Proyecto GEMINI: Plataforma avanzada de generación y ejecución de aplicaciones de diálogo hombre-máquina

R. de Córdoba, L.F. D'Haro, F. Fernández, V. Sama, J.M. Montero,
R. San-Segundo, J. Macías-Guarasa, J. Ferreiros, J.M. Pardo

Grupo de Tecnología del Habla. Departamento de Ingeniería Electrónica. UPM.
E.T.S.I. Telecomunicación. Ciudad Universitaria s/n, 28040 Madrid, Spain.
E-mail: {cordoba, lfdharo, efhes, vsama, juancho, lapiz, macias, jfl, pardo}@die.upm.es

Resumen

Presentamos una plataforma completa de generación semiautomática de sistemas de diálogo hombre-máquina en la que, a partir de una descripción de la base de datos del servicio, de un modelo de flujo de los diferentes estados de la aplicación final y una interacción guiada paso a paso con la intervención del diseñador, se generan simultáneamente diálogos para acceder a los datos de dicho servicio simultáneamente en varios idiomas y en dos modalidades, voz y web. Se describe cada uno de los módulos de la plataforma que hace que esto sea posible, haciendo un énfasis especial en las distintas estrategias seguidas para reducir el tiempo que se necesita para el diseño de la aplicación. Así mismo, presentamos el trabajo realizado en la plataforma en tiempo real, especialmente en cuanto a la adaptación de la misma para que interprete el lenguaje estándar de descripción de sistemas de diálogo VoiceXML, cuya difusión no deja de crecer, a través del intérprete de código abierto llamado OpenVXI.

Palabras claves: sistemas automáticos de diálogo, multimodalidad, multilingualidad, reconocimiento de habla, VoiceXML.

1. Introducción

El proyecto Gemini (Generic Environment for Multilingual Interactive Natural Interfaces, IST-2001-32343), objeto de esta comunicación (Gemini 03, Hamerich 04, D'Haro 04, Córdoba 04), es un proyecto de dos años (2002-2004), financiado por la Unión Europea en el que interviene nuestro grupo representando a la UPM.

El objetivo fundamental del proyecto ha sido el desarrollo de un sistema de generación de aplicaciones de diálogo de forma semiautomática partiendo de una descripción de la base de datos. Para ello se ha diseñado y creado una herramienta mediante la cual el diseñador puede especificar totalmente la aplicación de una forma rápida, flexible, intuitiva y cómoda, que permite reducir significativamente el tiempo necesario para su desarrollo y la productividad se maximiza.

Una aplicación de diálogo se puede definir como cualquier aplicación que ofrece de forma automática un servicio al usuario (contenido en una base de datos) mediante un diálogo hombre-máquina, estando la máquina en una ubicación remota. Por ejemplo, los datos de un cliente del banco para una aplicación bancaria, los datos de todos los viajes posibles en una aplicación de agencia de viajes, etc.

Este diálogo se puede hacer mediante varias modalidades: voz por teléfono (mediante técnicas de reconocimiento de habla y conversión texto-voz), a través de Internet, etc. Se habla en estos casos de multimodalidad, que es uno de los objetivos del proyecto: conseguir que, con un mismo diseño de la aplicación, nuestra plataforma genere simultáneamente el código para ofrecer el servicio a través del teléfono mediante voz y a través de Internet. Otro de los grandes objetivos del proyecto es la multilingualidad: hemos trabajado en cuatro idiomas: alemán, griego, español e inglés.

La primera parte del trabajo ya fue presentada en esta conferencia en el año 2003 (Córdoba 03). En la presente comunicación vamos a centrarnos en las nuevas funcionalidades y mejoras incorporadas a la herramienta de generación de diálogos. También se presenta el trabajo realizado en la plataforma en tiempo real. Por un lado, veremos la necesidad de adaptación de la misma al lenguaje estándar de descripción de sistemas de diálogo VoiceXML, que se genera como salida en la plataforma de generación de diálogos para la modalidad de voz. Para conseguirlo, se ha aprovechado el intérprete de código abierto llamado OpenVXI de la empresa SpeechWorks, ahora adquirida por Scansoft (OpenVXI 04). Así mismo, se destacarán las limitaciones encontradas en dicho intérprete.

2. Antecedentes

El presente trabajo es la continuación del proyecto europeo IDAS (Córdoba 01, Lehtinen 00), en el que se desarrolló un demostrador capaz de dar un servicio vocal interactivo de páginas blancas en el que se proporcionaba a los usuarios números de teléfono (o fax) de particulares y de empresas; Además de la experiencia obtenida en un proyecto con RENFE (San-Segundo, 01) en el que se hizo un sistema de reservas e

información de trenes. En esta ocasión hemos sido más ambiciosos al crear una plataforma multimodal y multilingüe para la creación de aplicaciones de diálogo independientes del servicio y el sistema en tiempo real del mismo.

