Plataforma de generación semiautomática de sistemas de diálogo multimodales y multilingües: Proyecto GEMINI

L.F. D'Haro, R. Córdoba, I. Ibarz, R. San-Segundo, J.M. Montero, 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.
{lfdharo, cordoba, ibarz, lapiz, juancho, macias, jfl, pardo}@die.upm.es

Resumen: Presentamos en este artículo 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 necesita el diseñador para el diseño de la aplicación. Finalmente, y como muestra de la eficacia e independencia de la plataforma, presentamos las aplicaciones de prueba realizadas: una aplicación bancaria en la que el usuario puede obtener información genérica de productos del banco, de su cuenta, hacer transferencias, etc., y una de atención al ciudadano.
Palabras clave: Sistemas automáticos de diálogo, generación automática, multimodalidad, multilingualidad, reconocimiento de habla, XML

Abstract: We present in this paper a complete platform for the semiautomatic generation of human-machine dialog systems, that using as input status a description of the database of the service, a flow model with the different states of the final application and a guided interaction step by step with the designer's intervention, generates dialogs to access the service data in different languages and two modalities, speech and web, simultaneously. We describe every module of the platform that makes this possible, emphasizing the different strategies followed to reduce the time needed to do the design. Finally, and as a demonstration of the platform efficiency and independency, we present the applications that have been developed: a banking application and a citizen care application.
Keywords: automatic dialog systems, automatic generation, multimodality, multilinguality, speech recognition, XML.

1 Introducción

El proyecto Gemini (Generic Environment for Multilingual Interactive Natural Interfaces, IST-2001-32343), objeto de esta comunicación (Gemini, 2003, Hamerich 2004), es un proyecto de dos años (2002-2004) financiado por la Unión Europea en el que interviene nuestro grupo representando a la UPM. El consorcio está formado por los siguientes socios: Knowledge S.A. (Grecia) como coordinador, Universidad de Patras – WCL (Grecia), TEMIC SDS (Alemania), Universidad Politécnica de Madrid – GTH (España), FAW (Alemania) y Egnatia Bank (Grecia). Es continuación del proyecto IDAS (LE4-8315), cuyos resultados se han presentado en distintos congresos (Córdoba 2001, Lehtinen 2000.)

El objetivo fundamental del proyecto ha sido el desarrollo de un sistema de generación de aplicaciones de diálogo de forma semi-automática partiendo de una descripción de la base de datos. Para ello, se ha desarrollado una herramienta mediante la cual el diseñador puede especificar totalmente la aplicación de una forma rápida, flexible, intuitiva y cómoda, buscando siempre reducir el tiempo necesario para su desarrollo.

Nuestro generador de diálogos es un sistema orientado a tareas que permite la creación de diálogos para subtareas fijas, como en (Hearst,

1999) y sigue un enfoque similar al del sistema Agenda (llamado ahora RavenClaw) de CMU (Rudnicky, 1999); en la misma línea, el diseñador puede crear el servicio utilizando una representación jerárquica de la tarea y sus subcomponentes, lo que supone facilidad de mantenimiento y crecimiento, y en el que cada estado se expresa mediante una serie de formularios con información relativa a sus restricciones y slots opcionales. Además, la plataforma utiliza algunas ideas del proyecto USI (Interfaz universal de habla) de (Toth, 2002) que se aplican a la generación de nombres de los diálogos generados automáticamente (se derivan de los atributos de la base de datos).

Generalmente los sistemas de diálogo pueden utilizar varias modalidades: voz a través de teléfono, Internet, PDA, WAP, etc. Se habla en estos casos de multimodalidad, uno de los objetivos del proyecto: conseguir que, con un mismo diseño de la aplicación, la plataforma genere simultáneamente el código para ofrecer el servicio, en este caso, a través del teléfono mediante voz y/o a través de Internet.

Otro de los grandes objetivos del proyecto ha sido la multilingualidad. De nuevo, a partir de un mismo diseño de la aplicación, el sistema debe generar en paralelo varios diálogos para atender a usuarios de países distintos. En el proyecto, estamos trabajando en cuatro idiomas: alemán, griego, español e inglés.

