AURAL INTERFACES TO DATABASES BASED ON VOICEXML

Beat Signer, Moira C. Norrie, Peter Geissbuehler, Daniel Heiniger
Institute for Information Systems
Swiss Federal Institute of Technology (ETH)
CH-8092 Zurich, Switzerland
signer@inf.ethz.ch, norrie@inf.ethz.ch

Abstract As part of a general framework for the development of global information systems, we include support for the development of aural interfaces. The framework uses an object-oriented database for the management of application, document content and presentation data. The access layer is based around an XML server and XSLT for document generation from default and customised templates. Specifically, aural interfaces are supported through a VoiceXML server that provides the speech recognition and synthesis mechanisms, together with XSLT templates for the generation of VoiceXML. In this paper, we describe the implementation of a generic voice browser for application databases as well as the development of a customised aural interface for a community diary managing appointments and events.

Keywords: web databases, aural interfaces, voice browser, VoiceXML

1. Introduction

We are rapidly moving towards an information society in which communities of users demand access to all forms of both personal and shared information from their work places, homes and on the move. The resulting application systems can be classified as *global information systems* that enable ubiquitous access to shared information.

To support the development of such global information systems, we require modern data management frameworks with a general and flexible interaction layer that allows access from various forms of client devices including not only desktop workstations, but also mobile phones, Tablet PCs and PDAs. We have therefore extended our OMS Java data management framework [Kobler and Norrie, 1999] with a generic database access layer to support universal client access. The resulting eXtensible

Information Management Architecture (XIMA) framework [Signer et al., 2001; Kobler et al., 2001] uses XML as an intermediate representation format and the Extensible Stylesheet Language Transformation (XSLT) for document generation from default and customised templates. The XIMA framework is very flexible and it is easy to support new forms of client devices since only the final rendering step to the client's desired output format, based on the XSLT, has to be adapted.

Recent innovations, such as the Voice Extensible Markup Language (VoiceXML), encouraged us to extend the XIMA framework with a user interface based on speech recognition and synthesised voice output [VXML, 2002]. Using voice as a potential input and output medium opens up many new possibilities in terms of, not only mobile access to information, but also access for the disabled. Aural interfaces just require the use of a regular telephone to access information stored in our databases. Today one can find telephones almost everywhere and mobile phones are far more portable and accessible than computers. Therefore the support of voice browsers was just a logical step towards realising our vision of *ubiquitous information — information for everyone, everywhere.*

Even in the case of WAP-enabled mobile phones, the use of a voice interface may be a much more convenient means of accessing information. Navigation by voice is by far more pleasant and faster than the use of touch-tone input or entering information using the small keypads of mobile WAP phones. Also, in some situations, voice output may be preferred over visual output. For example, a person may perform a manual task, while simultaneously receiving information via a voice interface. Just think of an employee driving to his office by car: He can listen to the news on the company's web portal site while his eyes are concentrating on the traffic.

With respect to the disabled, voice-enabled applications are valuable to users who can either not use their hands for keyboard input or their eyes to process visual output. Further, voice interfaces require no special instruction or experience. They also allow new forms of human-computer interaction based on a combination of visual and voice interfaces. We can build applications which are either fully based on voice or use speech technology to augment existing graphical user interfaces.

In this paper, we present an overview of the XIMA framework and describe how it has been extended with an aural interface based on VoiceXML. The enhanced framework strongly supports the database driven development of voice-enabled applications, which finally can be deployed using one of the numerously available voice server platforms [BeVocal, 2002; VoiceGenie, 2002]. We begin in section 2 by presenting our

overall vision and the main advantages of aural user interfaces. After discussing existing applications which already make use of aural input and output, we outline how VoiceXML can be used to simplify the development process of voice based applications. We describe the eXtensible Information Management Architecture (XIMA), which already supports different types of client formats (HTML for regular web browsers, WML for mobile WAP phones, and CHTML for PDAs and NTT DoCoMo's imode devices) in section 3. In section 4, we then point out how the XIMA framework has been enhanced with a generic voice browser interface that allows voice access to all our OMS Java databases. Section 5 explains how a customised voice interface can be developed by adapting the generic voice browser for a specific application and we describe how this was done for a community diary. Concluding remarks are given in section 6.