Como punto de partida estudiamos las características, alcances y limitaciones de diversas herramientas comerciales y educativas con el fin de aportar nuevas ideas en este campo; encontrando que las principales limitaciones se centraban en la estandarización, aceleración del diseño, escalabilidad y capacidad del servicio al usuario final, por lo que la información que presentamos en esta comunicación se centran en la superación de estas limitaciones.

3. La herramienta de generación de diálogos

Se ha creado una herramienta de generación de diálogos llamada AGP (Application Generation Platform, Plataforma de generación de aplicaciones) que es un conjunto integrado de herramientas y asistentes con los que automatizar el diseño de aplicaciones de diálogo. La arquitectura trata de conseguir los siguientes **objetivos fundamentales**:

- La participación del diseñador debe ser mínima o muy reducida.
- Se busca la utilización de estándares.
- Se deben proporcionar al diseñador una serie de bibliotecas o módulos con los que resolver situaciones cotidianas.
- Se busca independencia del servicio y la aplicación.

En la Figura 1 puede observarse la arquitectura elegida para el AGP.

1. Nivel superior. Aquí, el diseñador especifica los aspectos globales relacionados con la aplicación y los datos que va a utilizar. Tiene 3 asistentes:

Asistente de descripción de la aplicación (ADA): se definen las características básicas de la aplicación, como los idiomas, modalidades, bibliotecas, etc.

Asistente del modelo de los datos (DMA): se introducen las descripciones de las clases y atributos de la base de datos del servicio.

Asistente de conexión con el modelo de datos (DCMA): se definen las funciones de acceso a la base de datos específica. Su objetivo es independizar la base de datos de la aplicación.

2. Nivel intermedio. Es donde se define el diálogo de una forma independiente del idioma y la modalidad. Tiene tres módulos:

Asistente para el flujo de diálogo (SFA). Se especifican los estados por los que va a pasar el diálogo (a qué diálogos llama en cada estado) y qué datos (slots de la aplicación) se le van a preguntar al usuario en cada uno de ellos.

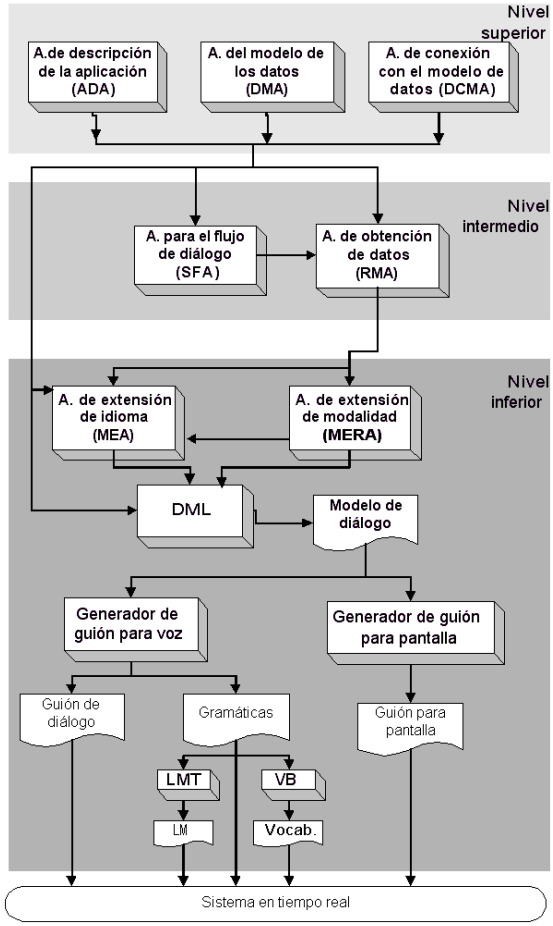

Figura 1. Arquitectura del AGP.

Asistente de obtención de datos (RMA). Utilizando la información del nivel superior, especialmente los estados definidos en el SFM, el diseñador describe todos los detalles del diálogo. Probablemente sea el asistente más complejo de la plataforma, por lo que será tratado más en detalle en la sección 3.1.

Asistente de modelado de usuario (UMA): Permite definir una serie de niveles de experiencia del usuario, con el fin de darle una atención más personalizada.

3. Nivel inferior (Dialogues Layer), donde se completa el diálogo introduciendo los aspectos dependientes del idioma y de la modalidad. Consta de estos módulos:

Asistente de extensión de modalidad (MERA): se completa el diálogo con los aspectos dependientes de la modalidad (los modos de confirmación, el manejo de errores, interfaz gráfica si es aplicable, cuántos resultados se presentan simultáneamente al usuario, etc., que son todos ellos diferentes entre la modalidad de voz y la de web).

Asistente de extensión de idioma (MEA): se completan los aspectos dependientes del idioma (frases que pronuncia el sistema, vocabularios y gramáticas utilizados en cada idioma, etc.)

