Personal Voice Call Assistant: VoiceXML and SIP in a Distributed Environment

Michael Pucher
FTW
Donau-City Straße 1/3
A – 1220 Vienna
+43/1/5052830-98
pucher@ftw.at

Julia Tertyshnaya
FTW
Donau-City Straße 1/3
A – 1220 Vienna
+43/1/5052830-45
tertyshnaya@ftw.at

Florian Wegscheider
FTW
Donau-City Straße 1/3
A – 1220 Vienna
+43/1/5052830-45
wegscheider@ftw.at

ABSTRACT
In this paper we introduce the architecture of a distributed service platform that integrates speech, web technology and voice-over-IP technologies and describe how a specific service can be built using these technologies.

The electronic assistant is an advanced voice-based service, that answers incoming calls and takes messages, consulting the user's calendar and address book. A novel contribution consists in the dynamic creation of the dialog (in form of VoiceXML pages) from calendar and other data residing in the platform's databases, providing in this way actualized information to the caller (at call time).

The described platform provides basic services (like call control) and allows developers to stack services, alleviating quick service generation. The assistant illustrates how several fairly basic building blocks can be combined into a powerful end-user application

Sticking to our assistant as a guiding example, the first part of this paper gives details on the platform's goals and architecture. The second part portrays the dynamic generation of user-friendly and intuitive VoiceXML pages and grammar from the calendar database, which proved to be an intricate task by itself.

Categories and Subject Descriptors
C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *distributed applications.*

E.1 [**Data**]: Data Structures – *distributed data structures.*

General Terms
Design, Experimentation.

Keywords
VoiceXML, SIP, CPL, Parlay, CCXML

1. INTRODUCTION
The implementation of new mobile communication technologies like UMTS and GPRS will have a strong impact on the Internet. Today we already access the Internet not only from a PC, but also via mobile phones, palmtops and other devices. New applications combining several basic services like telephony, e-mail, web browsing or instant messaging will emerge and include security and Quality of Service mechanisms.

We developed such an Internet-based telecom application using VoiceXML, which applies web development paradigms to voice dialog development and the Session Initiation Protocol (SIP).

The usage of these two technologies leads to a certain convergence between the Web and the sparsely existing Voice Web. Using VoiceXML voice interfaces, people can easily access Web resources, while SIP turns out to be the key technology for both Internet telephony and the core network of UMTS.

1.1 THE PERSONAL VOICE CALL ASSISTANT
The *Personal Voice Call Assistant* is an electronic secretary that can perform actions similar to those of a human assistant. Automatic Speech Recognition allows the user to interact with the assistant using natural language commands. Textual information extracted from a calendar is synthesized by a Text to Speech Synthesizer and played to the user.

The owner of an assistant activates it, if he cannot or does not want to answer calls. From this moment on all calls destined to the owner will be redirected to his Personal Voice Call Assistant.

Let us consider the following scenario: user B, owner of a Personal Voice Call Assistant, has to attend a meeting and enables his assistant during his absence, or configures his assistant to take calls after the – say – third ring if he does not pick up the phone. A colleague, user A, then calls him. A gets connected to B's personal assistant. The system recognizes the caller by his address. After establishing the connection, A can talk to B's Personal Voice Call Assistant. A can listen to a personalized message, left for him by B, leave a message for B, access his calendar to find out what he is currently doing (in a meeting till five o'clock), listen to the list of the events B has planned for today, or ask when B has free time.

If the caller (A) is not satisfied with the information the Personal Voice Call Assistant provides, he can ask the assistant to connect him with a colleague of B or any other person, defined by B in advance. For that the existing call between A and the Voice Call Assistant must be released and a new call between A and a colleague of B is established (see chapter 4 for more details).

Figure 1 shows all scenarios between two users owning Personal Voice Call Assistants. The users can either call each other (e.g. user A@Platform calls user B@Platform) or call their own Voice Call Assistants (e.g. user A@Platform calls assistant A@VoiceBox).