Lógicamente, como parte importante del proyecto se encuentra el poder demostrar la eficacia de la plataforma. Para ello, se ha utilizado dicha herramienta para generar dos aplicaciones que se han probado en entornos de funcionamiento reales: una aplicación bancaria, que permite un gran número de operaciones al usuario, y una aplicación de atención al ciudadano.

A lo largo del artículo, llamaremos usuario al cliente final de nuestro sistema, y diseñador a la persona que construye el sistema de diálogo. Nos centramos fundamentalmente en la herramienta de generación de diálogos y en el proceso de modelado del diálogo.

2 La herramienta de generación de diálogos (AGP)

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. Para ello, la interacción sistema-diseñador se basa en una serie de asistentes que automatizan y simplifican el trabajo, guiando al diseñador hacia su objetivo.
- Se busca la utilización de estándares, como veremos en el apartado 2.1.
- 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.

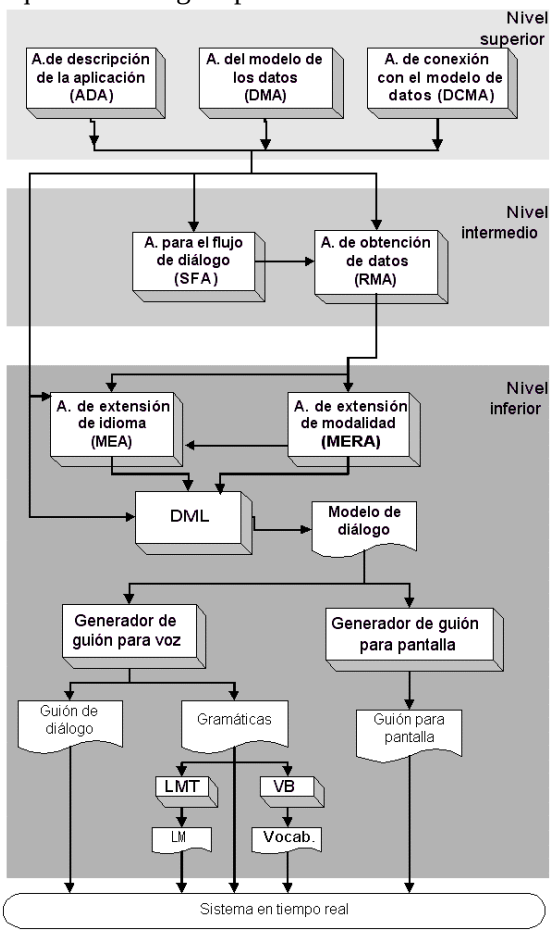

Figura 1. Arquitectura del AGP.

La arquitectura consta de **tres niveles básicos** e independientes:

1. Nivel superior. Aquí, el diseñador debe especificar todos los aspectos globales relacionados con la aplicación y los datos. Tiene 3 asistentes:

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

Asistente del modelo de los datos (DMA): se introducen las descripciones de las clases de la base de datos en que se apoya el servicio, así como sus atributos.

Asistente de conexión con el modelo de datos (DCMA): es donde se definen las funciones de acceso a la base de datos específica. Su objetivo es independizar la implementación de la base de datos de lo que es la aplicación. La aplicación llama a funciones definidas en este asistente. En este módulo trabajaría un experto en bases de datos más que el diseñador del diálogo.

2. Nivel intermedio. Se define el diálogo de una forma independiente del idioma y de la modalidad. Consta de tres módulos:

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

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 y es donde se ha hecho el mayor esfuerzo por proporcionar de facilidad y flexibilidad al diseñador, por lo que será tratado más en detalle en la sección 3.2.

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. Así, si es un usuario nuevo se ofrecería ayuda y preguntas más concretas y explicativas, mientras que para un usuario avanzado se ofrecerían diálogos más rápidos y preguntas mas generales. En este asistente se definen también los niveles de confianza en el reconocimiento por voz que determinan el tipo de confirmación (ver la sección 3.3.2.) De esta manera, se podría garantizar que para un dato crítico, como la cantidad en una transferencia bancaria, la confirmación sea más exigente.