Vinculador del modelo de diálogo (DML): simplemente une el resultado del RMA, MERA y del MEA para formar el modelo de diálogo completo. No hay interacción con el usuario.

A continuación, se generan automáticamente los guiones de ejecución en el estándar VoiceXML para voz y en xHTML para web.

Hay dos herramientas más: la Herramienta de modelado de lenguaje (LMT), donde se generan los modelos de lenguaje a utilizar en el reconocimiento, y la Herramienta de creación de vocabularios (VB), con la que se prepara el vocabulario a utilizar en el reconocimiento.

Así mismo, se ha buscado que la plataforma sea independiente del sistema operativo, portable y fácilmente escalable.

- Entorno de programación

Todos los componentes se han desarrollado usando Trolltech QT®, que es un entorno de desarrollo gráfico multiplataforma (Linux, Windows, Mac,…), compatible con C++, que permite al desarrollador escribir un programa ejecutable sobre diferentes sistemas operativos con una simple recompilación.

- GDXML y Rutinas básicas

GDXML (Gemini Dialog XML), es un lenguaje abstracto, orientado a objetos y escalable, basado en XML, que ha sido desarrollado específicamente para este proyecto con el fin de comunicar los diferentes módulos de manera inmediata. Permite la especificación de las diversas modalidades y sus correspondientes flujos y acciones, así como los diferentes elementos más básicos de un dialogo (variables, sentencias de control, acciones de presentación o recuperación de información, etc.).

- xHTML, VoiceXML y OpenVXI

Otra característica muy importante de la plataforma es que emplea una serie de lenguajes de etiquetas de uso común tanto en aplicaciones Web (xHTML) como de voz (VoiceXML 2.0), con lo que se consigue que el sistema sea compatible con los navegadores Web habituales y con las plataformas de voz más usadas.

Vamos a describir detalladamente los asistentes más ligados al modelado del diálogo y las estrategias que hemos seguido para acelerar la definición de un diálogo.

3.1. Asistente de obtención de datos (RMA)

El RMA es un asistente clave en el proyecto, al ser donde se definen en detalle los pasos de los que consta el servicio, es decir, el modelo del diálogo en cuanto a sus aspectos independientes de la modalidad o del idioma. Por ello, en general se trabajará al nivel de conceptos.

En la Figura 2 se observa la pantalla principal del asistente en el que ya está definida una aplicación bancaria, con su diagrama de flujo en árbol.

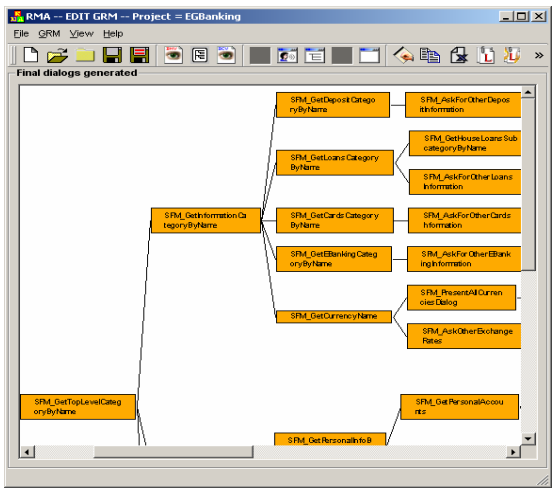

Figura 2. Pantalla principal del RMA.

Estos son los aspectos que contribuyen a acelerar el proceso de diseño:

1. Se crean automáticamente numerosos diálogos para facilitar el diseño

Se utilizan para obtener información del usuario (los llamamos DGet) y para proporcionar información al usuario (DSay.) Se basan en la información del Modelo de datos, es decir, las clases y atributos de los datos del servicio. Para cada clase y atributo se genera un diálogo DGet y uno DSay. Además, hay diálogos DSay de otros tipos: de una clase, que permite seleccionar sus atributos; un DSay configurable por el diseñador; un DSay para cada valor devuelto por una función de acceso a la base de datos (cuando dicho valor no pertenezca ya al Modelo de datos); y diálogos DSay predefinidos (bienvenida al usuario, la despedida, etc.)

2. Se aprovecha al máximo la información de los asistentes anteriores.

Además de la información del Modelo de datos que acabamos de comentar, la información fundamental que aprovechamos procede del Asistente para el flujo de diálogo (SFM). Para cada uno de los estados definidos en el SFM nosotros generamos automáticamente un diálogo. Al editar dicho diálogo, aparece una ventana llamada “Propuestas del SFM” (Figura 3), en la que se ofrecen al usuario toda la información que es específica de dicho estado. Tiene 4 partes:

- Slots preguntados en el estado actual y los estados a los que se salta.
- DGets para obtener del usuario la información de los slots a rellenar.
- Funciones de acceso a base de datos cuyo parámetro (o parámetros) de entrada coincida con alguno de los slots del estado.
- DSays que toman como entrada alguno de los valores devueltos por las funciones anteriores.

