SPEECH RECOGNITION APPLICATION USING VOICE XML

by
Marieta Gâta

Abstract. This paper presents basic elements of simple, realistic application example that uses VoiceXML programming. The core use cases were elaborated and diagrammed in UML. From the use cases, a basic object model was derived and diagrammed. This example is practical because it derives from a real application. This voice enabled application is the best practice of application design, development and software engineering.
Keywords: VoiceXML, Voice User Interface, speech recognition, speech synthesis

VoiceXML

HTML has roots in publishing. VoiceXML has a programming language background. It has the impression of a programming language: control constructs, variables, event handlers, nested scooping, and so on. At the beginning, VoiceXML was designed to be a programming language easy to learn, lightweight and interpreted for developing VUIs.

VoiceXML renders content as speech and interacts with the user using speech recognition and speech synthesis technologies.

The way of Voice XML structures for interaction with user is dialogs. A dialog is consisting of a sequence of prompts spoken by the computer and responses spoken by a person. Responses given by the person by speak or key using a keypad. In contrast with GUI windows which are multitasked and two-dimensional, VUI Dialogs are sequential and linear by nature.

Architecturally, VoiceXML interfaces are event-driven interfaces like GUIs. In a dialog, the computer speaks a prompt and then waits for the user to respond to it. The computer waits until a speech recognition event occurs. A speech recognition event is initiated by the speech recognition engine, which is continuously analyzing the user's speech and attempting to match it to expected responses in the dialog. There are a lot of possible speech recognition events, including “recognized responses ... ”, “got a responses but didn't recognize it”, “no response” and so on. Unlike GUIs and WUIs, where the events that drive the interface are low-level, non-equivocal incidences (button pressed, mouse clicked, and so on), events in VoiceXML interfaces are the result of complex, computation-intensive, possible processing with errors.

GUIs, WUIs, VUIs

GUI, WUI and VUI represent the major markup language-based browsing interfaces to the internet. GUI is the most fully developed of the three is exemplified by products such us Netscape Communicator and Microsoft Internet Explorer. WUI, the most next developed interfaces, is implemented by "microbrowsers" embedded in wireless phones and personal digital assistants (PDAs). VUIs (Voice User Interfaces) are just beginning to appear as browsing interfaces to the internet, and they are driven by the standardization of VoiceXML 1.0.

The markup languages for these three types of interfaces are all based on XML. XML is a markup metalanguage derived from SGML. XML simplifies some of the complex and little-used features of SGML, but it still provides a flexible and extensible base for defining specialized markup languages.

User interface mode	Browsing technology	Organizing paradigm	Usage
GUI (graphical)	HTML, XHTML	Page	Monitor, keyboard, mouse
WUI (wireless)	WML	Card	Portable handheld
VUI (voice)	VXML	Dialog	Phone, Microphone, Headset

The paper presents an application that uses a simplified personal information manager (SPIM). This application is a small personal information manager with the features like: address book, appointment calendar and to-do list. This application is accessible through voice and Web interfaces. The developing of such application means:

-Use Case Analysis
-Top-Level Use Case
-Edit Contact Information Use Case
-Running Late Use Case
-Review Schedule List Use Case
-Object Model

In the next sections is presented the simplified personal information manager application. The core use cases are elaborated and diagrammed in Universal Markup Language (UML).

Use Cases Analysis

A full personal information manager can be used in many ways, and full case analysis involve many use cases. In a sample application I will leave room for future growth. It can be implemented a variety of functions. In this application I will focus on three representative use cases for the simplified personal information manager application. These cases are representative and underlying technology involved and how it is used.

The use cases are as follows:

-editing information about a contact (for example: name, address, phone number, email)

-handling a scheduled appointment that you're late for

-reviewing your list of things to do today

Next sections elaborate on these cases.

Top-Level Use Case

The top-level use case diagram in Figure 1 shows the intentionally limited scope of the simplified personal information manager application. In a real simplified personal information manager application there would be many more use cases at this level, for example, “call a contact”, “schedule an appointment”, “add an address to an existing contact” and so on.

During use case analysis, SPIM User's access mode (eye or ear) is not assumed or implied unless the access mode is intrinsic to the requirements of the use case. For example, the use case in the next section (titled “Edit Contact Information Use Case”) should be the same regardless of access mode, because it is an essential function of the system. On the other hand, a hypothetical “Dial Roadside Assistance from My BrokenDown Car” use case could, dubitable, presume that the user is accessing the system through a VUI or WUI, but not a GUI.