Users calling Personal Voice Call Assistants are divided into owners and non-owners. The distinction is made basing on the address of the caller. The Personal Voice Call Assistant has different behavioral patterns for the owner and a non-owner.

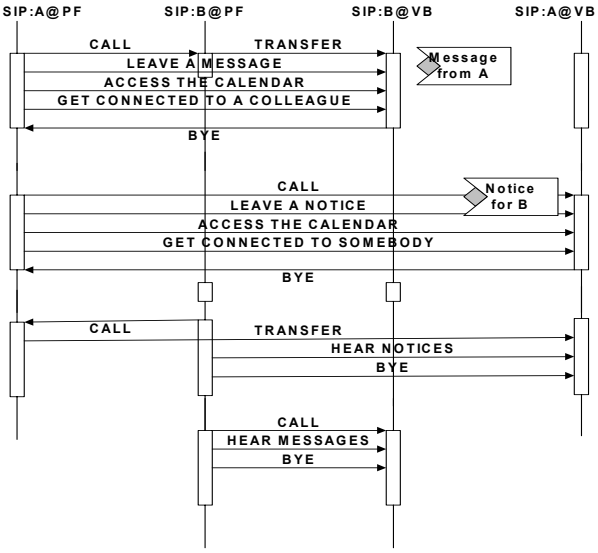

Figure 1. Possible Usage Scenarios

In general the owner-user can

- leave a notice for a person who will (probably) call while the owner is absent
- listen to messages left by other users (who called while the owner was not available)
- access his own calendar and listen to information about all the events or the next event, the owner has planned for today
- get connected to a person who has an entry in his address book.

and any user can

- leave a message for the owner
- access the calendar and listen to filtered information about public or restricted events
- get connected to a colleague of the owner
- hear notices the owner-user left for him
- access other VoiceXML applications (e.g. an interactive holiday diary).

The existing assistant is a prototype demonstrating the power of our architecture. It has been implemented with high level API's and provides a voice interface to dynamic data. The use of VoiceXML and other standardized components make it easily extensible**.**

1.2 ENABLING TECHNOLOGIES

VoiceXML is an XML-based language, designed for creating audio dialogs. It specifies functional requirements on implementations of VoiceXML interpreters and is driven by the W3C. Version 2.0 has the status of a last call working draft. Basic concepts of VoiceXML are dialogs, prompts and grammars. Dialogs are the basic building blocks that consist of forms, which are to be filled. Prompts consist of prerecorded audio or synthesized speech. Grammars determine what a user can say to fill in the fields of a form.[6]

SIP (Session Initiation Protocol) is a signaling protocol for Internet conferencing, telephony, presence, event notification and instant messaging. SIP was developed within the IETF MMUSIC (Multiparty Multimedia Session Control) working group, with work proceeding since September 1999 in the IETF SIP working group.[3]

CPL (Call Processing Language) is a simple XML-based language that allows an end-user to create scripts, which control the behavior of incoming and outgoing calls for the user. CPL provides functionality such as call redirection and call blocking. Every CPL script is a tree of statements, which may contain forks but no loops, since scripts potentially run on mission critical real time telecom switches. CPL is specified by the IETF.

Parlay provides APIs for accessing telecom services and network resources, defined in UML, OMG IDL and in future also WSDL formats. The Parlay APIs are jointly standardized by the Parlay Consortium, ETSI and 3GPP [1].

2. ARCHITECTURE

2.1 Service Platform

The *Service Platform* developed within the research project A1 is based on the Parlay/OSA model and CORBA Components model. The service platform can serve as a basis for an application service provider or a network operator to host applications for advanced call management, web conferencing, notification in case of traffic jams, etc.

The platform consists (amongst others) of the following components:

A *SIP Proxy Server* based on SIP version RFC2543 which implements the Parlay Generic Call Control Service. Other applications can use this service to access the network. The Generic Call Control Service provides a single point of access to the network, masking the network's complexity.

The *CPL Application* interprets CPL scripts written by the end-user and stored in a database in the Service Platform, in order to control the behavior of incoming and outgoing calls for the user. It registers with the Generic Call Control Service to be able to gain control of all calls for users who provide a CPL script.