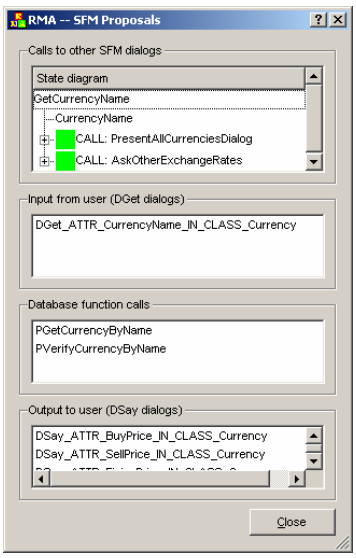

Figura 3. Ejemplo de "Propuestas del SFM".

En la Figura 3 puede verse lo accesibles que están los diálogos que va a necesitar el diseñador para proporcionar información de compra y venta de monedas en una aplicación bancaria; de esta manera se tiene: un DGet para preguntar al usuario el nombre de la moneda (CurrencyName), seguido de PgetCurrencyByName para acceder a la base de datos y obtener la información de dicha moneda, algunos DSays para proporcionar los resultados del acceso a base de datos y, finalmente, una llamada al diálogo siguiente (por ejemplo, AskOtherExchangeRates para preguntar por otras monedas).

3. Se automatiza el paso de variables entre los distintos diálogos.

Tenemos que encadenar distintos diálogos y procedimientos que van recibiendo los datos obtenidos en las etapas anteriores. Hemos diseñado el asistente de forma que el diseño de un diálogo típico se reduzca a tres operaciones de arrastrar y soltar seguidas de aceptaciones de los nombres de variables sin cambiar nada.

4. Tipos de diálogos permitidos

Hay cuatro tipos de diálogos permitidos, que cubren las posibilidades típicas de programación: basados en un bucle, en una secuencia de acciones (o subdiálogos), en información introducida por el usuario y en el valor de una variable.

Se pueden realizar las siguientes acciones en todos los diálogos:

- Definir variables locales y globales.
- Insertar llamadas a otros diálogos
- Introducir estructuras if-then-else.
- Introducir estructuras switch-case.
- Introducir bucles dentro del diálogo.
- Introducir asignaciones entre variables, incluyendo un asistente matemático.

Así mismo, se ofrece la posibilidad de clonar un diálogo, algo muy útil porque muchas veces es necesario crear un diálogo muy similar a uno existente.

5. Introducción de diálogos de iniciativa mixta y de sobre-respuesta

Una aplicación de diálogo moderna debe permitir una cierta iniciativa al usuario. Es lo que llamamos diálogos de iniciativa mixta, en los que se le pregunta al usuario dos o más slots de información a la vez.

Para hacer que un diálogo sea de iniciativa mixta basta con definir sus slots como tales en el SFM, y el RMA genera automáticamente un DGet para iniciativa mixta, que el usuario simplemente arrastrará al definir el diálogo, no tiene que especificar nada más.

Además, hemos querido hacer frente también a lo que se llama *overanswering* (sobre-respuesta o respuesta adicional): cuando al usuario se le pregunta un dato concreto, pero contesta con alguno más. Para sobre-respuesta, la solución es también muy simple. Al soltar cualquier DGet se le pregunta al diseñador si quiere algún otro slot definido en el estado actual o en los siguientes estados en el flujo como dato adicional que puede decir el usuario. En el sistema en tiempo real, antes de cada DGet, se comprueba si el dato a preguntar **ya** se ha obtenido en un estado anterior (lo que sucede cuando el usuario introduce un dato de más). Si es así, se pasa al siguiente estado de la aplicación.

3.2. Asistente de extensión de modalidad (MERA) para habla

En este asistente nos ocupamos de los aspectos del diálogo que son específicos de una aplicación de voz, es decir, situaciones que reciben un tratamiento muy diferente en función de si estamos en web o accediendo por voz. Hemos distinguido dos situaciones especiales: el tratamiento de las confirmaciones y la presentación de listas de objetos al usuario.

3.2.1. Presentación de listas de objetos

En este tipo de diálogos de presentación de listas distinguimos cuatros casos en función del número de elementos que contenga dicha lista:

1. La lista está vacía

Se informa al usuario de que no hay información disponible para saltar a continuación a un estado seleccionado por el diseñador en donde se le pregunta de nuevo información al usuario para conseguir que la búsqueda sea menos restrictiva.

2. La lista tiene un elemento

Se configura un diálogo DSay con la información completa o parcial del único elemento encontrado.