The next sections train another level in this scenario. Every case starts with the Authenticate use case. This shared case models the process of identifying the user to the system. Conventionally, authentication is performed by an username/password-based login, but it may use another mechanism for voice.

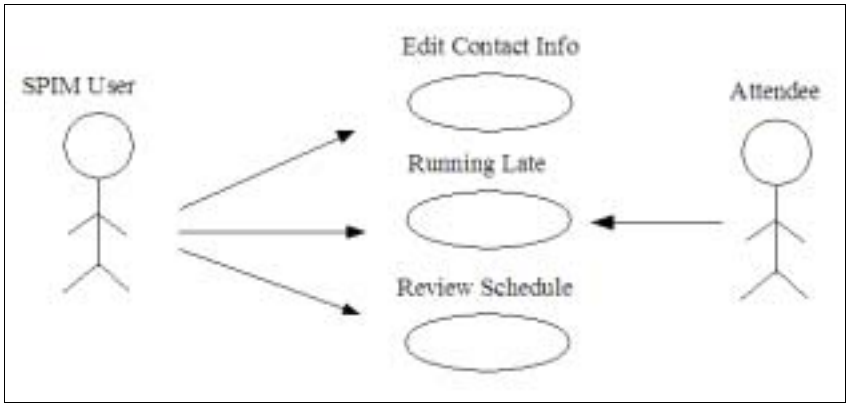

Figure 1

Edit Contact Information Use Case

We assume the next situation. If you received a change of address notice in the mail from a friend, you are not going to call the friend right now or schedule a meeting with him or her, but you want to record the address change. First, you look up the existing contact information for your friend, and then you selectively modify it. To look up it, you can either browse your address book (if it's small) or search your address book by name. Once you have the information at hand, you can either review it in summary form (the types of address you have on a file) or in detailed form (a full listing of address contents). Then you update the fields that have changed. Figure 2 shows this use case.

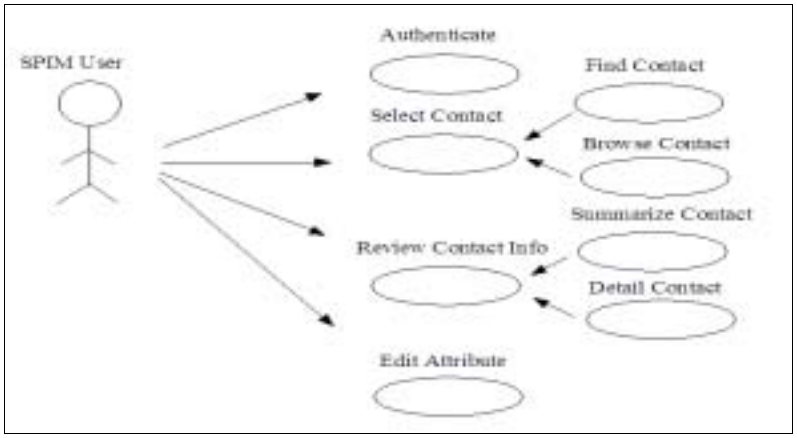

Figure 2

Running Late Use Case

Here we assume an ordinary experience. Your schedule is agglomerated and you've just escaped from a meeting that runs over. You don't have time to reschedule things, so you want to notify your next appointment that you'll arrive late. You access your SPIM, get the relevant contact information for your next appointment, and communicate the fact that you'll be late, by the best means possible: email, phone call, wireless text message, or voice mail. Figure 3 illustrate this case.

