongoing phone call and the other active processes.

The DM module communicates with ARM either via a C++ library or a TCP/IP-based protocol. Depending on which resource is the source of the event, the appropriate function is called or message is sent to the client. The main API function has two arguments: the input argument (`in`), which includes detailed information about the event generated from one of the resources, and the output string (`out`) which is the ARM command that the DM will generate as response to the input stimuli. The functions listed below are an example of C++ wrapper class implemented for a generic DM. Each resource module will define an input media source for the DM module (respectively GUI, ASR, LDAP, UMS, HTTP, and REN):

```
int DM::response(<InResClassTemplate> in, string& out);
```

This approach has been successfully used for two diverse dialogue manager approaches [4][5].

2.2 GENERIC MULTIMODAL INTERFACE

The Generic MultiModal Interface (GMMI) provides a Graphical User Interface for presenting information to the user and collecting input stimuli from the user. Using DHTML documents with embedded JavaScript function, it is able to display information and capture mouse clicks, stylus taps, and keyboard strokes. GMMI takes advantage of the Microsoft COM technology to integrate the following technologies:

- Standard ITU H.323 Terminal (Microsoft NetMeeting® COM component)
- Web Browser with Dynamic HTML support (Microsoft IE WebBrowser COM component)
- 3D-animations (Microsoft Agent COM component)

The GMMI logic acts like a message router, intercepting all the events from the underlying modules and redirecting the relevant events to the ARM module and vice versa. Since the ARM communicates with the DM, this architecture gives full control to the DM, which is able to make decisions about content generation based on the context of the interaction such as displaying DHTML pages launching animations, and even executing a function script in the DHTML

The GMMI software can be downloaded to the user's PC using a distribution package. When installed GMMI resides in the Windows' taskbar icon tray.

2.3 DATA LOGGER AND ANALYSIS TOOLS

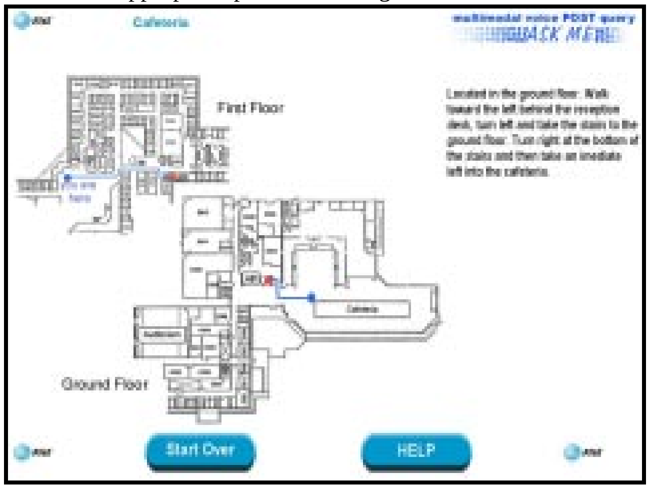
Detailed information about the user's sessions is automatically logged in a plain text file from a data logger system. DM inputs, including raw speech, database lookup, pen, mouse and keyboard activities are recorded with accurate timing information. A PERL script periodically checks whether there are new calls and eventually populates an SQL Relational Database System for further analysis. Collected data are immediately made available over the Intranet via several CGI scripts, which retrieve and summarize the collected data into readable HTML reports. Anomalous system latencies and barge-in occurrences are automatically detected and highlighted in different colors. Dialogue turns are available for analysis, including the user's speech, which can be played within the browser. Single turns may be manually annotated for further evaluation.

3. A CONVERSATIONAL MULTIMODAL-MULTIMEDIA DIRECTORY ACCESS SYSTEM

The application framework described above has been used to implement a large-vocabulary directory query application [1], the multimodal Voice Post Query (mVPQ). mVPQ is a spoken dialogue system for accessing information in the AT&T personnel database. The application can be accessed from a variety of devices, including telephones, desktop PCs, kiosks, and PDAs. Depending on the access device, the mVPQ interface allows the use of multiple input, modalities, including speech, typed input, touch, mouse clicks and stylus input. Output modes are audio only for voice telephony interfaces, and audio, graphics, text, and video for PC and PDA interfaces. The dialogue manager controls the application: it decides on what action to take based on user's input, and decides the content of output presentation in a device-dependent fashion. For example, a query seeking directions to the cafeteria from a telephone will result in a detailed audio description while the one from a kiosk will give a brief audio message accompanied by text and animated information on the screen (Figure 3).