3. Tiene más de un elemento pero menos que un máximo permitido

Es el caso más complejo, ya que se deben mostrar los datos uno a uno en grupos de elementos. Tras cada grupo, se le pregunta al usuario si desea escuchar más

elementos, repetir, comenzar de nuevo o si se desea seleccionar uno en particular, en cuyo caso se puede configurar el diálogo que muestra toda o la información parcial del ítem seleccionado.

Otro problema a tratar es cuando se llega al final de la lista y el usuario no ha escogido ningún elemento, en cuyo caso se informa al usuario y se realiza el mismo proceso que para el caso 1.

4. Tiene más elementos que el máximo permitido.

Al haber muchos elementos, la búsqueda debe ser más restrictiva. Podemos tener dos situaciones bien distintas: en primer lugar, si se han rellenado todos los slots, es necesario que el usuario modifique alguno de ellos para hacerlo más restrictivo (e.g., ha dicho que quiere volar la semana que viene y hay demasiados vuelos), para lo cual se le pregunta al diseñador cuáles desea volver a preguntar. Sin embargo, si todavía quedan slots sin rellenar se continúa con el flujo normal de la aplicación hasta tenerlos todos rellenados y repetir la consulta a la base de datos.

3.2.2. Gestión de las confirmaciones

Para confirmar los datos que genera el reconocedor se emplea un formulario. Se permiten dos tipos de confirmación: Sencilla y Completa. La sencilla es la recomendada para diálogos para los que se presupone una alta fiabilidad, por ejemplo, para respuestas tipo Sí/No. La completa permite utilizar varios niveles de confianza para delimitar el tipo de confirmación: ninguna, implícita, explícita o repetir directamente la pregunta, así como uno o más slots (diálogos con iniciativa mixta y/o sobre-respuesta).

La herramienta selecciona automáticamente los diálogos de obtención de datos (DGet) y genera automáticamente los diálogos para realizar la confirmación implícita, los de confirmación de datos (tipo “sí o no”) y los de corrección, entre otros.

4. La plataforma en tiempo real

Para poder utilizar el resultado del AGP en un sistema en tiempo real (un guión en VoiceXML) necesitamos, en primer lugar, un intérprete. Hemos elegido como intérprete OpenVXI 2.0.1, de SpeechWorks (OpenVXI 04), por ser una solución de código abierto y que por su portabilidad, no requiere de ningún motor de reconocimiento o síntesis de voz en particular, ni es específico de una plataforma telefónica concreta. Como inconvenientes cabe destacar la escasez de una documentación válida, lo que dificulta la introducción de los motores en el intérprete y la ausencia de una implementación de referencia con funcionalidad real.

Para las funciones relacionadas con entrada/salida (reconocimiento, síntesis y control telefónico) proporciona unas interfaces incompletas que podemos modificar para adaptarnos a las necesidades de cada plataforma. Vamos a describir ahora el trabajo realizado en estos módulos.

4.1. Módulo de reconocimiento de habla

Se ha hecho un gran esfuerzo en adaptar nuestro software de reconocimiento a OpenVXI. El sistema admite la carga dinámica y la utilización de distintos reconocedores, incluso de forma simultanea. En este sentido el sistema posibilita el tener modelos y gramáticas de reconocimiento diferentes en función de la pregunta al usuario, pudiendo adaptarlos a distintas situaciones: habla continua, habla aislada, dígitos conectados, fechas, etc.

La implementación que hemos realizado de la interfaz de reconocimiento de OpenVXI tiene las siguientes características:

1. Gestión y registro de errores y de eventos (i.e. fin de temporización).

Se han implementado los mecanismos de respuesta ante eventos como la solicitud de ayuda por parte del usuario, la solicitud de salida del sistema por parte del usuario en un momento determinado, la solicitud de repetición de la pregunta o del prompt formulado por el sistema, o la respuesta del sistema ante la no-detección de habla durante un proceso de reconocimiento.

2. Distintos canales de entrada

Hemos diseñado el módulo para poder realizar el reconocimiento bien a partir de muestras capturadas directamente desde un micrófono, desde el teléfono, o bien a través de muestras de voz pregrabadas.

La funcionalidad que le falta a OpenVXI y que ha sido necesario implementar es:

1. Un identificador del canal

Se necesita un identificador para el canal que estamos gestionando y poder asignarle de forma exclusiva un determinado dispositivo de audio de entrada. Se ha asignado un número para cada uno de los hilos que se crean para atender cada canal y se han modificado las cabeceras de funciones necesarias para conseguir disponer de este valor.

2. Carga previa del reconocedor

Para utilizar un determinado reconocedor debemos haberlo cargado previamente en el sistema (los modelos que lo definen) para no ralentizar el proceso de reconocimiento en tiempo real. Ni VoiceXML ni OpenVXI especifican un lugar donde cargar previamente el reconocedor. Sólo hay dos sitios posibles donde hacer dicha carga: al cargar la gramática y en la llamada al reconocedor. Como la carga de la gramática es previa al reconocimiento en sí, ha sido el sitio elegido para hacerlo.

