Universidad
Carlos III de Madrid

Ingeniería Informática

PROYECTO FIN DE CARRERA

Desarrollo de un servicio avanzado de callejero mediante el estándar VoiceXML

Autor: Jesús Martínez García
Tutor: David Griol Barres

Título:
Autor:
Director:

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Muchas gracias a mi familia por la insistencia en que acabara; a mis amigos, ya sea por ayudarme a buscar faltas o a cambiar el fondo de la presentación, y por supuesto muchísimas gracias a David Griol, por su paciencia, por su trabajo y por no perder la esperanza.

Resumen

El presente Proyecto Fin de Carrera incide en las potencialidades de la interacción oral con las aplicaciones informáticas, siendo la interacción oral una de las formas más naturales que poseemos los seres humanos para comunicarnos entre nosotros, y persiguiendo en este proyecto compaginarla con los avances tecnológicos. En este sentido, el proyecto se centra en el desarrollo de un **sistema de diálogo** que permita acceder a información y servicios disponibles en Internet.

El objetivo principal es conseguir que desde un teléfono el usuario pueda realizar una petición a un primer servidor IVR (Interactive Voice Response), encargado de la interacción oral con el usuario mediante la interpretación de sus elocuciones y su traducción a un formato entendible por las computadoras, en nuestro caso facilitado por el ***estándar VoiceXML***. A su vez, este primer servidor interaccionará internamente con los servidores web donde se encuentra la información o servicio web que requiere el usuario.

La aplicación práctica desarrollada utiliza la arquitectura y tecnologías descritas en el párrafo anterior con la utilidad principal de proporcionar una interfaz oral a un servicio de callejero y buscador de lugares. El usuario interacciona con la aplicación mencionando oralmente la calle y número de origen; eligiendo seguidamente si desea ir a otra calle y/o buscar lugares cercanos (restaurantes, metro, etc.). Si desea ir a un lugar cercano se requiere al usuario un radio de búsqueda y se le proporciona las distintas opciones encontradas. Una vez hecha la elección se le indica cómo llegar a ese lugar.

VoiceXML traduce las distintas entradas del usuario en peticiones HTTP a las APIs de Google Directions y Google Places. Al obtener la respuesta del servidor de Google, VoiceXML comunica los resultados obtenidos al usuario, consiguiéndose así información sobre distintos desplazamientos del mismo modo que si se estuviese utilizando directamente un ordenador.

Palabras clave: Interacción oral, sistemas de diálogo, VoiceXML, API Google Directions, API Google places, Servidor de llamadas, HTTP, ECMAScript.

Abstract

This Final Bachelor's Project intends in the potentialities of oral interaction with computer applications, being the oral interaction one of the most natural ways the human beings have for communication; we look to adapt it to the technology developments. In this sense, the project is focus in the development of a "dialogue system" that enables access information offered by computers and internet.

The objective is to make it possible for a telephone user to make a request to a first call server, responsible for the interaction with him by means of the interpretation of his utterances and transforming them into a comprehensible computer-format – the one elected in this case is named as standard VoiceXML -. Moreover, this first server internally interacts with a second one where the information or service required by the user is located.

Voice XML is the W3C standard XML format for voice dialogue interacting between human and computer and the application we are trying to develop will rely on it to transform text into voice, introduce requests, etc. Other computer languages such as SRGS – employed to define orally-recognizable entries -, ECMAScript – auxiliary functions for VoiceXML, and others, should also be taken into account.

The computer application which we are trying to develop uses the framework and technologies depicted in the precedent paragraph with the main purpose of providing an oral interface to a street & site searcher. The user interacts with the application by orally mentioning the street and the address number; afterwards, the user chooses if desired, another street or site nearby (such as restaurants, metro stations, etc.). If he wishes to locate a close site, he would be asked to introduce certain search parameters and the several different results obtained will be offered. Finally, once the user has made his choice, the application will tell him how to arrive at the place.

Voice XML transforms the different user entries into HTTP requests for the API of Google Directions and Google Places. Once the response is obtained from Google Server, Voice XML communicates the results back to the user, managing in this way information about site location and displacement in a very similar way as with a computer**.**

Keywords: Spoken interaction, dialogue systems, VoiceXML, API Google Directions, API Google places , Voice Center Network, HTTP, ECMAScript.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1: Introducción

1.1 Introducción

"La premisa básica de la evolución es la conservación de la especie. A lo largo de los millones de años de existencia de vida en nuestro planeta, los organismos vivos han ido poniendo en práctica sistemas para conservarse a sí mismos y a los demás miembros de su especie, apelando a un instinto de supervivencia que los ha hecho superar infinidad de obstáculos.

Esta cooperación entre miembros de la misma especie no puede realizarse si no es a través de un elemento que coordine los esfuerzos: el lenguaje" [DAN08].

El ser humano, entendido como ser social, ha demostrado una predisposición natural privilegiada a la utilización del lenguaje que ha empleado con el objetivo primordial de comunicarse con sus semejantes, transmitir sus ideas, pensamientos, impresiones y sentimientos. Pero, llegados a este punto de la evolución, el desarrollo tecnológico nos plantea una cuestión ineludible e insoslayable: ¿Es posible que el ser humano pueda comunicarse con los dispositivos y nuevas aplicaciones informáticas de forma similar a como lo haría con otro ser humano?

Es aquí donde se introduce, por tanto, el siguiente eslabón explicativo. Así, conviene, a continuación, ofrecer una definición más concreta sobre lo que implica "un sistema de diálogo hablado". Siguiendo la definición de R.Lopez Cozar y R.Granell, "*Los sistemas de diálogo hablado son programas informáticos que se diseñan con la finalidad de emular a un ser humano en un diálogo oral con otra persona*" [LOP04].

En este contexto, el estándar VoiceXML [VXML20] es un lenguaje basado en XML (*eXtensible Markup Languaje*) desarrollado por el W3C para gestionar la interacción por voz de la misma forma que HTML gestiona la interacción tradicional con los contenidos en la Web.

Cabe destacar la versatilidad de este estándar de interacción oral, que posibilita la interacción con otros elementos de la web (lenguajes, servicios, APIs, etc). Fundamentalmente, el proyecto que se presenta se servirá de la interacción con APIs de Google.

De este modo, VoiceXML será el encargado de obtener la información de entrada y transformarla en peticiones HTTP. Asimismo, también recibirá la información de esas peticiones y las comunicará posteriormente al usuario. Para conseguir este objetivo, VoiceXML se ayuda de dos lenguajes adicionales: SRGS y ECMAScript.

1.- SRGS (*Speech Recognition Grammar Specification*) [SRGS] es un lenguaje de definición de gramáticas que se ha usado para reconocer las posibles entradas del usuario y transformarlas a texto.

2.- ECMAScript [ECMASCRIPT] es un lenguaje soportado por VoiceXML, considerado como la base de JavaScript [JAVASCRIPT] y con el que definiremos funciones de tratamiento de cadenas, obtención de información de nodos en XML, etc.

Por otro lado, Google es una de las empresas tecnológicas más importantes e innovadoras del planeta, no siendo únicamente famosa por su potentísimo buscador, sino también por sus servicios online, entre los que cabría destacar Google Maps [GOOGLEMAPS], uno de los más utilizados actualmente, que se define como un servidor de mapas en la Web donde se pueden localizar distintos lugares mediante imágenes vía satélite. Como otros servicios de mapa, Google Maps permite la opción denominada como "*street view*" (visualización de las

calles desde la propia acera), así como la localización de lugares concretos y el cálculo de rutas óptimas una vez introducida una serie de parámetros o restricciones.

Para este proyecto nos ha interesado principalmente la información por pasos que Google Maps nos ofrece para llegar de un punto de origen a otro de destino. Esta funcionalidad permite al usuario disponer de una lista en la que, paso por paso, se detalla cómo llegar de una determinada localización espacial (punto A) a otra (punto B), calculando también el tiempo necesario para el desplazamiento – según la opción de transporte o desplazamiento escogida – e incluso informando con considerable precisión de la distancia recorrida entre las ubicaciones indicadas.

Las Figura 1 y 2 muestran una captura de pantalla del servicio Google Maps para indicar respectivamente la posición de un determinado lugar y las indicaciones sobre cómo llegar a un destino desde un determinado origen.

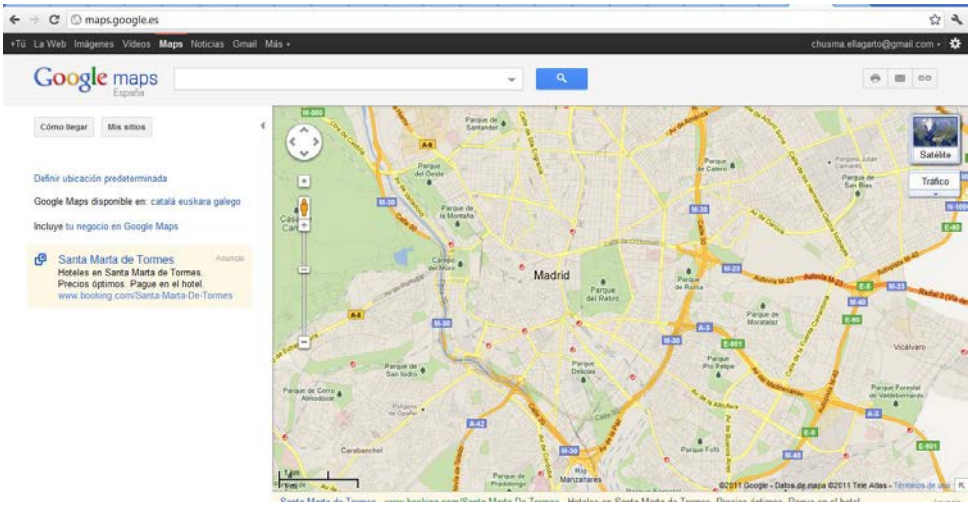

Figura 1. Captura de pantalla del portal Google Maps mostrando la posición de un lugar determinado

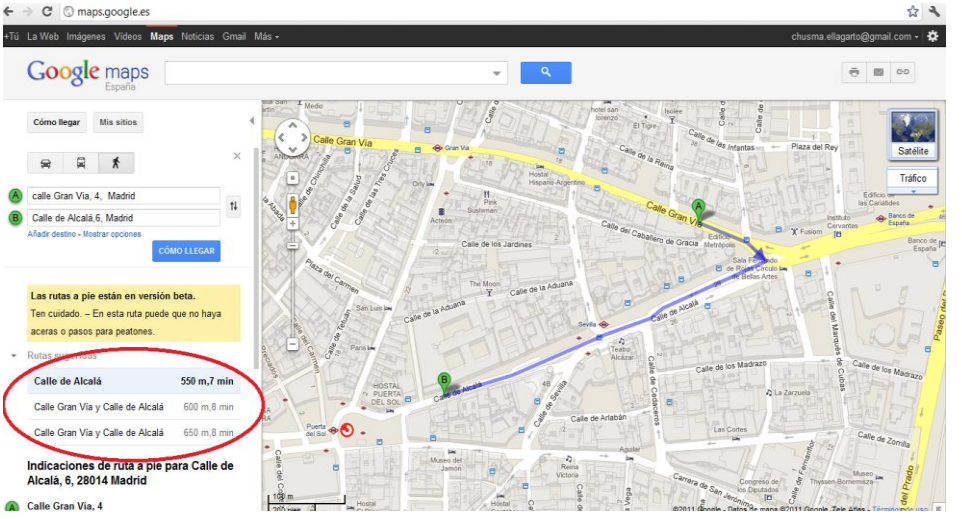

Figura 2. Captura de pantalla del portal Google Maps mostrando cómo llegar a un determinado destino

La Figura 2 muestra el resultado de la funcionalidad de Google Maps "Cómo Llegar". En ella, la aplicación nos solicita información sobre la localización concreta del punto de partida y el punto de destino. A partir de esta información el servicio Google Maps, aparte de la información gráfica, nos indica paso a paso en formato texto todas las indicaciones que hemos de seguir, la longitud de cada una de las rutas y el tiempo estimado. Generalmente, se ofrece la alternativa considerada como "óptima" en primer lugar, aunque se ofrecen distintas rutas alternativas por si fueren otras las preferencias del usuario.

Además, Google guarda información de todos los lugares de interés mediante las coordenadas de estos, y nos da la posibilidad de, a partir de un lugar y un radio, conocer los lugares de interés que se encuentran en ese rango, también podemos seleccionar el tipo de lugar de interés que se quiere (restaurantes y bares, tiendas, parques, bancos, paradas de metro, etc.), para ello Google nos brinda el API de Google Places [PLACESAPI].

Suele decirse que una imagen vale más que mil palabras. Cuando se trata un mapa, la aseveración es, si cabe, más acertada. Es evidente que no podemos pasar una imagen a formato oral, pero lo que sí que podemos hacer es pasar la información más importante contenida en esa imagen a un formato oral. En nuestro caso, esa información de relevancia serán precisamente las indicaciones de desplazamiento, con el cálculo de las distancias de cada tramo, que nos permitirán ir de un punto a otro con un conocimiento bastante ajustado del tiempo y del recorrido a realizar. Todo esto lo conseguiremos con el API de Google Directions [DIRECTIONSAPI], que se encarga de identificar con qué formato se tiene que hacer una petición al servidor de Google y de cómo devolver la información solicitada.

Nuestra aplicación unirá la potencia de los sistemas de diálogo con las APIs de Google Directions y Google Places, constituyendo un servicio de callejero por voz que indique cómo ir de un punto escogido a otro o de un punto escogido a un lugar de interés concreto (hemos restringido estos lugares a restaurantes y paradas de metro). Así, el usuario tendrá la opción de seleccionar el tipo de lugar de interés, la distancia máxima a la que se puede encontrar y una vez obtenidos estos parámetros, seleccionar una de las distintas opciones que le da nuestra aplicación. En definitiva, la aplicación que pretendemos poner en práctica con el presente proyecto puede calificarse como un “callejero-buscador” con acceso oral.

1.2 Objetivos del Proyecto

Podemos diferenciar dos tipos de objetivos, los establecidos como objetivos generales del Proyecto de Final de Carrera y los objetivos concretos que persigue la aplicación que se ha desarrollado.

1.2.1 Objetivos del Proyecto Final de Carrera

Como principal objetivo del PFC podemos establecer el estudio y desarrollo de un sistema de diálogo usando el lenguaje de programación VoiceXML, estudiando además la cohesión que éste tiene con otras tecnologías. Este objetivo ha supuesto un reto debido a la poca familiarización que disponía con este tipo de tecnologías. Como dominio práctico de la aplicación hemos seleccionado, tal y como se ha descrito en la sección anterior, el desarrollo de un servicio de callejero que sea accesible mediante voz, ya sea tanto por teléfono como por otro dispositivo similar (por ejemplo, la utilización de un ordenador y VoIP).

1.2.2 Objetivos específicos de la aplicación desarrollada

Podemos decir que estos objetivos derivan del objetivo genérico antes mencionado, ya que nuestra meta principal es familiarizarnos y desarrollar un sistema de diálogo. También se pretende que esta aplicación sea lo más sencilla posible y que, al mismo tiempo, cumpla con diversas expectativas relativas a los siguientes aspectos:

- Accesibilidad: Deseamos que la aplicación esté disponible en cualquier lugar, ya que al ser sólo necesario un teléfono para acceder a la aplicación, podremos acceder al callejero-buscador tanto en la calle, como desde un coche o desde casa.

- Eliminación de barreras: Queremos que las personas que no tengan acceso a un callejero mediante los interfaces tradicionales, bien sea por discapacidades visuales o motoras, puedan acceder a las nuevas tecnologías. Asimismo, incrementamos el número de entornos donde

poder utilizar la aplicación desarrollada sin la necesidad de los interfaces tradicionales.

- Simplificación: Los avances tecnológicos dejan cada vez más atrás a la gente que no ha podido ir familiarizándose con ellos, ya sea por cuestiones de edad o por falta de acceso a ellos. Es complicado que alguien que no maneje con frecuencia una computadora pueda acceder fácilmente a los recursos que hay en la red. Mediante nuestro sistema, el usuario sólo necesitará saber usar un teléfono y, siguiendo las instrucciones, podrá obtener la información que desea.

1.3 Planificación del Proyecto

Para la realización del proyecto, lo hemos dividido en tres fases principales:

- Fase de planificación, donde se ha realizado un estudio de las tecnologías necesarias y la toma de requisitos.

- Fase de desarrollo, donde se ha llevado a cabo la implementación de la aplicación y las pruebas necesarias para su corrección y validación.

- Fase de documentación, donde se ha completado la memoria y presentación del PFC.

La fase de documentación se ha dispuesto al final en la enumeración debido a que es donde más documentación se ha realizado al tener que completar la memoria, pero ha estado presente a lo largo de todo el proyecto.

1.3.1 Fase de Planificación

- **Estudio de los sistemas de diálogo:** lo primero que se ha realizado en el proyecto ha sido familiarizarse con los sistemas de diálogo para tener más conocimientos.

- **Estudio del lenguaje VoiceXML:** esta fase es importante debido a que para conocer qué es posible realizar se tiene que tener claro qué lenguajes y herramientas tenemos disponibles y sus limitaciones, especialmente en cuanto a

- **Estudio de la API de Google Directions:** necesitamos conocer qué se puede realizar y qué no con este servicio, el acceso a la información, cómo devuelve la información solicitada, etc.

- **Definición de los objetivos:** esta fase engloba todas las reuniones con el tutor para definir tanto el alcance de la aplicación como los objetivos que se han conseguido con el desarrollo de la aplicación. Se trata de un proceso dilatado en el tiempo, que se ha llevado a cabo a medida que hemos ido avanzando en la elaboración del proyecto.

1.3.2 Fase de Desarrollo

- **Diseño del flujo de la interacción:** se ha definido cómo queremos que el usuario interaccione con nuestra aplicación, cuándo nuestra aplicación dará la información al usuario y cómo la recibirá.

- **Diseño de las gramáticas:** las gramáticas se usan para identificar la información que el usuario puede comunicar a nuestra aplicación. Si lo que el usuario intenta comunicar no está definido en una gramática, la aplicación no lo comprenderá.

Se trata de una fase crucial en la que definir y limitar correctamente es completamente imprescindible. Se ha prestado una especial atención a facilitar, con la definición de las gramáticas, que la interacción con el usuario sea lo más natural posible.

- **Desarrollo de la aplicación:** implementar la aplicación conlleva, principalmente, escribir el código de la aplicación en VoiceXML y proceder al desarrollo de las gramáticas.

- **Evaluación y validación:** aunque a lo largo de todo el desarrollo se han ido realizando pruebas unitarias para comprobar el correcto funcionamiento de cada uno de los módulos, después se han realizado pruebas de integración para comprobar que su integración no originaba problemas. Por último, para comprobar la aplicación completa, se han realizado varias pruebas de validación con usuarios.

1.3.3 Fase de Documentación

Tal y como se ha mencionado anteriormente, la fase de documentación se ha realizado a lo largo del todo el proyecto, ya sea guardando información adquirida en el análisis para una incorporación futura en la memoria como mediante la bibliografía que nos ha ayudado en el desarrollo de la aplicación. Después de terminar las fases anteriores, se ha recopilado todos los conocimientos y la información acumulada para exponerla en esta memoria.

1.4 Medios empleados

En este apartado detallamos los diferentes medios empleados para el desarrollo de la aplicación desarrollada. Para ello, se hará referencia a temas diversos, ya sea en relación con el software, los

lenguajes de programación, el hardware o los distintos manuales consultados.

1.4.1 Lenguajes de programación

El proyecto está basado en la utilización del estándar VoiceXML, pero no sólo se ha usado este estándar, parte del código se ha realizado con ECMAScript, ya que se ha añadido funcionalidades que codificarlas sólo con VoiceXML no era posible. Para el desarrollo de gramáticas se ha optado por gramáticas SRGS, por su sencillez y compatibilidad.

1.4.2 Software

Se han usado los siguientes elementos:

- **Plataforma Voxeo Evolution:** ejerce la función de servidor intérprete de VoiceXML. Asigna también un número de teléfono con el que acceder a nuestra aplicación. En nuestro caso, también guardaremos nuestra aplicación en el servidor que nos proporciona. En el estado del arte se describirá este elemento con más detalle [VOXEOE].

- **Skype:** para poder realizar llamadas desde el ordenador y poder realizar pruebas más rápidamente.

- **Notepad++:** para el desarrollo de programación en XML.

- **Microsoft Office 2007:** para el desarrollo de la memoria.

1.4.3 Hardware

- Ordenador Portátil.
- Periféricos habituales: ratón y teclado.

- Auriculares con micrófono.
- Router.

1.4.4 Manuales

Principalmente se ha usado el manual de W3C "Voice *Extensible Markup Language (VoiceXML) Versión 2.0*" [VXML20]. Cuando se han tenido problemas específicos se ha recurrido a Internet, debido a la variada y detallada documentación que se puede encontrar, todas las referencias usadas se han ido recopilando para incluirlas en la bibliografía.

1.5 Planificación temporal.

Hemos planificado el proyecto para realizar el trabajo durante los fines de semana, debido al trabajo a tiempo completo durante la semana. La Figura 3 resume la planificación temporal del proyecto mediante un diagrama de Gantt.

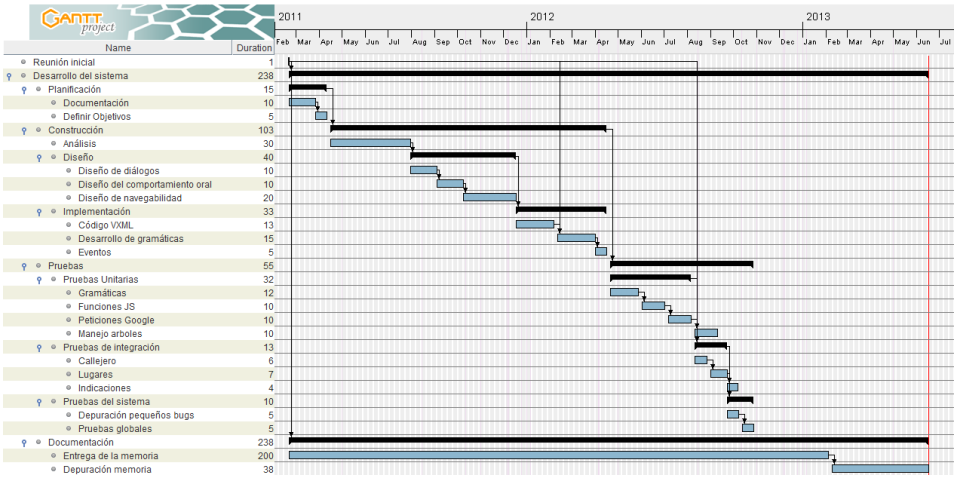

Figura 3. Diagrama de Gantt mostrando la planificación temporal del proyecto

1.6 Estructura de la memoria

Se incluye a continuación un breve resumen del contenido de cada capítulo de la memoria, con el objetivo de detallar la estructura de la memoria:

Capítulo 1: Introducción y objetivos. Este capítulo inicial describe las motivaciones y los objetivos del Proyecto Final de Carrera. Además, incluye las fases de desarrollo, la planificación temporal, los medios empleados y la descripción de la estructura de la memoria.

Capítulo 2: Estado del Arte. En este capítulo se hace un estudio detallado de los sistemas de diálogo, se analiza en detalle el lenguaje estándar VoiceXML y se presenta la plataforma utilizada para la implementación de la aplicación, Voxeo. También se describe con detalle el servicio Google Directions API.

Capítulo 3: Descripción general del sistema desarrollado. En este capítulo se proporciona una visión global de la aplicación, se analizan las tecnologías empleadas y se detallan las operaciones más comunes tratadas a lo largo de su implementación.

Capítulo 4: Conclusiones y trabajo futuro. Se exponen las principales ideas, cuestiones y conclusiones derivadas de la realización del proyecto, así como las posibles líneas de investigación que a partir de este proyecto se podrían generar.

Glosario. En este apartado se recopilan los principales términos y conceptos técnicos utilizados en la memoria, con el objetivo de facilitar su comprensión al lector.

Bibliografía. En este apartado se reflejan las citas bibliográficas que se han consultado para la realización tanto del proyecto como de la memoria.

Capítulo 2: Estado del arte

En todo proyecto, uno de los elementos más importantes a llevar a cabo antes de proceder al desarrollo del mismo, consiste en mostrar de una panorámica global que nos indique cómo se enmarca la aplicación desarrollada y qué aporta frente al resto del estado del arte.

Partiendo de esta idea, y centrándonos en nuestro caso en aplicaciones que facilitan la interacción oral con los usuarios, es necesario establecer las bases que nos permitan comprender cuál es la situación en la que se encuentran actualmente los sistemas de diálogo, haciendo hincapié en el análisis de sus módulos, determinando cuál es el procedimiento por el cual éstos interactúan entre sí y qué aplicaciones principales poseen. A continuación, resumiremos las características principales del lenguaje VoiceXML. Más tarde se describirá Voxeo, intérprete VoiceXML utilizado por el proyecto y que, como se ha descrito previamente, nos permite además asignar un número de teléfono a nuestra aplicación y disponer de un servidor para hospedar el código desarrollado en este lenguaje. Por último, describiremos las APIs de Google Directions y Google Places, de gran importancia para nuestra aplicación.

2.1 Sistema de diálogo

2.1.1 Introducción

Se entiende por sistema de diálogo o sistema conversacional (SDS, Spoken Dialogue System) “una determinada tecnología cuya finalidad primordial viene definida por la necesidad de facilitar un entorno natural de interacción entre un ser humano y una computadora” [LLI06].

En este sentido, debe entenderse que un sistema de diálogo es un programa informático que, por definición, proporciona la capacidad de emular el diálogo que se produce entre dos personas, y que tiene como objetivo elemental cumplir con una determinada tarea, que generalmente será la de suministrar una información concreta o proveer un servicio específico dentro de un dominio bien acotado [GRI07].

De estas definiciones introductorias, puede inferirse que en un sistema de diálogo ideal o pleno, deberían darse las siguientes características [LLI06]:

- Reconocimiento espontáneo del lenguaje oral o habla sin restricciones.
- Comprensión de enunciados sin restricciones de contenido.
- Capacidad de proporcionar respuestas en lenguaje natural con sentido, de construcción gramatical idónea y pragmáticamente adecuadas para cualquier dominio de interacción.

En definitiva, se deduce que en el caso de aplicaciones reales, la utilización de los sistemas de diálogo aporta tanto una serie de ventajas como de inconvenientes. Entre las ventajas destaca el hecho de que, por regla general, el sistema proveerá un acceso más natural a la información requerida que, además, en un entorno red, estará disponible 24 horas al día. Debemos añadir el hecho que, habitualmente, la provisión de información en Internet, suele ser una opción bastante económica para las empresas, frente a la que la necesidad de contratación de uno o varios empleados difícilmente podría competir.

No obstante, también existen desventajas cuando se sustituye la actividad de un ser humano por el proceder meramente mecánico y, algunas veces no intuitivo de una aplicación informática. Aquí puede

mencionarse, en primer lugar, el hecho que la aplicación sólo estará configurada para atender una serie de mensajes concretos, por lo que si nos salimos de las directrices marcadas, será imposible obtener una respuesta coherente. En segundo lugar, no debe olvidarse el hecho de que un porcentaje elevado de los usuarios de telefonía prefieren el trato humano frente a ser atendidos por una máquina que a veces es incapaz de detectar las matizaciones más específicas y las necesidades menos mecánicas de los individuos.

Puede decirse, en suma, que un sistema de diálogo suele presentar las siguientes limitaciones o restricciones [LLI06]:

- Se encuentran sujetos a las limitaciones en lo que respecta al reconocimiento automático del mensaje oral.
- Tanto la comprensión como la posible respuesta están restringidas a dominios específicos.
- Existe un condicionamiento claro debido a la naturalidad del habla sintetizada.
- Tienen una evidente dependencia de la necesidad de implementar estrategias de verificación.
- No pueden abstraerse de los problemas propios del diálogo espontáneo: elipsis, anáforas, deícticos.

Definidas estas limitaciones, cabe a continuación enumerar cuáles son las capacidades que un sistema de diálogo debe contemplar para ser capaz de comportarse de una manera análoga a la humana. Estas capacidades son [LLI06]:

- Habilidad para el reconocimiento de las elocuciones del usuario.
- Inclusión de una herramienta para la gestión del diálogo.
- Capacidad para el análisis lingüístico (morfológico, sintáctico, semántico, pragmático) de los enunciados.

- Creación de una representación interna de la historia del diálogo.
- Mecanismo para el tratamiento de la representación interna en función de la tarea.
- Implementación de un sistema para la generación de secuencias de respuesta.
- Herramienta de conversión del texto en lenguaje oral.

