

# Integrating Speech in Web Browsers

## Semester Project

Alex O. Prudencio Arispe  
Prof. Martin Rajman  
Assist. David Portabella

June 22, 2003

### Abstract

We have designed a Java Applet and an ActiveX control that integrate natural language interaction in web browsers. These applications are based on web services and a VoiceXML platform. In this report we describe the technologies used in this project and the steps that we have followed to implement these applications. Finally we explain the problems that have been encountered and those that haven't been fixed yet.

## 1 Introduction

In the last decade Internet and the Worldwide Web have evolved from a simple document-oriented platform to an interactive service-oriented one. At the beginning, the web was mainly intended to share text documents. Nowadays, the web has other functions and has to provide different services which require interaction with users. To perform an interactive service, Web-based applications are integrating all input and output modalities (multimodal interaction). Recently, natural language is being considered as a new modality to interact with the user in web applications.

Even if natural language interaction has already been used in some applications, for instance VoiceXML language has been used in some telephony centers, still there are only a few attempts to integrate natural language interaction in Web browsing applications [1]. At the EPFL's Artificial Intelligence Laboratory (LIA <sup>1</sup>) a prototype of a multimodal web application

---

<sup>1</sup><http://liawww.epfl.ch>

using VoiceXML and web services was developed. However, all the necessary applications require a complex set up.

This project's main objective is to implement a single module (plug-in) that wraps all the necessary components required to integrate multimodal interaction in Web browsers. In the development phase we have explored different technologies and we have finally chosen to implement a Java Applet because of its portability, and an ActiveX control for its high integration with Microsoft's Internet Explorer.

## 2 Multimodal Interaction

The Worldwide Web Consortium (W3C) defines multimodal interaction activity as extending the Web user interface to allow multiple modes of interaction, offering users the choice of using their voice, or an input device such as a key pad, keyboard or other input device. For output, users are able to listen to spoken prompts and audio, and to view information on graphical displays. Multimodal applications should be able to adapt to changing device capabilities, user preferences and environmental conditions [2].

In this project we integrate natural language interaction for output in Web-based applications. This modality will let users to listen spoken prompts. Our prototype application uses keyboard and mouse as input to interact with the web page content. Speech could be also implemented as an input device.

### 2.1 VoiceXML

VoiceXML is a language designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of DTMF (Dual Tone Multi-Frequency) key and spoken input, recording of spoken input, telephony, and mixed initiative conversations. Its major objective is to bring the advantages of web-based development to interactive voice response applications [3].

VoiceXML is an XML application, hence it promotes portability of services through abstraction of platform resources. To facilitate interoperability, a common grammar format is required, namely the XML Form of the W3C Speech Recognition Grammar Specification (SRGS).

## 2.2 Elvira VoiceXML platform

Elvira<sup>2</sup> is a VoiceXML platform which takes VoiceXML 2.0 specification as the reference for development. Elvira implements a VoiceXML interpreter which parses the input by means of active grammars and produces a semantic result.

Elvira is designed following a component-based architecture, i.e. this application is formed by set of components each one of whom do an specific task (see figure 1). The main component is called *Elvira Core* (elvira.dll), which interprets VoiceXML and fully controls all the other components. It is also responsible of the resource management for synchronous or asynchronous fetching of resources.