2. Aural Interfaces

There are several good reasons for investigating aural interfaces as a new access form for database management systems. As discussed in the introduction, such interfaces may provide benefits in terms of mobility and universality, extending potential user communities to those without access to computers or special devices and also to the physically handicapped, unable to use conventional input or output devices.

Aural interfaces are most frequently used for retrieving information from a database or invoking operations on the database. However, there are also applications and circumstances where voice input may be appropriate for the entry of new data or the editing of existing data. For example, entering a new person in an electronic address book could be done by spelling out the name and the phone number. This enables entry to be done quickly and conveniently "on the spot" rather than having to later enter the data when a computer plus keyboard or business card scanner are available. Also an aural interface enables a "hands-free" and "eyes-free" mode of interaction which may be useful in many practical work situations or for the augmentation of other visual applications.

For various technical reasons, it is only in recent years that it has become possible to create useful human-computer voice interaction. Due to advances in digital speech processing technologies, it is now possible to support the application and deployment of a variety of speech technologies for aural interfaces. Figure 2 shows how voice input is processed and an aural response is generated by a computer using a speech recognition component, a language analyser, an application server connected to a database and a speech synthesiser (text-to-speech or TTS).

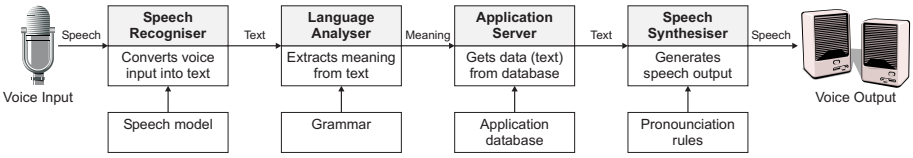

Figure 1. Voice Processing Components

The speech recognition is performed by a speech recogniser engine that processes incoming audio signals and tries to determine the most likely combination of basic spoken sound patterns (phonemes) that represent the audio input (word recognition). The language analyser then compares the resulting output to the active grammar containing all potential words and phrases which can be detected. Based on the extracted meaning, information can be fetched from an application database and a reply is built in textual form. Finally, based on prosody analysis and text-to-phoneme conversion, the output speech waveform is generated by the speech synthesiser. Today's synthesised voice output still sounds artificial, but it should continue to improve in the near future with the use of new voice modelling techniques and computing performance improvements.

The task of the application developer was further simplified when major vendors of speech technologies and tools (e.g. IBM Speech ML and Motorola VoxML) formed the VoiceXML Forum in 1999 and agreed to develop a common standard. This resulted in the specification of the VoiceXML standard which was published in March 2000.

The task of designing an aural interface to an application is not a trivial undertaking. Good speech user interfaces are significantly different from their visual counterparts. A lot of tasks that are visually possible — such as scrolling through a long list of elements — cannot be directly mapped to an aural user interface. Several projects try to adapt existing graphical user interfaces and make them accessible using voice browsers [Anderson et al., 2001; Freire et al., 2001]. The problem of such an approach is that the navigational concepts used by graphical applications are not suitable for access by voice recognition. Another problem is the audible output of large amounts of information. Since speech is transient (you hear it once and then it is gone), users tend to forget important information provided at the beginning of long dialogues. Books are available giving general guidelines for the design of aural interfaces [Balentine and Morgan, 1999].

The advantage of using the XIMA framework for the development of speech-enabled applications is that we adopt an "information-based"

approach. This means that we start by developing a general information model for the application in question using OM, an object-oriented data model [Norrie et al., 1996]. The OM model supports object and association constructs and encourages the application information to be represented in terms of fine-grained "information units" which are linked together by associations. Representing information objects as small, externally linked objects allows for a greater flexibility when it comes to generating document content and access patterns for a specific interface. Through the clear separation of content and visualisation, it is possible to use completely different navigational patterns for voice and visual user interfaces. In addition, the representation of application information in terms of small units of information provides a good basis for the development of aural interfaces.