The *Call Setup Service* is a high level API that can be used to establish calls via the underlying network. It simplifies the Parlay Call Control API. Its API is proprietary, however. It, too, uses the Generic Call Control Service for accessing the network.

Thus, the CPL Application and the Call Setup Service use the *Parlay Generic Call Control Service* as an interface to the network.

The *Address Book Service* allows the creation and use of personal address books. It is used for call screening, that is to verify if the caller is known and in consequence to filter the information and adapt or even completely modify the dialogue. In addition it provides the VoiceXML grammar used in the page which connects the caller to a 3rd party (see chapter 4 for more detail).

The *Calendar Service* performs the connection to an external calendar and generates natural language prompts, in VoiceXML compliant format, out of the calendar data. Calendar data is dynamic, since calendar events can be changing constantly (every

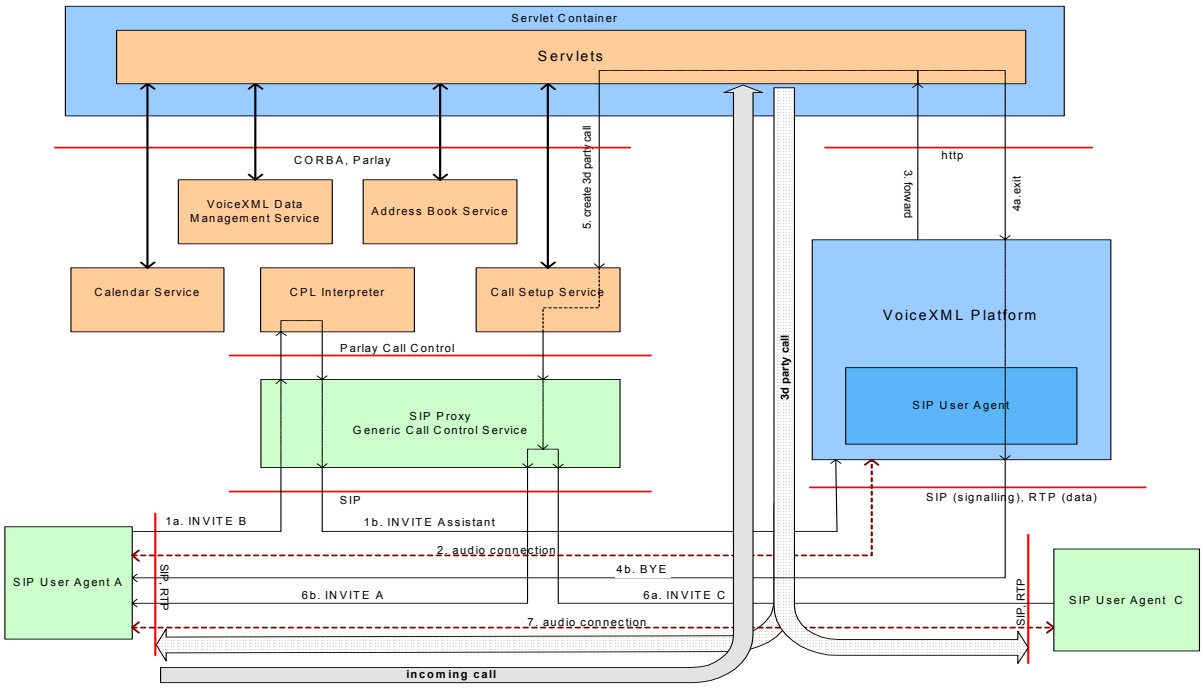

Figure 2. System Architecture

hour or day). These prompts are played to the caller by the Text to Speech system of the VoiceXML platform.

The *VoiceXML Data Management Service* manages users' VoiceXML files and audio messages which it stores in a database. This service is part of an effort to centralize the platform data storage instead of keeping data on several web servers.

We use this service to get a list of messages left for the owner-user. Out of this list we generate a dynamic VoiceXML prompt that informs the user who has left messages. After that it is possible to listen to the messages of a specific caller.