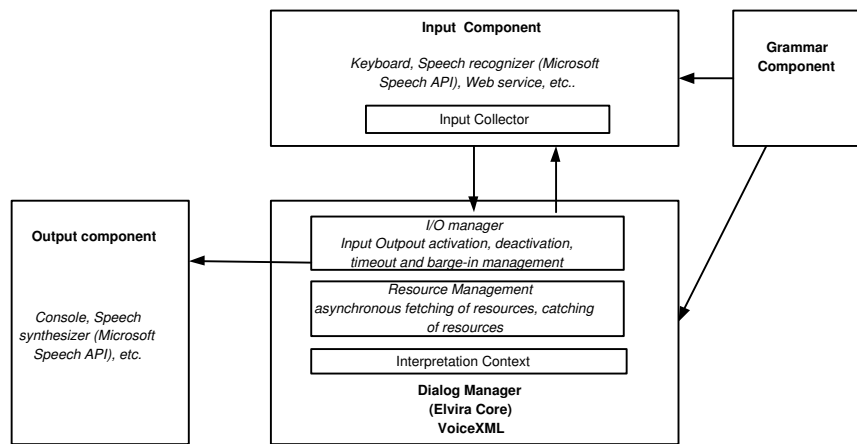


Figure 1: Architecture of the Elvira platform

Elvira Core also supports a mechanism for calling external functions written in C++. Elvira External Functions can be called from VoiceXML `<OBJECT>` tag, this allows a high integration of the external functions.

The framework for manipulation with components is called LSD Component Framework. The heart of the LSD Component Framework is the LSD Component Core (lsdcompcore.dll). It registers all available components (dynamic libraries with a special public interface) that are located in a specified directory. After registration, the core can create instances of the

<sup>2</sup><http://gin2.itek.norut.no/elvira/>

components. Every component is characterized by its unique name and by its category.

Elvira has also an Input/Output manager which is responsible for controlling the activation and deactivation of inputs and outputs. This part is also responsible for timeout and barge-in management and event queue management, including external events.

### **3 Web Services**

Web services are a stack of emerging standards such as SOAP or WSDL, that describe a service-oriented, component-based application architecture. Web services are components that semantically encapsulate discrete functionality and are distributed and accessible to applications over standard Internet protocols.

A Web service is identified by a URI(Uniform Resource Identifier), and its public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. Each web service component describes its own inputs and outputs in a way that other software can determine what it does, how to invoke its functionality, and what result to expect in return. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages transmitted by Internet protocols such as HTTP.

## **4 Review of technologies**

During the development phase of this project we have explored different technologies in order to find an efficient solution to deploy a module capable to integrate multimodal interaction in web browsers. We were confronted to choose between Java based technologies or Microsoft's COM technology, i.e. develop an ActiveX control.

### **4.1 Java technologies**

Java is a modern object oriented language which offers a number of advantages to develop portable code because of its virtual machine architecture.

### 4.1.1 Java Applets

Java Applets are programs intended to be embedded in Web pages. This is one of the first and most popular technologies that let web browsers to offer interactive content.

Generally, an applet has limited access permissions to machine resources and runs in what is called a "sandbox". The java virtual machine confines the applet in a restricted space in order to prevent malicious applications.

Signed Applets are applets that are granted permissions for some insecure or dangerous actions such as writing files, accessing native code through JNI, etc. Signed applets have to be distributed in a JAR (Java Archive) file and are signed using a certificate which can be delivered by companies such as Verisign<sup>3</sup>.

### 4.1.2 Java Native Interface

Java Native Interface<sup>4</sup> or JNI, is a library that allows java programs to communicate with programs written in some other languages also know as native code. JNI is useful when we've got some compiled code written in other language and we don't want to translate it to Java.

JNI also allows programs written in native code to use the java virtual machine to run java code. This is particularly useful because Java features a number of advantages over code written in languages like C or C++. For example network and multi-threading is easier to code and to maintain in the Java programming language than in most other languages.

### 4.1.3 JavaBeans

A *bean* is a reusable software component based on Sun's JavaBeans specification [6]. One of the particularities of the JavaBeans architecture is that beans can be manipulated visually in development environments such as JBuilder or Visual Café.

JavaBeans are, to a certain point, a Java equivalent to Microsoft's COM architecture. Beans are components that offers interfaces so other applications (containers) can call methods or read and modify variables through them.

---

<sup>3</sup><http://www.verisign.com>

<sup>4</sup><http://java.sun.com/j2se/1.4.1/docs/guide/jni/index.html>

Most beans share certain common features such as introspection, which allows a builder tool to analyze how a bean works. Customization allows the user to alter the appearance and behavior of a bean. Beans can also fire events and inform their container about both the events they can fire and the events they can handle.