2.1.2 Arquitectura de un sistema de diálogo

Para llevar a cabo las tareas descritas en el apartado anterior, un sistema de diálogo consta típicamente de una serie de módulos, cada uno especializado en un aspecto concreto de la interacción entre la persona y la máquina [LLI06]:

- Reconocimiento del habla.
- Comprensión del lenguaje.
- Gestión del diálogo.
- Generación del lenguaje.
- Conversión de texto en habla.

En la Figura 4 podemos observar cómo interactúan los distintos módulos de un sistema de diálogo.

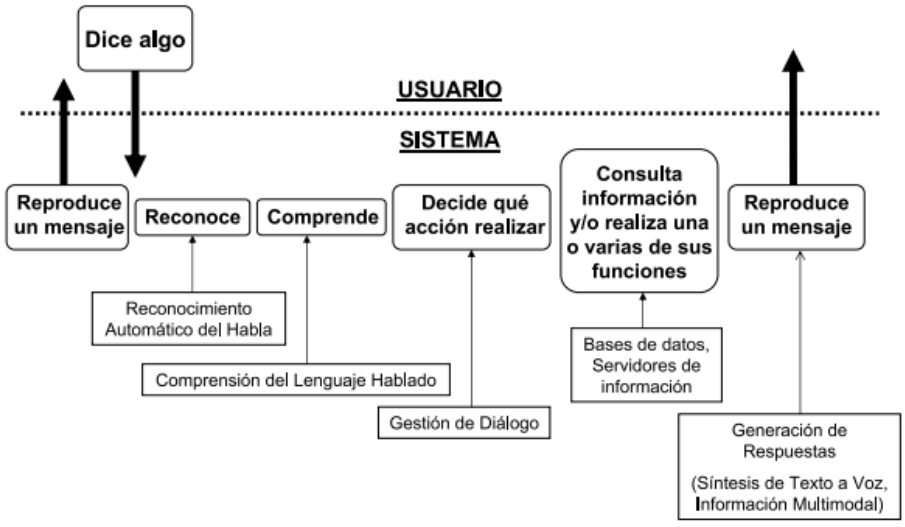

Figura 4. Funcionalidades y módulos asociados en la arquitectura típica de un sistema de diálogo hablado [GRI07]

Como vemos en la Figura 4 y tal como describe Joaquim Llisterri en el libro "*Los sistemas de diálogo*" [**LLI06**], el sistema de diálogo reproduce un mensaje solicitando información al usuario, a lo que el usuario responde de manera oral, al realizar el usuario esta acción el sistema utiliza el módulo de reconocimiento automático del habla. El cometido de este módulo es analizar las variaciones en el tiempo de la amplitud y la frecuencia en la señal de voz para intentar extraer de ellas el contenido de la información solicitada, es decir convertir la onda sonora en una representación escrita. Para ello, se necesita haber desarrollado previamente un conjunto de modelos fonéticos que recojan la variabilidad de cada segmento mínimo del habla –es decir, de cada alófono– y un modelo de lenguaje que establezca la probabilidad de combinación de una palabra con otra. Para obtener estos modelos se recurre al uso de corpus hablados, representativos tanto de los hablantes como del dominio de aplicación del sistema de diálogo. La tecnología utilizada en este módulo es, pues, el reconocimiento automático del habla (RAH o ASR, *Automatic Speech Recognition*), empleada también en otro tipo de aplicaciones como el dictado

automático de textos, y considerada una parte esencial de las tecnologías del habla.

Sin embargo, una representación escrita de lo dicho por el usuario no es suficiente para responder al usuario, pues además del reconocimiento es necesario que el sistema de diálogo la comprenda correctamente. Debe señalarse aquí que cuando en el contexto de las tecnologías lingüísticas se habla de comprensión, se hace referencia a la generación automática de una representación semántica abstracta de un enunciado. Esta tarea la realiza el módulo de comprensión o de interpretación semántica, recurriendo a las técnicas propias del campo conocido como comprensión del lenguaje natural (NLU, *Natural Language Understanding*), que forma parte del procesamiento del lenguaje, o de la comprensión del habla (SLU, *Spoken Language Understanding*), uno de los ámbitos de las tecnologías del habla.

El sistema debe considerar la historia previa del diálogo para decidir qué acción realizar a continuación, tarea que lleva a cabo el gestor del diálogo (DM, *Dialogue Manager*). Para ello, puede ser necesario acceder a las bases de datos u otros repositorios de información de la aplicación. A partir de la acción, es necesario construir una respuesta, que deberá corresponder a una frase gramaticalmente bien formada que también forma parte del procesamiento del lenguaje natural y que constituye el núcleo del módulo de generación de respuestas (NLG, *Natural Language Generator*).

En una última etapa, puesto que la consulta de información se realiza oralmente, la frase creada por el módulo de generación de respuesta tendrá que convertirse en su equivalente sonoro para que el usuario del sistema de diálogo pueda escucharla. De esta tarea se ocupa la conversión de texto en habla (TTS, *Text-to-Speech Synthesis*), uno de los campos de investigación de las tecnologías del habla que tiene como objetivo transformar textos escritos en su realización oral o, dicho de un modo más simple, efectuar la lectura en voz alta de un texto escrito. El módulo de conversión de texto en habla es, por tanto,

el último paso de un sistema de diálogo y el que se encarga, finalmente, de hacer llegar la información a la persona que espera una respuesta.

2.1.3 Tipos de sistemas de diálogo

Joaquim Llisterri [LLIAA], clasifica los sistemas de diálogo en:

Sistemas de diálogo guiados.

La interacción se realiza mediante alternancias cerradas entre preguntas y respuestas, por lo que hay una restricción en las iniciativas del usuario.

Sistemas de diálogo cooperativos.

Estos sistemas aceptan interrupciones y negociaciones por parte del usuario, hay un reparto equilibrado del turno de palabra e incorporan mecanismos de detección de incoherencias gramaticales.

Sistemas de diálogo adaptativos.

El sistema es capaz de aprender nuevas estrategias comunicativas en función del comportamiento del usuario (estados emocionales).

2.1.4 Retos actuales en el desarrollo de sistemas de diálogo

Llisterri presenta diez futuros avances con efecto a corto plazo [LLIAA]:

En el ámbito de la conversión de texto en habla.

- Síntesis guiada por el objetivo o la función del enunciado.
- Mejoras en la asignación de pausas, en la composición de frases y en la prosodia.

- Incorporación de emociones.
- Introducción de la síntesis a partir de conceptos para mejorar la conversión de texto en habla.

Herramientas.

- Desarrollo de plantillas y de asistentes para la creación de sistemas a partir de lenguajes como VoiceXML.
- Ayuda para incorporar las mejores prácticas en el diseño.

Identificación del locutor.

- Identificación a partir del habla.
- Reconocimiento del habla adaptado al perfil del locutor.
- Posibilidad de tratar varios hablantes en varias situaciones.

Procesamiento del lenguaje natural.

- Basado en principios lingüísticos más amplios (en lugar de correspondencias entre enunciados y patrones).
- Sistemas reutilizables (en vez de sistemas ad-hoc).
- Sistemas más robustos.

Aplicaciones multimodales.

- Integración de habla, texto y gráficos.
- Integración del tacto con el habla.
- Reconocimiento de expresiones faciales.

Aplicaciones integradas.

- Reconocimiento y síntesis del habla en asistentes digitales personales, vehículos, etc.
- Electrodomésticos.
- Entornos inteligentes.

Respuestas inteligentes.

- Resumen automático de información compleja procedente de diversas fuentes usando técnicas de Generación del Lenguaje Natural.
- Incorporación de razonamiento y planificación de respuestas.

Sistemas multilingües.

- Identificación automática del idioma del usuario.
- Traducción automática del habla.

Reconocimiento de grandes vocabularios.

- Combinación de vocabulario ilimitado, independencia del locutor y reconocimiento del habla continúa.
- Tareas de dictado y de recuperación de información a partir de archivos sonoros.

Interfaces conversacionales.

- Necesidad de realismo frente a las expectativas creadas.

2.1.5 Aplicaciones de los sistemas de diálogo

A continuación, vamos a exponer una serie de ejemplos que ilustran la aplicabilidad de los sistemas de diálogo en un gran número de dominios. Es necesario aclarar que, en este apartado, no es nuestra intención detallar las plataformas o componentes de las aplicaciones que nos servirán de ejemplo. Tampoco se ha buscado definir la arquitectura de las mismas. En definitiva, lo que se persigue es una exposición de la funcionalidad de las aplicaciones que nos permitan comprender qué es lo que puede conseguirse con los sistemas de diálogo.

La mayoría de los ejemplos que vamos a ofrecer han sido extraídos de la tesis doctoral del tutor del proyecto [GRI07]:

Información meteorológica

JUPITER: El sistema JUPITER [JUP20] es un sistema de diálogo que proporciona información meteorológica de diferentes ciudades alrededor del mundo (temperatura, velocidad del viento, etc.).

WebGALAXY: El sistema WebGALAXY [LGC97] proporciona información de vuelos, datos meteorológicos e información turística y de tráfico en Boston. Los usuarios interactúan con el sistema mediante la voz o entradas escritas.

Información planificación de viajes

MERCURY: El sistema de diálogo MERCURY [SEP20], creado en 1999, posibilita la búsqueda de información de horarios y precios de vuelos (trayectos entre más de 200 ciudades con posibilidad de escalas) por vía telefónica.

PEGASUS [PEG94]: Sistema de diálogo, también accesible telefónicamente, que proporciona información sobre vuelos en Estados Unidos. Esta información es en tiempo real, actualizándose continuamente a lo largo del día.

WHEELS: El sistema WHEELS [MBG96] realiza búsquedas en una base de datos de anuncios clasificados de automóviles. Estas búsquedas pueden realizarse vía telefónica o mediante comandos escritos.

BusLine [BCA02]: Facilita información de líneas de autobús de distintas zonas de Pittsburgh.

WAXHOLM [CHU96]: WAXHOLM proporciona información sobre el tráfico naval en el archipiélago de Estocolmo.

TRAINS [AJG00]: Informa sobre rutas y horarios de trenes de mercancías. Al usuario se le proporciona un mapa en pantalla mostrando ciudades, conexiones y localizaciones de trenes. Mediante la voz el sistema le informa de un conjunto de ciudades destino donde se requerían trenes. El objetivo del usuario era encontrar el conjunto de rutas más eficiente.

TOOT: El dominio del sistema TOOT [LIP02] es la búsqueda de información en la web sobre horarios de trenes utilizando el canal telefónico.

LIMSI RAITEL [LBR97]: El dominio del sistema desarrollado por el LIMSI (Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur) dentro del proyecto RAILTEL (*Railway Telephone Information Service*) es proporcionar información de la red ferroviaria francesa (horarios, precios y servicios de trenes) por vía telefónica.

Información restaurantes

DINEX [SEP96] es un sistema conversacional que proporciona información sobre restaurantes de Boston (nombre, dirección, horario, rango de precios, tipo de cocina, reserva...) y cómo llegar a ellos.

PENATES: El sistema PENATES [PENATES] proporciona información sobre cerca de mil restaurantes en el área metropolitana de Boston.

SAPLEN [LGD97]: desarrollado por el grupo de Tratamiento del Habla del Departamento de Electrónica y Tecnología de Computadores

de la Universidad de Granada. Está diseñado para atender las peticiones y consultas de un restaurante de comida rápida.

Información turística

VOYAGER: El sistema VOYAGER [VOY95] proporciona información turística y de viajes en el área metropolitana de Boston. Ejemplos de consultas que pueden realizar los usuarios del sistema son la búsqueda de lugares de interés, visualización de mapas y fotografías aéreas, petición de información sobre rutas locales y solicitud de datos en tiempo real sobre el tráfico.

August [GLL99]: Sistema de diálogo desarrollado para atender a los visitantes del Centro Cultural de Estocolmo dentro del programa de Capital Cultural Europea durante 1998 y 1999. Utiliza un agente animado que interactúa con el usuario a través de la voz, texto, gestos y movimientos de cabeza. Se desarrolló con la intención que proporcionase información sobre varios dominios (el más sencillo de ellos es información sobre restaurantes).

NJFUN [LSK00]: es un sistema de diálogo en tiempo real que proporciona información sobre actividades que pueden realizarse en New Jersey.

Información deportiva

NBA Game Update Line [BCA02]: Proporciona información sobre resultados y estadísticas de partidos recientes de la NBA.

Información sobre correo electrónico

Mailman / AthosMail [TMH00]: Es un cliente de correo multilingüe (inglés y finlandés) diseñado para facilitar la lectura de los correos electrónicos mediante el uso del teléfono.

ELVIS: La tarea del sistema ELVIS (*EmaiL Voice Interactive System*) [WDG97] es gestionar telefónicamente las consultas a un servidor de correo electrónico.

Información y servicios misceláneos

LISTEN [LISTEN] (*Reading Tutor*): se trata de un tutor de lectura dirigido a niños. El funcionamiento básico consiste en mostrar por pantalla cuentos e historias y escuchar si la pronunciación que se realiza de los mismos es correcta. El tutor interviene cuando el lector comete errores, duda en alguna palabra, solicita ayuda o el programa tiene la sensación de que la lectura se realiza con dificultad.

Adapt [GBB00]: Se trata de un sistema de diálogo multimodal para realizar la búsqueda de apartamentos dentro del mercado inmobiliario de Estocolmo. El sistema, que utiliza información real obtenida de la web, permite que el usuario pueda interactuar mediante la voz o mediante pulsaciones en un mapa interactivo. Utiliza agentes animados.

Olga [BEM97]: Se trata de una figura animada tridimensional con la que el usuario puede comunicarse mediante el habla. El sistema ayuda al usuario a encontrar la información presente en las bases de datos utilizadas (consejos sobre la utilización de hornos microondas). Además, puede tomar su propia iniciativa, por ejemplo, dando consejos al usuario.

Doorman [MKS01]: Sistema que proporciona información y guía a las personas que visitan el departamento a modo de portero virtual. Entre las funcionalidades del sistema se encuentran la lectura de los correos electrónicos, realización de operaciones como la apertura de puertas y suministro de información sobre la situación de despachos y personal del departamento.

ITSPOKE [LS04]: (*Spoken Dialogue for Intelligent Tutoring System*) es un proyecto de la Universidad de Pittsburgh con el principal objetivo de permitir a los estudiantes interactuar con un tutor virtual utilizando la voz.

CoBotDS: En el proyecto CoBotDS [KIS02], extensión del proyecto Cobot, se desarrolló un sistema de diálogo para el acceso a un servidor de chat en Internet llamado LambdaMOO. El sistema provee comunicación en tiempo real entre un usuario y múltiples usuarios del chat. Utiliza el canal telefónico y lenguaje natural.

DiSCoH [AFG06]: (*Spoken Dialogue System for Conference Help*) es un sistema de diálogo con iniciativa mixta desarrollado para proporcionar información sobre conferencias.

SENECA: La arquitectura del sistema de diálogo SENECA [MHH04], desarrollado para su utilización en el entorno de automóviles, la constituyen cuatro módulos: el gestor de comandos, un módulo de comunicaciones GSM, un intercambiador de Cds y un módulo DSP, que realiza las operaciones relativas al procesado de la señal y del diálogo.

2.2 VoiceXML

Tal y como se ha descrito previamente, VoiceXML ha sido el lenguaje de programación fundamental para el desarrollo de la

aplicación. La centralidad de su posición en el presente proyecto, hace que sea necesario un mayor grado de detalle para poder comprender la trascendencia del mismo en la aplicación desarrollada. Conviene señalar que la mayor parte de la información que se describe en esta sección proviene de la página web principal del W3C que incluye las especificaciones del estándar [VXML20].

Los orígenes de VoiceXML se remontan a 1995, cuando se introdujo su utilización como un idioma de diálogo basado en XML. El diseño estaba destinado a simplificar el reconocimiento de la voz humana, entendiéndose como una aplicación clave en el proceso de desarrollo de un proyecto de AT&T conocido como "*Phone Markup Language*" (PML). Posteriormente, Lucent y Motorola continuaron trabajando en diversas variantes del mencionado proyecto.

Tres años más tarde, en 1998, el W3C organizó una conferencia cuyo objeto principal era el análisis de los novedosos navegadores de voz. En aquel entonces, AT&T y Lucent ya tenían encauzadas distintas variantes del proyecto PML original. Por su parte, Motorola había desarrollado VoxML, mientras que IBM había procedido en paralelo con una versión completamente propia que respondía al nombre de SpeechML.

En la conferencia pudo apreciarse que muchos otros de los asistentes estaban en fases avanzadas para el desarrollo de otros lenguajes similares para el diseño de diálogo. Tal es el caso de los lenguajes TalkML y VoiceHTML, respectivamente desarrollados por HP y Pipe Beach.

En este contexto, AT&T, IBM, Lucent y Motorola deciden aunar sus esfuerzos de investigación, concertando el llamado Foro VoiceXML [FOROVXML], cuya misión se resumía en el intento de definir un lenguaje común que los desarrolladores pudiesen utilizar para construir

las tan ansiadas aplicaciones de conversación. Se eligió XML como base sobre la que concentrar estos esfuerzos, dado que era evidente para los organizadores del foro que ésta era la tecnología que más se había aproximado a los objetivos.

En el año 2000, el Foro VoiceXML presentó al público las especificaciones de la primera versión del estándar (VoiceXML 1.0) [VXML10]. Poco después, VoiceXML 1.0 se presentó al W3C como base para la creación de una nueva norma internacional. Por último, la versión VoiceXML 2.0 es resultado de este trabajo, basado en las aportaciones de las empresas que forman parte del W3C, de otros Grupos de Trabajo de W3C y del público. Actualmente están ya disponibles las especificaciones de la que será la tercera generación del lenguaje, VoiceXML 3.0 [VXML30].

El principal objetivo de VoiceXML es aunar las funcionalidades ofrecidas por la web con la capacidad de interacción oral. El lenguaje permite la integración de servicios por voz y de datos usando el paradigma cliente-servidor. La arquitectura comprende los componentes reflejados en la Figura 5.

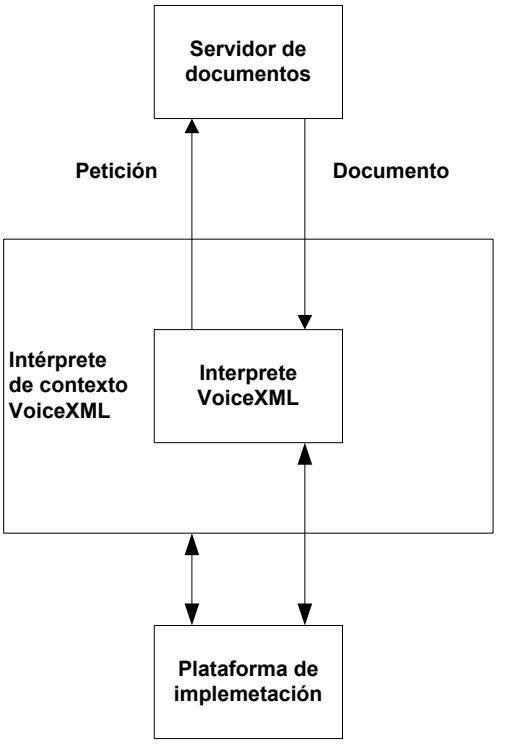

Figura 5. Arquitectura básica definida para el estándar VoiceXML

Un servicio de voz se puede definir como una secuencia de diálogos entre el usuario y la plataforma implementada. Estos diálogos son proporcionados por los servidores de documento (servidores web), que pueden ser externos a la plataforma implementada. Los servidores de documentos mantienen la lógica total del servicio, realizan operaciones en las bases de datos y gestionan los diálogos.

Un documento VoiceXML especifica cada uno de los estados del diálogo, que gestiona el intérprete VoiceXML. La entrada del usuario afecta a la interpretación del diálogo y se recoge en las peticiones realizadas a un servidor de documentos. El servidor de documentos responde con otro documento VoiceXML para continuar así la sesión del usuario con otros diálogos.

La plataforma de implementación está controlada por el intérprete de contextos VoiceXML. De esta forma, en una aplicación

interactiva de respuesta por voz, el intérprete de VoiceXML puede ser responsable de detectar una llamada entrante, adquirir el documento inicial de VoiceXML, contestar a la llamada y conducir el diálogo después de cada respuesta del usuario.

La plataforma genera eventos en respuesta a acciones del usuario (por ejemplo, proporcionar información, colgar, etc.) y eventos del sistema (por ejemplo, expiración por tiempo). Algunos de estos eventos actúan sobre el propio intérprete de VoiceXML, mientras otros actúan sobre el contexto.

Adicionalmente, VoiceXML proporciona las siguientes ventajas:

- Reduce al mínimo las interacciones cliente-servidor, especificando interacciones múltiples por documento.
- Separa el código de la interacción del usuario de la lógica del servicio.
- Promueve portabilidad del servicio a través de plataformas puestas en práctica. Es fácil de utilizar para las interacciones simples, pero proporciona características de lenguaje que posibilitan diálogos complejos.
- Las funcionalidades proporcionadas por el lenguaje incluyen conversión de texto a voz (text-to-speech), reproducción de archivos de audio, reconocimiento automático del habla, reconocimiento de entradas DTMF, grabación de entrada de voz, control del flujo de diálogo, y características de la telefonía tales como transferencia y desconexión de la llamada.

El lenguaje aporta los medios necesarios para recoger tonos DTMF y/o entrada de voz, asignando las entradas por voz a variables y tomando las decisiones que afectan a la interpretación de los

documentos. Además, un documento podrá ser enlazado a otros documentos a través de los identificadores uniformes de recursos (URIs).

VoiceXML se fundamenta en una serie de principios que están presentes desde la concepción primigenia de su propio diseño y que podemos resumir en los siguientes:

Portabilidad de servicios. El lenguaje promueve la portabilidad de servicios. Para ello, ha de servirse de la llamada técnica de abstracción de los recursos de la plataforma.

Simplificación de la interoperabilidad. El lenguaje acomoda diversidad de plataformas, formatos de audio, formatos de gramática, y esquemas de URI.

Claridad semántica. VoiceXML tiene una semántica bien definida. No se requiere la heurística del cliente para interpretarlo, puesto que queda perfectamente especificada de antemano.

Reconocimiento. El lenguaje reconoce las interpretaciones semánticas de las gramáticas y pone esta información a disposición de la aplicación.

Mecanismo de control de flujo. VoiceXML tiene un mecanismo de control de flujo. No está pensado para un uso intensivo de cómputo, ni de operaciones con bases de datos u operaciones de sistema. Estas operaciones se asume que serán gestionadas por los recursos externos al intérprete del documento. Se considera, en definitiva, que la lógica general del servicio, la administración del estado, la generación de diálogo y la secuencia del diálogo residen fuera del intérprete del documento.

Enlace documental. El lenguaje proporciona maneras de enlazar documentos utilizando URIs. Así mismo, también contempla la posibilidad de enviar datos a los scripts del servidor utilizando URIs.

Identificación de datos. VoiceXML proporciona las maneras de identificar exactamente qué datos enviará al servidor, y que método del HTTP (GET o POST) va a utilizar en el envío.

Para ilustrar de forma práctica el funcionamiento del VoiceXML presentamos a continuación dos ejemplos que nos acercan, de manera sencilla, al funcionamiento del lenguaje.

1.- En la Figura 6 podemos observar un ejemplo de codificación de un programa sencillo, tipo "hola mundo":

```
<? xml version = "1.0" encoding = "UTF-8"?>
<vxml xmlns= "http://www.w3.org/2001/vxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/vxml
http://www.w3.org/TR/voicexml20/vxml.xsd~~V "
version = "2.0">
   <form>
     <block> Hola mundo </block>
   </form>
</vxml>
```

Figura 6. Ejemplo *Hola mundo* en VoiceXML

En este primer ejemplo de la Figura 6 podemos apreciar con que la primera etiqueta que nos encontramos es <?xml>, esta etiqueta simplemente es la versión que se usa de *xml* y el tipo de codificación que tiene el documento, en este caso *UTF-8*, la siguiente etiqueta con la que nos encontramos es *<vxml>,* se detallan los posibles parámetros que puede tener en la Tabla 1.

Parámetro	Descripción
version	La versión de VoiceXML usada en el documento (obligatoria). La actual es la 2.1.
xmlns	El *namespace* designado por VoiceXML (obligatorio). En este caso es *http://www.w3.org/2001/vxml*.
xml:base	La dirección base para este documento (no obligatorio).
xml:lang	El lenguaje del documento, si es omitido será por defecto el específico de la plataforma (no obligatorio).
application	La dirección del documento raíz si tuviera (no obligatorio), más adelante entraremos más en detalle con este parámetro.

Tabla 1. Atributos de la etiqueta <vxml>

La información especificada en lenguaje es heredada en todo el documento: el valor de "xml:lang" se hereda por los elementos que también definen las propiedades de "xml:lang", tales como <*grammar*> y <*prompt*>, a menos que estos elementos especifiquen un valor alternativo.

Entre los diálogos, podemos distinguir dos tipos: formularios y menús. Los primeros – de formulario – presentan información y obtienen la información de entrada (en el caso de querer recoger información es necesario declarar una gramática). Por su parte, los diálogos de tipo menú ofrecen distintas posibilidades o alternativas y el usuario elige entre una de ellas sin ser necesario codificar una gramática, esta selección abrirá un diálogo nuevo basado en la selección.

En este caso, estamos ante un diálogo de formulario identificado como <form>, y con <block> reproduciremos el mensaje, dado que no hay diálogos sucesores, el programa no continuará más allá. Más adelante explicaremos con más detalle estas dos etiquetas.

2.- Nuestro segundo ejemplo en la Figura 7 requiere al usuario una selección de bebidas y transmite a continuación la selección del usuario a un script del servidor.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml20/vxml.xsd"
  version="2.0">
  <form>
      <field name="bebida">
        <prompt>¿Te gustaría café, té, leche o nada?
            </prompt>
            <grammar src="bebidas.grxml"
            type="application/srgs+xml"/>
      </field>
      <block>
            <submit next=
            "http://www.bebida.ejemplo.com/bebida2.asp"/>
      </block>
  </form>
</vxml>
```

Figura 7. Ejemplo de formulario VoiceXML

Apreciamos que en el ejemplo de la Figura 7 tenemos un diálogo de formulario. La etiqueta <*field*> define un elemento de entrada, es decir, recoge la información trasmitida por el usuario y la almacena en una variable con el mismo nombre que el definido para el campo ("*bebida*" en nuestro caso), para este ejemplo deberá existir una gramática "*bebidas.grxml*" que tendrá que reconocer los valores (café, té, leche o nada), cuando la maquina reconozca una de estas palabras se informará en la variable "*bebida*", como vemos esta gramática viene definida en la variable <*grammar*> más adelante entraremos en más detalle sobre esta etiqueta. Como vemos en <block> mandaremos el flujo a otro documento "*bebida2.asp*", usando la etiqueta <*submit*>.

Podemos imaginar cómo se estaría produciendo la interacción entre el usuario y la aplicación gracias al hipotético diálogo que presentamos a continuación:

C (computadora): ¿Te gustaría café, té, leche o nada?
U (Usuario): Zumo de naranja.
C: No le he entendido. (Es un mensaje por defecto de la plataforma, cuando lo que el usuario dice no es una de las opciones que espera el sistema (es decir, una opción no contemplada en la gramática "bebidas.grxml").
C: ¿Te gustaría café, té, leche o nada?
U: Té
C: (continua en el documento bebida2.asp)

De este modo, no debemos olvidar que para que las entradas del usuario sean comprendidas o reconocidas por la aplicación en cuestión, es necesario tenerlas definidas en gramáticas. Si esto no fuese así, el programa nos devolvería un mensaje de error, expresando su falta de entendimiento.

2.2.1 Conceptos adicionales

Un documento en VoiceXML (o un sistema de documentos relacionados entendido como una aplicación) puede entenderse como una máquina de estado conversacional. Es decir, el usuario está siempre en un estado conversacional –o de diálogo - constante con la computadora. Cada diálogo determina la transición al diálogo siguiente. Estas transiciones se especifican mediante URIs, que proporcionan el documento y/o diálogo posteriores al actual. Si un URI no apunta a un documento, se infiere que actúa sobre el documento actual en el que se incluye. Si no apunta a un diálogo, se considerará que actúa sobre el primer diálogo del documento. La ejecución terminará cuando un diálogo no haga referencia a su sucesor, o si se introduce algún elemento que provoque una salida explícita de la conversación mantenida.

2.2.1.1 Diálogos y subdiálogos

Un subdiálogo es equivalente a una llamada a una función, de forma que proporciona un mecanismo para invocar una nueva interacción y retornar seguidamente al formulario original. Las variables, gramáticas, y la información del estado de las variables se guardan y están disponibles para el documento que hizo la llamada. Los subdiálogos se pueden utilizar, por ejemplo, para crear un conjunto de componentes que se pueden compartir entre documentos en una aplicación, o para crear una biblioteca reutilizable de diálogos compartidos entre muchas aplicaciones.

Los subdiálogos permiten de este modo descomponer diálogos complejos para crear componentes reutilizables. Por ejemplo, la solicitud de información una cuenta bancaria puede implicar la recopilación de varios datos del usuario, tales como número de cuenta, número de teléfono, etc. Un servicio bancario puede dividirse en varias aplicaciones independientes que podrían compartir este bloque básico de petición de datos. De este modo, sería lógico construirlo como subdiálogo tal y como se ilustra en el ejemplo mostrado en la Figura 8 y Figura 9. El primer documento, *app.vxml*, busca conseguir la información de la cuenta. La información de la cuenta es obtenida usando un elemento subdiálogo que invoca otro documento de VoiceXML para solicitar la entrada del usuario.

Mientras que se está ejecutando el segundo documento, el diálogo que llama al primero se suspende, esperando devolver la información. El segundo documento proporciona los resultados de las interacciones del usuario utilizando el elemento < *return* >, y los valores resultantes se recuperan en la variable definida en la etiqueta <*subdialog*>.

Aplicación de Servicio al Cliente (app.*vxml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/vxml
 http://www.w3.org/TR/voicexml20/vxml.xsd"
 version="2.0">
  <form id="billing_adjustment">
    <var name="account_number"/>
    <var name="home_phone"/>
    <subdialog name="accountinfo"
            src="acct_info.vxml#basic">
            <filled>
<!—La variable definida como "accountinfo" es devuelta
como  un objeto ECMAScript  y contendrá las variables que
se hayan especificado en el elemento "return" dentro de
subdiálogo.-->
                  <assign name="account_number"
                        expr="accountinfo.acctnum"/>
                  <assign name="home_phone"
                  expr="accountinfo.acctphone"/>
            </filled>
    </subdialog>
  </form>
</vxml>
```

Figura 8. Ejemplo de aplicación de un servicio al cliente en VoiceXML

Como vemos en la Figura 8, lo primero que se hace es declarar una llamada a un subdiálogo mediante la etiqueta *<subdialog>*, en la Tabla 2 podemos ver cuáles son sus atributos más comunes.