The dialog manager (DM) was implemented using the DMD scripting language following the AT&T Mixed Initiative Design Architecture (AMICA, [5]). The DM maintains its state information in a recursive keyword-value pair data structure, called template. The DM provides unified control over user interactions across multiple device types by creating appropriate input solicitation and content generation based on knowledge of the access device type. The DM relies on a symbolic semantic representation, in template format, to unify input from various sources. For example, natural language text inputs – either typed or those generated from speech by ASR—are converted to this semantic representation by invoking a natural languageunderstanding module. Other input forms such as clicks and touch inputs as well as status messages (timeouts, rejects, waits) can be directly mapped in terms of the semantic representation at the source. In summary, an inter-lingua for communication between the DM and the media I/O was crucial in providing a unifying application interface.

Besides information presentation, error control (in cases of ASR or understanding rejections) and disambiguation strategies (for example when there are multiple matching listings for a user query) are provided by the DM in a device-dependent fashion. For example, in an audio only interaction it may take multiple disambiguation steps to resolve a query (the DM automatically decides the appropriate piece of missing information that is most



Located in the ground floor. Walk toward the left behind the reception desk, turn left and take the stairs to the ground floor. Turn right at the bottom of the stairs and then take an imediate left into the cafeteria.

Figure 3. mVPQ Kiosk Information Display

likely to resolve the request) while in an audio-visual environment multiple pieces of disambiguating information can be presented simultaneously. For example, if a user requests "room number for Rose", for which there are 5 matching listings, the audio-visual display will list all the matches with additional information – complete name and location here – to help resolve the request further as shown in Figure 4.