3. Asociación gramática-reconocedor

Se necesita una asociación directa entre la gramática o gramáticas activas y los modelos que queremos emplear en el reconocimiento. En OpenVXI se permite tener múltiples gramáticas activas, a la vez que diferentes modelos de reconocimiento, pero tanto VoiceXML como OpenVXI dejan este tema abierto.

Hemos aprovechado un método de la interfaz VXIrec (vacío en la distribución de OpenVXI) para hacerlo. Además, se permite establecer una relación entre una gramática y unos modelos específicos mediante la declaración del atributo "type" asociado a la etiqueta <grammar> empleada para declarar la gramática en cuestión.

<grammar src="datos/bigram.dat" type="Aislada"/>

Se indica al intérprete que los modelos que debe cargar en el reconocedor, teniendo activa la gramática referenciada en "src", son aquellos a los que hemos asociado el nombre de "Aislada".

4.2. Módulo de generación de voz

Este motor ofrece todos los servicios relacionados con la generación de voz contemplados por el estándar VoiceXML, incluyendo la reproducción de ficheros de audio y la conversión texto-voz que permite ofrecer al usuario mensajes de contenido variable sin necesidad de grabarlos previamente.

La implementación del mismo también cubre los siguientes aspectos:

- Permite al usuario disponer de un control prosódico básico sobre la voz sintetizada según las diversas posibilidades que ofrece el lenguaje de etiquetas SSML.
- Permite utilizar las funcionalidades de conversión texto a voz cuando la reproducción desde fichero falle.
- El procesamiento de la reproducción desde fichero y la síntesis de voz se hace de forma asíncrona, sin bloqueo, para permitir que el hablante interrumpa el diálogo (barge-in).
- Gestión y registro de los diferentes errores y eventos que puedan producirse.

La funcionalidad que ha sido necesario implementar en el intérprete es:

1. Un identificador del canal

Se necesita un identificador igual que para el reconocedor (ver sección 4.1.)

2. Cola FIFO

Se ha tenido que implementar una estructura tipo cola FIFO para almacenar los mensajes de voz que luego serán reproducidos en el procedimiento *Play()*.

3. Procedimiento *Play()*

La documentación oficial es incompleta e incorrecta para la reproducción sin bloqueo.

4.2.1. Etiquetas SSML

El estándar SSML (Speech Synthesis Markup Language) (Burnett, 2002) tiene como objetivo el proporcionar una manera estándar de controlar los elementos que entran en juego en la síntesis de voz como pueden ser la frecuencia fundamental, la pronunciación, y el volumen a través de las diferentes plataformas. Lo hace mediante una serie de etiquetas que se introducen con el texto a sintetizar. Hemos implementado la interpretación de las siguientes etiquetas en nuestro conversor:

1. "emphasis" (para dar énfasis a ciertos fragmentos), atributo "level", con los valores: "strong", "moderate", "none" y "reduced".
2. "break" (pausa de un tiempo definido), con el atributo "time", que es un entero seguido de "ms". Ej.: "300ms" para una pausa de 300 ms.
3. "prosody", con los atributos "pitch" (frecuencia fundamental), "rate" (velocidad) y "volume" (volumen). Hemos especificado los valores con un valor relativo, como porcentaje. Ej.: "+10%".

Ejemplo:

¿Ha dicho que quiere transferir <break time="100ms"/> <emphasis level="strong"> 100 euros </emphasis> desde su cuenta <prosody volume="+20%"> 3466 </prosody> <break time="200ms"/> a la cuenta con el número <prosody pitch="+40%"> 564 </prosody>?

4.3. Módulo de control telefónico

La norma VoiceXML sólo estandariza las acciones de desconexión y transferencia de llamadas, con lo cual el proceso de integración de los servicios telefónicos ha resultado relativamente sencillo.

La principal función que realiza la interfaz de línea es realizar la conexión del sistema con la línea telefónica, realizando la conversión de 2 a 4 hilos y adaptando los niveles de señal a ambos lados de la interfaz. Además, y debido a las necesidades del sistema, realiza otras muchas funciones como:

- Adaptación de señales para conectar dispositivos externos auxiliares de E/S.
- Control de la conexión/desconexión con la línea telefónica.
- Control del dispositivo externo para la grabación de la conversación sistema/usuario.
- Detección de tonos multifrecuencia (DTMF).

Por otra parte, se ha introducido una funcionalidad de gran utilidad para el desarrollo y prueba de aplicaciones: la simulación de la línea telefónica mediante un micrófono y un altavoz, con los que se puede probar el sistema sin necesidad de utilizar la línea telefónica, bastando una simple modificación de la configuración del sistema.

4.4. Módulo de comprensión

El módulo de comprensión se encarga de extraer los conceptos que posteriormente tomará el sistema OpenVXI para rellenar la estructura en la que se depositan los resultados del reconocimiento.