3. eXtensible Information Management Architecture

XIMA uses the OMS Java system for the management of application, document content and presentation data. OMS Java is an application framework for the Java programming environment that implements the abstractions and operations defined in the OM object-oriented data model. For the purposes of this paper, it is not necessary to describe OM or OMS Java in detail. However, it is important to appreciate the main constructs of the OM model and we therefore begin this section with a description of these.

Each OM object is an *instance* of one or more *types*. A type is a set of attribute and method properties. If a type t_2 is declared to be a subtype of type t_1, then the set of properties of t_2 is composed from the set of properties defined for t_2 and those inherited from t_1. The composition is in line with usual inheritance mechanisms in object-oriented languages. We refer to the set of properties defined for t_2, and not inherited, as the *type unit* of t_2.

If an object is an instance of type t_1, then it will have a corresponding *information unit* that gives the values of all attribute properties defined in the type unit of t_1. Access to an object is always associated with a *type context* that specifies one of its types. The corresponding type instance for that object will be constructed dynamically from the appropriate information units, i.e. from the information unit for the context type together with those inherited from supertypes.

A *collection* is an object that represents a semantic grouping of objects. It has a name, a member type and an extent which is the set of ids of all member objects. Only those objects which are instances of

the collection's member type can belong to the collection. The member type also specifies the type context for objects accessed through that collection.

For a given collection C, we can define one or more *subcollections* C_1, C_2,... This specifies a containment relation between the collections — every object that is a member of C_1 must also be a member of C.

An association is a *binary collection* together with constraints that specify a *source collection*, a *target collection* and cardinality constraints over the source and target. A binary collection B is a special form of collection in which the members are pairs of object ids, e.g. (O_1, O_2). Correspondingly, the member type of B must be a binary type of the form (t_1, t_2). Assume that collections C_1 and C_2 are specified as the source and target collections, respectively, of B. Then for every pair (O_1, O_2) that belongs to B, O_1 must be a member of C_1 and O_2 a member of C_2.

Although association constructs are present in a number of descriptive models such as Entity-Relationship models [Batini et al., 1992] and UML [Booch et al., 1998], they are rarely supported as a separate abstraction construct in models of operational systems. Our experience in working with object-oriented databases has shown the benefits of having a dedicated construct in order to be able to associate objects directly instead of via attributes and to perform operations over these associations.

The OMS Java framework was designed and developed to support application development through the provision of a high-level application programming interface based on the abstractions of the OM model. Further it was defined with openness and extensibility in mind which allowed us to extend and re-engineer the framework to provide universal client access and web content management facilities based on XML technologies.

In Figure 2, we present the architecture of the resulting eXtensible Information Management Architecture (XIMA). For a specific application, all client access is via a single Java servlet — the Entry Servlet. This means that all requests can be sent to the same URL, rather than having different URLs for different types of client devices. The Entry Servlet detects the user agent type from the HTTP request header and delegates the handling of the request to the appropriate servlet. For example, in Figure 2, we show servlets to handle requests from HTML browsers, WML browsers running on WAP-enabled mobile phones and voice browsers based on VoiceXML.

The request handling servlets then access the database by connecting to an OMS Java workspace via the OMS Java API. The connection may either be direct or via the OMS Java XML server. Direct connections