## 4.2 Microsoft's COM and ActiveX

Microsoft's COM (Component Object Model) is a specification for building component based software that can be dynamically interchanged. COM provides the standard that components and client applications follow to ensure that they can operate together through interfaces.

Other Microsoft's technologies such as ActiveX or OLE implement COM architecture[5]. An ActiveX control is an application that runs in a container such as Internet Explorer. Using ActiveX components permits to display interactive content in web pages.

In this project we have employed the ActiveX Template Library (ATL). ATL provides a set of reusable template<sup>5</sup> classes for building COM components and ActiveX controls.

## 5 Web Browsers

Since this project's goal is to integrate speech interaction in web browsers, we had to review the technologies that could be used for the most common browsers. Two major web browsers are available in almost all platforms, Microsoft's Internet Explorer and Netscape Navigator. Unfortunately, this two applications do not share the same technologies and have different standards to implement add-ins.

### 5.1 Microsoft's Internet Explorer

Internet Explorer is may be the most popular web browser. Every Windows operating system comes with Explorer as a default web browser and it is also available for other platforms like MacOS.

---

<sup>5</sup>C++ Templates are mechanisms for generating functions and classes based on type parameters, thus a single class or function can operate on data of many types, instead of having to create a separate class for each type.

Internet Explorer supports Java Applets by means of a Java plug-in which can be downloaded automatically if it's not already installed. Indeed, when the browser encounters an `<APPLET>` or an `<OBJECT>` tag that references the Java Plug-in, it searches the CLSID attribute (Unique Identifier of a component), to verify if the plug-in is installed. If the plug-in is not installed, it downloads it from Sun Microsystems' web page.

ActiveX technology is also supported by Internet Explorer and this is the principal characteristic that distinguish this web browser from other browsers that in general do not support ActiveX controls. Using ActiveX controls in web pages can integrate other functionalities in the web browser and allows multimedia and interactive content.

## 5.2 Netscape Navigator

Netscape is the other popular web browser. Netscape is a Mozilla-based browser <sup>6</sup> and is freely distributed. It is available for different platforms.

Netscape supports Java Applets via the Java Plug-in which is called when it encounters an `<APPLET>` or `<OBJECT>` tag. When the `<OBJECT>` tag is found, Netscape does not understand the Unique Identifier used along with the CLSID attribute. Mozilla-based browsers support the Netscape Plug-in architecture, which is not COM based like ActiveX (and thus, not invoked via a Unique Identifier) but rather, MIME type based. MIME (Multi-Purpose Internet Mail Extensions) is an extension of the original Internet e-mail protocol that lets people use the protocol to exchange different kinds of data files on the Internet. Mozilla-based browsers support the use of the object element along with a MIME type. The `<EMBED>` element has been used to invoke plugins since the early days of Netscape browsers. Typically, the `EMBED` element is nested within an object element, such that the outer object element invokes an ActiveX control for Internet Explorer, whereas the inner embed element invokes a Netscape Plugin.

Officially Netscape does not support ActiveX controls, but there are some non-official plug-ins that allows Mozilla-based browsers like Netscape Navigator to launch ActiveX controls <sup>7</sup>.

---

<sup>6</sup>Mozilla was Netscape Navigator's nickname before the product had a commercial name and, more recently, the name of an open source public collaboration aimed at making improvements to Navigator.

<sup>7</sup><http://www.iol.ie/~locka/mozilla/mozilla.htm>

## 6 Implementation of a Multimodal component

At the LIA we already had a prototype that implements multimodal interaction based in web services and VoiceXML (see figure 2). The VoiceXML interpreter is part of the Elvira VoiceXML platform.

