Extending a Standards-based IP and Computer Telephony Platform to Support Multi-modal Services

G. Di Fabbrizio, C. Kamm, P. Ruscitti, S. Narayanan, B. Buntschuh, A. Abella, J. Hubbell and J. Wright

AT&T Labs – Research
180 Park Avenue
Florham Park, NJ, 07932
{pino,cak,ruscitti,shri,bb,abella,jhubbell,jwright}@research.att.com

ABSTRACT

Despite recent advances in Computer Telephony (CT) and IP Telephony (IPT) standards at defining flexible architectures to support new technologies, the current CT paradigm does not adequately support the requirements of advanced spoken dialogue systems. This paper describes an application framework based on CT and IPT standards that defines new architectural components for information access, alerting, and multi-modal input/output integration. This framework permits separation of the application logic from low-level resource management in order to facilitate the design and development of advanced, multi-modal voice-enabled services.

1 INTRODUCTION

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) 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 Spoken Dialogue 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 architectural components in 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 paper 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 multi-modal 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 multi-modal graphic user interface for desktops and handheld devices (PDAs) that extends user inputs to mouse clicks, speech, and typed text. Output modes include speech, text, graphics, gestures and facial expressions by 3D-animated characters. Finally, we describe a speech-enabled IP telephony directory assistance application with a multi-modal graphical user interface running on a wireless PDA.

2 ARCHITECTURAL COMPONENTS

The application framework described in this paper was designed to satisfy the following goals:

- Providing access to the service from different devices;
- 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;
- 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;
- Facilitating the integration of the dialogue manager with system resources by providing a platform independent declarative language to represent the dialogue manager's actions;
- Adopting standard 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 includes a pool of media services such as the telephony line interface (System Call Router, SCR), the speech recognizer (ASR), the Signal Detector (SD), the Text-To-Speech (TTS) system, the prompt Player, and the Recorder. The upper layers (S.100 and above) are in 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 an ITU-H.323 gateway [8]. 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 Text-To-Speech synthesis (TTS) [3].

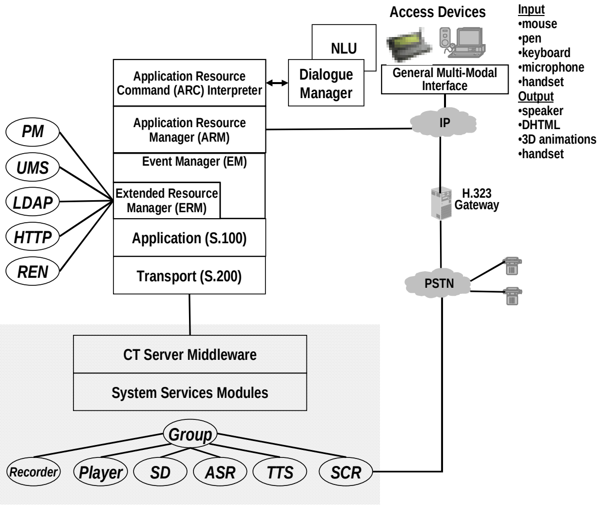


Figure 1. System Architecture

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. In the synchronous model, the thread in which a function runs blocks until the corresponding completion event arrives; in the asynchronous model, completion events are collected either by polling the event queue or by client-registered event handlers. We use the asynchronous model to guarantee maximum efficiency and to capture all user interface interactions.

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 Extended Resource Manager (ERM) adds services to the standard media resources. Each service is executed in a different thread and communicates with the main thread of execution via the session event queue. The completion event for a specific command executed in a thread will be 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. For example, an application may choose not to block during the database lookup, in order to play a "waiting" message to the user. This is accomplished by sending two commands simultaneously, to the Player and to the LDAP directory access module.

ERM includes the following services:

- Prompt Manager (PM). The PM service allows the application to combine different prompts in a single command. 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 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.

- HyperText Transfer Protocol (HTTP). This service implements a lightweight HTTP server. It can be used to transmit dynamically created web pages over IP. 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 thread execution. For example, to play a prompt to the user, activate the ASR, and display a 3D-animated character at the same time, the dialogue manager 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 speech recognition start. 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. Figure 2 depicts a message chart [9] for a typical situation

```
     PLAYER{$prompt=call,$name=di_fabbrizio
ASR{cancel} SD{pattern=* * 9} SCR{bridge=9733608517}
```

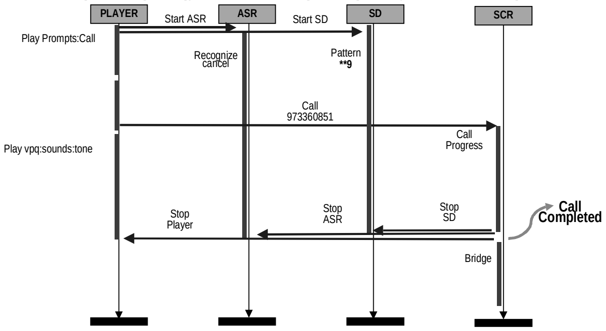


Figure 2. ARM Command Execution Example

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 9733608517. If the call completes successfully, ARM bridges the two parties. Figure 3 shows 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.

```
PLAYER{$prompt=call,$name=di_fabbrizio}
ASR{cancel} SD{pattern=* * 9} SCR{bridge=9733608517}
```