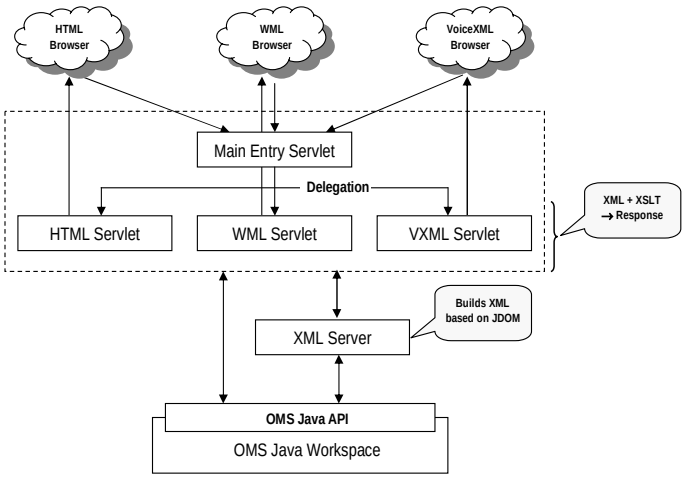

Figure 2. eXtensible Information Management Architecture

deal with requests that do not involve data retrieval such as checking the membership of an object in a collection or performing updates. Any requests that involve the retrieval of data go through the XML server. The XML server forwards requests to the OMS Java workspace and generates XML representations for any data objects. For all database constructs such as objects, instances, attributes, etc. there exist JDOM wrapper components, which generate a document object model (DOM) for the corresponding database object. Every wrapper object itself may use other wrapper components during its transformation process. For example, the JDOM wrapper of an *instance* database entry uses the *type*, *attribute*, *method* and *link* wrappers to add types, attributes, methods and links to the instance's document object model. The resulting DOM tree is returned to the requesting servlets (in Figure 4 of section 4 we present an XML instance representation). Finally, the servlets use the appropriate XSLT templates to transform the XML results to the desired client format.

There are a few points to note in this general architecture. First, we are not storing any XML documents, but rather generating them dynamically from the application data which is stored according to the information model. Since the information model is a loosely-coupled graph model based on object collections and associations, this gives much more flexibility than the rather restrictive hierarchical-based models imposed by XML structures. At access time, a particular hierarchical view on data is derived and the appropriate XML content generated. Second, since what we are interested in is the XML structure rather than

the document per se, what we generate as an intermediate form for the XSLT processor is actually the associated DOM (document object model) structure rather than the XML document.

The generated DOM tree conforms to an OMS/XML document type definition (DTD) which defines a representation of all database objects at a fairly high level, so that the resulting XML documents provide a good basis for the succeeding XSLT transformations. There exists a second "low level" OMS/XML DTD which is only used to transfer database content from one OMS database to another OMS database. Nevertheless, it is sufficient to implement only a wrapping mechanism for the "high level" document object model representation. The resulting DOM structure can easily be transformed to an XML document conforming to the "low level" DTD by applying a simple XSLT transformation.

Using generic XSLT templates for the various client devices, we are able to provide generic browsers and editors for the current set of client types. Adding a new client type involves implementing the corresponding servlet and writing the appropriate XSLT templates.

Specific application interfaces are supported through the customisation of XSLT templates. As an example, we have developed an application for a *community diary* that manages appointments and address book entries for a user community.

Many existing tools for web site engineering either rely on a specific form of client such as an HTML browser, or they require that different forms of clients be handled separately meaning that it requires significant development time to port an application to another form of client device. The goal of XIMA was to develop a universal client framework that requires minimal development effort to access an application via another form of client device. The effort required to support a particular client device goes into optionally customising the presentation to suit the device in question. Such general frameworks are particularly important when one considers how dynamic the world of mobile devices currently is. There are many questions as to whether technologies such as WAP will really become established or whether they will be replaced by either new technologies or new devices better capable of handling existing technologies.

4. Generic VoiceXML User Interface

The Voice Extensible Markup Language (VoiceXML) is an application of the Extensible Markup Language (XML) which enables interactive access to the web through standalone voice browsers or regular phones. Its major goal is to bring the advantages of well established web-based

content delivery techniques to Interactive Voice Response (IVR) applications. Application navigation works by voice recognition or the use of Dual Tone Multi Frequency (DTMF) keypad input. The resulting aural responses feature digitised audio or synthesised speech output.

During the design phase of the aural user interface, we used the IBM Voice Server Development Kit [IBM Voice Server, 2002] which fully supports the VoiceXML standard. It further allowed us to test the application on a personal computer using a standard microphone prior to installing it on a commercial voice server platform.