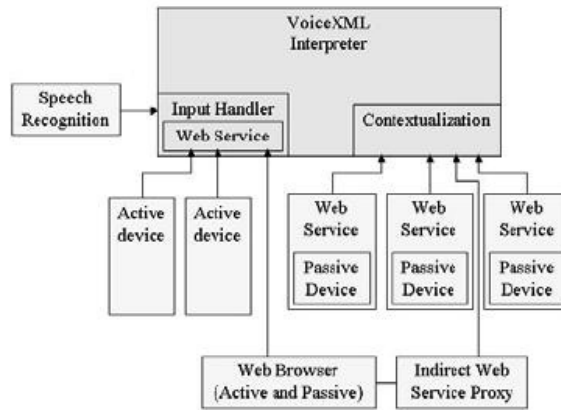


Figure 2: Framework for multimodal VoiceXML [4]

This prototype synchronizes Web applications with VoiceXML-dialogues through multimodal input/output using Web services [1]. This architecture has the advantage that multimodal interaction is not limited within the Web browser because we can wrap other kinds of input and output devices by Web services. Input devices may provide data which is then converted into streams recognized by the VoiceXML grammar, or they can be queried when the Dialogue Manager requires contextual information.

In this project we have chosen to work in Windows environment because we had to use Microsoft's Speech API. Another reason is that we have also implemented an ActiveX control which is not supported on Linux. Nevertheless, our Java-based solution could be portable to Linux since Elvira is also available on this platform.

In the first phase of this project, we chose to develop a java based application. First we tried to implement a Java bean component that wraps all the modules, specifically the Indirect Web service proxy and the interpreter. Unfortunately the JavaBeans specification <sup>8</sup> tells that JNI calls are not al-

<sup>8</sup><http://java.sun.com/products/javabeans/docs/spec.html>

lowed in order to ensure portability, which is the main purpose of Beans technology. As we were constrained to use JNI calls to initialize Elvira, we've decided to implement a Signed Java Applet.

In the second phase, we have considered the possibility of making an ActiveX control since COM technology offer a higher integration with Internet Explorer. However, ActiveX technology has a limited portability and is not supported by other browsers.

## 6.1 Elvira Multimodal Input

We have implemented an input component for Elvira, that gathers data from a Web Service server implemented in Java. We wrote this component in C++ and we used JNI to instantiate the Web service.

Since Elvira platform has a component-based architecture, we had to implement the necessary interfaces to communicate with the Elvira core. We have also used some of the tools that come in the Elvira Development Package to ensure a correct synchronization between this component and Elvira core, namely we used the Thread and Event classes.

This component creates an instance of the Java Virtual machine to call the Java class that launches the Web Service. The java class that implements the Web service is called *ElviraInput* and is part of the package `ch.epfl.lia.multimodality`. This class launches the web service that collects data, which is then treated by the VoiceXML interpreter.

Synchronization between the web service implemented in Java and the input component is achieved using a java class called Pipe that defines two synchronized methods (figure 3). Thus when the web service receives some data, it calls the synchronized method *send* which then sends a signal to the synchronized method *get* which is called by the multimodal input component.

## 6.2 Developing a Multimodal Applet

We have decided to implement a signed java Applet that initializes the VoiceXML interpreter and the Indirect Web service Proxy, see figure 4 for the architecture diagram.

During the first weeks we made some test programs in order to verify that we were able to call native functions from an applet. In order to access the system resources and make native calls, we have to sign the applet.