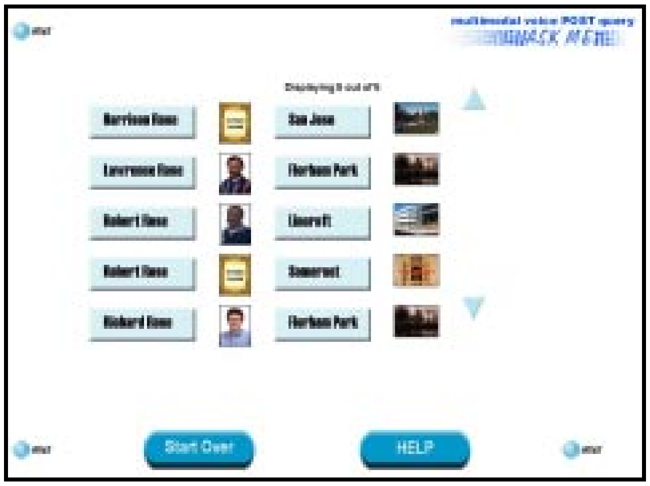


Figure 4. mVPQ Kiosk Information Display

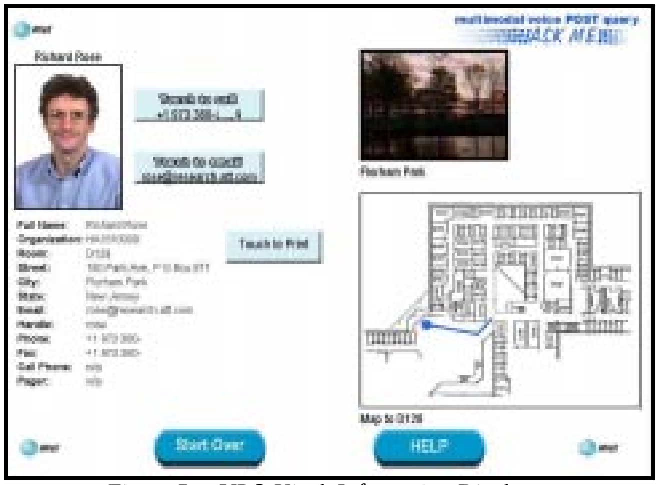


Figure 5. mVPQ Kiosk Information Display

The user can then speak or touch the desired listing to obtain the complete listing (in the example shown, the user could have said, “Richard”, “Richard Rose” or “Florham Park” to get the complete information shown in Figure 5). The disambiguation strategy for the same query in an audio-only telephone interface will cycle through soliciting information for disambiguation, one piece at a time in a sequential fashion (chosen automatically by the dialog manager to increase the likelihood of resolution). In the example shown, the system will solicit location information first. Should that not be provided by the user, the system will then request the first name. This is because the locations for the 5 retrieved listings are all distinct while there are two Rose’s with identical first names. If the user is unable to provide any information to disambiguate, and if the total number of listings is deemed reasonable (chosen to be a maximum of 5 listings in our design), the DM will summarize all the listings as a final fallback option; otherwise the request ends as incomplete.

Another key issue is achieving persistence in the interface. With audio-visual interfaces, the most recent information display remains as long as the user has the system on, although the ASR will turn off automatically after a prescribed waiting period. With audio-only interfaces, this is achieved by repeatedly prompting the user for appropriate information (based on the context of interaction) and the system automatically hangs up after a prescribed number of attempts.

In addition to the dialog manager strategies, the application dependent resources include the database, acoustic and language models for ASR and task domain data for NLU. Details of the knowledge sources used in the mVPQ application can be found in [1]. The application is capable of supporting information in the AT&T corporate directory, which has over 170,000 entries. With the expansion to include nicknames, last names only, multiple pronunciations, and the possessive form, the resultant lexicon for mVPQ has over 900,000 distinct entries. The ASR results, including the associated lexical semantic tags and scores, are passed by the ARM to the dialogue manager, which uses a natural language understanding (NLU) module to obtain a semantic representation of the natural language input. The dialogue manager is responsible for determining the structure of the interaction with the user, based on the current context of the interaction and the most recent semantic interpretation, dynamically adapting to the current conditions to resolve ambiguities, uncertainties and error conditions.

4. SUMMARY

This paper presents architecture and interface design for engineering conversational interfaces for network-based services that can be accessed from a variety of communication devices. We use an application framework that extends a standards-based CT platform to accommodate multimodal interactions by providing additional resources for information access and multimodal integration. The Application Resource Manager provides an effective method for separating the application logic from low-level resource management. These enhancements facilitate more efficient design and implementation of multimodal, multimedia interfaces to voice-enabled applications by making use of the same infrastructure, but at the same time permitting access to device-dependent differences in the user interface, in order to make optimal use of the available input and output modalities. The design concepts are illustrated for a large-scale directory access application with both audio-only and audio-visual interfaces.

In the future, our efforts will focus on seeking a generic standardized (e.g. SMIL-like [9]) approach for media I/O, including: both multimedia content presentation and multimodal input solicitation, with special focus on speech inputs.

5. REFERENCES

[1] B. Buntschuh, et al - *“VPQ: A Spoken Language Interface to Large Scale Directory Information”*, Proc. ICSLP 98, 1998.

[2] Wright, J. H., Gorin, A. L. and Abella, A., *“Spoken language understanding within dialogs using a graphical model of task structure”*, Proc. ICSLP 98, 1998.

[3] Sharp, R. D., Bocchieri, E., Castillo, C., Parthasarathy, S.,Rath, C., Riley, M. and Rowland, J. *“The Watson speech recognition engine”* Proc. ICASSP 97, 4065-4068, 1997.

[4] Abella, A. and Gorin, A. L., *“Generating semantically consistent inputs to a dialog manager”* Proc. Of EUROSPEECH 97, 1879-1882, 1997.

[5] Pieraccini, R., Levin, E. and Eckert, W., *“AMICA: the AT&T Mixed Initiative Conversational Architecture”*, Proc. Of EUROSPEECH 97, Rhodes, Greece, Sept. 1997.

[6] *“ECTF S.100 Media Services APIs”*, Rev 2, Vol 1-6, http://www.ectf.org

[7] *“Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service”*, ITU-T Recommendation H.323

[8] *“Formal Semantics of Message Sequence Charts”* ITU-T Annex B to Recommendation Z.120

[9] *“Synchronized Multimedia Integration Language (SMIL) Boston Specification”*, W3C Working Draft 25 February 2000, http://www.w3.org/TR/smil-boston/