Parámetro	**Descripción**
Name	El nombre del subdiálogo, este nombre será un objeto *ECMAScript* y se usará para recoger las variables informadas en el propio subdiálogo.
Src	La URI del subdiálogo.

Tabla 2. Parámetros más comunes de la etiqueta <subdialog>

Dentro de la llamada al subdiálogo recogeremos las variables que se hayan informado en este, esto se realizará mediante la etiqueta *<assign>* como vemos usamos el nombre del subdiálogo para recogerlas y las informamos en variables declaradas en el propio documento.

Documento que contiene la información del subdiálogo de la cuenta (*acct_info.vxml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/vxml
 http://www.w3.org/TR/voicexml20/vxml.xsd"
 version="2.0">
  <form id="basic">
    <field name="acctnum">
            <grammar type="application/srgs+xml"
                  src="/grammars/digits.grxml"/>
                  <prompt> ¿Cuál es su número de cuenta?
                  </prompt>
    </field>
    <field name="acctphone">
      <grammar type="application/srgs+xml"
            src="/grammars/phone_numbers.grxml"/>
      <prompt> ¿Cuál es su número de teléfono? </prompt>
      <filled>
<!—Como vemos enviamos las variables informadas en el
subdiálogo mediante el element “return”. -->
            <return namelist="acctnum acctphone"/>
      </filled>
    </field>
   </form>
</vxml>
```

Figura 9. Ejemplo de subdiálogo VoiceXML

Como vemos en la Figura 9, declaramos un documento que contendría el subdiálogo que vamos a usar, este subdiálogo requiere las variables número de cuenta al usuario, obtiene su valor mediante gramáticas y las transfiere al documento llamante mediante la etiqueta <*return*>.

Los subdiálogos agregan un nuevo contexto a la ejecución cuando son invocados. El subdiálogo puede ser un nuevo diálogo dentro del documento existente, o un nuevo diálogo dentro de un nuevo documento. Asimismo, los subdiálogos pueden estar compuestos por varios documentos.

En la Figura 10 se muestra el flujo de la ejecución donde se aprecia una secuencia de documentos (D) en transición a un subdiálogo

(SD). El contexto de la ejecución en el diálogo D2 se suspende cuando se invoca el subdiálogo SD1 en el documento *sdl.vxml* y se crea uno nuevo para los subdiálogos. El subdiálogo al que se llama es multi-documento, y después de ejecutar *sdl.vxml* se pasa la ejecución a *sd2.vxml* (a través del elemento <*goto*>). Una vez que el diálogo en *sd2.vxml* se ejecuta, el control de éste es devuelto directamente al diálogo D2 (mediante el elemento <*return*>).

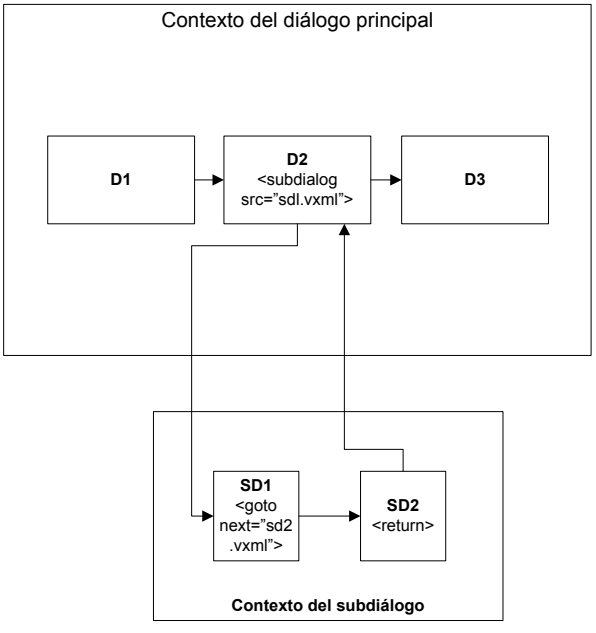

Figura 10. Esquema de funcionamiento de un subdiálogo integrado por varios documentos

Es posible que un subdiálogo llame a otro subdiálogo, en ese caso se creara otro contexto y se suspendería el del llamante. En la Figura 11 podemos observar cómo se daría esta casuística, en la ejecución normal del diálogo llamaremos desde D2 al subdiálogo SD1 (documento *sd1.vxml)*, se crearía un nuevo contexto y se suspendería el del diálogo, una vez en el subdiálogo SD1 llamaríamos al subdiálogo SD2 (documento *sd2.vmxl),* creándose un nuevo contexto y suspendiéndose el del actual. Una vez finalizado SD2, se devolverá el control a SD1, que a su vez lo devolverá de vuelta a D2, el diálogo original.

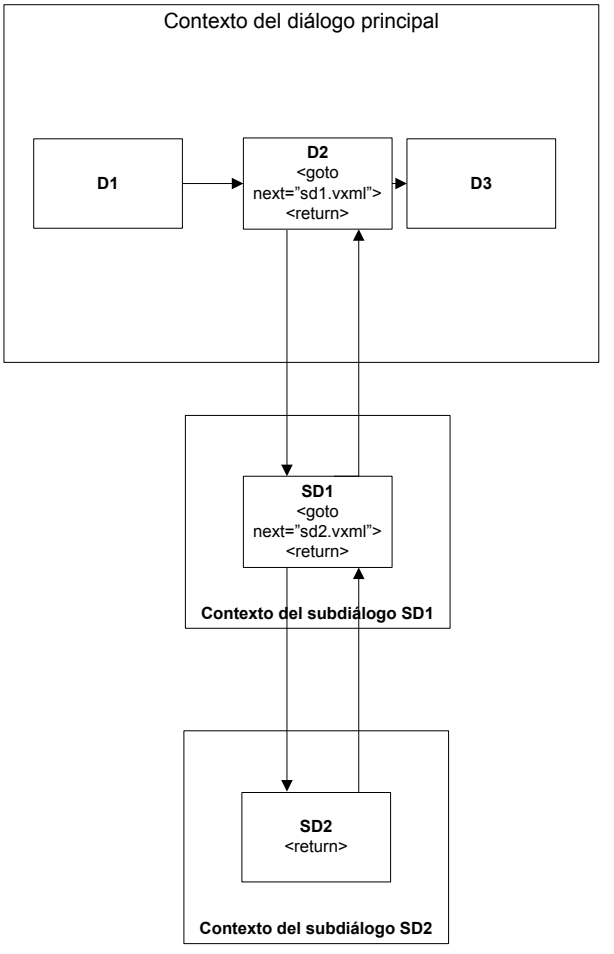

Figura 11. Ejemplo de subdiálogo integrado por varios documentos

2.2.1.2 Sesiones

Se inicia una sesión cuando el usuario comienza a interactuar recíprocamente con el intérprete de contexto VoiceXML. La sesión continúa hasta que se cargan y se procesan los documentos. Por último, la sesión finaliza cuando el usuario así lo requiere, aunque también es posible que sea un documento o el contexto del intérprete el que ponga fin a la misma.

2.2.1.3 Raíz

En el atributo "*application*" de la etiqueta *<vxml> podemos definir* como un conjunto de documentos que comparten el mismo

documento raíz. En este sentido, cada vez que el usuario interactúe con un documento en una aplicación, su documento raíz también se cargará, y permanecerá cargado mientras el usuario navegue por los distintos documentos de la misma aplicación, descargándose cuando el usuario navegue a un documento de otra aplicación. Mientras el documento raíz este cargado, sus variables estarán disponibles para el resto de documentos.

2.2.1.4 Gramáticas

Cada diálogo tiene una o más gramáticas de voz y/o de DTMF asociadas a él. En las aplicaciones dirigidas por el sistema, las gramáticas se activan solamente cuando el usuario está en el diálogo asociado. En aplicaciones mixtas, donde el usuario y la máquina se alternan para determinar qué hacer a continuación, los diálogos pueden activar sus gramáticas asociadas incluso cuando el usuario está en otro diálogo dentro del mismo documento, u otro documento se encuentra cargado de la misma aplicación. Así, si el usuario pronuncia algo reconocido por alguna gramática activa la ejecución se trasladará al diálogo asociado.

2.2.1.5 Eventos

VoiceXML proporciona un mecanismo de *relleno de formularios* para manejar las entradas previstas del usuario. Además, VoiceXML define un mecanismo para manejar los acontecimientos no cubiertos por estos formularios.

Los eventos pueden ser lanzados por la plataforma VoiceXML por distintas razones. Por ejemplo, cuando el usuario no responde, cuando no lo hace de manera inteligible, cuando se registran peticiones de ayuda, etc.

Además, el intérprete también lanza excepciones si encuentra un error semántico en un documento de VoiceXML. Las excepciones se recogen por los elementos "*Catch*". Se pueden especificar elementos "*Catch*" cada vez que se pueda llegar a una situación en la que se genere una excepción.

2.2.2 Elementos del lenguaje VoiceXML

La Tabla 3 muestra los distintos elementos que se pueden dar en VoiceXML junto con una pequeña descripción.

Elemento	**Descripción**
<assign>	Asigna un valor a una variable
<audio>	Ejecuta un clip de audio
<block>	Un contenedor de código ejecutable (no interactivo).
<catch>	Captura un evento.
<choice>	Define un elemento de menú.
<clear>	Limpia una o más variables en el formulario.
<disconnect>	Desconecta una sesión.
<else>	Usado dentro de una sentencia <if>.
<elseif>	Usado dentro de la sentencia <if>.
<enumerate>	Término para enumerar la selección de opciones en un menú.
<error>	Captura un error de evento
<exit>	Salir de una sesión.
<field>	Declara una entrada de campo en un formulario.
<filled>	Una acción es ejecutada cuando los campos son llenados.
<form>	Un diálogo para presentar información y recoger datos.
<goto>	Ir a otro diálogo en el mismo o en otro documento.
<grammar>	Especifica un reconocimiento de voz o gramática DTMF.
<help>	Captura un evento de ayuda.
<if>	Condicional simple.

<initial>	Declara inicializaciones lógicas sobre entradas en un formulario.
<link>	Especifica transiciones comunes para todos los diálogos en el alcance del link.
<log>	Genera información para depuración de errores.
<menu>	Un diálogo para elegir entre destinos alternativos.
<meta>	Define elementos metadata como el par nombre/valor.
<metadata>	Define información metadata utilizando un esquema de metadatos.
<noinput>	Captura un evento de no entrada.
<nomatch>	Captura un evento de no enlace.
<object>	Interactúa con una extensión personalizada.
<option>	Especifica una opción en un campo <*field*>
<param>	Parámetro en <*object*> o <*subdialog*>
<prompt>	Cola de síntesis de voz y salida de audio para el usuario.
<property>	Configuración para el control de la implementación de la plataforma.
<record>	Graba un audio básico.
<reprompt>	Realiza la ejecución de un "*prompt*", cuando un campo se visita nuevamente después de un acontecimiento.
<return>	Retorna de un subdiálogo.
<script>	Especifica un bloque de *ECMAScript* del lado del cliente.
<subdialog>	Invoca otro diálogo como también un subdiálogo en la sesión activa.
<submit>	Envía valores al servidor.
<throw>	Lanza un evento.
<transfer>	Transfiere las llamadas a otro destino.
<value>	Inserta los valores de una expresión en un "*prompt*".
<var>	Declara una variable.
<vxml>	Elemento ubicado en el nivel superior en cada documento de tipo VoiceXML.

Tabla 3 Elementos VoiceXML

2.2.3 Estructura y ejecución de documento VoiceXML

Distinguimos entre la ejecución dentro de un mismo documento o la ejecución de varios documentos.

2.2.3.1 Ejecución dentro de un documento

Tal y como se ha descrito previamente, la ejecución de un documento VoiceXML comienza en el primer diálogo por defecto. Cada diálogo ejecutado determina el siguiente diálogo. Cuando un diálogo no especifica un diálogo sucesor, la ejecución del documento finaliza. La Figura 12 muestra un ejemplo que ilustra el proceso descrito.

Tal y como se observa, se ha definido una variable de documento llamada "*hi*" para que lleve a cabo un saludo al usuario. Su valor se utiliza como *prompt* en el primer formulario. Una vez que el primer formulario realiza el saludo, va al formulario nombrado como "*say_goodbye*", mediante el cual se despide del usuario, precisamente con un "Adiós".

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/vxml
 http://www.w3.org/TR/voicexml20/vxml.xsd"
 version="2.0">
 <var name="hi" expr="'Hola mundo'"/>
 <form>
      <block>
            <value expr="hi"/>
            <goto next="#say_goodbye"/>
      </block>
 </form>
 <form id="say_goodbye">
      <block>
            Adiós
      </block>
 </form>
</vxml>
```

Figura 12. Ejemplo de documento VoiceXML con dos formularios

Alternativamente los formularios pueden ser combinados, tal y como muestra la Figura 13 en el elemento <*block*> que primero reproducimos la variable "*hi*" (hola mundo), y a continuación reproducimos el mensaje *"Adiós"*.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml20/vxml.xsd"
  version="2.0">
 <var name="hi" expr="'Hola mundo'"/>
 <form>
      <block>
            <value expr="hi"/> Adios
      </block>
 </form>
</vxml>
```

Figura 13. Ejemplo de formularios combinados

2.2.3.2 Ejecutar una aplicación multi-documento

Normalmente, cada documento funciona como una aplicación aislada. En los casos donde se requiera que los documentos múltiples trabajen conjuntamente como una aplicación, ha de seleccionarse un documento que actúe como *documento raíz de la aplicación.* El resto de los documentos pasarán automáticamente a ser considerados como *documentos hoja* de la aplicación.

Cada *documento hoja* señala al documento raíz de la aplicación en su cabecera <*vxml*>. Tal y como se ha descrito previamente, cada vez que se llama al intérprete para cargar y para ejecutar un documento hoja en la aplicación, lo primero que se cargará será siempre el documento raíz. El documento raíz permanecerá cargado hasta que el intérprete cargue un documento que pertenece a otra aplicación.

Son muchos los beneficios para el empleo de aplicaciones multi-documento. En primer lugar, las variables declaradas en el *documento raíz* estarán disponibles para ser usadas por todos los *documentos hoja* del documento. En segundo lugar, las etiquetas <*property*> y <*catch*>

del *documento raíz* especifican respectivamente valores por defecto de propiedades de los *documentos hoja* y la captura de excepciones por defecto en los documentos hoja. Además, se puede definir código ECMAScript en el *documento raíz* mediante la etiqueta *<script>*, y este código podrá ser usado en los *documentos hoja*. Por último, las gramáticas definidas en el *documento raíz* están activas cuando el usuario está en cualquier *documento hoja*, de modo que éste puede interactuar con los formularios, los links y menús del documento raíz.

La Figura 14 y 15 muestran un ejemplo de aplicación con respectivamente un documento raíz y uno hoja.

Documento raíz de la aplicación (app-root.vxml)

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml20/vxml.xsd"
  version="2.0">
  <var name="bye" expr="'adios'"/>
  <link next="operator_xfer.vxml">
  <grammar type="application/srgs+xml"
   root="root" version="1.0">
            <rule id="root" scope="public">operator</rule>
  </grammar>
 </link>
</vxml>
```

Figura 14. Ejemplo *documento raíz*

Hoja del Documento (*leaf.vxml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/vxml
 http://www.w3.org/TR/voicexml20/vxml.xsd"
 version="2.0" application="app-root.vxml">
 <form id="say_goodbye">
      <field name="answer">
            <grammar type="application/srgs+xml"
                  src="/grammars/boolean.grxml"/>
            <prompt>
                  ¿Debemos decir
                  <value expr="application.bye"/>?
            </prompt>
```

```
            <filled>
                  <if cond="answer">
                        <exit/>
                  </if>
                  <clear namelist="answer"/>
            </filled>
      </field>
 </form>
</vxml>
```

Figura 15. Ejemplo *documento hoja*

En el ejemplo mostrado en las Figura 14 y 15, la aplicación está diseñada para que el fichero *leaf.vxml* sea descargado en primer lugar. El atributo *"application" de* <*vxml*> del documento hoja indica que "*app-root.vxml*" debe ser utilizada como documento raíz. De esta forma, al cargarse el *documento raíz* "*app-root.vxml"*, se crea la variable "*bye"* y se define un vínculo (*link*) que enlaza al fichero "*operator_xfer.vxml"*, cuando el usuario diga "operador". La interacción comenzará, pues, en el formulario *"say_goodbye"*, tal y como se describe en el siguiente ejemplo:

C (computadora): ¿Debemos decir adiós?
H (humano): Si.
C: No le entiendo (mensaje por defecto de la plataforma).
C: ¿Podemos decir adiós?
H: adiós.
C: No le entiendo.
H: Operador.
C: (va a operator_xfer.vxml, que transfiere a un operador humano).

Debe tenerse siempre presente que cuando el usuario está en una aplicación multi-documento al menos dos documentos están cargados al mismo tiempo. El elemento <*vxml*> de un *documento raíz* no tiene el atributo "*application"* definido, sin embargo el del *documento hoja* sí (con el nombre del *documento raíz*). El intérprete

siempre tiene el documento raíz cargado pero no siempre tiene cargado un documento hoja.

2.2.4 Constructores del diálogo

2.2.4.1 Formularios

Los formularios son el componente principal de los documentos VoiceXML. Un formulario contiene:

- Un conjunto de campos, estos se dividen entre campos de entrada (los que recogen información del usuario) y campos de control.
- Declaraciones de variables (distintas de la variable de formulario).
- Manejadores de Eventos.
- Acciones "*filled*", bloques de procedimiento lógicos que se ejecutan cuando una combinación de variables de entrada se cumple (el usuario les asigna un valor).

Los atributos de los formularios aparecen en la Tabla 4.

id	El nombre del formulario. Si está especificado, el formulario podrá ser llamado desde el mismo documento, o desde otro documento, por ejemplo: <form id="weather">, <goto next= "#weather">.
Scope	El alcance por defecto de las gramáticas del formulario, puede ser sólo en este diálogo, en el formulario o en el root (estará activa en todos los documentos).

Tabla 4. Atributos de los formularios VoiceXML

Dentro de los formularios nos podemos encontrar elementos de entrada y de control.

Elementos de entrada

Un elemento de entrada especifica una variable para recoger la información del usuario. Los campos de entrada incluyen las gramáticas que definen las entradas permitidas y los manejadores de eventos. Un elemento de entrada puede también tener un elemento *<filled>* que defina una acción a tomar.

Los elementos de entrada se resumen en la Tabla 5.

<field>	Un elemento de entrada cuyo valor se obtiene vía gramáticas de ASR o de DTMF.
<record>	Un elemento de entrada cuyo valor es un clip de audio grabado por el usuario. Un elemento <*record*> puede recoger un mensaje de buzón de voz.
<transfer>	Un elemento de entrada que transfiere a un usuario a otro número de teléfono.
<object>	Este elemento de entrada invoca un "objeto" con varios parámetros. El resultado es un objeto ECMAScript.
<subdialog>	Un elemento de entrada <*subdialog*> es como una llamada a función. Invoca otro diálogo en la página actual, o de otro documento de VoiceXML. Devuelve un objeto de ECMAScript como su resultado

Tabla 5. Posibles campos de entrada en los formularios VoiceXML

Elementos de control

Los elementos de control sirven para dar avisos, controlar el flujo inicial etc. Tal y como muestra la Tabla 6, existen dos tipos de campos de control.

<block>	Una secuencia de código ejecutable usada para dar avisos (*prompting*) y realizar cálculos, pero no para recoger información de entrada.
<initial>	Este elemento controla la interacción inicial en un formulario.

Tabla 6. Posibles campos de control en los formularios VoiceXML

Variables y Condiciones de los Formularios

Cada formulario tiene una variable asociada que por defecto es fijada cuando se ejecuta el formulario. Esta variable contendrá el resultado de la ejecución del formulario.

Normalmente, los elementos de entrada se declaran mediante un nombre la variable asociada, sin embargo, los de control no. Generalmente a estas variables no se les asigna valores por defecto, ya que si esta variable tiene un valor no se ejecutará el elemento. Estas variables pueden liberarse del valor previamente asignado (usando <*clear*>).

Un elemento puede tener una condición de protección que se active solamente cuando no se han recogido datos o cuando se han llenado campos. De este modo, un elemento "*block*" podría ejecutarse solamente cuando cierta condición sea verdadera. Así, el control se puede realizar sobre la orden en la cual las posibles elecciones del formulario son seleccionadas. No obstante, en general, muchos diálogos pueden ser construidos sin usar este nivel de la complejidad.

En resumen, los formularios <*form*> tienen los atributos que se muestran en la Tabla 7.

name	El nombre de la variable asociada al formulario, como se ha indicado antes contendrá el valor de la ejecución de este.
expr	Valor inicial de la variable asociada al formulario, para que se ejecute el formulario esta variable no debe de venir informada, si la informamos y se quiere que se ejecute deberemos limpiarla con <*clear*>.
cond	Condición para la ejecución del formulario, por defecto está informada a true.

Tabla 7. Atributos de los formularios VoiceXML

2.2.4.2 Menús

Un menú es un tipo de diálogo donde se ofrecen al usuario diferentes opciones y dependiendo de la opción elegida el flujo continuará en un documento u otro. El menú que se muestra en la Figura 16 ofrece al usuario tres opciones: Deportes, tiempo o noticias, según la opción que se elija el flujo pasará a otro diálogo *sports.vxml, weather.vxml* o *news.vxml* respectivamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd">
<menu>
  <prompt>
    Bienvenido al sistema, elija una opción entre:
<enumerate/>
  </prompt>
  <choice
   next="http://www.sports.example.com/vxml/sports.vxml">
     Deportes
  </choice>
  <choice
   next="http://www.weather.example.com/weather.vxml">
     Tiempo
  </choice>
  <choice

next="http://www.stargazer.example.com/voice/news.vxml">
     Noticias
  </choice>
  <noinput>Por favor, seleccione una opción entre
<enumerate/></noinput>
</menu>
</vxml>
```

Figura 16. Ejemplo menú VoiceXML

Este diálogo podría seguir de la forma siguiente:

- C (computadora): Bienvenido a la aplicación, elija una opción entre: Deportes, Tiempo, Noticias.

- H (humano): Astrología.
- C: No entiendo lo que ha dicho. (mensaje predefinido.)
- C: Bienvenido al sistema, elija una opción entre: Deportes, Tiempo, Noticias.
- H: Deportes.
- C: (procesa http://www.sports.example.com/vxml/start.vxml)

El conjunto de atributos del elemento menú se resumen en la Tabla 8.

id	El identificador del menú. Permite que el menú sea llamado desde una etiqueta *<goto>* o una etiqueta *<submit>*.
scope	El alcance de la gramática de menú. Si es diálogo (por defecto), las gramáticas del menú están solamente activas cuando el usuario navega en él. Si el alcance es un documento, sus gramáticas están activas sobre el documento entero.
dtmf	Cuando se configura como verdadero, las primeras nueve opciones que no han especificado explícitamente un valor para el atributo del DTMF son dadas implícitamente como "1", "2", etc. Las opciones restantes no tienen especificaciones explícitas para los atributos del DTMF, es decir, no serán asignadas valores de DTMF. Si hay opciones que han especificado su propio DTMF como la utilización de secuencias como "*", "#", o "0", un *"error.badfetch"* será lanzado. El valor por defecto es falso.
accept	Cuando se configura como *"exact"* (por defecto), el texto de los elementos escogidos en el menú define la frase exacta que se reconocerá. Cuando se configura como *"approximate"*, el texto de los elementos escogidos define una frase aproximada del reconocimiento. Cada *<choice>* puede sobrescribir este atributo.

Tabla 8. Atributos del elemento <menu>

El elemento <*choice*> responde a varios propósitos:

- Puede especificar una gramática de "*speech*".
- Puede especificar una gramática de DTMF.
- El contenido se puede utilizar para enumerar las opciones disponibles para el usuario, con <*enumerate*> tal y como se observa en el ejemplo de la Figura 16.
- Especifica el lanzamiento de un evento o el URI al cual se redirecciona cuando se selecciona la opción.

Los atributos del elemento <*choice*> se resumen en la Tabla 9.

dtmf	Especifica una gramática DTMF para esta opción. Por ejemplo dtfm="1 2 3 #".
accept	Cuando se configura como "*exact*" (por defecto), el texto de los elementos escogidos en el menú define la frase exacta que se reconocerá. Cuando se configura como "*approximate*" el texto de los elementos escogidos definen una frase aproximada del reconocimiento.
next	El URI del diálogo o del documento siguiente.
expr	Especifica una expresión que evalúa a que URI ir, en vez de especificarlo en "*next*".
event	Especifica el lanzamiento de un evento en vez de especificar "*next*".
eventexpr	Una expresión de *ECMAScript* que evalúa el evento que se lanzará.
message	Un mensaje que proporciona información adicional sobre el evento que es lanzado. El mensaje está disponible dentro del alcance del evento (se recupera con *catch*).
messageexpr	Una expresión de *ECMAScript* que evalúa el mensaje del evento que se lanzará.

Tabla 9. Atributos del elemento <*choice*>

2.3 Plataforma Voxeo Evolution

Voxeo Corporation es una compañía que ha desarrollado la plataforma *Voxeo Evolution* [VOXEOE], la cual permite crear,

implementar y probar aplicaciones de voz desarrolladas en VoiceXML de forma telefónica y completamente gratuita.

Además, *Voxeo* facilita un servidor web para el alojamiento de las aplicaciones de voz, un número de teléfono dedicado para su uso durante la prueba de las aplicaciones, registro y depuración en tiempo real de la aplicación, conectividad de voz sobre IP, foros de soporte donde se puede interactuar con la Corporación *Voxeo* y con otros miembros de la comunidad y soporte extremo 24x7. Principalmente, para completar este apartado, nos hemos basado en su página oficial [VOXEOE].

2.3.1 ¿Cómo funciona Voxeo Evolution?

Para explicar el funcionamiento de la plataforma de voz *Voxeo Evolution* se va a comparar el modo de operación de un servidor Web con el de la plataforma.

En la Web, los exploradores realizan solicitudes a los servidores Web por medio de protocolos de Internet como HTTP. Los servidores Web alojan páginas estáticas y aplicaciones dinámicas HTML que se proveen al explorador. El resultado es una página visual que se visualiza en un navegador Web. Este mecanismo lo podemos observar en la Figura 17.