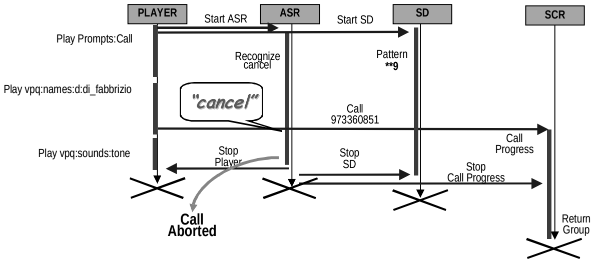


Figure 3. ARM Command Execution Example

The DM module communicates with the 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 source of events for the DM module (respectively ASR, LDAP, UMS, HTTP, and REN):

```
int DMInt::response(ASRRsp in,string& out);

int DMInt::response(LDAPRsp in,string& out);

int DMInt::response(UMSRsp in,string& out);

int DMInt::response(HTTPRsp in,string& out);

int DMInt::response(RENRsp in,string& out);
```

This approach has been successfully used for two diverse dialogue managers [5][6].

3 GENERIC MULTI-MODAL INTERFACE

The Generic Multi-Modal Interface (GMMI) provides a Graphical User Interface and collects the input stimuli from the user. The GMMI architecture is shown in Figure 4. 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)

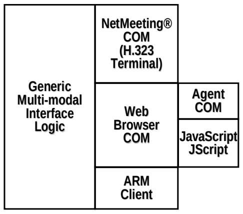

Generic Multi-modal Interface Logic
NetMeeting® COM (H.323 Terminal)
Web Browser COM
Agent COM
JavaScript JScript
ARM Client

Figure 4 . Generic Multi-Modal Interface Architecture

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 dialogue manager, this architecture gives full control to the dialogue manager, which is able to display DHTML pages, animate the 3D-character, and even execute a function script in the DHTML code. For example the ARM command *`URL{http://www.weather.com/weather/us/zips/07901.html} AGENT{script=move (upper,left,Table2)}`* would show the local weather forecast and it would move the agent from the current location to the upper left corner of the document object Table2. The GMMI software is downloaded to the user's PC using a distribution package. When installed GMMI resides in the PDA's taskbar icon tray.

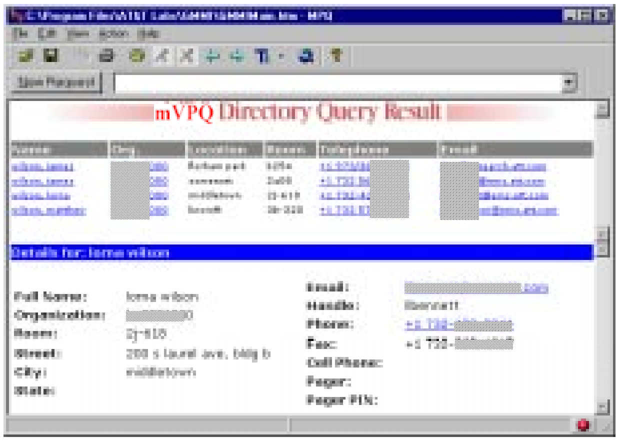
4 A MULTI-MODAL PORTABLE SPOKEN DIALOGUE SYSTEM FOR DIRECTORY ACCESS

The application framework described above has been used to implement a large-vocabulary directory query application [1], Multi-modal 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 and PDAs. Depending on the access device, the mVPQ interface allows the use of multiple input modalities, including speech, typed input, 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. This section describes multimodal access using a wireless PDA. The PDA we use is a Mitsubishi Amity VP, a Windows 95 PC with built-in microphone and an electromagnetic stylus for input and speaker and screen for output. A wireless LAN card was used to allow connectivity to the CT platform.

An explicit design goal for mVPQ is to have the user's initial interaction with the system be rather unconstrained and to rely on system-directed dialogue only when necessary. The active ASR grammar for the first interaction with the user allows for a variety of ways to request a) most attributes associated with an employee such as phone, fax or mobile number, e-mail address, location, etc., b) help, and c) actions such as calling, paging or faxing. Examples of queries mVPQ can handle include:

- What's Mike Armstrong's phone number?
- Call Pino Di Fabbrizio.
- I need the email address for Paolo Ruscitti.