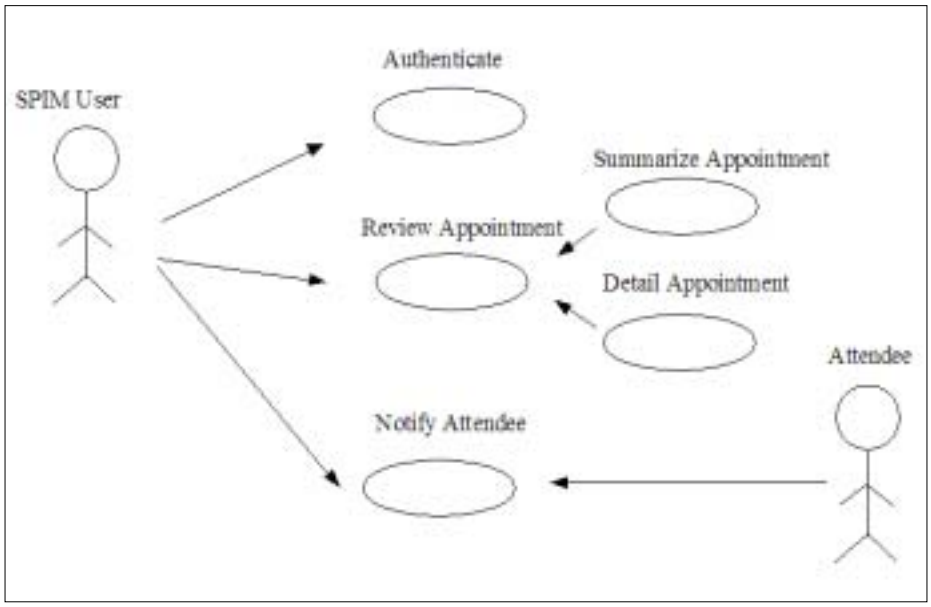

Figure 3

Review Schedule List Use Case

To refresh your memory on what you have scheduled for the upcoming week, you want to review your appointments on a day-by-day basis. In your SPIM, you identify the day you're interested in and then review that day's planned activities. As you progress through your list of appointments, you want to scan quickly, ignore things you already recall, and selectively focus on particular items of interest. This use case is illustrates in Figure 4.

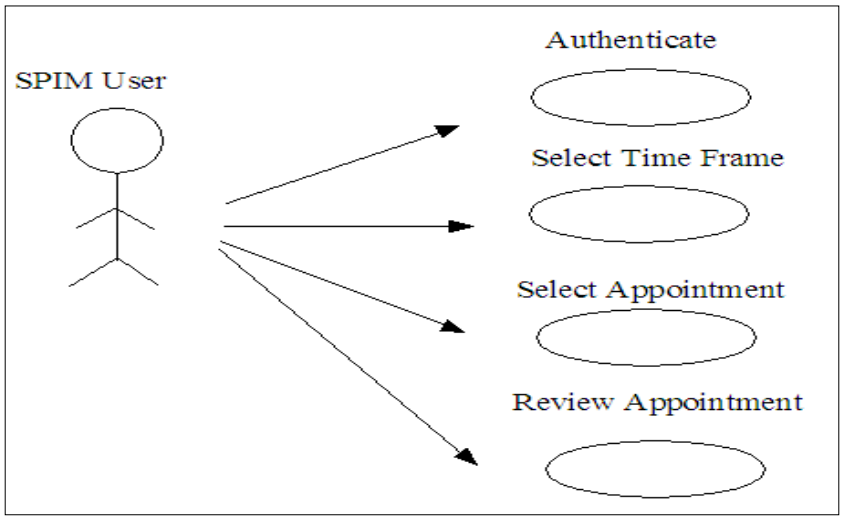

Figure 4

Object Models

The main points in this SPIM are follows:

-a *contact* is an entity that you communicate with. To initiate communication with a contact, you can use an address that is appropriate for you chosen communication method (a phone number for a phone call, an email address for email, and so on).

-an *appointment* is a scheduled event that involves you and one or more contacts with which you will interact. The venue specifies how the interaction will occur (physical meeting, teleconference, and so on).

-a s*chedule* is a list of appointments that fall in a given time span (this afternoon, today, tomorrow, and so on).

Figure 5 captures the relationships between these objects.