3. Nivel inferior, donde se completa el diálogo introduciendo los aspectos dependientes del idioma y de la modalidad. Consta de los siguientes 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 automática-mente los guiones de ejecución en el estándar VoiceXML para voz y en xHTML para web.

Hay dos herramientas adicionales en la arquitectura: 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.

2.1 Estandarización

A continuación se describen algunas de las características que han permitido que la plataforma sea independiente del sistema operativo, portable y fácilmente escalable.

2.1.1 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.

2.1.2 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.). Estas características han permitido presentar el

lenguaje ante el foro W3C con gran aceptación. En (Katsurada, 2002) se utiliza también una arquitectura independiente de la modalidad con un lenguaje de marcas basado en XML.

2.1.3 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 como de voz, con lo que se consigue que el sistema sea compatible con los navegadores Web tradicionales y con las plataformas de voz más usadas. Para ello, el sistema final genera guiones en xHTML para Web y VoiceXML 2.0 para voz. Además, como intérprete de VoiceXML usamos un sistema de código abierto llamado OpenVXI.

3 El modelado de diálogo

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 flujo de diálogo (SFM)

En este asistente el diseñador especifica a un alto nivel los estados por los que va a pasar el diálogo. Simplemente, le da un nombre a cada estado e indica qué otros estados pueden venir a continuación en el flujo del diálogo, pero sin especificar las condiciones que determinan los saltos ni otros aspectos más detallados. Además, introduce qué datos se le van a preguntar al usuario en cada estado (slots de la aplicación.) Para la introducción de estos datos, al diseñador se le ofrece siempre que elija los atributos del Modelo de datos introducido en el DMA. Esto ahorra tiempo, dado que en la mayoría de los casos los datos que se le preguntan al usuario son un atributo de una clase del modelo de datos. Así mismo, el hecho de que los slots coincidan con atributos del Modelo de datos ayuda al siguiente asistente en las propuestas que va a hacer al diseñador, como veremos a continuación.

3.2 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.

Para que el sistema sea productivo, es necesario que este asistente sea lo más intuitivo posible y que automatice el diseño del servicio.

En la Figura 2 se puede observar la pantalla principal del asistente en el que ya está definida una aplicación bancaria, con su diagrama de flujo en árbol. Al pulsar con el botón derecho se puede ampliar o reducir la visualización, ocultar o mostrar más información acerca de los nodos y refrescar la representación, entre otros.

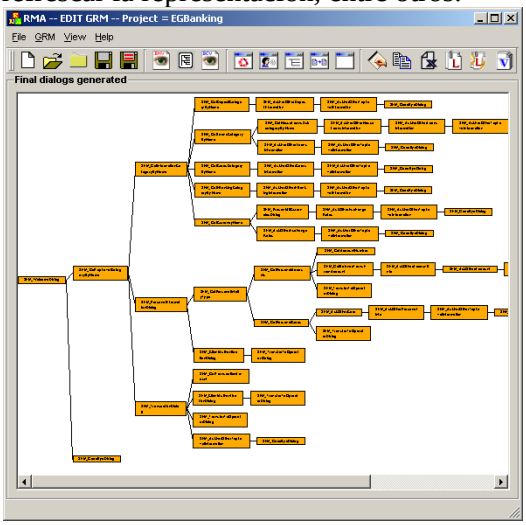

Figura 2. Pantalla principal del RMA.

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

1. Se crean automáticamente numerosos diálogos para ayudar al diseñador: para obtener información del usuario (los llamamos DGet) y para proporcionar información al usuario (DSay.)

Los diálogos DGet y 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. Estos diálogos incluyen una etiqueta que le indica al Asistente de extensión de idioma que debe pedir al diseñador que introduzca el texto a presentar al usuario (para DSay) y la pregunta a hacer al usuario y la gramática a utilizar en el reconocedor de voz (en el caso del DGet) en cada uno de los idiomas utilizados en la aplicación. De este modo, se consigue la independencia del idioma en este asistente.

Hay diálogos DSay de varios tipos: de una clase, que permite seleccionar sus atributos; de cada uno de los atributos de cada clase; 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.)