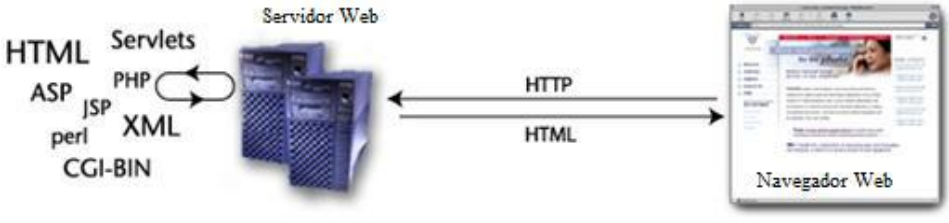

Figura 17. Interpretación de una página web [VoxeoHIW]

En la plataforma *Voxeo*, las aplicaciones de voz siguen el mismo modelo de navegador que las páginas Web estándares. La diferencia es

que, en vez de HTML, se utilizan lenguajes de marcado por voz (VoiceXML, CCXML [CCXML] o CallXML [CallXML]) para desarrollar las aplicaciones, y en vez de navegador, el usuario puede acceder a la información a través del teléfono. La Figura 18 describe este procedimiento. *Voxeo Evolution* es el enlace entre el teléfono y el servidor Web, llamado "*Voice Center Network*".

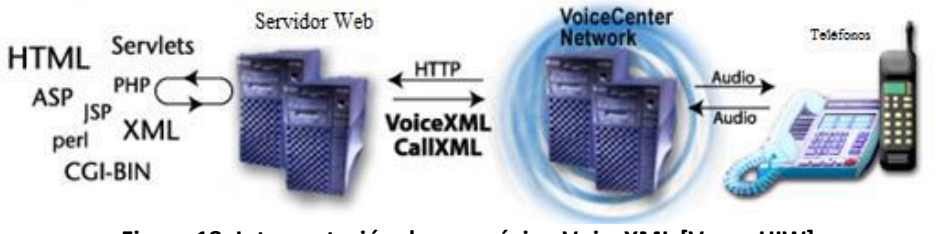

Figura 18. Interpretación de una página VoiceXML [VoxeoHIW]

2.3.2 Desarrollo de una aplicación VoiceXML

En este apartado se describe cómo desarrollar y configurar una aplicación VoiceXML usando *Voxeo Evolution*. La explicación se basa principalmente en la "guía rápida" que facilita la aplicación [VOXEOGUIA].

Los pasos fundamentales para la utilización de la plataforma *Voxeo* son:

- Crear una cuenta de desarrollador (no se entrará en más detalle, dada la sencillez del procedimiento).
- Seleccionar la plataforma de la aplicación. En nuestro caso, VoiceXML, aunque no debemos olvidar que *Voxeo Evolution* ofrece muchas más opciones.
- Desarrollar el código VoiceXML y hospedarlo en el servidor. Lo explicaremos con más detalle en el siguiente apartado.
- Asignar a la aplicación un número de teléfono. Lo explicaremos con más detalle en un apartado más adelante.
- Esperar un breve tiempo para que se actualice la información en la red y poder llamar a la aplicación.

2.3.2.1 Desarrollo de la aplicación y hospedarla en la plataforma.

Después de haber dado de alta en *Voxeo Evolution* un usuario y haber seleccionado nuestra plataforma, habrá que desarrollar una aplicación en VoiceXML y hospedarla en una de las carpetas que nos ofrece *Voxeo Evolution*. Así, al estar en su servidor, estará disponible 24 horas al día. En la página principal, mostrada en la Figura 19, seleccionamos la funcionalidad "*Files, Logs & Reports*" (Figura 20).

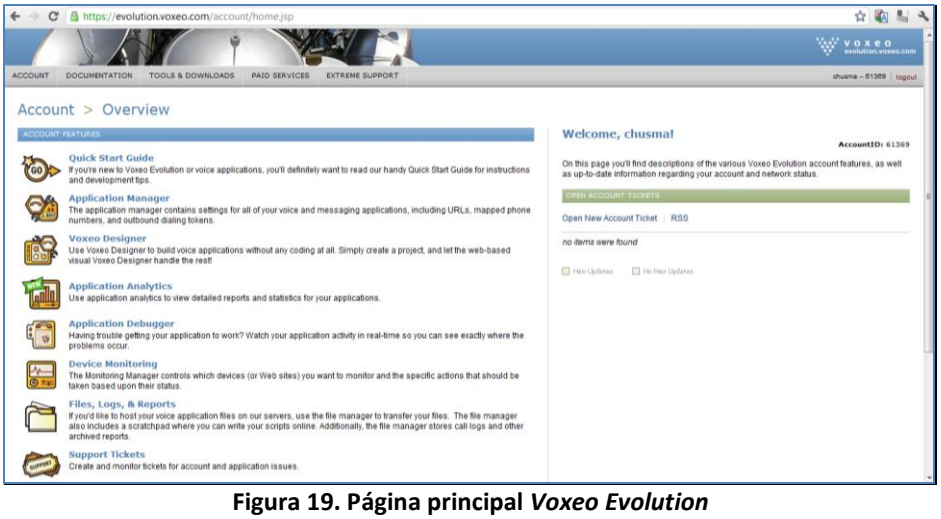

Figura 19. Página principal *Voxeo Evolution*

Figura 20. Utilidad Files, Logs & Reports

Como se observa en la Figura 19, la funcionalidad “Files, Logs & Reports” permite seleccionar un archivo de nuestra computadora y hospedarlo en la plataforma. Esta función también permite crear nuevos directorios para tener la información más ordenada, así como crear directamente ficheros en plataforma, renombrarlos o modificar su contenido. Como ejemplo práctico procederemos a subir un fichero con el ejemplo “Hola mundo” mostrado anteriormente.

Tal y como muestra la Figura 21, se ha almacenado el fichero en la carpeta “ejemplos”, junto a otras aplicaciones que se han desarrollado a lo largo del proyecto, en el apartado siguiente se verá cómo asignarle un número de teléfono para poder acceder oralmente.

Figura 21. Opciones ofrecidas por la funcionalidad “*Files, Logs & Reports*”

2.3.2.2 Asignación de número teléfono y acceso a la aplicación

En el apartado anterior se creó una aplicación, ahora se asignará un número de teléfono para acceder a ella telefónicamente. Para ello, el primer paso es entrar en la página principal de la plataforma, previamente mostrada en la Figura 19.

En este caso, seleccionamos la funcionalidad *"Application Manager"*, que es donde vamos a poder administrar todas nuestras aplicaciones (Figura 22).

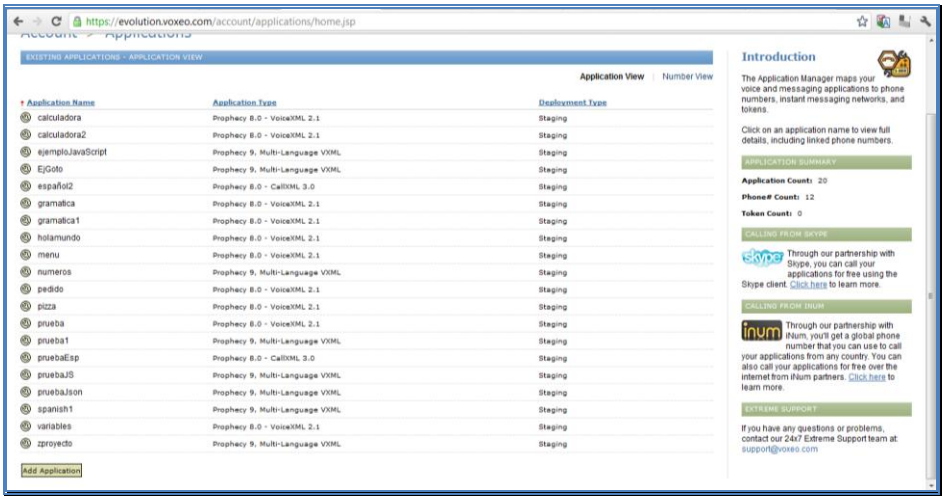

Figura 22. Funcionalidad "*Application Manager*"

En la pantalla mostrada en la Figura 22 se tendrán todas las aplicaciones que hayamos ido desarrollando con nuestro usuario. En nuestro caso, habremos de pinchar en "*Añadir una nueva aplicación*" para crear una aplicación nueva (Figura 23).

Figura 23. Añadir una nueva aplicación mediante Voxeo Evolution

Indicamos el nombre de la aplicación y seleccionamos como tipo *"Voice phone calls",* ya que en este ejemplo se quiere solo permitir únicamente llamadas telefónicas a la aplicación. Seguidamente, podremos configurar el conjunto de opciones mostrado en la Figura 24.

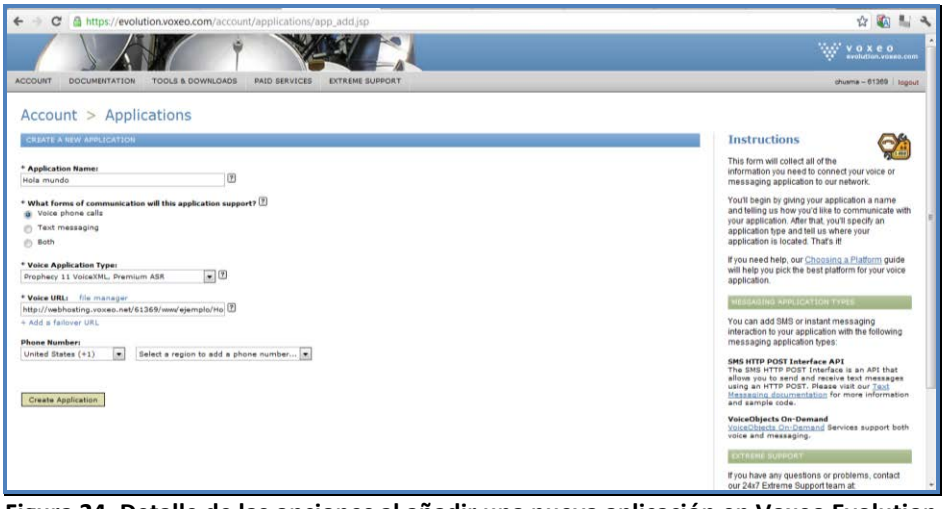

Figura 24. Detalle de las opciones al añadir una nueva aplicación en Voxeo Evolution

En "*Voice Aplication type*" seleccionamos *"Prophecy 11 VoiceXml Premium ASR/TTS"*, y en *"Voice URL"*, si el archivo se encontrará en un servidor indicaríamos la dirección URL, en nuestro caso al haberlo cargado en las carpetas que nos ofrecen pinchamos en *"file manager"* buscamos el fichero cargado y seleccionamos el enlace *"Create Application"*. Una vez realizado esto ya estará creada la aplicación, acto seguido seleccionamos *"Application"* (Figura 25), buscamos la aplicación que hemos creado y la seleccionamos.

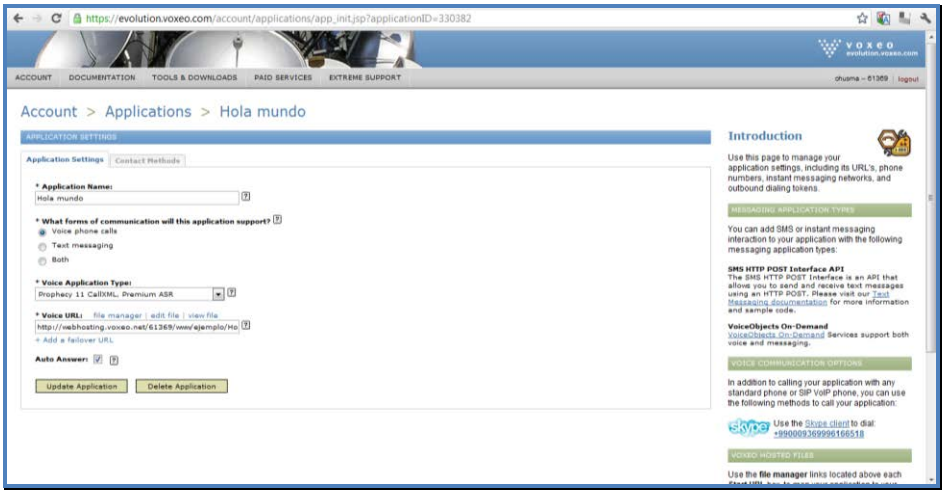

Figura 25. Funcionalidad "*Applications Settings*"

Se abrirá la aplicación que hemos creado con la información sobre ella, si seleccionamos la opción *"Contact Methods"* (Figura 26).

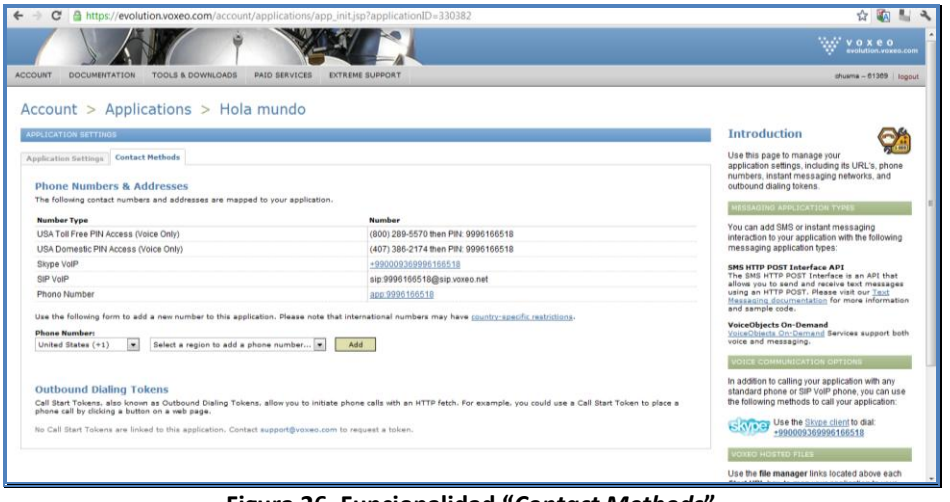

Figura 26. Funcionalidad "*Contact Methods*"

Con ello, podremos visualizar los números de teléfono asociados a esta aplicación, si pinchamos sobre Skype VoiP y se dispone del programa "*skype*" encontrándose abierto, automáticamente realizará una llamada a nuestra aplicación y podremos interactuar con ella.

Después de este apartado ya conocemos cómo asignar números de teléfonos a nuestra aplicación, existiendo un conjunto adicional de funcionalidades en las que no hemos entrado en profundidad.

2.4 APIs de Google

Google es una compañía principalmente conocida por su buscador y por los diferentes servicios que provee (Google Maps, correo, Google Reader, etc.). Para nuestra aplicación nos hemos centrado en dos APIs para desarrolladores que Google nos ofrece: *Google Directions* y *Google places*.

2.4.1 Google Directions API

Como referencia para este apartado hemos tomado la página oficial de Google Directions API [DIRECTIONSAPI].

2.4.1.1 Introducción

Google Directions API es un servicio que utiliza una solicitud HTTP para calcular rutas y llegar de una ubicación a otra. Las peticiones deben especificar orígenes, destinos e hitos (lugares por los que se quiere que pase la ruta) como cadenas de texto (por ejemplo, "Chicago, IL" o "Darwin, NT, Australia") o como coordenadas de latitud/longitud (40.4697200,-3.6867200). *Google Directions* puede devolver rutas segmentadas especificando los hitos por los que se quiere pasar. En nuestra aplicación no hemos usado hitos debido a que, al ser una aplicación oral, puede resultar tedioso y lento el manejo de muchas opciones, aunque nada impide un desarrollo posterior para incorporar esta funcionalidad.

Límites de uso

El uso de *Google Directions* API está sujeto a un límite de consultas de 2.500 solicitudes al día. Estas solicitudes enviadas pueden incluir 8 hitos intermedios como máximo. Los clientes de *Google Maps Premier* pueden consultar hasta 100.000 solicitudes de rutas al día y cada una de ellas puede incluir hasta 23 hitos.

Asimismo, hay que tener en cuenta que las URL de *Google Directions* API tiene una restricción de 2.048 caracteres. Dado que algunas URL del servicio de indicaciones pueden incluir varias ubicaciones a lo largo de una ruta, se recomienda tener en cuenta este límite a la hora de crear las URL.

Estas limitaciones nos afectarán si queremos hacer público el proyecto y éste se hace "popular", ya que actualmente es muy complicado que accedamos más de 2.500 veces en un día. Por su parte, la restricción de caracteres no nos afecta, debido a que trabajamos con nombres de calles normales y no usamos hitos que pudieran llevarnos a rebasar ese límite.

2.4.1.2 Solicitudes de rutas

Una solicitud a Google Api Directions tiene la siguiente forma:

```
http://maps.googleapis.com/maps/api/directions/output?parameters
```

En esta solicitud, "*output*" puede ser uno de los valores que se indican a continuación:

- *Json*: el formato de salida en Notación de objetos JavaScript (*JavaScript Object Notation*, JSON).

- *Xml*: indica el formato de salida como un archivo XML.

Para acceder a través de HTTPS, utilizamos la siguiente dirección:

```
https://maps.googleapis.com/maps/api/directions/output?parameters
```

Se recomienda utilizar HTTPS para aplicaciones que incluyan datos de usuario sensibles, en nuestro caso al no manejar información que necesite ser encriptada usaremos el protocolo HTTP.

Parámetros de solicitud

Algunos parámetros son obligatorios, mientras que otros son opcionales. Como en las URL estándar, todos los parámetros se separan con el carácter *"&"*. A continuación, se muestra una lista de los parámetros y sus valores posibles.

- ***Origin*** (obligatorio): define la dirección o el valor de latitud/longitud de la ubicación desde la que se deseé calcular la ruta.
- ***Destination*** (obligatorio): define la dirección o el valor de latitud/longitud de la ubicación hacia la que desee calcular la ruta.
- ***Sensor*** (*obligatorio*): indica si la solicitud desde la que se realiza la ruta dispone de un sensor de ubicación. Este valor debe ser *"true"* o *"false",* en nuestro caso será siempre "false", ya que realmente realizaremos la petición desde un servidor (proporcionado por VOXEO), no desde el dispositivo que realicemos la llamada.
- ***Mode*** (*opcional*, el valor predeterminado es *"driving"*): especifica el tipo de transporte que se utilizará para calcular las rutas, hay cuatro opciones "*driving*", "*walking*", "*bicycling*" y

"*transit*", como veremos más adelante en nuestra aplicación usaremos siempre "*walking*".

- ***Waypoints*** (*opcional*): especifica un conjunto de hitos. Para modificar una ruta, los hitos establecen las ubicaciones específicas por las que debe pasar. Es posible especificar un hito mediante coordinadas de latitud/longitud o como una dirección que se codificará de forma geográfica.
- ***Alternatives*** *(opcional*): si se establece en "*true*", indica que el servicio de rutas puede devolver más de una ruta alternativa. Ha de tenerse en cuenta que la obtención de rutas alternativas puede incrementar el tiempo de respuesta del servidor.
- ***Avoid*** (*opcional*): indica que la ruta o las rutas calculadas deben evitar determinados elementos. Actualmente, este parámetro admite los siguientes argumentos (hay que especificar alguno):
    - ***Tolls*:** indica que la ruta calculada debe evitar los peajes de carretera y de puentes.
    - ***Highways*:** indica que la ruta calculada debe evitar las autopistas y las autovías.
- ***Units*** *(opcional):* especifica el sistema de unidades que se utilizará para mostrar los resultados, hay dos posibles opciones: "*metric*" si se quiere usar el sistema métrico o "*imperial*" si se quiere usar el sistema inglés, por defecto el sistema usará la unidad del país sobre el que se está realizando la consulta, en nuestro caso se usará el sistema métrico.
- ***Region*** (*opcional*): es el código de región que se especifica como un valor de dos caracteres ccTLD ("*country code top-level domain*"), por ejemplo, en el caso de reino unido el código es "*uk*", especificar la región es importante ya que si se realizará una petición especificando "Toledo", no se sabría si se refiere a la ciudad de España o a la que se encuentra en el estado de Ohio, por lo que no devolvería ningún resultado. En nuestro caso no será necesario, ya que especificaremos lugares dentro de Madrid.

- ***Language*** (*opcional*): es el idioma en el que se proporcionan los resultados, en nuestro caso será siempre "*es*" ya que los proporcionaremos en español.

Modos de viaje

Al calcular indicaciones, se puede especificar el modo de transporte que quiere utilizarse con *"mode"*. De forma predeterminada, las rutas se calculan como indicaciones para llegar en coche *"driving"*. Actualmente, se admiten los siguientes modos de viaje:

- ***Driving*** (predeterminado) proporciona indicaciones para llegar en coche a través del sistema de carreteras.
- ***Walking*** solicita rutas para llegar a pie a través de rutas peatonales y aceras (en los lugares en los que estén disponibles).
- ***Bicycling*** solicita rutas para llegar en bicicleta a través de carriles bici y vías preferenciales para bicicletas (actualmente, sólo disponible en EE.UU).
- ***Transit*** solicita rutas por vías públicas de tránsito (en los lugares en los que estén disponibles).

Para nuestra aplicación sólo hemos utilizado la opción *"walking"* por distintas razones:

- No es recomendable realizar largas distancias (debido al coste de la llamada telefónica).
- Nuestra aplicación, en cierto modo, es incompatible para usarse cuando el usuario está montando en bicicleta, y aunque podría usarse mientras el usuario conduce no lo recomendamos.

En definitiva, consideramos que nuestra aplicación se usará principalmente para distancias cortas y en momentos puntuales, por lo

que solo hemos definido la opción para ir andando, en el futuro se podrán realizar mejoras en este aspecto si procede.

2.4.1.3 Respuestas de indicaciones

Las respuestas de las indicaciones se proporcionan en el formato que indica la marca de *"output"* en la petición URL. Como hemos comentado antes tiene dos opciones "*json*" o "*xml*", vamos a proceder a explicar cada una de ellas.

Salida JSON

Aunque parezca la opción más sencilla (debido a que con VoiceXML se puede manejar *ECMAScript*) no se consiguió que el diseño funcionase como se deseaba, por lo que procedimos a descartarla desde las primeras fases del proyecto. No se va a entrar en detalle en lo que respecta a esta capacidad debido a que hemos optado por obtener la información mediante XML.

Salida XML

Ha sido la opción elegida para la elaboración del proyecto. La respuesta XML está formada por una única etiqueta <*DirectionsResponse*> que engloba toda la respuesta y por dos etiquetas de nivel superior:

- <*status*> contiene los metadatos de la solicitud. Más adelante entraremos en detalle sobre los códigos de estado.

- Cero o más elementos <*route*> que contengan un conjunto único de información de las rutas entre el origen y el destino cada uno.

Debemos prestar atención al procesar árboles XML para asegurarnos de hacer referencia a las etiquetas y a los nodos

adecuados, cuando no se obtenga ningún resultado XML lo devolverá ninguna etiqueta <*route*>.

Códigos de estado

La etiqueta <*status*> de la respuesta a la petición contiene el estado de la solicitud y puede incluir información para ayudarnos a descubrir el motivo por el que no se ha obtenido resultado. El campo "*status*" puede contener los siguientes valores:

- *OK*: indica que la respuesta contiene un resultado válido.
- *NOT_FOUND*: indica que al menos una de las ubicaciones especificadas en los orígenes, el destino o los hitos de la solicitud no se pudo codificar de forma geográfica.
- *ZERO_RESULTS*: indica que no se pudo encontrar ninguna ruta entre el origen y el destino.
- *MAX_WAYPOINTS_EXCEEDED*: indica que se proporcionaron demasiados hitos en la solicitud. El número máximo de "*waypoints*" permitidos es ocho, además del origen y del destino.
- *INVALID_REQUEST*: indica que la solicitud enviada no es válida.
- *OVER_QUERY_LIMIT*: indica que el servicio ha excedido el límite de peticiones.
- *REQUEST_DENIED*: indica que el servicio ha denegado el uso del servicio de rutas a tu aplicación.
- *UNKNOWN_ERROR*: indica que no se ha podido procesar una petición debido a un error del servidor. Es posible que la petición se ejecute con éxito si se vuelve a intentar.

Rutas (*route*)

Cuando Google Directions A*PI* proporciona resultados, las respuestas XML están formadas por cero o más etiquetas <*route*>),

puede ser que esta respuesta contenga varias etiquetas <*route*>, indicando cada una de las rutas.

Cada elemento del conjunto de *"route"* contiene un resultado único del origen y del destino que hemos especificado. Esta ruta puede constar de uno o varios *"legs"*, en función de los hitos que se hayan especificado. Además, la ruta también contiene información sobre derechos de autor y sobre advertencias que se deben mostrar al usuario junto con la información de rutas.

Cada ruta del campo *"route"* puede contener los siguientes campos:

- *Summary*: contiene una breve descripción sobre la ruta que permite identificarla y distinguirla de otras alternativas.
- *legs*[]: contendrá cada tramo definido por los hitos. A todos los hitos o los destinos especificados les corresponde un tramo distinto. (Una ruta sin hitos contendrá un tramo dentro del conjunto de "*legs*"). Cada tramo consta de una serie de pasos ("*steps*"). Entraremos en detalle más adelante
- *waypoint_order*: Indica el orden de los hitos de la ruta calculada. En nuestro caso vendrá vacía.
- *overview_polyline*: contiene un objeto que consta de un conjunto de puntos geográficos que representan la ruta aproximada (suavizada) de las indicaciones resultantes. Es útil cuando se quiere realizar una representación en un mapa de la ruta.
- *Bounds*: contiene los límites de ventana que se deberían de respetar para mostrar la ruta en un mapa (así aparecerá cuadrada dentro del mapa).
- *Copyrights*: contiene los derechos de autor que se mostrará con la ruta.

- *warnings*[]: contiene un conjunto de avisos que se visualizará cuando se muestren las rutas

Tramos (*leg*)

Cada elemento del conjunto de "*leg*" especifica un tramo único del trayecto desde el origen al destino de la ruta calculada. Las rutas que no contengan hitos constarán de un único tramo, mientras que las rutas que definan uno o varios hitos constarán de uno o varios tramos correspondientes a los tramos específicos del trayecto.

Cada tramo del campo "*leg*" puede contener las siguientes etiquetas:

- *steps*[]: contiene un conjunto de pasos en los que se divide el tramo, entraremos en detalle más adelante.
- *Distance*: es un campo que indica la distancia total que abarca el tramo y que consta de los siguientes elementos:
  - *Value*: indica la distancia en metros (con un valor).
  - *Text*: contiene la distancia en texto, expresada en las unidades utilizadas en el origen (o como aparecen anuladas en el parámetro "*units*" de la solicitud), en el idioma especificado en la solicitud. (Por ejemplo, se utilizarán millas o pies para cualquier origen dentro de los Estados Unidos). Téngase en cuenta que el campo "*distance*.*value*" siempre contendrá un valor expresado en metros. Si no se conoce la distancia, es posible que estos campos no aparezcan.
- *Duration*: es un campo que indica el tiempo total necesario para recorrer un tramo y que consta de los siguientes elementos:
  - *Value*: indica la duración en segundos (con un valor).
  - *Text*: contiene información de la duración expresado en texto. Si no se conoce la duración, es posible que estos campos no aparezcan.

- *start_location*: contiene las coordenadas de latitud/longitud del origen del tramo. Dado que Google Directions API utiliza la opción de transporte más cercana a los puntos de partida y de llegada (normalmente carreteras), para calcular las rutas entre dos ubicaciones es posible que el valor "*start_location*" no coincida con el origen del tramo si por ejemplo, no hay carreteras cercanas al mismo.
- *end_location*: contiene las coordenadas de latitud/longitud del destino dado del tramo. Al igual que "*start_location*", puede que no coincida con el final del tramo.
- *start_address*: contiene una dirección interpretable por humanos (normalmente una calle) que refleja el valor "*start_location*" de dicho tramo.
- *end_addresss*: contiene una dirección interpretable por humanos (normalmente una calle) que refleja el valor "*end_location*" de dicho tramo.

Pasos (*step*)

Cada etiqueta <*leg*> puede contener varios pasos <*step*>, cada una define un paso único de las rutas calculadas. Un paso es la unidad más atómica de una ruta, que consta de una instrucción específica y única del trayecto. Por ejemplo, "Gira a la izquierda en la calle W. 4th St.". Un paso no sólo describe una instrucción, sino que también contiene información sobre la distancia y sobre el tiempo con respecto al paso siguiente. Por ejemplo, es posible que el paso etiquetado como "Tome la interestatal 80 oeste" especifique una duración de "60 kilómetros" y de "40 minutos" para indicar que el siguiente paso se encuentra a 60 kilómetros/40 minutos.