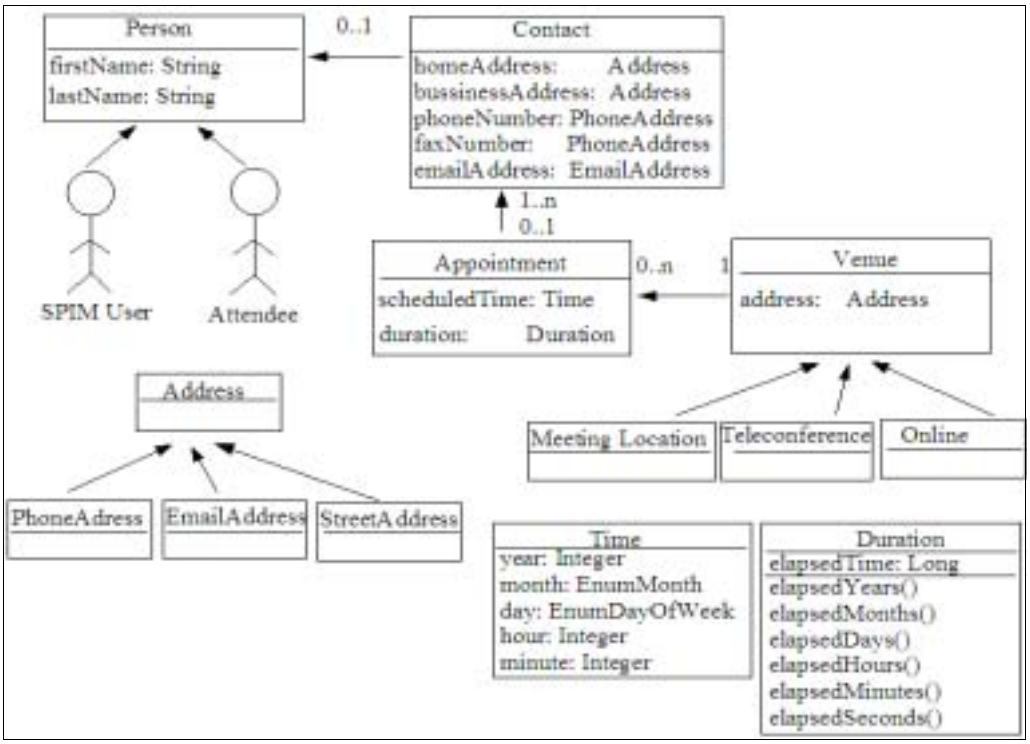

Figure 5

Anatomy of the Application

The database is named spim.mdb and this database has four tables: Appointment, Contact, LocaleLabels and MediaLabels. The table Contact has Id, FirstName, LastName, HomePhone, FAXPhone, BusinessPhone, CellPhone, EmailPrimary, EmailSecondary, HomeAddress1, HomeAddress2, HomeCity, HomeState, HomeCountry, HomePostalCode, BusinessAddress1, BusinessAddress2, BusinessCity, BusinessState, BusinessCountry, BusinessPostalCode, Nickname, s_ColLineage, s_Generation, s_GUID, s_Lineage, Appointment. MediaLabels has Id, MediaType, s_ColLineage, s_Generation, s_GUID, S_Lineage, Appointment. LocaleLabels has Id, LocaleLable s_ColLineage, s_Generation, s_GUID, S_Lineage, Appointment. Appointment has Contact, Id, Owner, StartTime, EndTime, Medium, Locale, Description, Gen_Description, s_ColLineage, s_Generation, s_GUID, s_Lineage. Each table has a generated primary key (Id) that uniquely identifies the entity. The Appointment table uses a Contact ID as a foreign key to the

Contact table. This models whom the appointment is with. Auxiliary tables map integer-coded values for meeting locales and media to text string.

Root element for the application contains subelements ownerId, Appointment and Contact. Each of these contains various attributes and subelements that capture all the information needed to generate HTML or VXML. Appointment contains ScheduledTime, AttendeeContact, AttendeeName (which means PersonalNameType), locale (which means AddressType), Medium, Description. Contact is compound from name, HomeAddress, BusinessAddress, HomePhone, FAXPhone, BusinessPhone, MobilePhone, E-mailAddress. Each person has a PersonalNameType with Nickname, FirstName and LastName. An AddressType consist of an optional one or two AddressLine followed by a City, State, Country and PostalCode (any or all can be omitted). Most elements are optional to allow for partial information being stored in the database. Furthermore, an integrity constraint (for example, there must be a city) probably should be enforced in the database, not the middle tier.

To associate Appointments and Contacts, XML ids are used. An id is an intradocument link that can be used to uniquely identify an XML element within a single document.

When data is retrieved through a database query, the data is mapped into a single XML format. The XML format consists of a header (which identifies the SPIM user) and one or more Appointment elements followed by one or more Contact elements. Depending on the context for the query, the application determines how to interpret the relationships between Appointments and Contacts. For example, if the user is reviewing his or her calendar, there may be multiple appointments and multiple contacts (corresponding to the appointments within a given time period). For the Running Late function in the application, there will be at most one Appointment (scheduled for the current time) and one corresponding Contact.

The main menu of the application has three choices: Calendar, Address Book, To-Do List and one link Running Late like in Figure 6.