In this section, we focus on the design of the generic voice browser which enables any OMS Java application database to be accessed using speech input and output [Geissbuehler and Heiniger, 2001]. We therefore had to design general interactive sequences for navigating through a database and accessing the constructs of the OM model i.e. objects, collections and associations.

Most of the general voice interface's output is generated dynamically based on XIMA's XML server results. To avoid a discordant mixture of digitised audio and synthesised speech, we decided to use synthesised speech for all outputs. For a specific application, as described in the next section, it still makes sense to use digitised audio output for the representation of static content (e.g. help information) to improve the user experience.

To define valid navigation commands, we use rather simple grammars consisting of only basic words and phrases. In our generic voice browser, almost all grammars are built dynamically based on the returned XML content and therefore do not contain complex sentences. The main advantage of a simple grammar is better voice recognition performance. On the other hand, we had to spend additional effort in designing the system prompts so that they guide the user to only choose one of the valid responses specified by the corresponding grammar. Keeping an end-user within "bounds" is an important aspect of overcoming fragility in the interface and may have strong effects on user satisfaction [Roe and Wilpon, 1994]. Since speech is slow, only essential data should be spoken. The prompts in our general aural interface are rather terse and normally not longer than a few seconds. This leads to compact dialogue structures, which is relevant if simple grammars are used as is the case here.

A problem of the generic voice browser is that collections of objects can become quite large and therefore an aural representation of such a collection's content may take some time. As stated in [Schmandt, 1994], voice response systems must strive to minimise the amount of time required to retrieve information: They must be brief, selective about

content, and they should almost always be interruptible. For this reason, we decided to use a full-duplex (barge-in) implementation which lets a user interrupt the voice output if he hears the desired object or if he already knows his next step before the aural output is finished.

We further improved the navigational component by introducing special commands which are always active and therefore accessible in any situation. The user can get information about these always-active command by using the *commands* keyword. Nevertheless we did not use any of the built-in commands of the IBM Voice Server Development Kit to remain independent of a specific voice server framework.

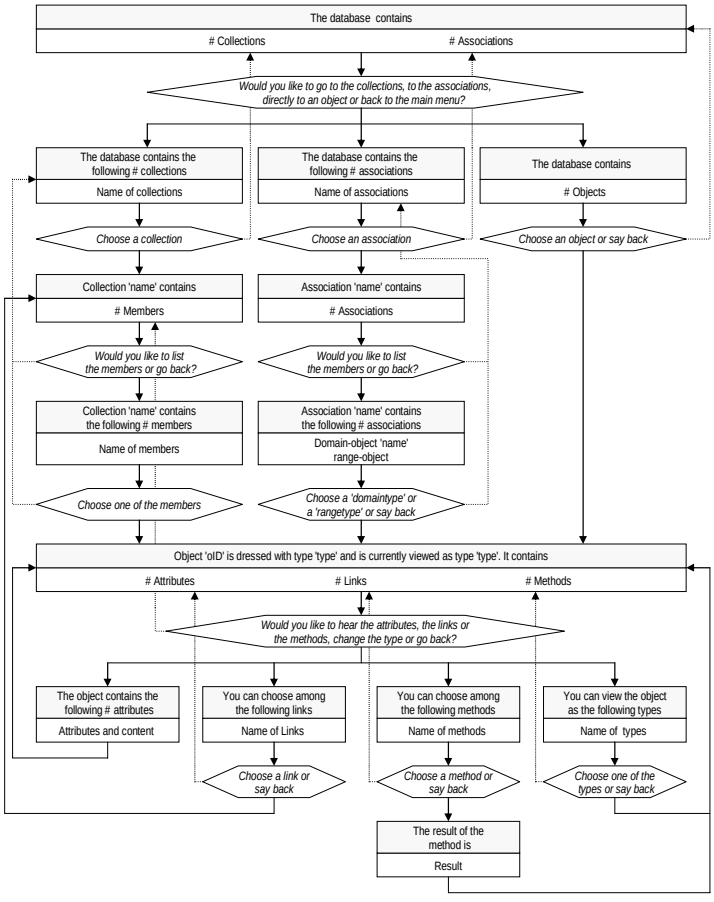

Figure 3. Generic Information Flow

An information flow chart of the generic voice interface is shown in Figure 3. The introductory prompt reads the number of collections and associations that the OMS Java database contains. A user can choose