The corporate directory is preprocessed to populate the main ASR grammar with all last names in the directory, and all full names, where full name is defined as a first name - last name pair. In addition, full names composed of preferred name (i.e., nickname) - last name pairs are also included where that information is available. The pronunciations for each name are derived using the front end of the Watson TTS engine, which generates multiple pronunciations for names that are likely to be pronounced more than one way (e.g., Weinstein would be entered as both /waynstin/ and /waynstayn/). The corporate directory has over 120,000 entries, but with the expansion to include nicknames, last names only, multiple pronunciations, and the possessive form, the resultant lexicon for mVPQ has over 700,000 distinct entries. The ASR grammar represents proper names in a phonetic form delimited by lexical semantic tags. The recognized phonetic form of a name is easily isolated in parsing and can then be efficiently searched for in the database to retrieve all candidate entries. The schema for the mVPQ LDAP directory has been designed to facilitate retrieval of all entries that correspond to a specific utterance, by indexing each entry on the phonetic representation of the names (for the ASR results) as well as the orthographic representation (for typed requests on devices with keyboards).


C:\Program Files\AT&T Labs\GMMI\GMMain.htm - MPQ
File Edit View Action Help
Show Request

mVPQ Directory Query Result

Name	Org	Location	Room	Telephone	Email
wilson, james	[redacted] 080	florham park	b25a	+1 973[redacted]	[redacted]@att.com
wilson, james	[redacted] 080	somerset	2a08	+1 732 [redacted]	[redacted]@ems.att.com
wilson, lorna	[redacted] 080	middletown	2j-619	+1 732 [redacted]	[redacted]@ems.att.com
wilson, matthew	[redacted] 080	lincroft	3b-328	+1 732 [redacted]	[redacted]@ems.att.com

Details for: lorna wilson

Full Name:	lorna wilson	Email:	[redacted].com
Organization:	[redacted]0	Handle:	lbennett
Room:	2j-610	Phone:	+1 732-[redacted]
Street:	200 s laurel ave, bldg b	Fax:	+1 732-[redacted]
City:	middletown	Cell Phone:	
State:		Pager:	
		Pager PIN:	

Figure 5. GMMI screen

The ASR result, 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. 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 ASR/NLU result, dynamically adapting to the current conditions to resolve ambiguities, uncertainties and error conditions. Strategies for ambiguity resolution are both data- and access-device dependent. Access devices capable of presenting text output (e.g., PDA or PC) make use of the visual mode to facilitate disambiguation, as shown in the following sample interaction.

To access mVPQ from the PDA, the user connects to the CT platform by tapping on the GMMI icon, which sends notification to ARM to make an IPT call to the PDA's IP address. At the same time, the ARM instructs the GMMI to display a "Welcome" screen, the Player resource to play the prompt "VPQ, how may I help you?", and the ASR resource to load the initial grammar. The user speaks the initial request, "Call Wilson". The ASR result `_ACTION {call} _NAME {wilsin}` is sent to ARM and passed to the dialogue manager and NL modules. The dialogue manager determines that there is sufficient information (i.e., a NAME) to search the database, and instructs ARM to send an LDAP query for the name pronounced /wilsin/. In this case, the LDAP query results in four matching listings that are returned to the dialogue manager. The dialogue manager then directs ARM to send instructions to the Player to play the prompt "I have four listings for Wilson. Which one do you want?", to GMMI to display the summary listings shown in the top panel of the screen shown in Figure 5, and to the ASR resource to create and activate a grammar from subset of legal responses to the question (in this example, James, Jim, Lorna, Matthew, Matt). Subsequent recognition of the spoken first name or a stylus tap on a highlighted name in the upper part of the display are sent through ARM to the dialogue manager, which then instructs ARM to display the detailed information for the selected name in the lower GMMI panel, and to set up a phone call to the phone number associated with the selected entry.

5 SUMMARY

This paper has presented an application framework that extends a standards-based CT platform to accommodate multimodal interactions by providing additional resources for information access and multi-modal integration. The Application Resource Manager, which translates directives from the dialogue manager into S.100 and other API calls, provides an effective method for separating the application logic from low-level resource management. These enhancements facilitate more efficient design and implementation of multi-modal and telephony-only interfaces to voice-enabled applications by making use of the same infrastructure, but at the same time permitting access device-dependent differences in the user interface, in order to make optimal use of the available input and output modalities.

6 REFERENCES

1. B. Buntschuh, C. Kamm, G. Di Fabbrizio, A. Abella, M. Mohri, S. Narayanan, I. Zeljkovic, R. D. Sharp, J.Wright, S. Marcus, J. Shaffer, R. Duncan and J. G. Wilpon - "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. S. Downey, A. P. Breen, M. Fernandez, E. Kaneen, "Overview of the Maya Spoken Language System" *Proc. ICSLP 98*, 1998.
5. Abella, A. and Gorin, A. L., "Generating semantically consistent inputs to a dialog manager." *Proc. Of EUROSPEECH* 97, 1879-1882, 1997.
6. Pieraccini, R., Levin, E. and Eckert, W., "AMICA: the AT&T Mixed Initiative Conversational Architecture", *Proc. Of EUROSPEECH 97*, Rhodes, Greece, Sept. 1997.
7. "ECTF S.100 Media Services APIs" - Revision 2- Volumes 1 through 6, *http://www.ectf.org*
8. "Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service", ITU-T Recommendation H.323
9. "Formal Semantics of Message Sequence Charts" ITU-T Annex B to Recommendation Z.120