Cada etiqueta "*step*" puede contener los siguientes campos:

- *html_instructions*: contiene instrucciones de este paso, presentadas en formato de cadena de texto HTML.
- *Distance*: contiene la distancia que hay que recorrer desde un paso hasta el siguiente. Si no se conoce la distancia, es posible que este campo no esté definido.
- *Duration*: contiene el tiempo normal necesario para realizar un paso antes de pasar al siguiente. Si no se conoce la duración, es posible que este campo no esté definido.
- *start_location*: es un conjunto único de campos de "lat" y de "lng" (latitud/longitud) que indica la ubicación del punto de partida de un paso determinado.
- *end_location*: idéntico formato que el campo anterior para indicar en este caso la ubicación del punto de llegada de un paso determinado.

2.4.2 Google Places API

Google Places proporciona información acerca de lugares (establecimientos, lugares geográficos o puntos importantes de interés) con peticiones HTTP. Estas peticiones se hacen usando coordenadas de latitud-longitud. Como referencia para completar este apartado hemos tomado la página oficial de Google Places API [PLACESAPI].

El API de Google Places proporciona cinco tipos distintos de peticiones posibles:

- ***Place Searches*:** Proporciona una lista de lugares cercanos basados en la localización del usuario. Es el tipo de petición que vamos a usar para nuestra aplicación, las demás las comentaremos a nivel informativo.
- ***Place Details*:** Proporciona información más detallada sobre un lugar específico, incluyendo comentarios de los usuarios.

- ***Place Actions***: Permite añadir información a la disponible en la base de datos de Google places, se pueden crear eventos, borrar lugares, etc.
- ***Place Photos***: Da acceso a las millones de fotos que guarda la base de datos de Google Places.

2.4.2.1 Autenticación

Para realizar peticiones *a Google Places*, Google nos exige una clave API, para poder identificar la aplicación que lo está llamando y controlar los distintos accesos. La clave API viene relacionada con una cuenta de Google, para obtener la clave hay que visitar la página de la consola API de Google [APICONSOLA].

Al igual que *Google Directions*, el API de *Google Places* tiene límite de consulta, los usuarios con una clave de API pueden realizar 1.000 solicitudes en un periodo de 24 horas.

2.4.2.2 Búsquedas Place

Tal y como hemos mencionado anteriormente, en nuestro proyecto sólo haremos búsquedas de lugares cercanos basándonos en la localización del usuario y de un radio de búsqueda, esta petición nos devolverá una lista de opciones.

Una solicitud de búsqueda del lugar es una dirección URL HTTP de la siguiente forma:

```
https://maps.googleapis.com/maps/api/lugar/search/salida?parámetros
```

El parámetro salida puede tomar los siguientes valores:

- JSON indica que la producción en *JavaScript Object Notation* (JSON)

- VXML indica la salida en formato XML, al igual que en el API de Directions esta es la opción que hemos usado, ya que es más fácil de manejar con VXML.

Algunos parámetros son necesarios para iniciar una petición de búsqueda del lugar. Como es habitual en las direcciones URL, todos los parámetros se separan mediante el carácter "*&*".

Parámetros obligatorios:

- *key*: La clave de API que se nos exige para realizar búsquedas.
- *Location*: La latitud/longitud alrededor de la cual recuperar la información de lugar.
- *Radius*: Define el radio (en metros) en el que para obtener resultados *Place*. El radio máximo permitido es de 50.000 metros, no tendremos problema con esta restricción ya que el radio máximo que hemos definido para nuestra aplicación es de 999 metros.
- *Sensor*: Indica si la solicitud lugar se ha realizado desde un dispositivo con un sensor de localización (por ejemplo, un GPS) para determinar la ubicación envió esta solicitud. Este valor debe ser "*true*" o "*false*", para nuestras peticiones serán siempre "*false*".

También hay una serie de parámetros opcionales:

- *Keyword*: Una palabra clave que se compara con todo el contenido que Google tiene indexado a este lugar, incluyendo nombre, tipo, dirección, así como comentarios de los clientes y otros contenidos de terceros. Este parámetro no lo hemos usado porqué complicaría mucho la aplicación (definir gramática para todas las posibles palabras).

- *Language*: El código del idioma. En nuestro caso el lenguaje será siempre "*es*", ya que hemos utilizado el español.
- *Name*: Se busca este nombre entre los nombres de los lugares, y solo nos devolverá los que coincidan, al igual que la *keyword* no lo usamos para no complicar mucho con gramáticas la aplicación.
- *Rankby: Especifica el orden en que los resultados están en la lista. Los valores posibles son:*
    - *Prominence* (por defecto): Ordena los resultados según su importancia, esta clasificación favorecerá lugares destacados en la zona especificada. Este parámetro puede verse afectado por la clasificación de un lugar en el índice de Google, el número de *check-in* desde la aplicación, la popularidad mundial y otros factores. Nos hemos quedado por defecto con esta opción, ya que con la opción *Distance* no se usa radio y tendríamos que requerir al usuario muchas más opciones.
    - *Distance*: Esta opción clasifica los resultados en orden ascendente según la distancia al lugar especificado. Especifica un radio de 50 km y va proporcionando en paquetes de 20 todos los resultados.
- *Types*: definimos tipos de los que queremos que nos devuelva la petición, los resultados cumplirán al menos uno de ellos. Los tipos pueden ser separados por el símbolo "/" (tipo1 | tipo2 | etc.). En nuestra aplicación sólo haremos peticiones con los tipos 'subway_station' (parada de metro) o 'restaurant' (restaurante).
- *Pagetoken*: Proporciona los siguientes 20 resultados de una búsqueda realizada anteriormente, este parámetro no lo utilizaremos, ya que dar más de 20 opciones por teléfono sería muy tedioso, por lo que nos ceñiremos a los 20 primeros resultados.

Un ejemplo de petición que realizaría nuestra aplicación sería:

```
https://maps.googleapis.com/maps/api/place/search/xml?loca
tion=40.4667000,-
3.6886900&radius=200&types=food&sensor=false&key=AIzaSyDZ_
qFhjceA9xWgeBUBRImcQJvOd0k_jaI
```

Los atributos que se han utilizado en el ejemplo son:

- *Location = 40.4667000,-3.6886900*: Corresponde a la latitud y longitud de la calle de Agustín de Foxá, 4, Madrid.
- *Radius= 200:* El radio que queremos de búsqueda.
- *Types=food*: El tipo de lugar que queremos buscar.
- *Sensor= false*: Ya que no se tiene sensor GPS en el dispositivo.
- *Key=AIzaSyDZ_qFhjceA9xWgeBUBRImcQJvOd0k_jaI:* Clave que hemos obtenido de Google.

Como vemos, indicamos "*xml*" para que la respuesta esté en este lenguaje.

2.4.2.3 Respuestas a petición

Como hemos indicado antes, las respuestas del API de Google Places pueden ser en XML o en JSON, pero nosotros nos centraremos en explicar cómo se realizan las respuestas XML, que son las que usa nuestra aplicación.

Una respuesta XML consta de una única etiqueta <*PlaceSearchResponse*> que contiene hasta cuatro etiquetas de nivel superior:

- <*status*> contiene metadatos sobre la solicitud. Más adelante se explicaran sus posibles valores.
- <*result*> puede aparecer una o varias veces, cada una de las apariciones contiene información acerca de un único establecimiento. Es el elemento más importante, donde vendrá

la información, más adelante explicaremos con detalle cómo se dividen.

- *next_page_token* contiene un *token* para poder obtener 20 resultados siguientes, no lo usaremos en nuestra aplicación.
- *html_attributions* contiene un conjunto de atributos sobre esta respuesta que deben ser mostrados al usuario, en nuestro caso no se lo mostraremos al usuario, ya que no lo tendremos en cuenta.

El código de <*status*>, al igual que en *API Google Directions,* obtiene la información de cómo ha terminado la búsqueda, sus posibles valores pueden ser:

- *OK*: indica que no se han producido errores, se ha obtenido al menos un resultado.
- *ZERO_RESULTS*: indica que la búsqueda fue exitosa, pero no obtuvo ningún resultado
- *OVER_QUERY_LIMIT*: indica que está por encima del límite (se han realizado demasiadas consultas).
- *REQUEST_DENIED*: indica que su solicitud fue denegada.
- *INVALID_REQUEST*: generalmente indica que un parámetro de consulta necesario (*location* o *radius*) no se encuentra.

En la etiqueta <*results*> obtendremos los datos de cada uno de los lugares obtenidos (uno por etiqueta <*result*>), esta etiqueta pude tener:

- <*icon*>: contiene la dirección URL de un icono del lugar, se puede mostrar al usuario la hora de indicar este resultado.
- <*id*>: contiene un identificador único para este lugar.
- <*geometry*> contiene información de geometría del resultado, como la ubicación (*geocode*) del lugar (latitud y longitud).
- <*name*>: contiene el identificador del lugar. Para establecimientos se trata por lo general del nombre de la empresa.

- <*opening_hours*>: puede contener la siguiente información:
  - <open_now>: valor booleano que indica si el lugar está abierto actualmente.
- <*rating*>: contiene la calificación de Google del lugar, de 0 a 5, en base en comentarios de los usuarios.
- <*reference*>: contiene un único *token* que se puede utilizar para obtener información adicional acerca de este lugar en una solicitud de detalle del lugar . Se puede almacenar y utilizar este token en cualquier momento del futuro para actualizar los datos almacenados en caché sobre este lugar.
- <*type*>: los tipos asociados al lugar dado. Se puede repetir varias veces, ya que por ejemplo, un sitio puede estar indicado como un sitio de "*food*" y "*restaurant*".
- <*vicinity*>: contiene un nombre característico de las inmediaciones. A menudo se refiere a una calle o barrio dentro cerca del lugar.

Por ejemplo, la respuesta a la petición descrita anteriormente se muestra en la Figura 27.

```
<PlaceSearchResponse>
       <status>OK</status>
       <result>
              <name>Asador De Aranda</name>
              <vicinity>Plaza de Castilla, 3, Madrid
             </vicinity>
              <type>restaurant</type>
              <type>food</type>
              <type>establishment</type>
              <geometry>
                     <location>
                            <lat>40.4660620</lat>
                            <lng>-3.6902160</lng>
                     </location>
              </geometry>
              <rating>4.3</rating>
              <icon>
http://maps.gstatic.com/mapfiles/place_api/icons/restaurant-
71.png
              </icon>
              <reference>
```

```
CnRtAAAAVHIwdDbGmLp8etZFAMpqyGm8gJBIzgSImLV4DdvsCdReLMmwi6RGdQ2A
FTqw-2Cu3OTzWXwcTVtMZHESjkfY1EBXHlHXo4-2O-
WJjm8uXXHvLMmJq6XdWptDn6nDPPXXHLIQ9-N-0pRNj21MDn1xKBIQ30tgbsfGt-
xcMgs-BT1MWRoUreb-aQmuugLqzgybZJvrK_sEMqU
            </reference>
            <id>480fc0d78d6e88b907bb228f1136dae45ee60a2f</id>
      </result>
      <result>
            <name>Burger King</name>
            <vicinity>Calle de Bravo Murillo, 377, Madrid
          </vicinity>
            <type>restaurant</type>
            <type>food</type>
            <type>establishment</type>
            <geometry>
                  <location>
                        <lat>40.4656970</lat>
                        <lng>-3.6909480</lng>
                  </location>
            </geometry>
            <icon>
http://maps.gstatic.com/mapfiles/place_api/icons/restaurant-
71.png
            </icon>
            <reference>
CnRpAAAAF5OOQGIdj1FfSihVDSq2qfZsxAqnuqF8c8TwzzwgoxF2Y7dCmbfmr5cW
bIlODQ83hiduNuj5w_CWX40bNrG6hUn2VtbKv7z4IPXpuqvU4Mo1SNI8Wj8ukrdt
AIB8vbavARIcWQIXflEHhHbGjCqoYhIQbt2Wh5FRqg_XM4ZCaCzmOxoUI6sZtXAX
_TlXkoG9EZsA1tIYLCQ
            </reference>
            <id>054b05786d86b2ca47b0f6153cee45c993f22072</id>
      </result>
</PlaceSearchResponse>
```

Figura 27. Ejemplo respuesta Google Places

Se ha acortado la respuesta dejando sólo dos etiquetas <*result*>. Como puede observarse, la información que viene en el <*status*> está definida con valor OK, por lo que ha sido correcta, y cada etiqueta <*result*> nos transmite la información de cada lugar encontrado.

Capítulo 3: Descripción general del sistema desarrollado

Una vez analizado en profundidad el estado del arte sobre el que se asienta el desarrollo de nuestra aplicación, se describen en este capítulo sus características esenciales, su arquitectura, conjunto de funcionalidades, tecnologías implicadas y la interacción entre los diferentes módulos que la conforman.

Para ello, se empezará explicando las características generales de la aplicación, que servirán como fundamento para profundizar, posteriormente en una descripción más detallada de cada uno de los módulos que la integran.

3.1 Presentación del sistema

La aplicación que hemos desarrollado en el presente proyecto fin de carrera ha de ser entendida como un callejero por voz que nos va a ofrecer los siguientes servicios:

- Conocer cómo ir de una calle a otra.
- Localizar lugares de interés cercanos a una dirección.

En el desarrollo de la aplicación se ha utilizado una arquitectura cliente-servidor, tal y como se esquematiza en la Figura 28.

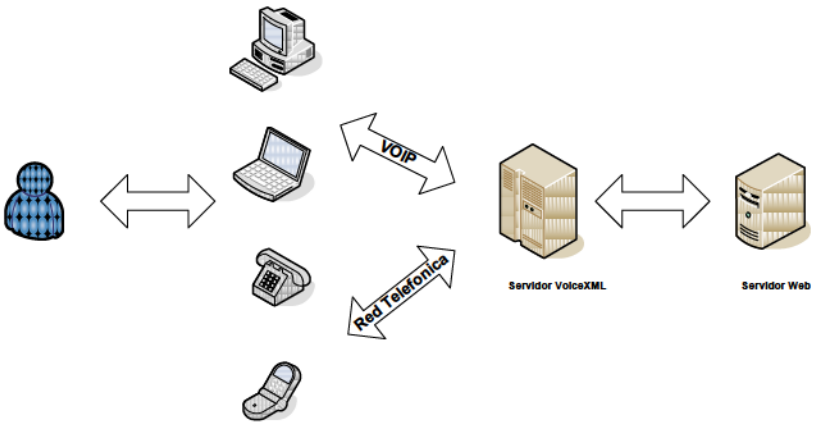

Figura 28. Arquitectura cliente-servidor utilizada para el desarrollo de la aplicación

Como puede observarse en la figura anterior, el usuario se puede comunicar con la aplicación mediante un ordenador de sobremesa, un portátil, tablets PC, teléfono fijo o móvil. Estos dispositivos se comunican, a su vez, con el servidor VoiceXML, que interactuará con un servidor web para generar, de forma dinámica, las respuestas requeridas por el usuario.

Presentada la definición básica del diseño, se hace necesario realizar una primera aproximación para proceder a la presentación de los módulos principales de la aplicación desarrollada. Para ello, no sólo se analizarán sus características individualizadas, sino que también se estudiará la interacción existente entre los módulos.

En la Figura 29 se presenta un esquema que permite obtener una visión inicial sobre las funciones de cada uno de los módulos y sobre la interacción entre los mismos para que la aplicación ofrezca el servicio deseado.

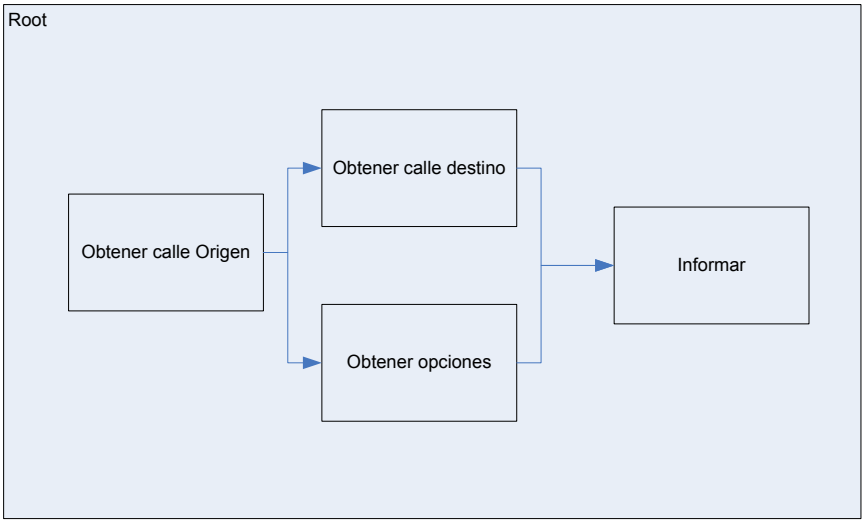

Figura 29. Módulos principales de la aplicación

Tal y como se puede intuir a partir del esquema de la Figura 29, cada módulo en el que se ha dividido la aplicación está conformado por una o más páginas VXML. De la figura se deriva que los principales módulos de la aplicación son:

- Módulo *Obtener calle origen*: Este módulo es el encargado de obtener toda la información referente a la primera calle, que se define como la vía en la que el usuario se encuentra en el momento de iniciar la aplicación. Mediante la interacción con el usuario, la aplicación obtendrá su posición actualizada y se realiza una interacción del módulo con el servidor de Google para obtener las coordenadas. Acto seguido, la aplicación preguntará al usuario qué desea hacer, existiendo aquí dos posibilidades de respuesta: el usuario podrá elegir entre seleccionar una calle de destino (entonces la aplicación le redirigirá al módulo *Obtener calle destino*) o bien podrá interesarse por la localización de un punto o lugar de interés concreto (en este caso, la aplicación le informará sobre las distintas opciones disponibles – metro o restaurante – y el

usuario, una vez escogida su opción, será redirigido al módulo *Obtener opciones lugares de interés*).

- Módulo *Obtener calle destino*: Cuando el usuario escoge desplazarse de la calle origen a una calle destino, será este módulo de la aplicación el que entre en funcionamiento. Es el encargado de obtener toda la información referente a la calle destino e interactúa con el servidor de Google para obtener las coordenadas. Terminada su tarea, la aplicación invocará al módulo *Informar*.
- Módulo *Obtener opciones lugares de interés*: Este módulo entra en juego cuando el usuario escoge localizar un lugar de interés. El módulo obtendrá del usuario el radio de búsqueda y se realiza una petición a Google para obtener los lugares que cumplen esas características, este le propone al usuario los distintos lugares y el usuario elige uno de ellos. Posteriormente, el módulo vuelve a realizar otra petición a Google para obtener las coordenadas del lugar seleccionado. Cuando el usuario escoja una opción la aplicación llamará al módulo *Informar*.
- Módulo *Informar*: Este módulo será el encargado de indicarle al usuario paso a paso cómo llegar desde la calle origen hasta la calle destino o hasta la opción de punto de interés que haya elegido. Para ello realiza una petición a Google para obtener mediante coordenadas de origen y destino cómo llegar de un punto a otro.
- Módulo "*Root*" o *Raíz*: Este módulo contiene a todos los demás y siempre estará presente, ya que guarda la información que se transmite de módulo a módulo (recordemos que cada uno de éstos está en una página distinta y, por tanto, no se puede enviar directamente). Además, contiene los scripts que se usaran en las páginas y se encarga, asimismo, de la gestión de eventos. No interaccionará con el usuario ni con el servidor de Google en ningún momento.

Hasta el momento, nos hemos limitado a explicar cómo es la interacción interna entre los módulos de la aplicación. La Figura 30 ilustra las relaciones que existen entre los distintos módulos, sus accesos al servidor web de Google y su ubicación en la aplicación.

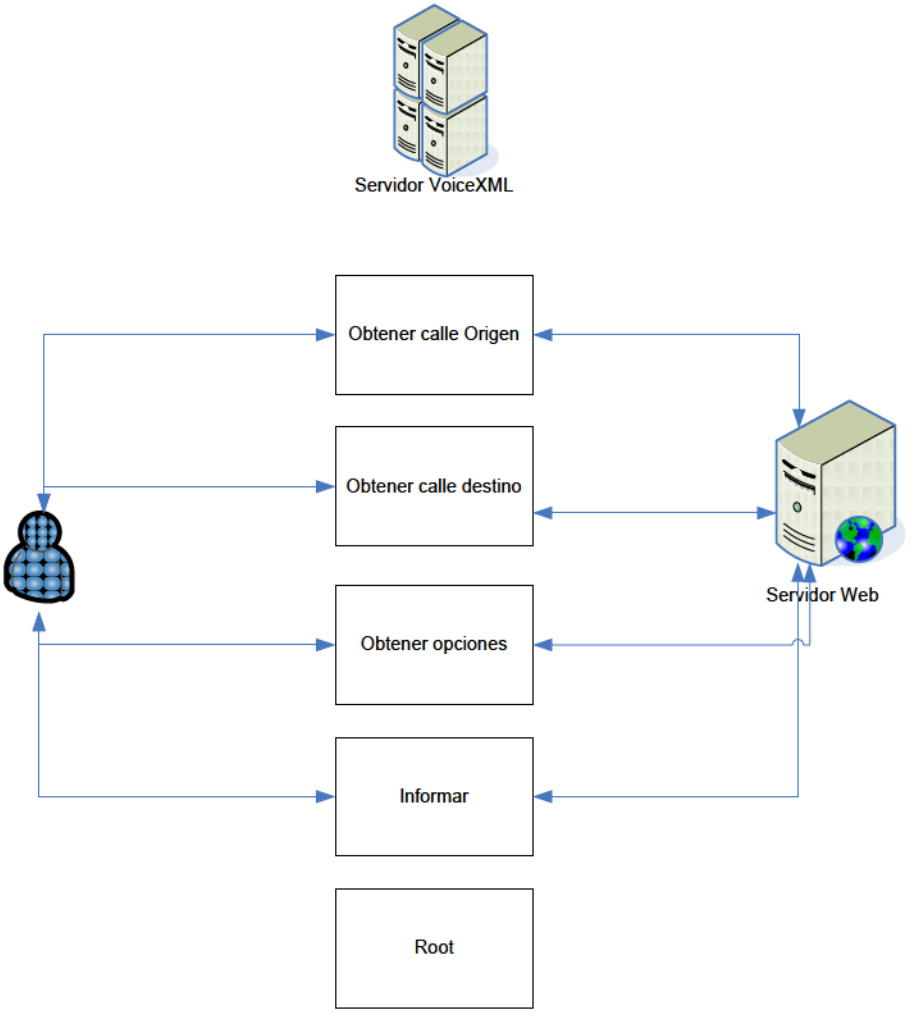

Figura 30. Interactuación entre los módulos de la aplicación con el servidor de Google y el usuario

A continuación se describen con mayor detalle los módulos de la aplicación.

3.2 Módulo *Obtener calle origen*

Como se ha descrito anteriormente, este módulo es el encargado, en primer lugar, de obtener del usuario la calle origen y el número desde donde quiere iniciar el desplazamiento, que por regla general coincidirá con el lugar en el que se encuentra en ese momento (aunque también existe la posibilidad de que el usuario quiera anotar los pasos para llegar de un lugar a otro, sin estar en el punto de partida que nos señala). En segundo lugar, el módulo se encargará también de preguntar al usuario qué opción desea, si dirigirse a otra calle o dirigirse a un lugar de interés.

En la Figura 31 apreciamos los distintos componentes en los que se divide este módulo.

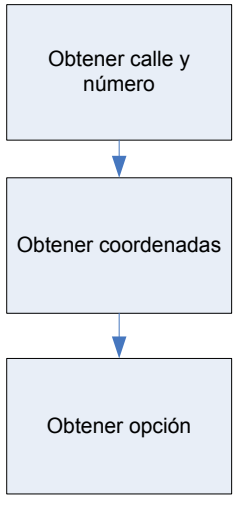

Figura 31. Componentes del módulo *Obtener calle origen*

La Figura 31 nos muestra una división del módulo en tres componentes, que se procede a explicar separadamente a continuación.

3.2.1 Obtener calle y número.

Como su propio nombre indica, este componente será el encargado de obtener la calle y el número a partir del cual habrá de calcularse el desplazamiento. La Figura 32 nos muestra cómo se realizará la interacción del usuario con el componente para obtener la calle y número de origen. Asimismo, se detallan los pasos seguidos para obtener la información deseada.

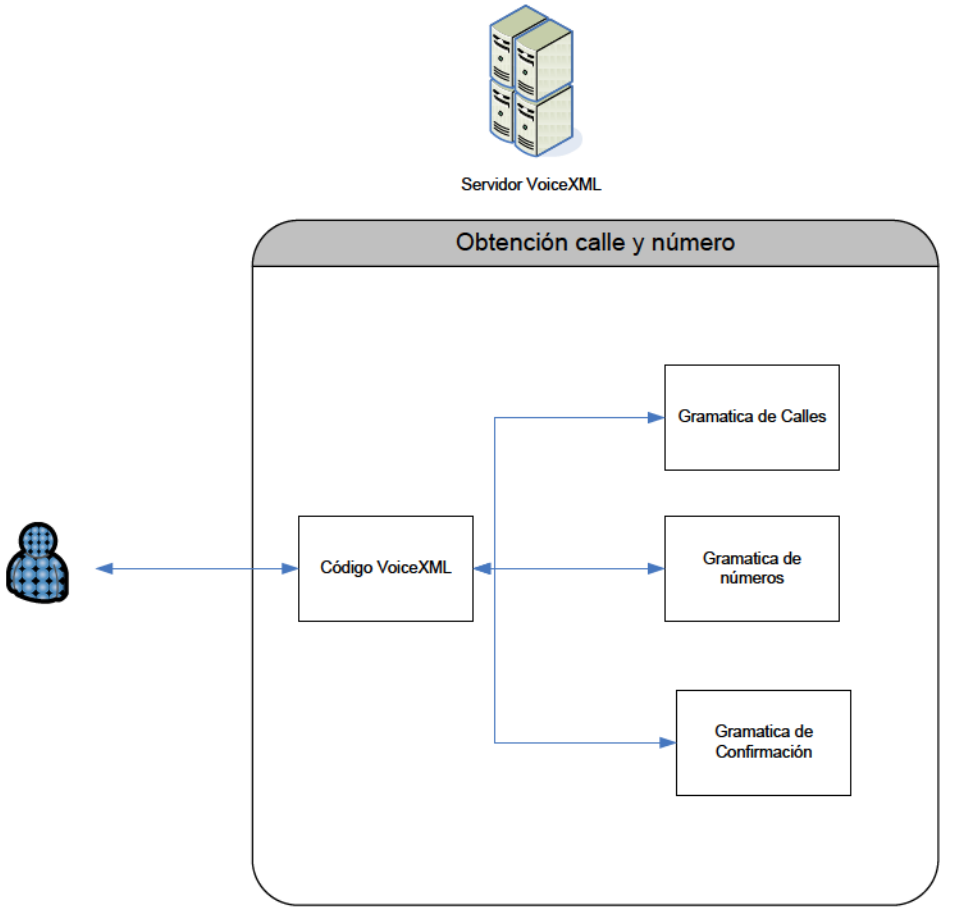

Figura 32. Detalle de ficheros principales del módulo *Obtener calle origen*

De lo expuesto en la Figura 32 se deduce que el módulo *Obtener calle origen* se divide en diversos componentes, todos ellos hospedados en el servidor VoiceXML.

3.2.1.1 Petición del nombre de la calle.

Este módulo requiere inicialmente la calle de entrada. Para ello se utiliza una variable "origen" donde guardaremos el nombre de la calle. Primero, mediante la etiqueta "<*prompt*>", se pedirá el nombre de la calle. Después, mediante la gramática de calles, se procederá a reconocerla. La Figura 33 muestra el código VoiceXML utilizado para requerir el nombre de la calle.

```
<!--le pedimos la calle de Entrada-->
<field name="origen">
    <nomatch>
            La calle no ha producido ningún resultado,
            por favor, repítala, o diga el nombre de otra
             calle.
    </nomatch>
    <help> Por favor, diga la calle origen: </help>
      <noinput> Por favor, diga la calle origen:
</noinput>
    <prompt> Por favor, diga la calle origen: </prompt>
      <grammar src=
"http://webhosting.voxeo.net/61369/www/grammar/calles.grxm
l"/>
    <filled>
      <assign name="origenAux" expr="origen"/>
    </filled>
</field>
```

Figura 33. Código VoiceXML para obtener el nombre de la calle origen

Como vemos en la Figura 33, dentro del campo "<*field*>" se llama a la gramática de "calles.grxml", mediante la etiqueta "<*grammar*>". Pueden ocurrir varias situaciones:

- Que la calle que ha pronunciado el usuario no produzca ningún resultado en la gramática: esta situación se contempla con el evento "<*nomatch*>", que, como se puede observar, cuando se active comunicará un mensaje al usuario confirmándole que la calle no ha producido ningún resultado.