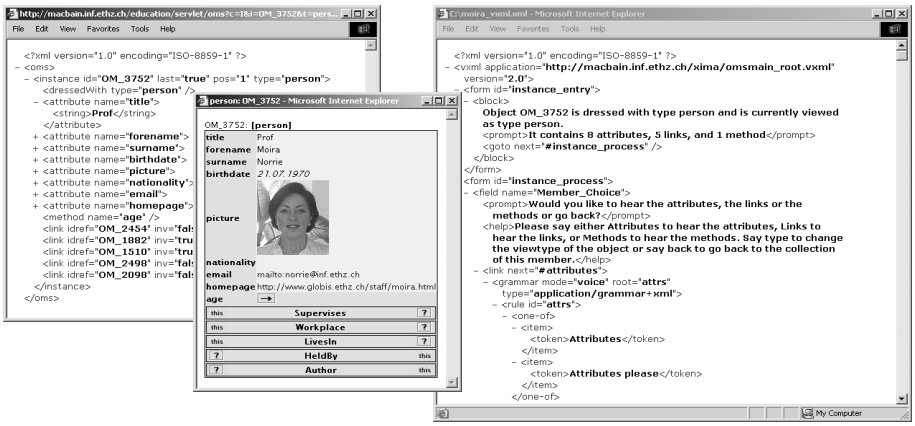

Figure 4. XML, HTML and VoiceXML Instance Representation

to go to the collections, the associations or directly to an object if he already knows the object of interest. One problem is to decide how individual objects can be referenced since it is not reasonable to expect the user to know the object ids. We have handled this by specifying an attribute within the OMS Java database, which acts as a *label* for each object type. For example, one might choose the *name* attribute of a type *person* to be the label attribute. Then a person instance could be accessed directly by giving the name of the person as the input.

An object instance contains attributes, methods and links to other objects. The links are generated dynamically based on the association information and allow easy browsing between related objects. An example of an object instance is shown in Figure 4. On the left hand side we show the XML representation of a database instance. The HTML transformation of this XML document is presented in the centre. A VoiceXML representation of the same XML content, which can be interpreted by a voice browser, is shown on the right hand side.

A first prototype of the general voice browser has been tested by several users. The test persons were given error recovery sheets on which they could mark errors or ambiguities in the application control flow. A first problem detected during this initial test phase was that the voice recognition did not always work correctly. The most common reason for voice recognition errors was the occurrence of multiple similar sounding words in our grammar. By refining the static part of the grammar and using synonyms for the "problematic" words, we could significantly improve the voice recognition performance. It was not possible to solve this problem for dynamically generated grammars. To support users in

the case of voice recognition errors, we provide the always-active *back* command allowing them to browse back step by step.

Another common problem was that some users were not understood because they added additional filler words such as *please* to their commands. By extending our grammar with optional parts handling such filling constructs, we could further improve the voice recognition reliability.

In our first prototype, the *help* command always led to an aural presentation of the always-active commands. For most of the inexperienced test persons, this was not sufficient because they expected context dependent help for any potential input they had to provide. In the redesigned version, the application therefore does not switch into a separate help dialogue (help mode) when the user demands help, but uses self-revealing help. This means that suitable help information is integrated within each application dialogue and provides a specific output for the current situation. With this approach, no mode management and no audio formatting is necessary. The provided help is more specific and is closer to the dialogue style than in a help mode.

Users were often unsure as to whether the application had understood a command or if they had to repeat it. We therefore introduced an aural feedback after most of the user inputs. The application confirms every input with an *OK* prompt.

The current generic voice browser functions quite well and after a short amount of time, users are able to locate desired information within a database. However, the interaction can have a tendency to rapidly become tedious and, if little is known about the database content in advance, it is difficult for a user to keep track of their navigation within the database. In any case, it is important to remember that this generic voice browser is mainly intended as a first step in the rapid development of application specific interfaces as described in the next section.