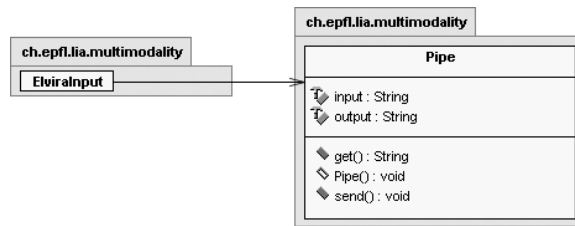


Figure 3: Class Pipe UML description

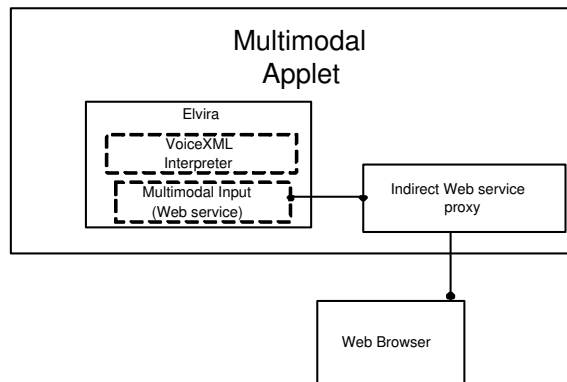


Figure 4: Multimodal Applet schematic model

### 6.2.1 Signing an Applet

To distribute a signed an applet we need a certificate delivered by a trusted institution which certifies that the applet is safe. For testing purposes, we have created a self-certified applet using the `keytool` and `jarsigner` programs, which are included in the Java Standard Edition Development Kit (Java SDK 1.4.1).

To create a certificate and a private key we use the `keytool` program.

```
keytool -genkey -keystore MyCert.store -alias multimodality
```

Once we have generated the certificate, we have to create a policy file to define the permissions. Java Plug-in uses a policy file of the form:

```
keystore "file:MyCert.store", "JKS";
```

```
grant signedBy "multimodality" {  
    permission java.security.AllPermission, signedBy "multimodality";  
};
```

We can use the *policytool* application to generate this file. Any signed code that can be verified with the public key associated with the alias *multimodality* is now granted these permissions.

Now we can sign the applet which has to be included in a JAR file.

```
jar cvf multimodality.jar *.class  
jarsigner -keystore MyCert.store multimodality.jar -alias multimodality
```

When the web browser launches the applet, a pop-up dialog appears and requests the permissions defined in the policy file. This way, the applet can access the necessary resources on the client's machine.

### 6.2.2 Calling native functions using JNI

Since the VoiceXML interpreter is written in C++, we have to use JNI to initialize it. Thus we had to create a dynamic library (DLL) that initializes the VoiceXML interpreter and which can be loaded from the Multimodal Applet.

To call a native function from a signed applet, the DLL has to export this function using a special JNI header that is generated with the program *javah*. We have tested this procedure with some simple functions written in C using the JNI specification and which were compiled in a DLL.

After verifying that a signed applet can call native functions, we began to tailor the interpreter for our needs. We have implemented a DLL that exports a function that receives the VoiceXML grammar's URI as argument and that initializes all the components, namely the Elvira core, the VoiceXML interpreter and the input and output modules. This DLL is loaded by the Multimodal Applet.

We have got some problems during the implementation of this DLL because Elvira Core couldn't initialize the necessary modules. The problem was that Elvira looks for its libraries in the path of the program that loaded Elvira LSD component core, in this case the java virtual machine. To solve this problem we submitted our requirements to Elvira's developer who changed

the proprietary code in order to implement an interface that enables us to indicate the components' location. However we didn't succeed to launch the interpreter from the applet.

At this point we have got an applet with the model described in figure 5. The Applet that we have implemented doesn't instantiate the VoiceXML interpreter yet. We have succeeded to register Elvira's components from the applet, but we're not allowed to launch the interpreter.