- Que el usuario no diga nada: al no obtener ningún resultado se contemplará con la etiqueta "<*noinput*>", evento que hará que el sistema repita el mensaje de petición de la calle origen.
- Que el usuario pida ayuda: En este caso se activa un evento de ayuda (lo veremos más adelante en el apartado *Gestión de eventos*), que se capturará con esta etiqueta "<*help*>" y detalla al usuario la petición de la calle origen.
- Que la calle pronunciada coincida con alguna de las descritas en la gramática, en ese caso se guardará en la variable "origen". Como se observa en la etiqueta "<*filled*>" esta variable se copiará a la variable "origenAux", ya que es una variable a nivel global (de página) y se usará más adelante.

La gramática de la aplicación comprende todas las calles de Madrid además de un conjunto de reglas para cubrir diferentes posibilidades. Para ello, se ha creado una gramática en formato GRXML, llamada "calles.grxml". Esta gramática está formada por tres tipos de reglas:

- Tipo: son el tipo de vía, (calle, plaza, vía, etc.). Puede ser repetido cero o una vez.
- Preposición: es la preposición (de, del). Puede ser repetida cero o una vez.
- Nombre: es el nombre de la calle (Castilla, Gran Vía, etc.). Es repetido una vez siempre.

Después de definir el conjunto de símbolos terminales para desarrollar la gramática se han definido las siguientes reglas de producción:

- Resultado = "Nombre"
- Resultado = "Tipo" +"Nombre"
- Resultado = "Tipo" +"Preposición"+"Nombre"

Los nombres de las calles se han obtenido de la página oficial de la comunidad de Madrid, donde se encuentran listados con las diferentes calles [MADRID].

La Figura 34 muestra un ejemplo concreto de uso de esta gramática.

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar root="inicio"
version="1.0" xml:lang="es-es" tag-format="semantics/1.0">
      <rule id="inicio" scope="public">
            <item repeat="0-1"><ruleref uri="#tipo"/>
            </item>
            <item repeat="0-1"><ruleref uri="#prep"/>
            </item>
            <item ><ruleref uri="#nombre"/></item>
      </rule>
      <rule id="tipo">
            <tag>$="";</tag>
                  <one-of xml:lang="es-es">
                        <item>Calle</item>
                        <item>Plaza</item>
                        <item>Travesía</item>

                  </one-of>
      </rule>

      <rule id="prep">
            <tag>$="";</tag>
                  <one-of xml:lang="es-es">
                        <item>de <tag>$="de ";</tag>
                        </item>
                        <item>del <tag>$="del ";</tag>
                        </item>
                  </one-of>
      </rule>

      <rule id="nombre">
            <one-of xml:lang="es-es">
                  <item>Castilla</item>
                  <item>Gran Via</item>
            </one-of>
      </rule>
</grammar>
```

Figura 34. Ejemplo gramática definición de calles

Como vemos en la figura anterior, el "*<grammar root>*", que es donde empieza la gramática, está enlazada a "inicio", que es donde se trataran los distintos tipos de reglas (tipo vía, preposición y nombre). También se procede a informar cuántas veces se repetirá cada regla. Seguidamente, en la gramática definiremos el contenido de cada de una de las reglas.

3.2.1.2 Petición del número de la calle

Una vez que se ha obtenido el nombre de la calle, se solicita al usuario el número de la calle. Para ello se ejecutará el código expuesto en la Figura 35.

```
<!--le pedimos el numero origen-->
<field name="numeroOrigen">
      <nomatch>
            No he entendido correctamente el número,
            por favor repítalo.
      </nomatch>
      <help>
            Por favor, diga el número de la calle origen:
      </help>
      <noinput>
            Por favor, diga el número de la calle origen:
      </noinput>
      <prompt>
            Por favor, diga el número de la calle origen:
      </prompt>

    <grammar src=
      "http://webhosting.voxeo.net/61369/www/grammar/numer
os.grxml"/>
    <filled>
        <assign name="origenAuxNum" expr="numeroOrigen"/>
    </filled>
</field>
```

Figura 35. Código VoiceXML para la petición del número de calle

Como se observa en la Figura 35, dentro del campo "*<field>*" se crea la variable "numeroOrigen", donde se guardará el número. Se pedirá mediante la etiqueta *"<prompt>*" y se reconocerá lo que diga el usuario realizando una llamada a la gramática de "numeros.grxml",

mediante la etiqueta "<*grammar*>". Al igual que cuando se requiere el nombre de la calle, cabe distinguir los siguientes casos:

- Que el número que el usuario ha pronunciado no produzca ningún resultado en la gramática: esta situación se contemplará con la etiqueta "<*nomatch*>" que, tal y como se puede apreciar, cuando se active lanzará un mensaje diciendo que no se ha entendido el número.
- Que el usuario no diga nada: al no obtener ningún resultado, se contemplará con la etiqueta "<*noinput*>", y el sistema volverá a pedir el número.
- Que el usuario pida ayuda: cuando el usuario pida ayuda se lanzará un evento de ayuda (lo veremos más adelante en el apartado *Gestión de eventos*), pero se recogerá con la etiqueta "<*help*>" que, tal y como vemos, pedirá que se diga el número de la calle origen.
- Que la calle pronunciada por el usuario coincida con alguna de las descritas en la gramática. En ese caso se guardará en la variable "numeroOrigen". Tal y como se muestra, en la etiqueta "<*filled*>" asignaremos esta variable a "origenAuxNum".

La realización de esta gramática ha sido costosa, ya que debe considerar una gran casuística y es necesario realizar muchas distinciones. Se han definido distintas reglas:

- "Un digito": está formado por los dígitos del 0 al 9.
- "Dos dígitos simple": está formado por los dígitos simples, del 11 al 19, y el 20, 30, 40, 50, 60, 70, 80, 90 y 100 (éste último lo incluimos en este grupo).
- "Dos dígitos compuestos": estos reconocerán el primer dígito cuando el número está formado por "veinti", "treinta y", "cuarenta y", "cincuenta y", "sesenta y", "setenta y", "ochenta y" y "noventa y".

- “Tres dígitos”: está formado por el primer dígito, -cuando son tres- , y podrá ser “doscientos”, “trescientos “, “cuatrocientos”, “quinientos”, “seiscientos”, “setecientos”, “ochocientos “, “novecientos”
- “Cero”: está formado por el cero.

Mediante estas definiciones se han creado las siguientes reglas de producción:

- Resultado = “Un digito”
- Resultado = “Dos dígitos simple”
- Resultado = “Dos dígitos compuestos” + “Un digito”
- Resultado = “Tres dígitos compuestos” + “Dos dígitos simple”
- Resultado = “Tres dígitos compuestos” + “Un digito” + “cero”
- Resultado = “Tres dígitos compuestos” + “Dos dígitos compuestos” + “Un digito”

Tal y como se ve en estas combinaciones, se puede obtener números del 0 al 999. En [CALLES] se puede consultar que la calle más grande de España sería la Gran Vía de les Corts Catalanes de Barcelona con 700 portales, y en segunda posición estaría la calle de Alcalá, ésta sí, en Madrid, que tiene 10,5 km y comprende 544 portales. Por lo tanto, reconocer números de tres dígitos es suficiente para nuestra gramática.

3.2.1.3 Confirmación de los datos obtenidos

Por último, se tendrá que confirmar con el usuario lo que el sistema ha comprendido. Para ello se utiliza el código indicado en la Figura 36.

```
<!--le decimos lo que se ha entendido-->
<block name="ConfirmacionEntradaOrigen">
    <prompt bargein="false">
```

```
      La calle de origen es:  <value expr="origenAux"/>,
      número:  <value expr="origenAuxNum"/>.
     </prompt>
</block>
```

Figura 36. Código VoiceXML para la comunicación del origen entendido por la aplicación

Como se observa en la Figura 36, se le está comunicando al usuario lo que se ha guardado en las variables "origenAux" y "origenAuxNum", que corresponden al nombre y número de la calle. A continuación, se tendrá que recoger del usuario si lo que se ha entendido el sistema es correcto o no. Con este objetivo, se le preguntará al usuario si la calle seleccionada es correcta, utilizando para ello el código indicado en la Figura 37.

```
<!--Confirmamos-->
<field name="ConfirmaOrigen">
       <nomatch>
              No le entiendo, diga Sí o No.
       </nomatch>
       <help>
              Por favor, diga Si o No.
       </help>
       <noinput>
              Por favor, diga Si o No.
       </noinput>
       <prompt bargein="false">
              ¿es correcto?
       </prompt>
       <grammar src=
"http://webhosting.voxeo.net/61369/www/grammar/SiNo.grxml"/>
       <filled>
              <if cond="ConfirmaOrigen=='No'">
                     <clear namelist="origen"/>
                     <clear namelist="origenAux"/>
                     <clear namelist="numeroOrigen"/>
                     <clear namelist="origenAuxNum"/>
                     <clear namelist=
                   "ConfirmacionEntradaOrigen"/>
                     <clear namelist="ConfirmaOrigen"/>
               <goto nextitem="origen"/>
              </if>
   </filled>
</field>
```

Figura 37. Código VoiceXML para la confirmación del origen entendido por la aplicación

Tal y como se aprecia en la Figura 37, para la confirmación llamaremos a la gramática "SiNo.grxml". Esta gramática reconocerá "Sí" o "No" por parte del usuario, que serán las dos opciones posibles para saber si se ha entendido correctamente el nombre y número de la calle o no. Como en las anteriores gramáticas, habrá distintas posibilidades que contemplaremos con distintas etiquetas:

- "<*nomatch*>": Lo que ha dicho el usuario no se reconoce como "Sí" o "No", en ese caso volveremos a pedirle la confirmación.
- "<*noinput*>": Que el usuario no diga nada, en este caso se volverá a pedir al usuario que repita la confirmación.
- "<*help*>": Que el usuario pida ayuda. Se lanzará el evento ayuda y se capturará con esta etiqueta, se le pedirá al usuario que repita la confirmación.
- En el caso de que el sistema entienda "Sí" o "No" se realizaran distintas acciones. En el caso de que el usuario confirme que lo que ha reconocido el sistema es correcto –Sí-, se seguirá con la ejecución normal. En caso que el usuario indique que el sistema no ha entendido correctamente –No-, se volverá a pedir al usuario la calle de origen. Para ello, en la etiqueta <*filled*> las variables: "origen", "origenAux", "numeroOrigen", "origenAuxNum", "ConfirmacionEntradaOrigen", "ConfirmaOrigen" se inicializaran y más tarde se llamará al ítem "origen", así se volverá a pedir toda la información relativa a la calle de origen de nuevo.

3.2.2 Obtener coordenadas

Una vez que se ha obtenido la calle y el número, se obtendrán las coordenadas. La razón principal de que se necesite obtener las coordenadas es para unificar el proceso de información: aunque para

realizar una petición a *Google Directions* se puede usar la calle y número, si accedemos a los lugares de interés, *Google Places* responderá con coordenadas (también informa la calle y número, pero esa información no es fiable, ya que a veces no está completa o está desajustada).

Para obtener las coordenadas se necesitará una dirección origen y una dirección destino. Esto es así es porque Google no posibilita obtener las coordenadas de una dirección directamente. Por esta razón, se tendrá que realizar una petición con *Google Directions* utilizando la dirección de la que se quiere obtener las coordenadas y una dirección destino establecida por defecto (coordenadas de la Puerta del Sol). En la Figura 38 se detalla el funcionamiento de esta funcionalidad.

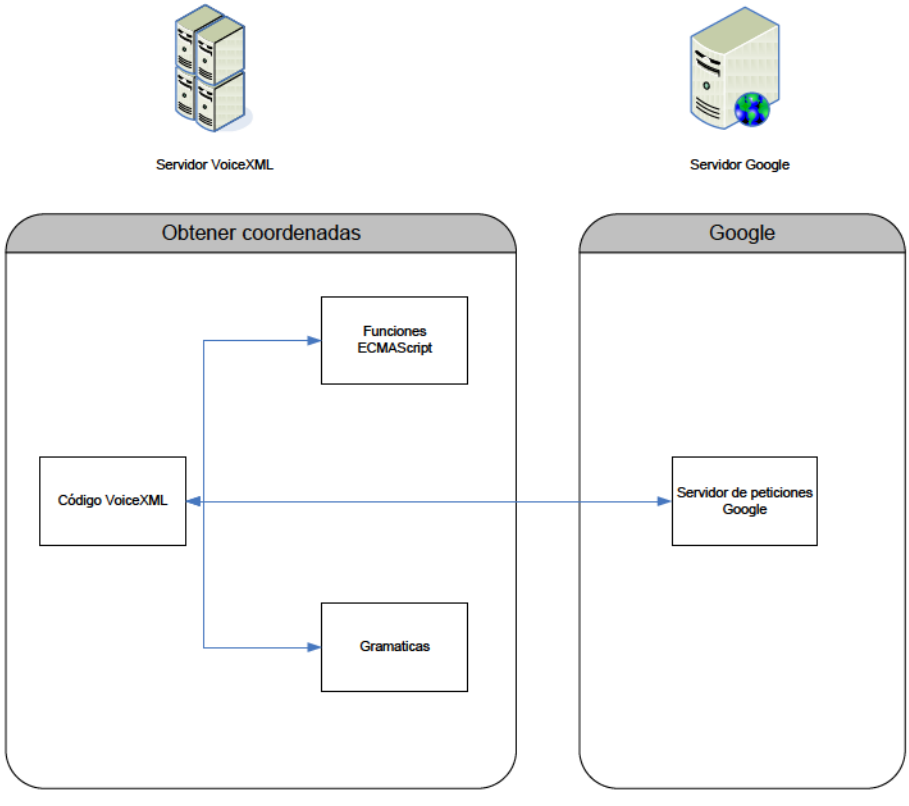

Figura 38. Procedimiento para la obtención de las coordenadas

Como vemos en la Figura 38, el usuario no interactuará con este componente, pero sí se realizará una petición a Google. Esta petición será a *Google Directions* para obtener las coordenadas. En la Figura 39 mostramos el código necesario para proceder en este sentido.

```
<block name="EntradaOrigen">
      <var name="origin"
      expr="origenAux+','+origenAuxNum+',Madrid'"/>

      <assign name="origin"
      expr="quitarEspacios(origin)"/>

      <var name="destination" expr="application.latSol
      +','+application.longSol"/>

      <!--Hacemos la petición a google-->
      <data name="xmlSubjects" srcexpr="MyXMLDest"
      namelist="origin destination language mode sensor"/>

      <var name="quote"
      expr="xmlSubjects.documentElement"/>

      <if cond="getStatus(quote)=='OK'">
            <assign name="application.longOrigen"
            expr="getLongitud(quote)"/>
            <assign name="application.latOrigen"
            expr="getLatitud(quote)"/>
      <else/>
            <goto next="#NoResultados"/>
      </if>
</block>
```

Figura 39. Código VoiceXML para la realización de la petición de coordenadas a Google Directions

Tal y como se puede apreciar en la Figura 39, primero se unen el nombre de la calle y el número obtenido en el módulo anterior, y se asigna a la variable "*origin*". Una vez guardada esta variable, se llamará a la función "quitarEspacios()" definida en la página de Scripts "funciones.js". Es necesario aclarar que se requiere quitar los espacios debido a que se va a realizar una petición http, donde se hace imprescindible la supresión de los mismos. En la Figura 40 mostramos la función definida en ECMAScript.

```
function quitarEspacios(cadena)
{
var c = cadena.indexOf(' ');
      while(c!=-1){
            cadena= cadena.replace(" ","+");
            c =cadena.indexOf(' ');
      }
      return cadena;
}
```

Figura 40. Función ECMAScript función para eliminar espacios

Como vemos en la Figura 40, la función "quitarEspacios()", tiene como funcionalidad localizar los espacios y sustituirlos por '+' hasta que no queden más espacios.

Una vez que se tiene la dirección de origen en formato válido para realizar una petición, se definirá una dirección destino. Para ello, en el módulo *Raíz*, se encuentran definidas las coordenadas (latitud y longitud) de la Puerta del Sol, ya que por defecto es la dirección que vamos a utilizar como destino (hay que tener en cuenta que Google no tiene un servicio para obtener las coordenadas de una dirección, por lo que usamos una dirección destino ficticia).

Por lo tanto, se tendrá en la variable "*origin*" la calle y número de la que se quiere obtener las coordenadas, mientras que en la variable "*destination*" se tendrán recogidas las coordenadas de la Puerta del Sol. Para realizar la petición mediante *Google Directions* se necesitan más variables, que definiremos en nuestro módulo *Raíz* tal y como mostramos en la Figura 41.

```
<var name="language" expr="'es'"/>
<var name="sensor" expr="'false'"/>
<var name="mode" expr="'walking'"/>
<var name="MyXMLDest"
expr="'http://maps.google.es/maps/api/directions/xml?'"/>
```

Figura 41. Código VoiceXML para variables utilizadas en las consultas a Google Directions

Tal y como se observa en la Figura 41 se definirán las variables necesarias para realizar la petición de la siguiente manera:

- "*language*": Inicializada a "es", ya que se quiere que el lenguaje sea el español.
- "*sensor*": Inicializada a "false", debido a que el usuario no realizará la petición desde un dispositivo con sensor de ubicación.
- "*mode*": Inicializada a "*walking*", ya que se quiere obtener las rutas caminando.

Por último, se inicializará otra variable más en la que se guardará la URL a la que hay que realizar la petición:

- "MyXMLDest": "http://maps.google.es/maps/api/directions/xml?" que es la dirección base de la petición HTTP de Google.

Una vez que se tienen todas las variables definidas, se realizará la petición a Google, mediante la etiqueta "<*data*>". Para ello, se establecerá el atributo "*name*" al valor "xmlSubjects". Esto se realiza de esta manera porque lo que recogeremos será un árbol XML. En el atributo "*srcexpr*" se indicará la dirección de las peticiones de Google y en "namelist" se indicaran todas las variables necesarias para realizar la petición a Google. La petición la realizaremos tal y como muestra la Figura 42.

```
<!--Hacemos la petición a google-->
<data name="xmlSubjects" srcexpr="MyXMLDest"
namelist="origin destination language mode sensor"/>
```

Figura 42. Código VoiceXML del detalle de la petición a Google Directions

En la variable "xmlSubjects" usada en el código de la Figura 42 se guardará la información devuelta por Google a nuestra petición. En este

apartado nos centraremos en cómo obtener las coordenadas, ya que más adelante se desglosará está información con más detalle.

En la Figura 43 se puede observar cómo es la primera parte de una respuesta típica de Google Directions.

```
<DirectionsResponse>
      <status>OK</status>
      <route>
      <summary>Calle de Mauricio Legendre</summary>
      <leg>
            <step>
                  <travel_mode>WALKING</travel_mode>
                  <start_location>
                        <lat>40.4697600</lat>
                        <lng>-3.6870300</lng>
                  </start_location>
```

Figura 43. Ejemplo de respuesta proporcionada por Google Directions

Tal y como muestra la figura anterior, lo primero que nos da la respuesta de *Google Directions* son las coordenadas dentro de las etiquetas "*<start_location>".*

Para obtener las coordenadas se usarán funciones ECMAScript, ayudándonos de la función que proporciona VoiceXML "*documentElement*", que referencia a la raíz del árbol del DOM (*Domument object Model*), que es la forma que se ha elegido para tratar la información.

Una vez que se tiene un puntero dirigido sobre la raíz, ya se podrán obtener la información importante. El primer dato que interesa saber es si la petición ha tenido éxito. Esa información se proporcionará en la etiqueta "<*status*>". Para obtener esta información, se usará la función "*getStatus*()" descrita en la Figura 44.

```
function getStatus(quote)
{
      var result =quote.getElementsByTagName("status");
      var status;
      status = result.item(0).firstChild.data;
      return status;
}
```

Figura 44. Función ECMAScript para obtener la etiqueta status

Tal y como se observa en la figura anterior, se obtendrá la información obtenida en la etiqueta "<*status*>" mediante la función "*getElementsByTagName*()", y seguidamente su valor con la función "*firstChild.data*".

Como ya comentamos en el estado del arte, la etiqueta "<*status*>" puede tener muchos estados de solicitud, pero en nuestra aplicación sólo se ha tenido en cuenta si ha sido correcta ("OK") o incorrecta (cualquier otro valor). Los errores más típicos son "NOT-FOUND" y "ZERO-RESULTS", que ocurrirán respectivamente cuando insertemos una dirección errónea o ésta no esté reconocida por Google. Los casos "MAX_WAYPOINTS_EXCEEDED" o "INVALID_REQUEST" no se pueden dar, el primero porque no usamos hitos y el segundo porque, al hacer las peticiones automáticas, no es posible que se escriba algún parámetro mal.

Otra forma de error es que el propio servidor de Google rechace la petición. Esto se puede dar con "*OVER_QUERY_LIMIT*", "*REQUEST_DENIED*" y "*UNKNOWN_ERROR*". El caso de "*OVER_QUERY_LIMIT*" ocurrirá cuando se pase de 2.500 peticiones, que únicamente ocurrirá a partir de 1.250 usos de nuestra aplicación (cada vez que la usamos hacemos dos peticiones). Aun así, no se tendrá en cuenta porque nos parecen peticiones más que suficientes para la presentación de este proyecto. En el caso de "*REQUEST_DENIED*" y "*UNKNOWN_ERROR*" no se conocerá el motivo del rechazo de nuestra petición, pero si se vuelve a realizar es posible que funcione

correctamente, por lo que hemos decidido ponerlas en el mismo bloque de error.

En el caso de que la petición haya sido correcta (“OK”) se procederá a obtener las coordenadas. Para ello, nos valdremos de las funciones “*getLongitud*()" y *getLatitud*()" mostradas en la Figura 45.

```
function getLatitud(quote)
{
      var result =
      quote.getElementsByTagName("start_location");

      var latitud;
      latitud =
      result.item(0).getElementsByTagName("lat").
      item(0).firstChild.data;

      return latitud;
}
function getLongitud(quote)
{
      var result =quote.
      getElementsByTagName("start_location");

      var longitud;
      longitud =
      result.item(0).getElementsByTagName("lng").
      item(0).firstChild.data;

      return longitud;
}
```

Figura 45. Función ECMAScript para obtener la latitud y la longitud

Tal y como se aprecia en la figura anterior, con la función “*getLatitud*()” obtendremos la información dada en la etiqueta “<*start_location*>”. Dentro de esta etiqueta se obtendrá la información de la etiqueta “<*lat*>”. Para obtener la longitud se realizarán los mismos pasos que en el caso anterior, sólo que obteniendo la información de la etiqueta “<*lng*>”.

Una vez que se han obtenido las coordenadas, se guardarán en el módulo *Raíz*, ya que se necesitará esta información en otros

módulos. Acto seguido, se pasará a obtener la opción, pero antes es necesario explicar qué pasará en el caso de que la petición que se ha realizado a Google sea errónea.

Cuando el código de la operación no sea correcto, se le comunicará al usuario que su consulta no ha obtenido ningún resultado y se le preguntará si desea realizar otra consulta o si desea salir de la aplicación. En la Figura 46 se observa cómo se lleva a cabo esta operación.

```
<form id="NoResultados" scope="document">
      <field name="otra">
            <nomatch>
                  No le he entiendo, por favor
                  diga otra o salir.
            </nomatch>
            <help>
            Por favor, diga otra o salir:
            </help>
            <noinput>
            Por favor, diga otra o salir:
            </noinput>
            <prompt bargein="false"> <break time="1.5s"/>
                  Su consulta no ha obtenido resultados.
                  Si quiere realizar otra consulta diga,
                  otra. Si quiere salir diga salir:
            </prompt>
            <grammar src=

      "http://webhosting.voxeo.net/61369/www/grammar/otra.
grxml"/>

            <filled>
                  <if cond="otra=='Otra'">
                        <clear namelist="longOrigen"/>
                        <clear namelist="latOrigen"/>
                        <clear namelist="longDestino"/>
                        <clear namelist="latDestino"/>
                        <clear namelist="radio"/>
                        <clear namelist="tipo"/>
                        <goto next="ObtenerOrigen.vxml"/>
                  <else/>
                        <prompt bargein="false">
                              Gracias por usar el callejero
                              de la ciudad de Madrid,
                              un saludo.
                              </prompt>
```

```
                    <exit/>
                </if>
               </filled>
        </field>
</form>
```

Figura 46. Código VoiceXML para la interacción de una petición sin resultados

Como se observa en la Figura 46, para preguntar si el usuario quiere salir o repetir consulta, llamaremos a la gramática "otra.grxml". Esta gramática reconocerá los valores "otra" o "salir" por parte del usuario, que serán las dos opciones posibles para saber qué desea hacer éste. Como en las anteriores gramáticas habrá distintas posibilidades que se contemplan con distintas etiquetas:

- <*nomatch*>: Lo que ha dicho el usuario no se reconoce como "otra" o "salir", en ese caso volveremos a pedirle que diga "otra" o "salir".
- <*noinput*>: Que el usuario no diga nada, en este caso se volverá a pedir al usuario que diga "otra" o "salir".
- <*help*>: Que el usuario pida ayuda. Se lanzará el evento ayuda, y al capturarse con esta etiqueta se le pedirá al usuario que diga "otra" o "salir".

En el caso de que el sistema entienda correctamente "otra", se realizarán las siguientes opciones:

- Se limpiará de información todas las variables importantes con "<*clear namelist=…*>", ya que si no se liberan de valor estas variables, no se podrá volver a pedir información.
- Se volverá a llamar a "ObtenerOrigen", por lo que el usuario pasará de nuevo por todas las etapas desde volver a obtener calle y número de origen.

3.2.3 Obtener opciones

Una vez que se han obtenido las coordenadas de la dirección origen, ya se tiene lo necesario para saber qué quiere realizar el usuario con esa información. La Figura 47 describe las diferentes opciones.

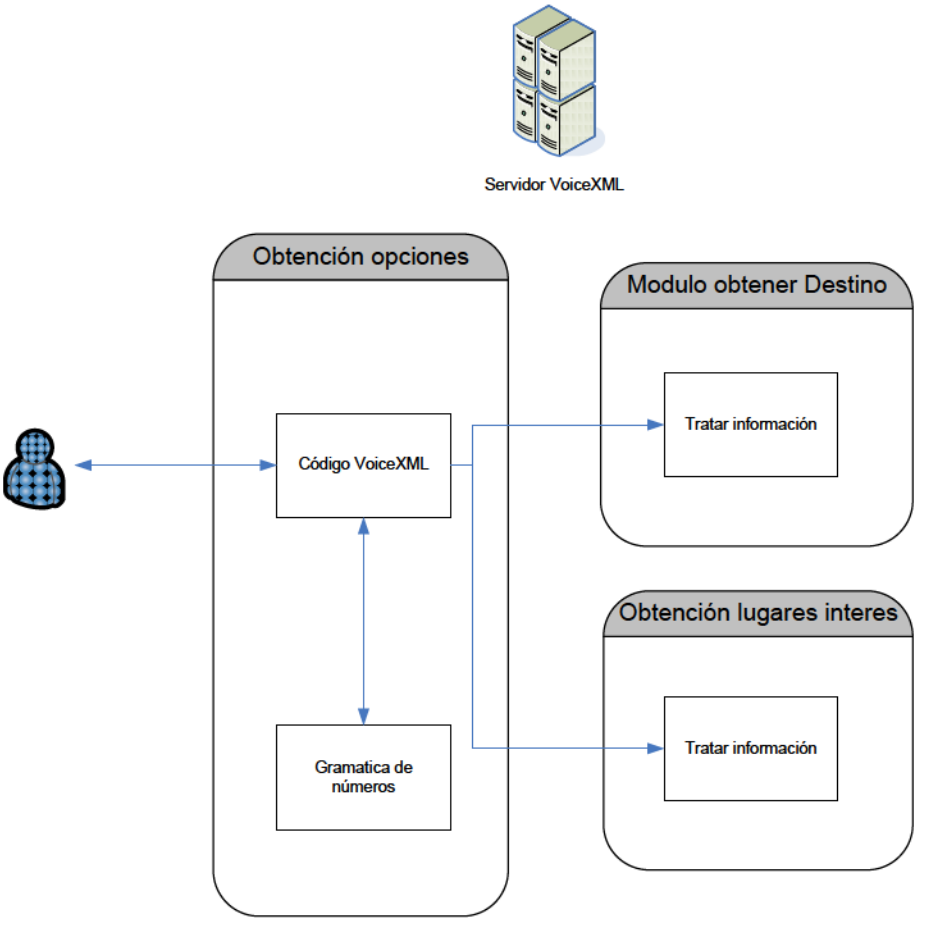

Figura 47. Esquema de la funcionalidad Obtener opciones

Como se observa en el esquema de la Figura 47, según la información obtenida por el usuario, podremos proceder al módulo *Obtener calle destino* o al módulo *Obtener opciones lugares de interés*. Para ello, primero se procederá a pedirle al usuario las distintas opciones y, según la respuesta obtenida, se tratará la información de una forma u otra. Las opciones que se le ofrecen al usuario son:

- Ir a una calle destino: El usuario más adelante tendrá que informar a la aplicación de otra calle y número, pero esta vez serán del destino.
- Buscar un restaurante: Se llamará al módulo *Obtener opciones lugares de interés*, donde se le informará de las distintas opciones de restaurantes que tiene en un radio determinado desde la dirección origen.
- Buscar una parada de metro: Se llamará al módulo de *Obtener opciones lugares de interés*, donde se le informará de las distintas opciones de paradas de metro que tiene en un radio determinado desde la dirección origen.

En la Figura 48 podemos ver el código mediante el cual se realiza el tratamiento de las diferentes opciones.