To record messages the <RECORD> tag in VoiceXML is used and the recordings are posted to the Web Server as binary multi-part/form-data. The VoiceXML Data Management Service stores the recorded messages in the database.

All services are currently implemented as Parlay services (using CORBA on the remote access layer) but could also be realized as web services using SOAP over HTTP.

2.2 External components

The *external components* are integrated with the service platform:

A *VoiceXML Platform* consisting of a TTS (Text to Speech) server, an ASR (Automatic Speech Recognition) system and a VoiceXML Browser.

A *Web Server* and a *Java Servlet Container* constitute an interface between the VoiceXML Platform and the Service Platform.

The *Calendar* allows owner-users to store events. Every event is characterized by the parameters starting and ending time, title and location.

The combination of services and external components enables the *Personal Voice Call Assistant*.

3. DYNAMIC VOICEXML GENERATION

In general there are two different types of domains for which one can generate dynamic content. The domain is either open (e.g. the Web) or closed (e.g. a calendar). The problem of generating dynamic VoiceXML dialogues for open domains is that we only know the structure of the documents, not the contents. (We only know that it is HTML or XML structured data). The structure of the content could be defined using the Resource Description Framework defined by W3C's Semantic Web Activity group. [2]

If we have a closed domain data structure like a calendar on the other hand, we know the content structure of the domain. Although the domain is closed it is dynamic, i.e. changes over time, so that we need dynamic VoiceXML generation to build a voice interface for it.

Accessing the Calendar

A calendar event is characterized by its starting and ending time, its title and location, its duration, which can be normal (some hours, i.e. starts and ends on the same day), long (2 days and more) or whole day and its visibility, which can be public, restricted or private.

Registered users and the owner-user have separate interfaces for accessing the calendar, i.e. they are supposed to use different sets of operations (event request types):

The owner-user can:

- get list of events for the day;
- find out their next event today.

The registered user can:

- get the list of events of the owner-user for the day;
- find out the current event of the owner-user;
- and find out when the owner-user is available.

Both interfaces could be extended if necessary.

The dynamic VoiceXML prompt generation process flows as following: the Calendar Service accepts the user request, connects to the calendar, authorizes itself as owner-user, using his user identifier (uid) and password and obtains structured data from the calendar. At this point the Calendar Parser (one of the modules of the Calendar Service) is invoked and an extraction process begins: necessary information, such as starting and ending event time, event title and event location is extracted from the data obtained from the calendar earlier. Then the appropriate template is found and fed with the extracted data, which fills up blank spaces in the template. Thus, the Calendar Parser is fed with raw calendar data and is responsible for generation of VoiceXML prompts out of it