Hemos adaptado este módulo a la aplicación bancaria desarrollada en el proyecto. La comprensión está basada en reglas dependientes de contexto, utilizándose

un único diccionario para todos los *prompts* del sistema. Este diccionario está etiquetado semánticamente en función de la información que se desea extraer. A las palabras que no aportan información de utilidad se les asigna la categoría "basura", al igual que a las posibles palabras fuera de vocabulario con las que se pudiera encontrar el sistema.

A la hora de etiquetar el diccionario se ha tenido en cuenta la posibilidad de que una misma palabra puede aparecer en distintos *prompts.* Se ha llegado a una posible solución al respecto estableciendo que una palabra puede tener múltiples categorías, siendo una de ellas la de "basura" para que en caso de necesitar utilizar esa palabra pueda ser transformada por las reglas de comprensión antes de la eliminación de los elementos basura. Así se podrían utilizar las palabras que se necesitasen y en caso de no ser necesarias se perderían tras el mencionado proceso de eliminación.

Podríamos resumir la secuencia que sigue este módulo para procesar una intervención del usuario, junto con un ejemplo, en los siguientes pasos:

1) Preprocesado de la cadena procedente del reconocedor (asignación de categorías, procesado de dígitos, eliminación de interjecciones, etc.)

2) Reglas previas a la eliminación de palabras con categoría "basura"

reescribe3 ("ID_change", "basura", "ID_moneda", "cambio_moneda");

3) Eliminación de palabras con categoría "basura",

4) Reglas posteriores a la eliminación de "basuras"

reescribe2 ("ID_quiero", "cambio_moneda", "SLOT_cambio_moneda");

5) Escritura de los conceptos resultantes

Como se puede observar se utilizan reglas de reescritura en las que se indican las categorías de origen al inicio de la regla y el elemento final sería la categoría de destino del resultado de esa regla. Así, en la regla del paso 2 se utilizan tres palabras que se encuentran en el orden indicado en la regla (las tres primeras) y van a dar como resultado un elemento con la categoría "cambio_moneda". Esta regla recogería expresiones del tipo "cambio de moneda". Posteriormente, y tras la eliminación de las palabras con categoría "basura", vemos como en el paso 4 se utilizan dos elementos que darán como resultado el SLOT_cambio_moneda que finalmente será el resultado de la comprensión en el paso cinco.

4.5. Módulo de modelado de lenguaje

Disponemos de los habituales modelos de lenguaje estocásticos. Además, para este proyecto hemos desarrollado gramáticas de tipo JSGF (Java Speech Grammar Format).

Este tipo de gramáticas son independientes de plataforma y están basadas en el concepto de gramáticas de reglas (*rule grammar*) adoptando determinadas convenciones de Java a la forma de las gramáticas tradicionales. Se han utilizado para representar cómo suelen responder los usuarios a las preguntas del sistema. Al igual que en el módulo de comprensión, tenemos una gramática *JSGF* para cada *prompt* del sistema, además de una especifica para números.

Un posible ejemplo de gramática JSGF para un prompt del castellano podría ser:

```
#JSGF V1.0 ISO8859-1;
grammar GeneralInformationCathegory;
public <infocathegory> =
    <Deposit_Products> {this.$value="DepositProducts"}|
    <Cards>            {this.$value = "Cards"} |
    <EBanking>         {this.$value = "EBanking"} |
    <Exchange_Rates>   {this.$value="ExchangeRates"};


<Deposit_Products>= [infórmeme (sobre | de)] depósitos
            [<Polite>];
<Cards>= [<I_want>] tarjetas ;
<EBanking>= [(a cerca de)] banca [electrónica];
<Exchange_Rates>= [<I_want>] cambio [<Monedas>] ;
<Monedas> = de (moneda | divisas)
<I_want> = [(quiero | deseo)] información (de | sobre);
<Polite> = (por favor);
```

Está gramática corresponde al prompt en el que el usuario debe elegir entre varios servicios (depósitos, tarjetas de crédito, cambio de moneda o información sobre banca electrónica) y para su escritura nos hemos basado en una captura de textos. En ella podemos ver cómo los elementos opcionales aparecen entre corchetes y los paréntesis se han utilizado para agrupar o desambiguar expresiones. A su vez, entre llaves encontramos etiquetas de información específica de la aplicación y las barras verticales indican una alternativa.

5. Limitaciones de VoiceXML

VoiceXML es un lenguaje de gestión de diálogos muy potente, que ofrece una gran variedad de posibilidades de control sobre los servicios de reconocimiento y generación de voz. Sin embargo, se han detectado una serie de limitaciones en la definición de dicho lenguaje que merece la pena comentar (consultar también (Hamerich 03)):

- Hay una carencia o falta de estandarización en la manera de lanzar los distintos reconocedores activos. Este tema se puede resolver de una manera bastante confusa, mediante la definición de propiedades (<property>).
- Tampoco hay estandarización en los aspectos relativos al control telefónico de llamadas.
- Modelado del retorno en el flujo de diálogo: VoiceXML ofrece un mecanismo para gestionarlo pero desafortunadamente sólo está permitido dentro de un <form>. Consideramos que las llamadas a subdiálogo dentro de <fields> o <blocks> son claramente un requisito indispensable.