Además de los diálogos DGet y DSay, el diseñador puede aprovechar mediante "arrastrar y soltar" otros diálogos: los obtenidos de las bibliotecas que se hayan cargado y los procedimientos de acceso a base de datos definidos en el asistente DCMA. En la Figura 3 puede verse la pantalla auxiliar del RMA desde la cual el diseñador puede arrastrar y soltar toda una serie de diálogos.

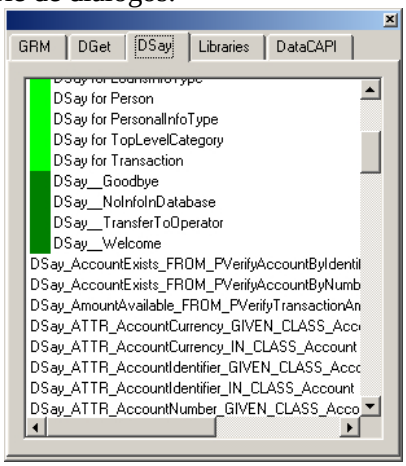

Figura 3. Pantalla auxiliar del RMA.

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 4), en la que se ofrecen al usuario toda la información que es específica de dicho estado evitándole que tenga que perder tiempo buscando los diálogos adecuados en la ventana general (Figura 3). Tiene 4 partes:

- Slots preguntados en el estado y estados a los que se salta.
- DGets específicos del estado, que se determinan a partir del slot (todos los DGets de nombre similar al del slot.)
- 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.

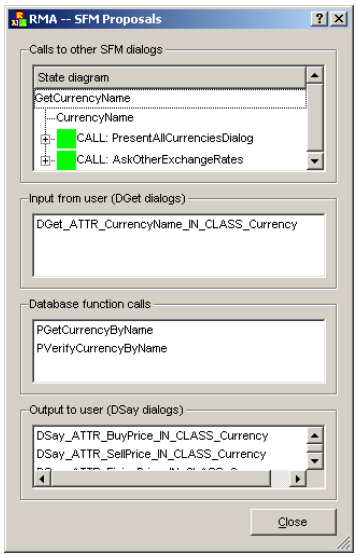

Figura 4. Ejemplo de "Propuestas del SFM".

En esta figura puede verse lo accesibles que están los diálogos que va a necesitar el diseñador para proporcionar información de la moneda: 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.

Este es un aspecto crítico del diseño de aplicaciones de diálogo. Tenemos que encadenar distintos diálogos y procedimientos que van recibiendo los datos obtenidos en las etapas anteriores.

Veamos un ejemplo con una tarea típica: obtener el saldo de la cuenta corriente del usuario. En primer lugar, deseamos obtener un dato del usuario, el número de cuenta (AccountNumber). Como pertenece al Modelo de datos, basta con arrastrar y soltar su DGet correspondiente. El sistema nos preguntará automáticamente si deseamos asignar al valor devuelto por la función a una variable con el mismo nombre que el atributo. Lo habitual es contestar que sí (pulsando Aceptar.)

A continuación, arrastramos la función de acceso a base de datos en la que a partir del número de cuenta se obtiene el saldo de dicha

cuenta. Al soltarla, el sistema detecta automáticamente que el nombre de la variable AccountNumber coincide con el de una variable ya existente (la que acabamos de crear) por lo que simplemente le pide al diseñador que confirme si desea usar dicha variable con una pulsación. Para el parámetro de salida, el saldo, basta también con aceptar la propuesta del sistema (la variable devuelta por la función).

Por último, tenemos que decirle al usuario el saldo de su cuenta. Nos basta con arrastrar el DSay correspondiente al atributo saldo de la clase cuenta y asignar como parámetro de entrada el valor obtenido en el paso anterior.

En resumen, gracias al asistente, el diseño del diálogo se reduce a tres operaciones de arrastrar y soltar seguidas de aceptaciones de los nombres de variables sin cambiar nada.