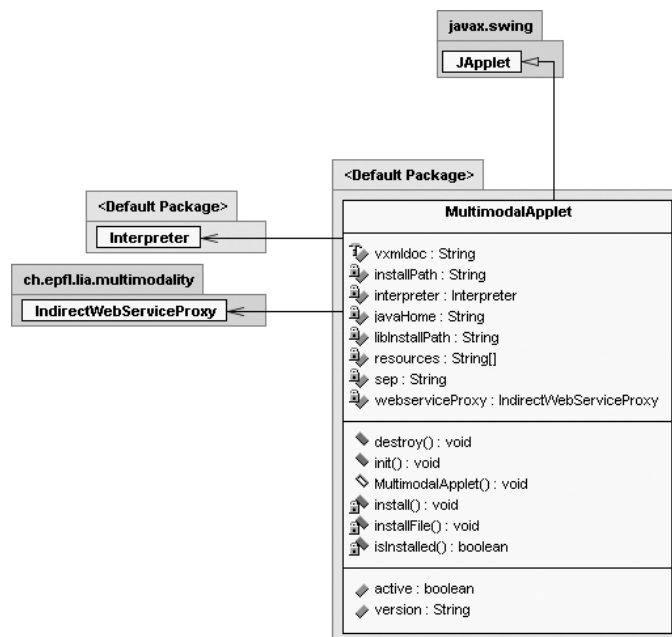


Figure 5: Multimodal Applet UML description

### 6.2.3 Deploying the applet

After implementing all the modules that we needed to ensure interoperability between native and java codes, beside the problem with launching the interpreter, we concentrated ourselves in the deploying strategy. The ideal solution would be to distribute the applet in one JAR file that includes the multimodal applet and Elvira's native libraries. However we were confronted to the problem that JNI calls cannot access DLLs stored in JAR files.

We decided to deploy our applet in two JAR files, one containing the native

libraries, which is quite big and have to be cached in the client's machine. The second JAR file contains the java applet which before starting the Indirect Web Service Proxy and the VoiceXML interpreter, extracts the native libraries from the first JAR package and installs them on the machine.

The Java Plug-in offers the possibility of caching applets<sup>9</sup> that are contained in JAR files, thus improving loading time in next references. When invoking the Java Plug-in, we can specify the attribute `<PARAM NAME="cache_archive">` to indicate which JAR files should be stored in the cache directory specified by the Java Plug-in. There is also the attribute `<PARAM NAME="cache_version">` which is used to tell the Java Plug-in that a new version of the JAR files is available. In fact, there is now version number attached to a JAR file in the web server, the `cache_version` is used only to trigger an update. Otherwise the Java Plug-in would continue to load the file stored in cache.

### 6.3 ActiveX control

The fact that our Java Applet can not be distributed in a single package without managing some file installation led us to try ActiveX technology. We decided to use java for its simplicity and its portability, but ActiveX offers the advantage of being more integrated to Microsoft's Internet Explorer. We decided to implement a ActiveX control based on the ActiveX Template Library (ATL).

The architecture of this component is described on figure 6. In general, writing an COM component is not easy because of its architecture which requires to implement different interfaces. This process is simplified using the ATL library and Microsoft Visual Studio .NET.

We have implemented an ActiveX control that creates two threads. One of them uses JNI to initialize the Indirect Web service Proxy which is written in Java. The other thread initializes the Elvira interpreter. Here we have got the same problem with the VoiceXML interpreter, i.e. we couldn't initialize the interpreter.

We have used Microsoft Visual Studio .NET to create this component. This development environment has many tools that facilitates the process of writing an ActiveX control. Visual Studio automatically creates the code needed to define the interfaces specified by COM. It also generates the files needed to register the component. Then we completed the class that implements the interface showed by our ActiveX control, namely *IMultimodalCtl*.

---

<sup>9</sup>[http://java.sun.com/j2se/1.4.1/docs/guide/plugin/developer\\_guide/applet\\_caching.html](http://java.sun.com/j2se/1.4.1/docs/guide/plugin/developer_guide/applet_caching.html)

We define three methods through this interface. The first one is the method *run* which launches the Indirect Web service proxy and the interpreter. The second method is *get\_State* which indicates if the interpreter is running. Finally *put\_VxmlDoc* defines the VoiceXML grammar location.