- Carece de algunas herramientas típicas de programación, como los bucles (i.e., while y for). Generalmente se recomienda emplear en su lugar guiones ECMA, pero esta solución no sirve cuando se necesitan llevar a cabo operaciones entre fields o diálogos.
- Flujo de datos entre los guiones VoiceXML y bases de datos externas. El acceso a recursos externos, como por ejemplo bases de datos, solamente es posible a través de guiones CGI.
- La importación de datos podría mejorarse haciendo posibles las peticiones HTTP de forma explícita extendiendo las posibilidades de <subdialog> y <data>.

6. Conclusiones

Se ha desarrollado un sistema de generación de diálogos que es extremadamente potente, capaz de generar de una forma semiautomática diálogos válidos para múltiples idiomas y dos modalidades partiendo de una descripción de la base de datos y de una interacción reducida con el diseñador. El resultado es un sistema potente y abierto con el que desarrollar diálogos de forma rápida y amigable. Así mismo, la utilización de estándares como VoiceXML y una sintaxis basada en XML nos coloca en una buena posición en el creciente mercado de los sistemas de diálogo basados en VoiceXML.

Así mismo, se ha desarrollado una plataforma completa en tiempo real capaz de ejecutar guiones en VoiceXML aprovechando la funcionalidad ofrecida por OpenVXI. Se han presentado todos los detalles necesarios para conseguir la adaptación de un sistema existente a dicho entorno, destacando una serie de carencias tanto en OpenVXI como en VoiceXML que pueden ser aprovechados por los responsables del estándar.

Se han presentado los módulos que componen dicho sistema, destacando sus aspectos más novedosos y la utilización de estándares, como SSML y JSGF.

7. Referencias

Burnett, D. C., M. R. Walker, A. Hunt. 2002. “Speech Synthesis Markup Language Version 1.0”. W3C Working Draft, http://www.w3.org/TR/speech-synthesis.

Córdoba, R., R. San-Segundo, J.M. Montero, J. Colás, J. Ferreiros, J. Macías-Guarasa, J.M. Pardo. 2001. “An Interactive Directory Assistance Service for Spanish with Large-Vocabulary Recognition”, EUROSPEECH, pp. 1279-1282. 2001.

Córdoba, R., L. F. D'Haro, J. M. Montero, J. Ferreiros, J. Macías-Guarasa, J. D. Romeral, J. M. Pardo. 2003. “Generación semiautomática de aplicaciones de diálogo multimodales: Proyecto Gemini”. XIII Jornadas Telecom I+D. Noviembre 2003. ISBN: 84-89315-28-0.

Córdoba, R. de, F. Fernández, V. Sama, Luis F. D’Haro, R. San-Segundo, J. M. Montero. 2004. “Implementation of Dialog Applications in an Open-Source VoiceXML Platform”. ICSLP. Aceptado y pendiente de publicación.

D’Haro, L.F., R. de Córdoba, R. San-Segundo, J. M. Montero, J. Macías-Guarasa, J. M. Pardo. 2004. “Strategies to reduce Design Time in Multimodal/Multilingual Dialog Applications”. ICSLP. Aceptado y pendiente de publicación.

Gemini 2003. Página web del proyecto Gemini: www.gemini-project.org.

Hamerich, S. W., Y.F.H. Wang, V. Schubert, V. Schless, S. Igel. 2003. “XML-Based Dialogue Descriptions in the GEMINI Project”. Proceedings of the “Berliner XML-Tage 2003”, pp. 404-412.

Hamerich, S. W., V. Schubert, V. Schless, R. de Córdoba, J. M. Pardo, L. F. d’Haro, et al. 2004. “Semi-Automatic Generation of Dialogue Applications in the GEMINI Project”. 5th SIGdial Workshop on Discourse and Dialogue, pp. 31-34.

Lehtinen, G., S. Safra, …, J.M. Pardo, R. Córdoba, R. San-Segundo, et al. 2000. “IDAS: Interactive Directory Assistance Service”, VOTS Workshop.

OpenVXI 2004. http://fife.speech.cs.cmu.edu/openvxi/.

San Segundo, R., Montero, J.M., Colás, J., Gutiérrez, J.M., Ramos, J.M. and Pardo. J.M. 2001. Methodology for Dialogue Design in Telephone-Based Spoken Dialogue Systems: A Spanish Train Information System. In Proceedings of the European Conference on Speech Communication and Technology (Eurospeech). pp 2165-2168.