5. Application Specific Interfaces

To build a new application, the developer first prototypes the information model using OMS Pro [Würgler, 2000], a prototyping system for the OM model. During this early development stage, the model can already be tested by entering concrete data. In the next step, the prototype is ported to the OMS Java database management system which requires the implementation of a simple Java class for each type defined in the information model. The model and data can be exported from OMS Pro and imported in OMS Java using the Data Definition Language (DDL) and the Data Manipulation Language (DML), respectively. The OMS

Java application database is then already accessible by XIMA's generic voice browser. A final step is to customise the dialogues to improve the user experience as described in the following paragraphs.

The generic voice browser has been customised for a specific *community diary* which manages appointments and events for a user community and further offers some basic email functionality. The initial development phases of this application involved the steps described above and it was then the task of the developer to customise the XSLT stylesheets to refine the voice user interface. Note that this involved no changes to the results returned from XIMA's XML server component.

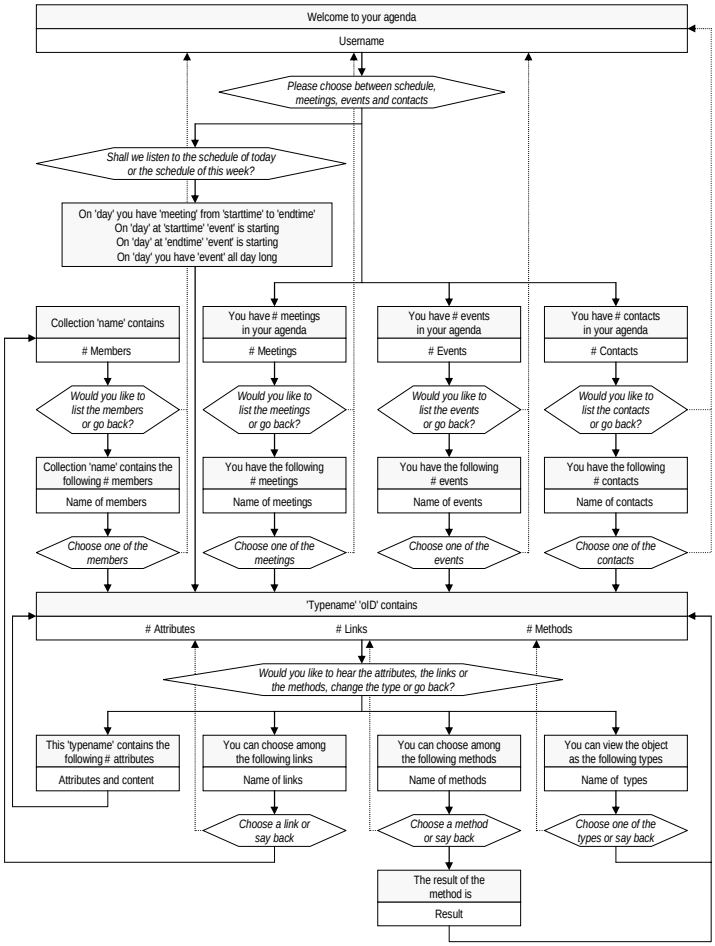

Figure 5. Diary Information Flow

By importing the generic XSLT stylesheets, we could inherit the base functionality from the existing stylesheets and only had to override specific parts. The final diary application allows a user to receive contacts information and to get a schedule of a day or the whole week. An information flow of the diary application with the customised dialogues is shown in Figure 5.

It is no longer possible to get access to all collections and associations defined in the diary's information model as explained in the description of the generic voice browser. Rather a few collections such as *meetings* or *contacts* can be accessed. We further had to add a component which allows access to the schedule of a day or a week as generated by operations defined within the database. While some parts of the generic voice browser interface have been refined, one can see that the lower part of the information flow diagram is exactly the same as the one shown before and therefore could be inherited from the generic voice browser interface.