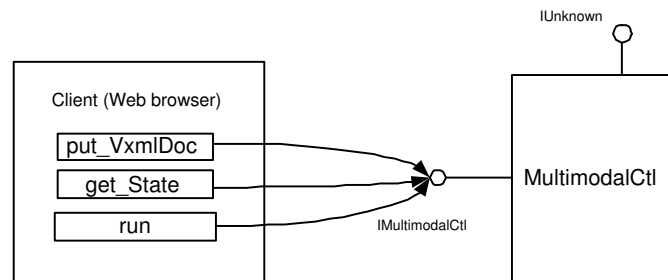


Figure 6: ActiveX control model

Deploying an ActiveX control for Internet Explorer is easier than an Applet because Explorer has already the necessary tools to cache and install the components needed by the control. Generally an ActiveX control is distributed in a Cabinet file (CAB file) which contains the installation and registration directives.

### 6.3.1 Deploying an ActiveX control

As we mentioned before, one of the main advantages of creating an ActiveX control is that it can be easily integrated in Microsoft software, for instance Internet Explorer offers a number of facilities to integrate ActiveX control in Web pages. Microsoft has developed their software using COM architecture so some of Microsoft's applications are containers in which other components can be called.

To distribute a component over the internet the easiest way is to make a signed cabinet file (CAB file). The advantages of a CAB file are compression and, if used with an INF file (Information file), the bundling of all necessary code together. When Internet Explorer finds an <OBJECT> tag, it looks for the CLSID number in the system's registry, and if the component is not installed, it downloads it from the location specified by the parameter CODEBASE. Once the component is downloaded, the browser installs it using the directives included in the CAB file. There are 2 formats to include the installation instructions in a CAB file. The first one is writing an information

file (.INF file) and the second one is writing an Open Software Description (.OSD file).

### 6.3.2 Information file

An information file <sup>10</sup> (.INF file) provides installation instructions that the Internet Component Download service provided in Microsoft's Internet Explorer 3.0 or later uses to install and register software components downloaded from the Internet, as well as any files required by the components.

Beside an Information file, Cabinet file (.cab) can also contain an Open Software Description (OSD) file (or both). CAB files may contain software components such as ActiveX Controls , a .dll, or an .exe application.

When a cabinet file containing an INF file is referenced by the `CODEBASE` attribute of an object element, Internet Explorer automatically treats the cabinet file as a software distribution unit.

INF files provide the ability to create customized software installation instructions, which include registry entries and destination directories. By pointing to the URLs of files to download, an INF file provides instructions that Internet Explorer uses to install your software components. The INF file also specifies the files that need to be downloaded and set up for the component to run.

For our project we have written an INF file to install the multimodality component. The CAB file includes Elvira's libraries and the package `multimodality.jar` which is called by the `multimodal.input` component to instantiate the Web service input.

### 6.3.3 Creating and Signing a CAB file

Internet Explorer's default security settings do not allow to launch unauthenticated ActiveX controls. This is due to the fact that ActiveX components have complete access to the machine resources, thus dangerous code can easily cause damages in the client machine.

To distribute ATL controls via the Internet, we should package them as signed Cabinet (CAB) files and a Software Publisher Certificate is needed to sign a CAB file. To obtain such a certificate, one must apply for a certificate to a Certification Authority. For testing purposes, we have created

---

<sup>10</sup><http://www.microsoft.com/technet/prodtechnol/ie/reskit/ie6/part4/c16ie6rk.asp>

a certificate using tools available with Microsoft's Visual Studio .NET and in the CAB SDK <sup>11</sup>.

To create a CAB file we use the CABARC application. We have included the libraries that the ActiveX control depends on. Once we have created the CAB file, we sign it as follows :

```
makecert -sv myNew.pvk -ss myNewStore myNew.cer
cert2spc myNew.cer myNew.spc
signcode -spc myNew.spc -v myNew.pvk multimodality.cab
```

## 6.4 Problems Encountered

We have already mentioned that we had some problems in trying to integrate Elvira in our applications. We have been working to fix these problems but unfortunately we didn't succeed to solve them all. Since Elvira core has proprietary code, we had to work with the owner to make some changes in the code.