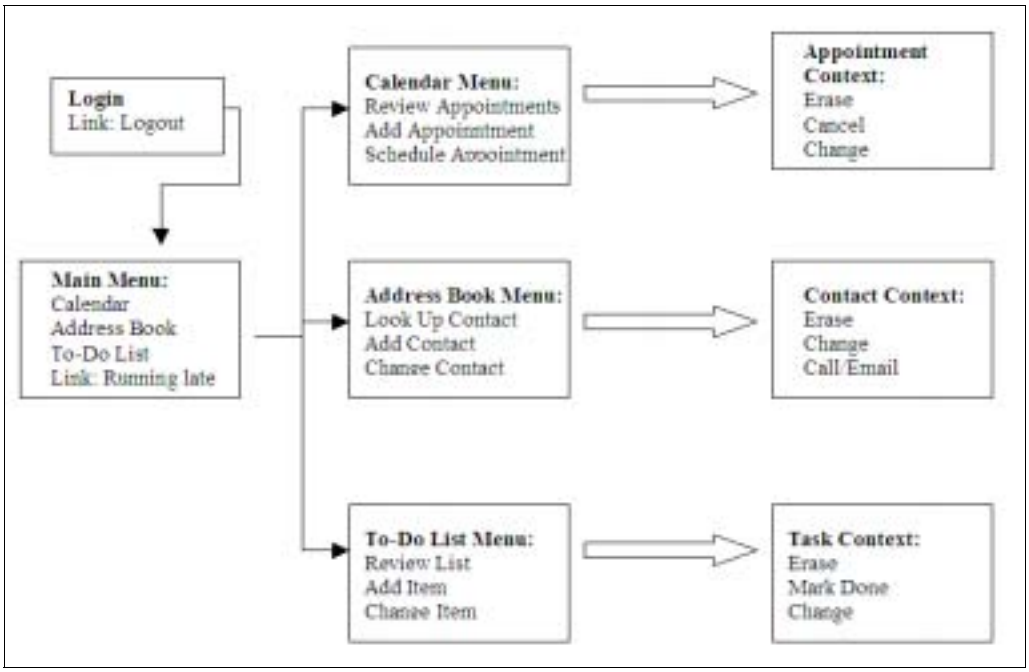

Figure 6

When the VoiceXML interpreter executes the document, it starts executing the first form or menu in the document.

A <menu> consist of a spoken prompt and a set of choices. When the <menu> is executed, the prompt is read and then the system listens for the user to say one of the choices. The choices are specified in the body of the <choice> tag. When a choices is recognized, the system starts executing the form or menu identified by the value of the next attribute, which is interpreted as an URI.

A menu or form field specifies a prompt/response exchange between the caller and the computer. A link is a transition that the caller can activate at any time the computer is listening for a response. Whereas a menu choice generally corresponds to only one spoken phrase, a link specifies a grammar. The grammar may be as simple as a single word or phrase, or it may be more complex.

In a VoiceXML browser the Running Late Link will lead you to a dialog that starts with a prompt like this: C (Computer): You have a scheduled appointment with Mary Abay at 14:00. Do you want to call home, call business, fax a message, or send an email notice?

Future Refinements

In future it could be done some refinements like the following example:

-Appointments are currently modeled with just two people: the owner and the contact. In reality, there would be one owner and a set of contacts.

-Appointments might be with people who are not registered contacts. That is, people who don't have an entry in the Contact table.

The Purpose of the Application

The purpose of the application is to demonstrate the feasibility of the architecture and provide a model for developing a "real" implementation. You can use this application and the information as a concrete pedagogical example, or you can actually install the components and make it work on your own computer.

This application brings together representative technologies necessary to create a small-scale, multititer, working Web application. For the implementation I used: XML Spy, A Web browser, a VoiceXML browser, a Web server and a servlet container, an ODBC data source. XML Spy is the XML IDE used to develop and bench-test XML, XSL, and schemes. The Web browser is Microsoft Internet Explorer. The VoiceXML browser is the IBM WebSphere Voice Server SDK. JRun plays the part of Web server and servlet container. Microsoft Access, acting as a nonproprietary ODBC/JDBC data source, plays the role of database. Apache Cocoon provides the framework for generating and transformation XML in the Web server.

References:

1. Abbott, K.R. (2002). Voice Enabling Web Applications: VoiceXML and Beyond, Apress, Berkely, USA.

2. Phillips, L.A. (2000). Special Edition Using XML, QUE Corporation, USA.

3. Becchetti, C. ad Ricotti, L.P. (1999) Speech Recognition Theory and C++ Implementation, John Wiley & Sons, West Sussex, England

4. Rabiner, R L. and Juang, B.H. (1993) Fundamentals of Speech Recognition, Prentice Hall, New Jersey, USA

Author:

Marieta Gâta - North University of Baia Mare, Romania, Email address: marietag@ubm.ro