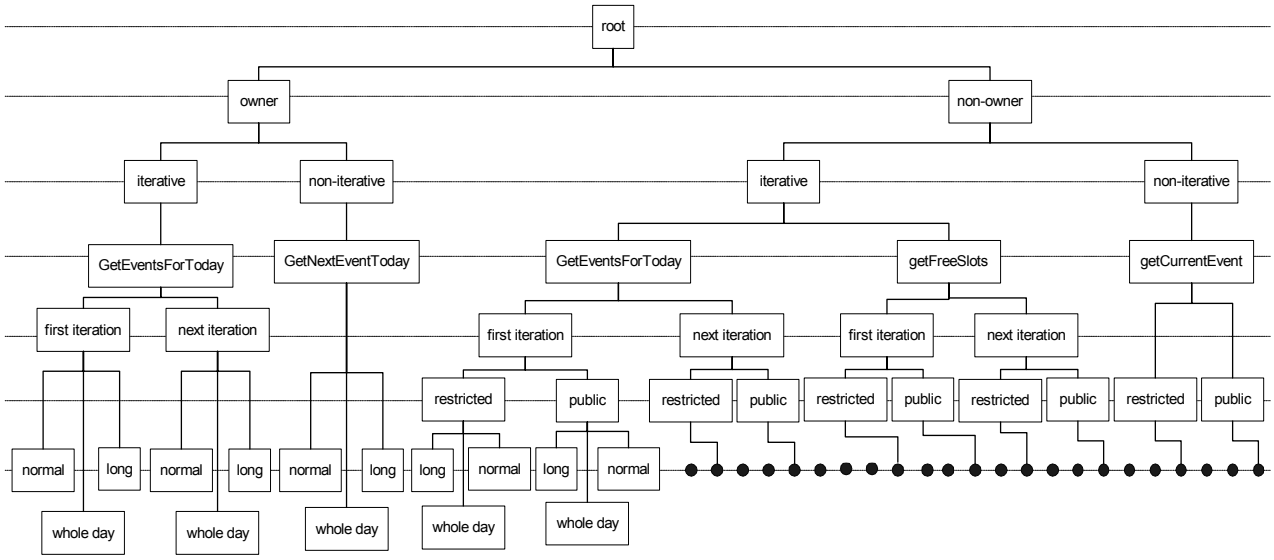

Figure 3. Template tree

Let's consider the extraction process in more details.

Extraction can be either iterative or non-iterative. Iterative extraction takes place if the event response contains more than one event, otherwise extraction is non-iterative. In the iterative extraction case the extraction cycles are repeated as many times as there are events in the event response. Non-iterative extraction means that there's a single extraction cycle since there's only one event in the event response.

In the iterative extraction case the final VoiceXML prompt is constructed from multiple templates found and filled during every iteration cycle and appended all together afterwards.

Every iteration cycle consists of three steps:

(1) extracting dynamic bits of information from the event response;

(2) finding a template, walking through the decision tree (see Figure 3) guided by the given request type, event duration, iteration number (first or next) and user access rights;

(3) replacing the template slots by the calendar event values, using regular expressions.

Below we provide more details on each of the three steps.

Step 1.

In every iteration cycle the extract() method of the Extractor class is called. During every cycle dynamic event data such as title, location, start and end time of the event is obtained in the correct format. For different request types various extractor implementations are instantiated. Eeach concrete child class of the Extractor should decide what kind of information and in which format to return. The decision depends on the event duration type.

For example, for normal events we need the short format of start and end time: just hours and minutes. Since the request always comes for today we do not need to specify year, month and day. For long events, however it is necessary to provide the long format – day, month, year.

Examples:

Normal event: Meeting from 10.00 till 13.00 at office

Long event: Vacation from 16/07/2002 till 4/08/2002

Step 2.

During every iteration cycle the template path is constructed based on the following criteria: event duration (long, normal, whole day), user access rights (for non-owners we have to filter out private events and hide particular information from restricted events), request type (getFreeSlots, getNextEvent, etc.) and iteration number (first or next iteration). Every leaf of the XML tree reached after all the decisions is a template. We need a substantial amount of the templates to provide natural sounding prompts for the various use cases.

Here are some examples of templates:

Template path:owner/iterative/getEventsForDay/

first-iteration/normal/

Template: Today you have [title] from [start_time] till [end_time] [location]

Template path: non-owner/non-iterative/getCurrentEvent/

restricted/normal/

Template: Now the callee is restricted and will be available at [end_time]

Step 3.

Content inside square brackets is to be replaced by dynamic calendar data.

If the prompt creation is iterative the extraction result is appended to the already constructed prompt. The prompt that is produced during the iteration cycle is included in a VoiceXML page and loaded by the VoiceXML browser.

We believe that this extraction method is general enough to be used in similar contexts, where one has a closed, yet dynamic data structure and wants to develop a voice interface for it.

4. VOICEXML, SIP AND CCXML

As opposed to VoiceXML, which allows transferring calls from the voice application to a different terminal and back to the VoiceXML page, SIP does not support this behavior[1]. In order to simulate VoiceXML's transfer to some address, we have to terminate the call and create a new call to the requested address from outside the VoiceXML pages. This is currently done by our servlets via the Call Setup Service. We call this a "third party call" as a third party (the servlet) establishes a connection between two parties.