As we mentioned before, Elvira needs to register its components. The first problem that we found was that Elvira looks for its components in the directory of the application that loads `lsdcompcore.dll` library. This was a serious problem because in our case, when we launch the applet, it is the Java Virtual Machine that loads `lsdcompcore.dll`, hence this component looks for the other Elvira components in the Java Virtual Machine's directory. For the ActiveX control we have the same problem, but in this case , "lsdcompcore" component looks for the components in Internet Explorer's path.

We have submitted our problem to the code's developer, and he has written an interface that allows us to register components in a specific directory. Thus, now we can specify the components' directory path in which Elvira has to look for.

We have tested the methods written to register components in a specific directory, and it seems to be working right. Even if we have succeeded to register components located in other directory, we can't launch the VoiceXML interpreter from a dynamically loaded library. These problem seems to be due to the fact that we can not register other component inside a dynamically loaded library. We didn't have enough time to find what exactly the problem is.

---

<sup>11</sup><http://msdn.microsoft.com/library/en-us/dncabsdk/html/cabdl.asp>

## 7 Summary and conclusions

In this project we have tested different technologies to implement a module that could be embedded in a Web Browser in order to offer multimodal interaction. This application would be instantiated every time that a web page attempts to implement multimodal interaction with the user.

Our first solution is to implement a Java applet. Applets are a java technology that lets the browser to run an embedded application in a web page. We have created an applet that installs the necessary libraries to launch the Elvira VoiceXML interpreter and starts the web services required by the interpreter to communicate with the browser. This solution has the advantage of being portable to different browsers because the Java Plug-in is available for different browsers and platforms.

During the development phase of the applet, we have been confronted to some problems that could not be resolved because of limitations of the technology. Namely, we couldn't deploy the interpreter native libraries in a JAR package because the java virtual machine can't load libraries stored in compressed packages.

We have also implemented a COM-based solution, i.e. an ActiveX control, that has the advantage of being more integrated with Internet Explorer. It seems that ActiveX controls have a better performance in terms of speed and memory requirements than Applets. Distribution is also simplified using CAB files.

We couldn't finish our prototype due to some problems encountered with Elvira. Since it has proprietary code, we did not have enough time to coordinate with the code owner to solve our problem. We think that the registering process causes some problems. We have already solved some problems of the registering process, namely now the LSDcompcore component can register components from a given directory. However, even if the components seem to be registered, we can not instantiate the interpreter from a dynamically loaded library.

We have defined two possible architectures to distribute a module capable to integrate speech in web browsers. We couldn't make further tests, but once the problem with Elvira is solved, we think that both solutions seem appropriate to deploy a component that offers multimodal interactivity, namely natural language interaction.

## References

- [1] Susan Armstrong, Alex Clark, Giovanni Coray, Maria Georgescu, Vincenzo Pallotta, Andrei Popescu-Belis, David Portabella, Martin Rajman, Marianne Starlander, March 2003: Natural Language Queries on Natural Language Data: a Database of Meeting Dialogues, *NLDB2003 conference*.
- [2] Dave Raggett, 2002, Multimodal Interaction Working Group Charter, Internet: <http://www.w3.org/2002/01/multimodal-charter.html>
- [3] World Wide Web Consortium, Voice Extensible Markup Language (VoiceXML) Version 2.0, Internet: <http://www.w3.org/TR/voicexml20>
- [4] David Portabella, 2003, Speech dialogue and multimodality, Internet: <http://liawww.epfl.ch/portabel/dialogue>
- [5] Rogerson, Dale. *Inside COM*, Microsoft Press, 1998.
- [6] Horstmann Cay S. and Gary Cornell, *Core Java Vol.2. Advanced Features*, Sun microsystems press, 2002.
- [7] Norton, Ivor. *Beginning Java 2*, Wrox Press, 2002.
- [8] Schildt, Herbert. *The complete reference C++*, Osborne McGraw Hill, 1998.