```
<form id="ObtenerOpcion">

      <!--seleccionamos una opción entre metro,
      restaurante, callejero-->

      <field name="opcion">
            <nomatch>
                  No le entiendo, por favor diga una
                  opción.
            </nomatch>
            <help>
                  Por favor, seleccione una opción
            </help>
            <noinput>
                  Seleccione una opción
            </noinput>
            <prompt bargein="false">
                  Si desea ir a otra calle diga uno,
                  si desea ir a una parada de metro
                  cercana diga dos, si desea ir a un
                  restaurante cercano diga tres
            </prompt>
            <grammar src=
      "http://webhosting.voxeo.net/61369/www/grammar/numer
os.grxml"/>

            <filled>
                  <if cond="opcion=='1'">
                        <goto next="ObtenerDestino.vxml"/>
```

```
                  <elseif cond="opcion=='2'"/>
                        <assign name="application.opcion"
                        expr="'una parada de metro'"/>

                        <assign name="application.tipo"
                        expr="'subway_station'"/>

                        <goto next="ObtenerOpciones.vxml"/>

                  <elseif cond="opcion=='3'"/>
                        <assign name="application.opcion"
                        expr="'un restaurante'"/>

                        <assign name="application.tipo"
                        expr="'restaurant'"/>

                        <goto next="ObtenerOpciones.vxml"/>

                  <else/>
                        <prompt bargein="false">
                              Diga una de las opciones
                              indicadas
                        </prompt>
                        <clear namelist="opcion"/>
                        <goto next="#ObtenerOpcion"/>
                  </if>
            </filled>
      </field>
</form>
```

Figura 48 Código VoiceXML para obtener opción

Tal y como queda reflejado en la Figura 48, lo primero que se hará es comunicarle al usuario las distintas opciones. Se ha reaprovechado la gramática de números y no se ha creado una nueva para las opciones, para que resulte lo más fácil posible agregar nuevos lugares de interés. De esta forma, si se quiere, por ejemplo, añadir "cajeros" no se tendrá que modificar ninguna gramática, sino sólo este componente en concreto.

Al igual que en las anteriores ocasiones en las que se han usado gramáticas, se controlarán los distintos eventos que se pueden producir.

En el caso que haya seleccionado la opción de ir a una calle destino, la ejecución de la aplicación seguirá en el módulo *Obtener calle destino*. En el caso de que haya seleccionado buscar una parada de metro o un restaurante, guardaremos las variables correspondientes en el módulo *Raíz*:

- Variable tipo: es el que guarda la información sobre el tipo de lugar de interés que se ha seleccionado, se guardará en inglés debido a que se usará para hacer la petición a *Google Places*, las opciones posibles son "*subway_station*" y "*restaurant*".
- Variable opción: se guardará el tipo de lugar de interés que se ha seleccionado, pero esta vez en español, ya que la usaremos para informar al usuario más adelante, sus posibles valores son "una parada de metro" y "un restaurante".

Una vez guardadas estas variables se procederá a llamar al módulo *Obtener opciones lugares de interés*.

3.3 Módulo Obtener calle destino

En el caso de que el usuario quiera ir a una calle destino se tendrá que obtener de él a que calle destino quiere dirigirse. Al igual que cuando se obtuvo la calle de origen se realizarán dos pasos:

- Obtener calle y número del usuario.
- Obtener coordenadas de la calle y número.

Como la operativa es exactamente igual que en el módulo *Obtener calle origen*, se ha optado por no volver a repetir toda la información.

3.4 Módulo Obtener opciones lugares de interés

Una vez que se ha obtenido la calle de origen y el usuario ha seleccionado la opción de lugares de interés se procederá a obtener del usuario cuál es el radio de búsqueda en la que buscar estos lugares, realizar la petición correspondiente a *Google Places* e informar al usuario los resultados obtenidos. Seguidamente, el usuario tendrá que elegir a qué lugar de los ofrecidos está interesado en obtener las indicaciones.

La Figura 49 nos muestra una división del módulo en tres componentes, que se proceden a explicar a continuación.

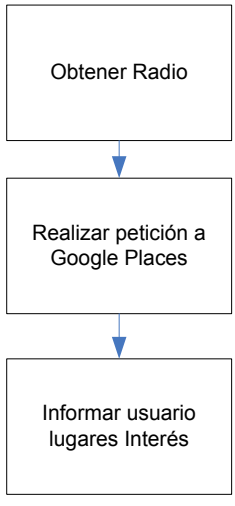

Figura 49. Componentes del módulo *Obtener opciones lugares de interés*

3.4.1 Obtener radio

Como su propio nombre indica, este componente será el encargado de obtener el radio a partir del cual se realizará la búsqueda de lugares de interés. La Figura 32 nos muestra cómo se realizará la interacción del usuario con el componente para obtener el radio.

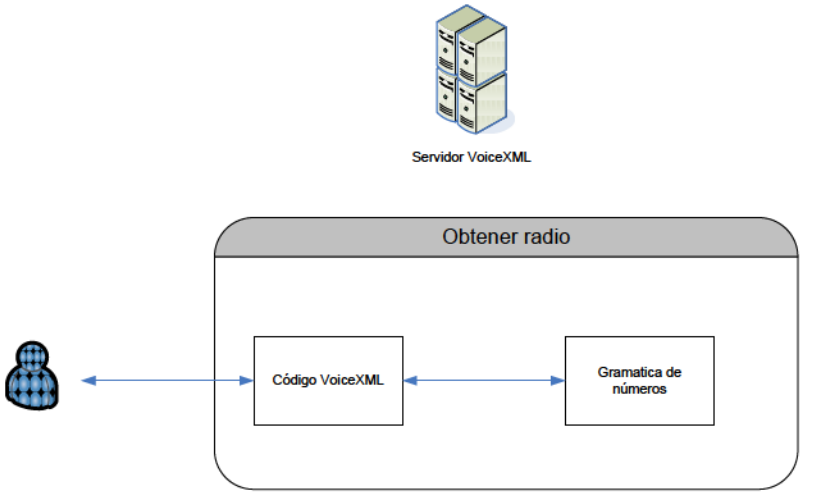

Figura 50. Componentes de *Obtener radio*

Como se observa en la Figura 50, para obtener el radio se realizará la petición de este al usuario y mediante la gramática de obtención de números se identificará su valor.

Para la obtención del radio se ejecutará el código indicado en la Figura 51.

```
<field name="distancia">
       <nomatch>
             Por favor diga la distancia en metros
       </nomatch>
       <help>
             Por favor, diga la distancia en metros
       </help>
       <noinput>
             Por favor, diga la distancia en metros
       </noinput>
       <prompt>
             Por favor, diga la distancia
             en metros en la que encontrar
             <value expr="application.opcion"/>
       </prompt>

       <grammar src=
       "http://webhosting.voxeo.net/61369/www/grammar/numeros.grx
ml"/>

       <filled>
        <assign name="application.radio" expr="distancia"/>
       </filled>
</field>
```

Figura 51. Código VoiceXML para obtener el radio

Como se observa en la Figura 51, lo primero que realizamos es mediante la etiqueta "*<promt>*" pedir al usuario que diga un radio. Para ello, usamos la variable *"application.opcion"*, ya que se le quiere indicar al usuario para qué lugar de interés está buscando el radio, como se indicó anteriormente los dos posibles valores de esta variable pueden ser 'una parada de metro' o 'un restaurante', por lo que las dos posibles opciones que se podrían dar al usuario son:

- Por favor, diga la distancia en metros en la que encontrar una parada de metro.
- Por favor, diga la distancia en metros en la que encontrar un restaurante.

Para reconocer los posibles valores que pueda decir el usuario, se usará la gramática "números.grxml", que se ha detallado previamente.

Se puede observar en la Figura 51 que la acción a realizar cuando se ha reconocido un número correctamente es guardar la variable "distancia" en la variable "application.radio".

Una vez que se ha obtenido correctamente el radio, ya se tendrá toda la información necesaria para realizar una llamada al API de Google Places.

3.4.2 Realizar petición a Google Places

Una vez que se ha obtenido la calle y el número desde donde se quiere partir para la búsqueda de los lugares de interés y el radio, se realizará la petición a *Google Places* para obtener esta información.

En la Figura 52 se muestra cómo se ha estructurado este componente.

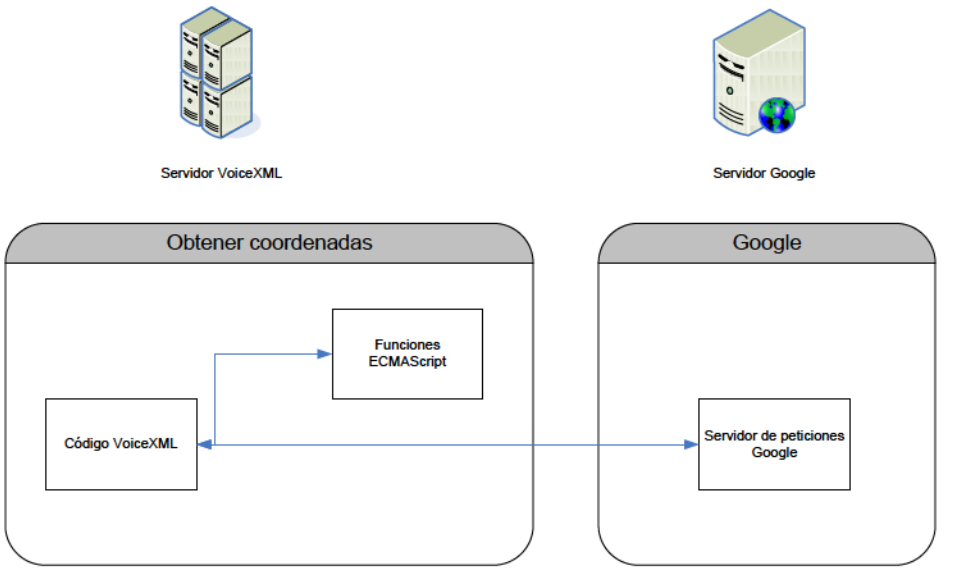

Figura 52. Relación componente petición Google Places

Como vemos en la figura anterior, el usuario no interactuará con este componente, pero sí se realizará una petición a Google. Esta petición se realizará a *Google Places* para obtener los lugares de interés. En la Figura 53 mostramos el código necesario para proceder en este sentido.

```
<block name="places">
      <var name="location" expr="application.latOrigen +',
      '+application.longOrigen"/>

      <var name="radius" expr="application.radio"/>
      <var name="types" expr="application.tipo"/>
      <!--Realizamos la peticion a google-->
      <data name="xmlSubjects"
      srcexpr="application.MyXMLPlaces"
      namelist="location radius types
      language sensor key"/>

      <assign name="quote"
      expr="xmlSubjects.documentElement"/>
```

Figura 53. Código VoiceXML para la realización de una petición a Google Places

Tal y como se puede apreciar en la Figura 53, primero se unen el nombre de la calle y el número obtenido en el módulo *Obtener calle origen*, y se asigna a la variable "*location*", también se grabará el radio obtenido anteriormente a la variable "radius", y el tipo de lugar de interés ("*subway_station*","*restaurant*") obtenido en el módulo anterior. El resto de las variables están definidas en el "*Root*" tal y como se muestra en la Figura 54.

```
<var name="language" expr="'es'"/>
<var name="sensor" expr="'false'"/>
<var name="MyXMLPlaces"
expr="'https://maps.googleapis.com/maps/api/place/search/x
ml?'"/>
<var name="key"
expr="'AIzaSyDZ_qFhjceA9xWgeBUBRIedfeded0k_jaI'"/>
```

Figura 54. Código VoiceXML para variables utilizadas para la interacción con Google Directions

Tal y como se observa en la Figura 54 se definirán las variables necesarias para realizar la petición de la siguiente manera:

- "*language*": Inicializada a "es", ya que se quiere que el lenguaje sea el español.
- "*sensor*": Inicializada a "false", debido a que el usuario no realizará la petición desde un dispositivo con sensor de ubicación.
- "*key*": es la clave que identifica quién está realizando la petición. En el caso de *Google Places,* tal y como se comentó en el Capítulo 2, es obligatorio.

Por último, se inicializará otra variable más en la que se guardará la URL a la que hay que realizar la petición:

- " *MyXMLPlaces* ": "https://maps.googleapis.com/maps/api/place/search/xml"

Una vez que se tienen todas las variables definidas se realizará la petición a *Google Places*. Mediante la etiqueta "<*data*>", se establecerá el atributo "*name*" al valor "*xmlSubjects*". La petición la realizaremos tal y como muestra la Figura 55.

```
<!--Hacemos la peticion a google Places-->
<data name="xmlSubjects" srcexpr="application.MyXMLPlaces"
namelist="location radius types language sensor key"/>
```

Figura 55. Código VoiceXML del detalle petición de información a Google Places

En la variable "xmlSubjects" usada en el código de la Figura 55 se guardará la información devuelta por Google a nuestra petición.

En la Figura 56 se puede observar cómo sería un resultado de una respuesta típica de Google Places.

```
<PlaceSearchResponse>
      <status>OK</status>
      <result>
            <name>El Asador de Aranda</name>
            <vicinity>
                  Plaza de Castilla, 3, Madrid
            </vicinity>
            <type>restaurant</type>
            <type>food</type>
            <type>establishment</type>
            <geometry>
            <location>
            <lat>40.4663310</lat>
            <lng>-3.6905850</lng>
            </location>
            </geometry>
            <rating>3.8</rating>
            <icon>

      http://maps.gstatic.com/mapfiles/place_api/icons/res
taurant-71.png
            </icon>
            <reference>
            CoQBcQAAAPgh92EmpoLTwufwO9RvNm4wy9h8vyY0QSnOG
            EIl0jC5_qm66HCFocSM-HaUQzokOJ3WHRGIUlAw01qEyfD
            biQG-LQKo2thIa_pTreLRgTsBV-wzTDQ2Bj0wrTZKA88re
            wEGK9z2T-pUMujrgbdkNMxnHCCCHeP0u5Nna9WbMz42EhC
            SeGRA6b_FGN7q5Kv-BGopGhShPPAAgkEdNOT_9qy-dBlI1
            NrHtQ
```

```
            </reference>

      <id>480fc0d78d6e88b907bb228f1136dae45ee60a2f</id>
            <opening_hours>
            <open_now>false</open_now>
            </opening_hours>
            <photo>
                  <photo_reference>
            CnRoAAAALcNLHbf4JSx8ArQLdMVqHfHErCcExdwK8oW_Y
            btv9IUtwD80xVzlmTqI2GE1x0RjaD7nrrjH3E3OrUZV3O
            JzhagiQdBlYKFmP99EMl9kTeo4r3RVvOIk94wfA38XaF7
            13-SqKSbLIpzs6_Olr0z6fhIQVA9pVZeAajxob9MbmqUo
            ixoU2jeTAaYpwH6a-DLBSSnEDzfeZ2c
                  </photo_reference>
                  <width>1224</width>
                  <height>1632</height>
                  <html_attribution>
                        Usuario de Google
                  </html_attribution>
            </photo>
      </result>
```

Figura 56. Ejemplo respuesta proporcionada por Google Places

Tal y como se observa en la Figura 56, lo primero que nos indica *Google Places* es el "<*status*>" "OK", mostrando que la consulta ha sido realizada correctamente.

En segundo lugar vendrá la etiqueta "<*result*>", que se repetirá tantas veces como resultados se hayan obtenidos. Dentro de esta etiqueta cabe destacar los siguientes elementos:

- <*name*>: Se trata del nombre, esta etiqueta es importante para nuestra aplicación ya que al usuario se le informará de los nombres de los resultados obtenidos para que elija una. Tal y como mostramos en la Figura 56, el nombre del resultado expuesto es "El Asador de Aranda".
- <*type*>: Se trata de los tipos, esta etiqueta se puede repetir varias veces ya que un resultado puede corresponder a varios resultados. En el ejemplo mostrado en la Figura 56 se puede observar que es de los tipos "*food*", "*restaurant*" y "*establishment*". Es importante esta etiqueta ya que al realizar la petición e indicarle el tipo, sólo obtendremos los resultados

que lo cumplan (en el ejemplo mostrado se pidieron "restaurant").

- *<location>*: Las coordenadas para nuestra aplicación son importantes, ya que se guardará esta información para indicarle al usuario cómo llegar. Se dividen en:
  - *<lat>*: Latitud.
  - *<lng>*: Longitud.
- *<rating>*: La nota que tiene para Google este lugar de interés. Aunque no se vaya a utilizar, se menciona debido a que se podría pedir que los resultados fueran devueltos en función de esta característica del lugar, y en desarrollos futuros podría ser interesante.

Para obtener el árbol de resultados se usarán funciones ECMAScript, ayudándonos de la función que proporciona VoiceXML "*documentElement*", que referencia a la raíz del árbol del DOM (*Domument object Model*) y es la forma que se ha elegido para tratar la información. La Figura 57 muestra el código que realiza esta acción.

```
<assign name="quote" expr="xmlSubjects.documentElement"/>

<if cond="getStatus(quote)=='OK'">
      <assign name="nombres" expr="getNombres(quote)"/>
<else/>
      <goto next="ObtenerOrigen.vxml#NoResultados"/>
</if>
```

Figura 57. Código VoiceXML para la obtención árbol resultados mediante Google Places

Una vez que se tiene un puntero dirigido sobre la raíz, ya se podrán obtener la información que es importante para nosotros. El primer dato que interesa saber es si la petición ha tenido éxito. Esta información se proporcionará en la etiqueta "*<status>*". Para obtener esta información, se usará la función "*getStatus*()", ya explicada anteriormente. En el caso de que la petición haya sido correcta, se guardarán en la variable "nombres" un vector con los nombres de todos

los resultados obtenidos, esta acción se realizará con la función "*getNombres*()". Se puede observar el código de esta función en la Figura 58.

```
function getNombres(quote)
{
      var result =quote.getElementsByTagName("name");
      var nombres = new Array;
            for(i=0;i<result.length;i++)
                  nombres[i]= utf8_decode(result.item(i).
                  firstChild.data);

      return nombres;
}
```

Figura 58. Función ECMAScript desarrollada para la obtención de nombres

Tal y como se observa en la figura anterior lo primero que se hace es obtener en la variable "*result*" todos los elementos con la etiqueta "<*name*>" que nos haya devuelto la petición. Para ello, nos ayudamos de la función "*getElementsByTagName*()". Una vez realizado esto, se recorrerán todos los elementos "<*name*>" obtenidos y mediante la función "*fistChild.data*" iremos recogiendo las variables en el vector "nombres". Como vemos es necesario decodificarlos a UTF-8, ya que si no es posible que nuestro sistema no pueda pronunciarlos.

En el caso de que en la etiqueta "<status>" no venga informado "OK", se mandará la ejecución a "NoResultados", donde se le informará al usuario de que su petición no ha obtenido ningún resultado.

3.4.3 Informar sobre lugares de interés

Una vez que se han obtenido los nombres de todos los lugares de interés en el radio de búsqueda que el usuario ha establecido, se procederá a informarlos al usuario para que elija uno. En la Figura 59 se puede ver como interactúa este componente con el resto de la aplicación.

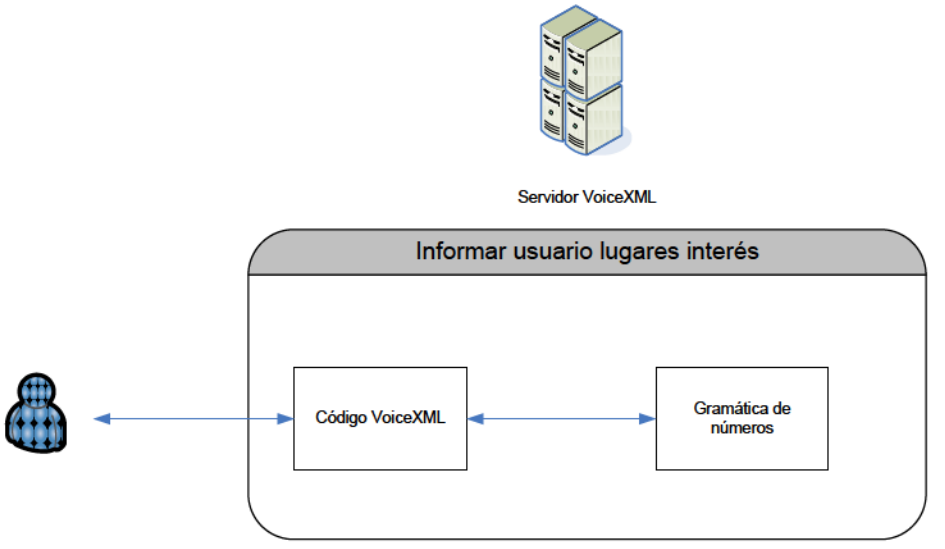

Figura 59. Informar usuario lugares de interés

Para informarle al usuario de los distintos valores se procederá a leer mediante un bucle los nombres obtenidos. En la Figura 60 podemos ver el código mediante el cual se realiza este tratamiento.

```
<field name="opcion">
      <grammar src=
"http://webhosting.voxeo.net/61369/www/grammar/numeros.grx
ml"/>

      <prompt>
            Por favor, diga el número de la opción
            que quiere
            <break time="40ms"/>
      </prompt>

      <prompt>
            <foreach item="nombre" array="nombres">
                  <assign name="contador"
                  expr="contador + 1"/>

                  <if cond="contador &gt; nombres.length">
                        <assign name="contador" expr="1"/>
                  </if>
                  <value expr="contador"/>
                  <break time="30ms"/>
                  <value expr="nombre"/>
                  <break time="40ms"/>
            </foreach>
      </prompt>
```

```
      <noinput>
            Lo siento, no le he oido.
      <reprompt/>
      </noinput>

      <nomatch>
            Por favor seleccione una opción de las dadas.
            <reprompt/>
      </nomatch>

</field>
```

Figura 60. Código VoiceXML para informar sobre lugares de interés al usuario

Como observamos en la Figura 60, mediante la gramática "números.grxml", se le pedirá al usuario una opción que se guardará en la variable "opción", no vamos a entrar en detalle de esta gramática ya que se ha explicado anteriormente. Usando la variable "contador" definida al principio del módulo como "0" se irá recorriendo el vector de "nombres", en el caso de que el contador sea mayor que el número total de elementos se asignará a "1", para informar al usuario se leerá primero el contador y después el nombre correspondiente a esa posición en el vector "nombres".

En el momento en que el usuario detecte un número dicho por el usuario se comprobará si es un valor correcto. En la Figura 61 se puede ver como se realiza esta comprobación.

```
<block>
      <if cond="opcion &gt; nombres.length">
            <prompt>
                  Esa opción no existe, por favor
                  diga una que exista.
                 <break time="30ms"/>
            </prompt>
            <clear namelist="opcion"/>
            <goto nextitem="opcion"/>
      </if>
      <assign name="longDestino"
      expr="getLongitudNum(quote,opcion)"/>
      <assign name="latDestino"
      expr="getLatitudNum(quote,opcion)"/>
      <goto next="Informar.vxml"/>
</block>
```

Figura 61. Código VoiceXML para cuando el sistema detecta una opción no valida

Como vemos en la Figura 61, lo primero que se realiza es comprobar si la opción dicha por el usuario es mayor que el número de resultados que tenemos. En caso de ser así, se le informará que la opción no es válida, y se enviará la ejecución al bucle donde se le informa de los posibles valores, limpiando las variables ya informadas.

En el caso de ser un valor correcto, quiere decir que la variable "opción" es la posición que ocupa el lugar de interés al que se quiere desplazar el usuario, por lo tanto se recuperarán las coordenadas de ese lugar de interés, para ello nos ayudaremos de las funciones creadas para este propósito "*getLongitudNum*()" y "*getLatitudNum*()", mostradas en la Figura 62.

```
function getLatitudNum(quote,pos)
{
      var result =quote.getElementsByTagName("lat");
      var latitud;
      latitud = result.item(pos-1).firstChild.data;

      return latitud;
}

function getLongitudNum(quote,pos)
{
      var result =quote.getElementsByTagName("lng");
      var longitud;
      longitud = result.item(pos-1).firstChild.data;

      return longitud;
}
```

Figura 62. Funciones ECMAScript obtener latitud y longitud

Como se observa en la figura anterior, ayudándonos de la función "*getElementsByTagName*()"se obtendrán las etiquetas "<*lat*>" y "<*lng*>", y mediante la función "*firstChild.data*" obtendremos los valores de éstas. En ECMAScript los vectores empiezan a tener información desde 0, como nosotros tenemos en cuenta desde 1, se tendrá que restar 1 para dar con la información correcta.

Una vez que tenemos las coordenadas del lugar interés, las guardamos en las variables “longDestino” y ”latDestino”, y se podrá proceder al módulo *Informar*, donde se le comunicará al usuario cómo llegar al lugar deseado.

3.5 Módulo Informar

Una vez que se han obtenido las coordenadas de origen y las coordenadas destino (ya sea mediante calle destino o lugares de interés) se puede proceder a hacer la petición a Google para obtener los pasos para llegar de un lugar a otro y, una vez obtenida la información, comunicársela al usuario. Este módulo, por lo tanto, diferenciará dos pasos: realizar la petición a Google y comunicar esa información al usuario.

3.5.1 Realizar la petición a Google.

En este paso se realizará una llamada a *Google Directions*. La única diferencia con peticiones anteriores es que la realizaremos con coordenadas (latitud, longitud) que se han obtenido en los módulos anteriores. En la Figura 63 vemos una pequeña descripción de cómo se lleva a cabo.

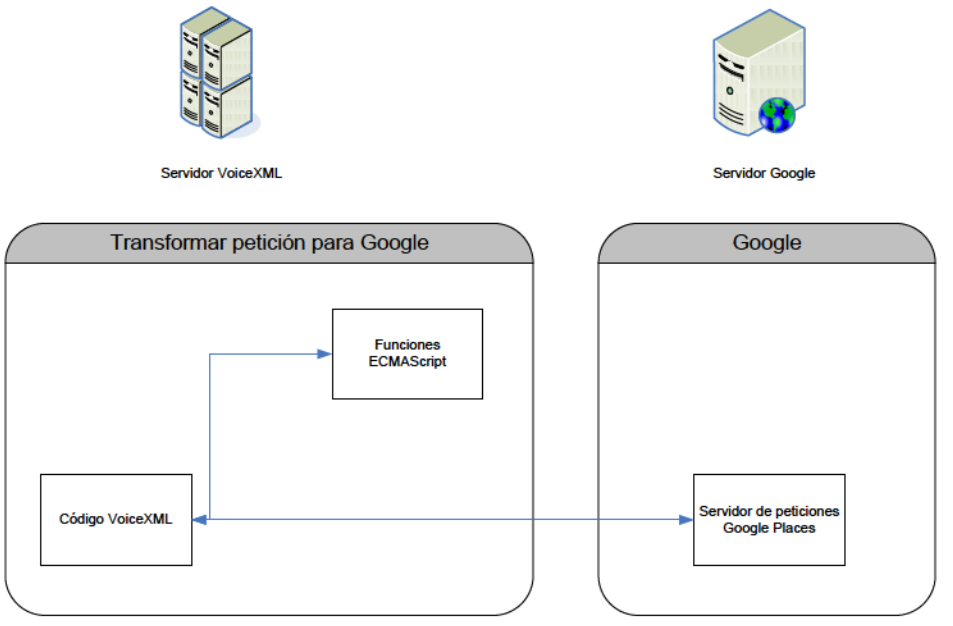

Figura 63. Relaciones en la realización de la petición a Google Directions

Como en los casos anteriores, una vez que hemos obtenido la calle origen y destino (esta vez en coordenadas), se tendrá que transformarlas en una petición HTTP. Para ello, simplemente uniremos las coordenadas correspondientes a origen y destino en las variables *"origin"* y *"destination"* respectivamente.

Se definen las dos variables presentadas tal y como se muestra en la Figura 64.