4. Tipos de diálogos permitidos

El usuario puede añadir todo tipo de diálogos con los que configurar el servicio. Hemos considerado cuatro tipos básicos que cubren las posibilidades típicas de programación: basados en un bucle, en una secuencia de acciones (o subdiálogos), en información introducida por el usuario, en el valor de una variable. Además, se permite tener diálogos en blanco, lo que está pensado para permitir la llamada a un diálogo que se va a definir posteriormente. Por otra parte, se ofrece la posibilidad de clonar un diálogo, algo muy útil porque muchas veces es necesario crear un diálogo muy similar a uno existente.

En función del tipo de diálogo elegido, el diseñador tendrá que introducir una serie de datos diferente. En la Figura 5 podemos ver un ejemplo de estas pantallas, en este caso dedicada a introducir un bucle en el servicio.

Aparte de la posibilidad de añadir los diálogos automáticos ya comentados, 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 para operaciones matemáticas y otro para operaciones con cadenas de texto.

De ahí la existencia de tantas opciones en la ventana de la Figura 5. Y todas estas acciones se pueden realizar sin escribir nada en nuestra sintaxis interna. Lo introducido por el usuario se reduce al máximo.

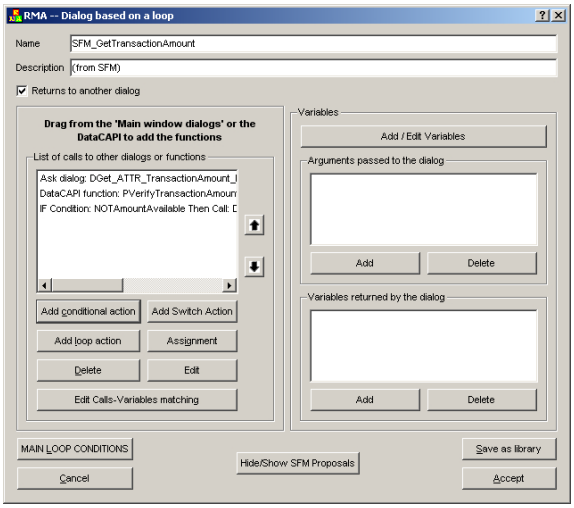

Figura 5. Ventana en la que se define un diálogo basado en un bucle de acciones.

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

Los sistemas de diálogo más sencillos son los de iniciativa del sistema, donde el sistema lleva toda la iniciativa. Nosotros hemos ido un paso más adelante y también nos hemos enfrentado a los diálogos de iniciativa mixta, donde el usuario puede introducir información adicional y el sistema aprovecharla. La clave es que el usuario pueda proporcionar dos o más slots de información a la vez. 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, pero contesta con algún dato más. Por ejemplo, para determinar un viaje se le pregunta '¿De dónde desea salir?' y el usuario contesta 'De Madrid a Barcelona'.

Para implementar estos diálogos dentro del sistema se han tenido que introducir unas etiquetas especiales en la sintaxis GDialogXML y en el archivo que genera el asistente, pero para el diseñador es todo transparente.

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. Esa acción tan simple se traduce en el código final en:

- Se le preguntan los slots al usuario.
- Si falla el reconocimiento de voz, se pasa directamente a preguntar los slots individualmente.

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 (con un límite de dos niveles en la jerarquía) como dato adicional que puede decir el usuario. Además, los slots definidos como opcionales en el SFM se convierten en slots de sobre-respuesta en el RMA. El tratamiento en el sistema en tiempo real es que antes de cada DGet se compruebe si el dato a preguntar **ya** se había obtenido en un estado anterior (lo que sucede cuando el usuario dice un dato de más). Si el dato ya es conocido, se pasa al siguiente estado de la aplicación.

3.3 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.

3.3.1 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, dado que para cada uno es necesario generar acciones diferentes que el diseñador debe ir decidiendo utilizando un sencillo formulario. Vamos a ver cuales son dichos casos:

1. La lista está vacía

Se informa al usuario de que no hay información disponible para posteriormente saltar 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. Existe también la opción de mencionar al usuario cuántos elementos se han encontrado y cuantos desea escuchar. De este modo, puede reducir dinámicamente la cantidad de información proporcionada.

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 (se borrarán) y se vuelve a ejecutar el proceso de búsqueda. Sin embargo, si todavía quedan slots sin rellenar se continúa con el flujo normal de la aplicación hasta tenerlos rellenados y repetir la consulta a la base de datos.