On Figure 2 we can see the different steps of this 3rd party call scenario:

In **steps 1 and 2** the user A calls user B. If B is not available and his *Personal Voice Call Assistant* is active, the caller gets connected to the assistant (**step 1**). The SIP Proxy handles the network call and notifies the CPL application about an incoming call. The call is routed according to a CPL script belonging to B. The CPL script contains information concerning where to route calls for B. In our scenario the CPL script redirects all the incoming calls to the VoiceXML platform. More complex scenarios (e.g. call forwarding on busy, depending on caller, time of day etc.) are possible and will be generally used. An audio connection between A's User Agent and the Voice Platform is established (**step 2**); the caller enters a voice dialog with B's *Personal Voice Call Assistant* and can from now on interact with the system.

Let us assume now that A wants to be redirected to a colleague (user C) of B. All he has to do is say the appropriate command[2]. The grammar accepts sentences like "I want to talk to somebody" or "I want to get connected to a colleague".

If this command is recognized an intermediate VoiceXML page is loaded that contains a SIP address and immediately submits the SIP address to a Servlet. It then loads yet another VoiceXML page containing an <EXIT> tag (**step 4a).**

In **steps 4b** the voice browser sends a SIP "BYE" message to A's User Agent, which causes the call between the caller and Voice Platform to be released.

The Servlet that produced the exit page initiates the 3rd party call with the desired user in **step 5**. To do this, it needs an instance of the *Call Setup Service*.

In **steps 6a-6b** the *SIP Proxy* sends SIP "INVITE" messages to the colleague of the owner-user and to the caller. If both users accept the incoming call the audio connection between the parties is established (**step 7**).

A disadvantage of using VoiceXML and SIP in separated interpreter contexts is that the VoiceXML application must exit when the new call is initiated. If the new call attempt fails, the application cannot re-enter its VoiceXML part. To get back to the voice dialog the caller would have to repeat the original call.

For asynchronous notification between the VoiceXML and call control contexts one can use extended call control languages like CCXML (Call Control eXtensible Markup Language) which specifies an event handling mechanism [5].

5. CONCLUSION

This paper has shown how VoiceXML and other standardized technologies can be used to create applications for next generation telecom networks.

We introduced an extraction method that can be used to build voice interfaces for closed domain, dynamic data structures.

We also presented an extensible and scalable architecture that uses high level and standardized interfaces and languages (Parlay, VoiceXML, CPL). Looking beyond the call assistant we described, this architecture allows easy creation of various applications and deployment of multiple scenarios basing on combination of the above-mentioned generic elements.

The integration of CCXML or a similar call control language will permit implementing even more functionality in a standardized, easy to understand scripting interface.

6. ACKNOWLEDGMENTS

This work was supported within the Austrian competence center program **K**plus.[3]

7. REFERENCES

[1] Parlay APIs 2.1, The Parlay Group (AT&T, BT, Cisco, IBM & others), June 2000, http://www.parlay.org

[2] Resource Description Framework, http://www.w3.org/RDF

[3] Schulzrinne/Schooler/Rosenberg/Handley, "SIP: Session Initiation Protocol", IETF Draft, ietf-sip-rfc2543bis-02.ps, November 2000

[4] Sparks R., The SIP Refer Method, Internet-Draft (expires January 15, 2003), July 2002 http://www.ietf.org/internet-drafts/draft-ietf-sip-refer-06.txt

[5] W3C: Voice Browser Call Control : CCXML Version 1.0, Working Draft 11 October 2002, http://www.w3.org/TR/ccxml

[6] W3C: VoiceXML Version 2.0, Working Draft, 24 April 2002, http://www.w3.org/TR/voicexml20/

[7] Zeiss Joachim, "Cross-domain Service Deployment with OSA/Parlay and CORBA components", FTW 2002

[8] Niklfeld Georg, Pucher Michael, Finan Robert, Architecture for adaptive multimodal dialog systems based on VoiceXML, Eurospeech 2001, Aalborg, Denmark

[1] At least not until the refer method is supported.

[2] Of course the assistant informs the caller of this possibility.

[3] Parts of the content of this paper are presented at the poster session of the www2003 conference in Budapest.