In another project, we developed a special voice application service based on the XIMA framework for the Swiss Federal Institute for Snow and Avalanche Research (SLF) in Davos [Geissbuehler and Heiniger, 2002]. SLF gathers information from automatic weather stations and manual observers throughout the Swiss Alps and generates both national and regional avalanche forecasts. These reports are made available to members of the public, mountain guides and resorts in various ways including publication in newspapers and on the web. There is also a limited telephone service for national forecasts based on audio reports generated by someone reading prepared texts. Our application based on XIMA and VoiceXML will enable them to offer a more flexible telephone service at lower costs and with faster dissemination of information.

6. Conclusions

We have presented an approach for developing aural interfaces to applications using a general framework for universal client access to databases. The framework is based on an approach to application development that relies heavily on the central notion of an information model and the extensive use of prototyping and generic templates.

Our initial experiences have been very positive. We were surprised at the quality of speech-based applications that could be developed within a relatively short period of time given the approach that we followed. Not only that, but this was achieved as part of a general framework that also provides access to the applications from a range of desktop and mobile devices. At the same time, these experiments highlighted

many challenges in terms of the best ways of designing speech dialogues — especially in the case of generic database browsers — and we intend addressing these in future projects.

References

Anderson, C. R., Domingos, P., and Weld, D. S. (2001). Personalizing Web Sites for Mobile Users. In *Proceedings of the 10th International World Wide Web Conference*, Hong Kong.

Balentine, B. and Morgan, D. P. (1999). *How to Build a Speech Recognition Application: A Style Guide for Telephony Dialogues*. Enterprise Integration Group.

Batini, C., Ceri, S., and Navathe, S. B. (1992). *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings.

BeVocal (2002). BeVocal, Inc., http://www.bevocal.com.

Booch, G., Rumbaugh, J., and Jacobson, I. (1998). *Unified Modelling Language User Guide*. Addison-Wesley.

Freire, J., Kumar, B., and Lieuwen, D. (2001). WebViews: Accessing Personalized Web Content and Services. In *Proceedings of the 10th International World Wide Web Conference*, Hong Kong.

Geissbuehler, P. and Heiniger, D. (2001). Voice-Based User Interface for the eXtensible Information Management Architecture (XIMA). Semester work, Institute for Information Systems, ETH Zurich.

Geissbuehler, P. and Heiniger, D. (2002). Providing A Telephone Avalanche Warning Service Using XIMA, VoiceXML and WML. Diploma thesis, Institute for Information Systems, ETH Zurich.

IBM Voice Server (2002). Voice Server SDK, http://www-4.ibm.com/software/speech.

Kobler, A. and Norrie, M. C. (1999). OMS Java: Lessons Learned from Building a Multi-Tier Object Management Framework. In *Proceedings of the Workshop on Java and Databases: Persistence Options*, Denver, USA.

Kobler, A., Norrie, M. C., Signer, B., and Grossniklaus, M. (2001). OMS Java: Providing Information, Storage and Access Abstractions in an Object-Oriented Framework. In *Proceedings of the 7th International Conference on Object-Oriented Information Systems (OOIS'01)*, Calgary, Canada.

Norrie, M. C., Steiner, A., Würgler, A., and Wunderli, M. (1996). A Model for Classification Structures with Evolution Control. In *Proceedings of 15th International Conference on Conceptual Modelling (ER'96)*, Cottbus, Germany.

Roe, D. B. and Wilpon, J. G. (1994). *Voice communications between humans and machines*. National Academy Press.

Schmandt, C. (1994). *Voice Communication with Computers, Conversational Systems*. Van Nostrand Reinhold Verlag.

Signer, B., Grossniklaus, M., and Norrie, M. C. (2001). Java Framework for Database-Centric Web Engineering. In *Proceedings of the 4th Workshop on Web Engineering (in conjunction with 10th International World Wide Web Conference)*, Hong Kong.

VoiceGenie (2002). VoiceGenie Technologies, http://www.voicegenie.com.

VXML (2002). VoiceXML Forum, http://www.voicexml.org.

Würgler, A. (2000). *OMS Development Framework : Rapid Prototyping for Object-Oriented Databases*. PhD thesis, ETH Zurich.