También hemos considerado un caso simplificado: si la lista depende únicamente de un slot introducido por el usuario, por ejemplo, una lista de los últimos movimientos bancarios. En este caso, se presenta la versión simplificada de las ventanas anteriores en la que el diseñador no tiene que especificar los slots a borrar.

3.3.2 Gestión de las confirmaciones

Para confirmar los datos que genera el reconocedor se emplea el formulario de la Figura 6, en el cual se especifican 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 para ellos hace un análisis del tipo de diálogo que permite saber si la opción escogida por el

diseñador es factible. Además, la aplicación detecta en qué casos se permite el uso de confirmaciones implícitas y en cuáles no (por ejemplo, no se permite cuando el siguiente paso en el flujo es ya la consulta a la base de datos, ya que el usuario no podría corregir un fallo en el reconocedor.) El asistente genera automáticamente los diálogos para realizar la confirmación implícita, los de confirmación de datos (tipo "sí o no") y los de corrección, entre otros.

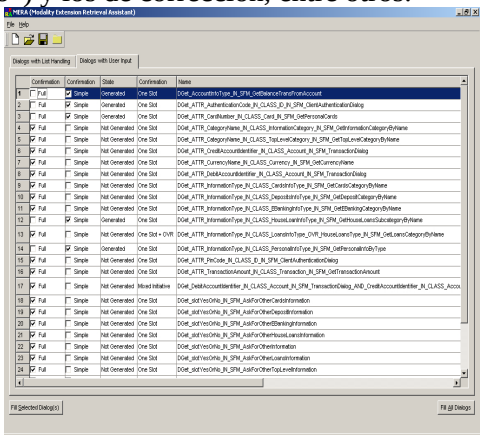

Figura 6. Formulario para la gestión de las confirmaciones.

4 *Las aplicaciones*

Como prueba de la eficiencia de la herramienta y su independencia de la aplicación se han propuesto dos aplicaciones distintas. La primera, una aplicación bancaria que ofrece información general de productos del banco: créditos personales, créditos para compra de coche, hipotecas; depósitos y sus tipos de interés; tarjetas de crédito / débito, etc.; autenticación del usuario: el usuario debe introducir su número de cuenta y un PIN; consultas y transacciones para las cuentas del cliente: consulta de saldos y movimientos para las cuentas y tarjetas de crédito, realización de transferencias entre cuentas, etc. Se ha realizado en cuatro idiomas: griego, alemán, inglés y español.

La segunda aplicación es de ayuda al ciudadano, en la que se ofrece información por Web y voz sobre diferentes entidades gubernamentales y personas, incluyendo información sobre direcciones, horarios de atención, números de teléfonos, etc. Esta aplicación se encuentra disponible para alemán e inglés.

En el momento de escribir esta ponencia no se había realizado todavía la evaluación de las aplicaciones con usuarios reales.

5 *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, de un flujo básico de la aplicación y de una interacción reducida con el diseñador.

El resultado es una plataforma flexible, portable y abierta con la 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 y en xHTML.

6 *Referencias*

Córdoba, R., R. San-Segundo, et al. 2001. "An Interactive Directory Assistance Service for Spanish with Large-Vocabulary Recognition", Eurospeech, pp. 1279-1282.

Gemini. 2003. Página web www.gemini-project.org.

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, USA, pp. 31-34.

Hearst M. A, Allen J, Horvitz, E, Guinn C. 1999. "Mixed-initiative interaction". IEEE Intelligent Systems, Vol 14(5): pp. 14-23.

Katsurada, K., Y. Ootani, Y. Nakamura, S. Kobayashi, H. Yamada, T. Nitta. 2002. "A Modality Independent MMI System Architecture", ICSLP, pp. 2549-2552.

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

Rudnicky, A. and Xu W. 1999. "An agenda-based dialog management architecture for spoken language systems". IEEE ASRU Workshop, pp. I-337-340.

Toth, A.R., T. K. Harris, et al. 2002. "Towards every-citizen's speech interfaces: an application generator for speech interfaces to databases". ICSLP, pp. 1497–1500.