```
<var name="origin" expr="application.latOrigen
+','+application.longOrigen"/>

<var name="destination" expr="application.latDestino
+','+application.longDestino"/>
```

Figura 64. V Código VoiceXML variables *"origin"* y *"destination"*

Una vez definidas las variables se procede a realizar la petición a *Google Directions* tal y como mostramos en la Figura 65.

```
<data name="xmlSubjects" srcexpr="MyXMLDest"
namelist="origin destination language mode sensor"/>
```

Figura 65. Código VoiceXML para la realización de petición a Google Directions

Las variables "*origin*" y "*destination*" han sido definidas anteriormente y las demás variables ya tienen su valor correspondiente en "*Root*".

- *MyXMLDest*: Esta variable se define como "srcexpr", ya que será la URL a la que habrá que realizar la petición, en este caso está definida como la URL a Google Directions: 'https://maps.googleapis.com/maps/api/place/search/xml?'.
- Language: Es el lenguaje, que en nuestro caso será el español 'es'.
- *Mode*: El medio de transporte. En nuestro caso será andando lo señalamos como "*walking*".
- *Sensor*: Si el dispositivo desde el que llamamos tiene GPS o no, por defecto pondremos que no ya que no nos va a aportar información adicional "*false*".

La respuesta de la petición se guardará en "*xmlSubjects*", y será la que se tratará para extraer la información devuelta.

Por ejemplo, si se quiere realizar una petición desde la calle "San Ramón Nonato, 4" a la calle "Mauricio Legendre, 11" se tendría primero que conseguir sus coordenadas (paso que se realiza en módulos anteriores):

- San Ramón Nonato, 4, Madrid: 40.4697600,-3.6870300.
- Mauricio Legendre, 11, Madrid: 40.4697200,-3.6867200

En la Figura 66 se puede ver cómo quedaría una petición a Google Directions codificando todo en una URL con los datos exactamente igual a como los definidos.

```
http://maps.google.com/maps/api/directions/xml?origin=40.4
697600,-3.6870300&destination=40.4697200,-
3.6867200&mode=walking&language=es&sensor=false
```

Figura 66. Ejemplo URL petición a Google Directions

Una vez que hemos realizado la petición a Google, se recogerá la información de la petición y se le comunicará al usuario.

3.5.2 Recepción de la información

Después de haber realizado la petición a Google, se procederá a tratar la información para comunicársela al usuario. La interacción de este módulo se muestra en la Figura 67.

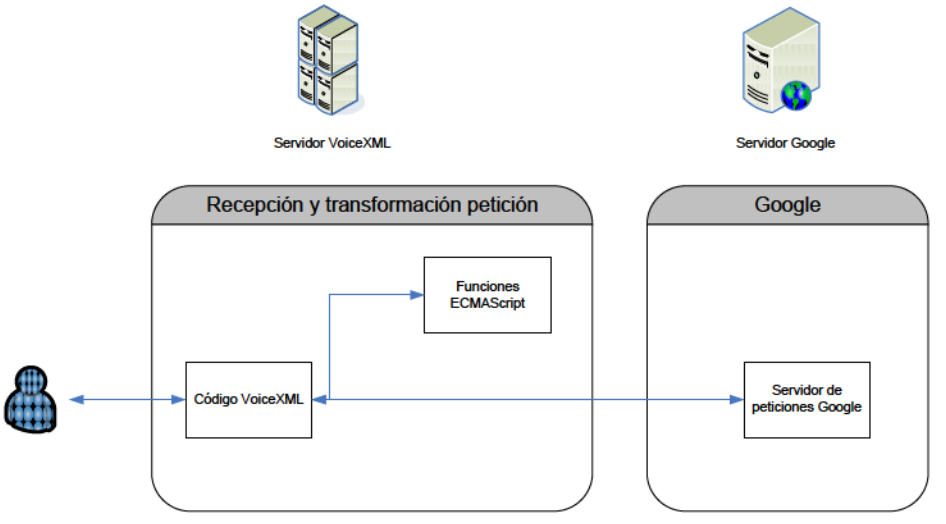

Figura 67. Recepción de información Google Directions y comunicación al usuario

Lo primero que realiza este módulo es procesar la información facilitada por Google para generar la respuesta al usuario. Por ejemplo, si se realiza la petición mostrada en la Figura 66, Google nos respondería con la información mostrada en la Figura 68 como un árbol XML. Se ha elegido para realizar la petición calles colindantes, para no obtener una respuesta muy larga.

```
<DirectionsResponse>
      <status>OK</status>
      <route>
            <summary>Calle de San Ramón Nonato</summary>
            <leg>
                  <step>

      <travel mode>WALKING</travel mode>
                        <start_location>
```

```
                        <lat>40.4697600</lat>
                        <lng>-3.6870300</lng>
                    </start_location>
                    <end_location>
                        <lat>40.4697200</lat>
                        <lng>-3.6867200</lng>
                    </end_location>
                    <polyline>
                        <points>
                            _g_vF|boU?CFy@
                        </points>
                    </polyline>
                    <duration>
                        <value>19</value>
                        <text>1 min</text>
                    </duration>
                    <html_instructions>
                        Dirígete al <b>este</b> por
                        <b> Calle de San Ramón
                        Nonato</b> hacia<b>Calle
                        de Mauricio Legendre</b>
                    </html_instructions>
                    <distance>
                        <value>26</value>
                        <text>26 m</text>
                    </distance>
                </step>
            <duration>
                <value>19</value>
                <text>1 min</text>
            </duration>
            <distance>
                <value>26</value>
                <text>26 m</text>
            </distance>
            <start_location>
                <lat>40.4697600</lat>
                <lng>-3.6870300</lng>
            </start_location>
            <end_location>
                <lat>40.4697200</lat>
                <lng>-3.6867200</lng>
            </end_location>
                    <start_address>
                        Calle de San Ramón Nonato,
                        6, 28046 Madrid, España
                    </start_address>
            <end_address>
                Calle de Mauricio Legendre, 11,
                28046 Madrid, España
            </end_address>
            </leg>
                <copyrights>
```

```
                        Datos de mapa ©2012 Google,
                        Tele Atlas
                  </copyrights>
                  <overview_polyline>
                  <points>_g_vF|boUF}@</points>
            </overview_polyline>
            <warning>
                  Las rutas a pie están en versión beta.
                  Ten cuidado. En esta ruta puede que
                  no haya aceras o pasos
                  para peatones.
            </warning>
            <bounds>
                  <southwest>
                        <lat>40.4697200</lat>
                        <lng>-3.6870300</lng>
                  </southwest>
                  <northeast>
                        <lat>40.4697600</lat>
                        <lng>-3.6867200</lng>
                  </northeast>
            </bounds>
      </route>
</DirectionsResponse>
```

Figura 68. Ejemplo de respuesta proporcionada por Google Directions

Como se observa en la Figura 68, al hacer una petición en la que se pide información de dos calles adyacentes, Google nos envía mucha información, desde coordenadas, tiempo estimado, etc. Aunque ya se explicó en el estado del arte con detalle cómo está formada esta respuesta, se va a realizar un pequeño resumen. La información es por un único <DirectionsResponse> y por dos elementos de nivel superior:

- <*status*>: nos indica si ha finalizado con éxito la petición, en la Figura 68 se puede observar que es "*OK*".
- Cero o más elementos "<*route*>": Contienen un conjunto único de información de las rutas entre el origen y el destino cada uno, se escogerá el primero ya que será el más rápido. En la Figura 68 podemos ver que en el ejemplo sólo tenemos un elemento" <*route*>".

A su vez las rutas están formadas por:

- <*Summary*>*:* Contiene una breve descripción textual sobre la ruta que permita identificarla y distinguirla de otras alternativas, (en el ejemplo de la Figura 68 tiene un valor de "Calle de San Ramón Nonato").
- <*legs*>[]: Contiene un conjunto que consta de información sobre un tramo de la ruta comprendido entre dos ubicaciones de la ruta proporcionada. A todos los hitos o los destinos especificados les corresponde un tramo distinto. Una ruta sin hitos contendrá exactamente un tramo dentro del conjunto de *legs*. Cada tramo consta de una serie de pasos (*steps*).

En nuestro caso de la Figura 68, "<*legs*>" sólo tiene una iteración ya que especifican tramos en los trayectos, y este "<leg>" o tramo estará formado por:

- <*steps*>[]: Contiene un conjunto de pasos que proporciona información sobre cada uno de los pasos del tramo de un trayecto. En nuestro ejemplo de la Figura 68 sólo hemos mostrado una iteración.
- <*distance*>: Es un campo que indica la distancia total que abarca el tramo y que consta de los siguientes elementos:
    - <*value*>: indica la distancia en metros. Como se puede observar en la Figura 68, el valor de la primera ocurrencia es "26".
    - <*text*>: contiene una representación interpretable por humanos de la distancia, expresada en las unidades utilizadas en el origen. Como se puede observar en la Figura 68, el valor de la primera ocurrencia es "26 m".

Si no se conoce la distancia, es posible que estos campos no aparezcan.

- <*duration*>: Es un campo que indica el tiempo total necesario para recorrer el tramo y que consta de los siguientes elementos:
    - <*value*>: Indica la duración en segundos, el valor de la primera ocurrencia es “19”.
    - <*text*>: contiene una representación interpretable por humanos de la duración, el valor de la primera ocurrencia es “1 min”.

Si no se conoce la duración, es posible que estos campos no aparezcan.

- <*start_location*>: Contiene las coordenadas de latitud/longitud del origen del tramo. En el ejemplo de la Figura 68, podemos observar tiene los valores:

  <lat>40.4697600</lat>
  <lng>-3.6870300</lng>
- <*end_location*>: Contiene las coordenadas de latitud/longitud del destino dado del tramo, En el ejemplo de la Figura 68, podemos observar que tiene los valores:

  <lat>40.4697200</lat>
  <lng>-3.6867200</lng>
- <*start_address*>: Contiene una dirección interpretable (normalmente una calle) que refleja el valor “*start_location*” de dicho tramo. En el ejemplo de la Figura 68, contiene los valores:

  Calle de San Ramón Nonato,
  6, 28046 Madrid, España

- <*end_address*>: Contiene una dirección interpretable (normalmente una calle) que refleja el valor “*end_location*” de dicho tramo. En el ejemplo de la Figura 68, contiene los valores:

  Calle de Mauricio Legendre, 11,
  28046 Madrid, España

Por último, dentro de "<*steps*>[]" se definirá cada paso. Este elemento contiene:

- <*html_instructions*>: Contiene instrucciones de formato para este paso, presentadas en forma de cadena de texto HTML. En el ejemplo de la Figura 68 contiene los valores:

  Dirígete al este por
   Calle de San Ramón
  Nonato haciaCalle
  de Mauricio Legendre

- <*distance contiene*>: La distancia que hay que recorrer desde un paso hasta el siguiente. El ejemplo de la Figura 68 contiene los mismos valores que los de la etiqueta "<*leg*>"
- <*duration*>: Contiene el tiempo normal necesario para realizar un paso antes de pasar al siguiente. Si no se conoce la duración, es posible que este campo no esté definido. El ejemplo de la Figura 68 contiene los mismos valores que los de la etiqueta "<*leg*>".
- <*start_location*>: Es un conjunto único de campos de lat y de lng que indica la ubicación del punto de partida de un paso determinado. En el ejemplo de la Figura 68 contiene los mismos valores que los de la etiqueta "<*leg*>".
- <*end_location*>: Es un conjunto único de campos de *lat* y de *lng* que indica la ubicación del punto de llegada de un paso determinado. En el ejemplo de la Figura 68 contiene los mismos valores que los de la etiqueta "<*leg*>", por lo que la se puede decir que el tramo definido en la etiqueta "<*leg*>" y el definido en "<*step*>" es el mismo.

De toda la información que se nos devuelve de la petición sólo nos quedaremos de cada paso (<*steps*>) lo siguiente:

- <*html_instructions*>: Las instrucciones en forma de texto, como "Dirígete hacia el Este en Calle de San Ramón Nonato hacia Calle de Mauricio Legendre"
- <*distance*>: La distancia, que vendrá expresada en metros o en kilómetros, dentro de "<*distance*>" se obtendrá el valor de "<*values*>", por ejemplo "26".

Más adelante se verá cómo se trata la información devuelta por Google para obtener estos datos. Se ha optado por obtener sólo estos valores para no sobrecargar al usuario y proporcionarle la información más importante.

Para obtener esta información nos ayudaremos de la función proporcionada por VoiceXML "*documentElement*", que referencia a la raíz del árbol del DOM (*Domument object Model*).

Una vez que el puntero está dirigido a la raíz del árbol ya se podrá operar para obtener los elementos que nos interesan. Para conseguir todas las indicaciones por texto se usará la función que hemos creado "*getElements*()", mostrada en la Figura 69.

```
function getElements()
      {
            var result =quote.
            getElementsByTagName("html_instructions");

            var indicaciones = new Array;
            for(i=0;i<result.length;i++)
                  indicaciones[i]=
                  quitarHTML(
                  CambiarAbreviaturas(
                  result.item(i).firstChild.data));

            return indicaciones;
}
```

Figura 69. Función ECMAScript para obtener elementos

La función mostrada en la Figura 69 proporciona una matriz con todas las instrucciones de texto devueltas por Google. Para ello, usamos las funciones proporcionadas por ECMAScript como "*getElementsByTagName*", que obtiene todos los elementos que corresponden a una etiqueta dada (en este caso "<*html_instructions*>"). También se usará la función "*ítem*()", que devuelve el elemento que está en la posición del número indicado en el paréntesis. Mediante "firsChild.data" se conseguirá la información de texto de cada elemento.

La información "<html_instructions>" que devuelve el servidor de Google está desarrollada para visualizarla de forma escrita y no de forma oral, por lo que se tendrá que transformar a formato oral, esto lo realizaremos ayudándonos de dos funciones:

- *CambiarAbreviaturas*(): Esta función cambiará las abreviaturas por palabras completas, como "Av" por "Avenida" o "Sta" por "Santa" ya que, si no fuese así, el usuario tendría problemas para entenderlas.

- *QuitarHTML*(): El servidor de Google devuelve las indicaciones en formato HTML. Por ejemplo, palabras entre <b> (negrita), etc. Por ello se borrará todo lo que sea devuelto entre "< >", asegurándonos así que el mensaje oral no contenga símbolos sin sentido.

Una vez que hemos se ha obtenido en una vector las indicaciones de cómo llegar, se procesan de forma parecida las distancias. Lo realizaremos mediante la función "*getDistancias*()", mostrada en la Figura 70.

```
function getDistancias()
{
      var result =quote.
      getElementsByTagName("distance");

      var distancias = new Array;
      for(i=0;i<result.length;i++)

            distancias[i]=result.item(i).
            getElementsByTagName("value").
            item(0).firstChild.data;

      return distancias;
}
```

Figura 70. Función ECMAScript para obtener distancias

Como se observa en la la figura anterior, al igual que en la función "getElements()", con las herramientas que nos proporcionan ECMAScript se obtendrán los elementos de la etiqueta "<*distance*>" mediante la función "*getElementsByTagName*()". Una vez obtenidos todos los elementos se seleccionan dentro de estos elementos los que tengan la etiqueta "<*value*>". De éstos valores, con "*ítem(0).firsChild.data*" se obtendrán los valores de las distancias en un vector.

Por lo tanto, se tendrá la información que le queremos comunicar al usuario en dos vectores, uno con las indicaciones y otro con las distancias.

3.5.3 Comunicación de la información al usuario.

Una vez que el sistema dispone de toda la información que le vamos a trasmitir al usuario, se procede a comunicársela paso a paso. Para ello, recorreremos los vectores usando un índice. Para comunicar al usuario cada paso se ha realizado un bloque "<*block*>" llamado "decirIndicacion", mostrado en la Figura .

```
<block name="decirIndicacion">
      <if cond="indice&lt;longitud">
            <assign  name="indicacionAux"
            expr="getElement(indicaciones,indice)"/>

            <prompt bargein="false" xml:lang="es-es"> .
                  <value expr="getElement(
                  indicaciones,indice)"/>.

                  <value expr="getElement(
                  distancias,indice)"/>, metros

            </prompt>
            <if cond="indice + 1 == longitud">
                  <goto nextitem="otra"/>
            <else/>
                  <goto nextitem="repetir"/>
            </if>
      <else/>
            <prompt bargein="false" xml:lang="es-es">
                  Ha habido un problema con el
                  API de GOOGLE, lo sentimos.
            </prompt>
            <exit/>
      </if>
</block>
```

Figura 71. Código VoiceXML para informar una indicación al usuario

Como se observa en la Figura 71, lo primero que se hará será comprobar si el índice es mayor que el número total de indicaciones que se han obtenido (se guardará en la variable "longitud"), en vez de usar el símbolo de menor que ("<") tendremos que usar entidades HTML, que son códigos que sustituyen a los símbolos. En este caso usaremos "<" (*less than*). En el caso de que efectivamente sea menor, se procederá a comunicar la indicación al usuario. Para ello, se usará la función "*getElements"*, la única acción que realiza la función "getElement(array, id)" es al pasarle el nombre de un vector en la variable "*array*" (en nuestro caso sólo puede ser o de nombres o de distancias) y un número con la posición en el "*array*" que ocupa el dato que se quiere obtener. Esta función devolverá el valor deseado.

Cuando se obtiene la instrucción deseada, se procederá a informar al usuario. Una vez que se ha informado la instrucción, se

comprobará si ésta era la última. Se realizará comprobando si la indicación que se acaba de comunicar + 1 es igual al número total de instrucciones. De ser así, le ofreceremos al usuario la posibilidad de realizar otra consulta, o salir ya de la aplicación, dado que la última instrucción que nos comunica Google es para informar que ha llegado al destino correctamente, y no proporciona valor añadido.

En caso de que no sea la última consulta, se interactuará con el usuario para ofrecerle la posibilidad de repetir o de obtener la siguiente indicación. Para ello, se crea la variable "repetir" y mediante la gramática "Repetir.grxml" se obtendrá los deseos del usuario. Esta gramática sólo reconocerá "siguiente" y "repetir", podemos ver como se interactúa con esta función mediante el código mostrado en la Figura 72.

```
<field name="repetir">
       <nomatch>
              No te entiendo, diga siguiente o repetir.
       </nomatch>
       <help>
              Ayudando, Por favor diga siguiente o repetir:
       </help>
       <noinput>
              Por favor  diga siguiente o repetir.
       </noinput>

       <grammar src=
       "http://webhosting.voxeo.net/61369/www/grammar/Repetir.grx
ml"/>

        <filled>
              <if cond="repetir=='Siguiente'">
                     <assign name="indice"
                     expr="indice+1" />

              </if>
              <clear namelist="destino"/>
              <clear namelist="repetir"/>
              <goto nextitem="decirIndicacion"/>

       </filled>
</field>
```

Figura 72. Código VoiceXML para repetir una indicación

Al igual que en las anteriores ocasiones en las que se han usado gramáticas, se controlarán los distintos eventos que se pueden producir.

Si el sistema mediante la gramática ha entendido correctamente lo que nos ha dicho el usuario, habrá dos opciones: si el usuario ha pedido la siguiente instrucción, se aumentará la variable en una posición y se volverá a llamar a "decirIndicacion", donde se comunicará la siguiente indicación. En caso que el usuario haya pedido que se repita la indicación, se volverá a llamar también a "decirIndicacion" pero sin aumentar el índice, por lo que éste se repetirá.

Una vez que se ha llegado al final de las indicaciones, se le preguntará al usuario si quiere realizar otra consulta o salir, para ello se utiliza la gramática "otra.grxml". En caso que el usuario quiera otra gramática limpiaremos todas las variables de información y mandaremos el flujo de la ejecución al primer módulo. En caso de querer salir, le daremos las gracias al usuario por usar nuestro callejero y saldremos de la aplicación.

3.6 Gestión de eventos.

En este apartado se va a explicar cómo se gestionan los eventos fuera del ciclo normal de ejecución. Para ello se creará en el módulo *Raíz* eventos y se capturarán otros para poder ser usados en cualquier momento y lugar de la ejecución.

Para la definición y captura de eventos nos ayudaremos de la etiqueta "<*link event*>" y de la etiqueta "<*catch event*>":

- <*link event*>: Es donde se definirá el evento que podrá ser capturado en cualquier momento. Para definirlo, se tendrá que

establecer una gramática para que el sistema reconozca cuándo se ha lanzado un cierto tipo de evento.

- <*catch event*>: Es donde se captura un evento dado y las acciones a realizar.

Para ese sistema se han creado dos eventos: evento "salida" y evento "ayuda". En la Figura 73 mostramos como se definiría el evento salida "*exit*". Como se puede observar en la figura, se define una gramática que reconoce solo una palabra "salir", cuando el usuario diga "salir" en cualquier momento de la ejecución, el sistema lanzará el evento salir.

```
<link event="exit">
      <grammar root="main">
            <rule id="main" scope="public">
            <one-of>
                  <item>salir</item>
            </one-of>
            </rule>
      </grammar>
</link>
```

Figura 73. Código VoiceXML del evento salida

En la Figura 74 mostramos como definiremos el evento ayuda "*help*".

```
<link event="help">
      <grammar root="main">
            <rule id="main" scope="public">
                  <item repeat="0-1">
                        por favor
                  </item>
                  <item repeat="0-1">
                        necesito
                  </item>
                        ayuda
                  <item repeat="0-1">
                        me
                  </item>
                  <item repeat="0-1">
                        por favor
                  </item>
            </rule>
      </grammar>
</link>
```

Figura 74. Código VoiceXML del evento ayuda

La palabra principal de la gramática para que el sistema lance el evento ayuda es “ayuda”, para no caer en limitaciones innecesarias, también se posibilita el reconocimiento de sentencias como:

- “por favor, necesito ayuda”
- “ayúdame por favor”

Una vez definidos los eventos, es necesario contemplar cómo capturarlos. Mostramos en la Figura 75 la captura del evento salida.

```
<catch event="exit">
      <prompt>
            Ha seleccionado salir, gracias por usar el
            sistema de Callejero por Voz de la ciudad
            de Madrid, Adios.
      </prompt>
      <exit/>
</catch>
```

Figura 75. Código VoiceXML para captura del evento de salida de la aplicación

Como vemos en la figura anterior, cuando el usuario diga “salir” y se lance el evento de salida, lo capturaremos, se despedirá al usuario y terminaremos con la ejecución.

La captura del evento ayuda se ha ido definiendo a lo largo de todos los módulos, ya que se requiere que para cada caso actúe de una forma diferente.

3.7 Ejemplos diálogos de la aplicación

A continuación mostraremos ejemplos de iteración con nuestra aplicación, para ello lo dividiremos en varios apartados cada uno mostrando una funcionalidad de la aplicación. Se intentará explorar la máxima funcionalidad posible (mediante silencios o mensajes erróneos). Cuando la aplicación sea la que reproduzca una elocución la indicaremos como M, cuando sea un usuario le indicaremos con una U.

3.7.1 Ejemplo 1: ir de una calle a otra

M) Hola, bienvenido al callejero de la ciudad de Madrid.
M) Por favor diga la calle origen.
U) [Calle inexistente].
M) La calle no ha producido ningún resultado, por favor repítala o diga el nombre de otra calle.
U) San Ramón Nonato.
M) Por favor diga el número de la calle origen.
U) Cuatro.
M) La calle de origen es San Ramón Nonato número cuatro, ¿es correcto?
U) Sí.
M) Si desea ir a otra calle diga uno, si desea ir a una parada de metro diga dos, si desea ir a un restaurante cercano diga tres.
U) Repetir.
M) Diga una de las opciones indicadas, si desea ir a una parada de metro diga dos, si desea ir a un restaurante cercano diga tres.
U) (silencio)
M) Seleccione una opción.
U) Uno.
M) Por favor diga la calle destino.
U) (Silencio)
M) Por favor diga la calle destino.
U) Agustín de Foxa.
M) Por favor diga el número de la calle destino.
U) (Silencio).
M) Por favor diga el número de la calle destino.
U) Dos.
M) La calle destino es Agustín de Foxa número dos, ¿es correcto?
U) (silencio)
M) Por favor diga sí o no.
U) Cinco.
M) No le entiendo diga sí o no.
U) Sí.
M) A continuación procederemos a decirle las indicaciones, para escuchar la siguiente indicación diga siguiente, para volverla a escucharla diga repetir.
M) Diríjase al este por calle San Ramón Nonato a calle Mauricio Legendre, 103 metros.
U) (silencio).

M) Por favor diga siguiente o repetir.
U) Repetir.
M) Diríjase al este por calle San Ramón Nonato a calle Mauricio Legendre, 103 metros.
U) Siguiente.
M) Gira a la derecha hacia calle de Mauricio Legendre, 101 metros.
U) Siguiente.
M) Ha llegado a su destino.
M) Si quiere realizar otra consulta diga otra, si quiere salir diga salir.
U) Salir.
M) Gracias por usar el callejero de la ciudad de Madrid, que pase buen día.

3.7.2 Ejemplo 2: buscar una parada de metro

M) Hola, bienvenido al callejero de la ciudad de Madrid.
M) Por favor diga la calle origen.
U) San Ramón Nonato.
M) Por favor diga el número de la calle origen.
U) Cuatro.
M) La calle de origen es San Ramón Nonato número 4, ¿es correcto?
U) Sí.
M) Si desea ir a otra calle diga uno, si desea ir a una parada de metro diga dos, si desea ir a un restaurante cercano diga tres.
U) Dos.
M) Ha seleccionado la opción buscar una parada de metro, por favor diga la distancia en metros en la que encontrar una parada de metro.
U) 800.
M) Por favor diga el número de la opción que quiere:
1) Ventilla.
2) Duque de Pastrana.
3) Valdeacederas.
U) (Silencio)
M) Lo siento no le he oído.
U) Uno.
M) A continuación procederemos a decirle las indicaciones, para escuchar la siguiente indicación diga siguiente, para volverla a escucharla diga repetir.
M) Diríjase al sur por Paseo de la Castellana hacia Calle de Carmen de Sánchez Carrascosa. 138, metros.
U) Siguiente.

M) Gire a la derecha para continuar en Paseo de la Castellana. 72 metros.
U) Siguiente.
M) Gire a la izquierda para continuar por Paseo de la Castellana. 78 metros.
U) Siguiente.
M) Gire a la derecha hacia Calle de San Benito. Pase una rotonda. 365, metros.
U) Siguiente.
M) Gire a la izquierda hacia Calle del Cañaveral. 140, metros.
U) Siguiente.
M) Gire a la derecha hacia Avenida de Asturias, el destino está a la derecha. 232 metros.
U) Siguiente.
M) Ha llegado a su destino.
M) Si quiere realizar otra consulta diga otra, si quiere salir diga salir.
U) Salir.
M) Gracias por usar el callejero de la ciudad de Madrid, que pase buen día.

3.7.3 Ejemplo 3: buscar un restaurante.

M) Hola bienvenido al callejero de la ciudad de Madrid.
M) Por favor diga la calle origen.
U) San Ramón Nonato.
M) Por favor diga el número de la calle origen.
U) Cuatro.
M) La calle de origen es San Ramón Nonato número 4, ¿es correcto?
U) Sí.
M) Si desea ir a otra calle diga uno, si desea ir a una parada de metro diga dos, si desea ir a un restaurante cercano diga tres.
U) Tres.
M) Ha seleccionado la opción buscar un restaurante, por favor diga la distancia en metros en la que encontrar un restaurante.
U) 200.
M) Por favor diga el número de la opción que quiere:
1) El pato laqueado.
2) Restaurante Fresco.
3) Mesón restaurante el Laconero.
U) Uno.

M) A continuación procederemos a decirle las indicaciones, para escuchar la siguiente indicación diga siguiente, para volverla a escucharla diga repetir.
M) Diríjase al sur por Paseo de la Castellana hacia Calle de Carmen de Sánchez Carrascosa. 138, metros.
U) Siguiente.
M) Gire a la derecha para continuar en Paseo de la Castellana. 72 metros.
U) Siguiente.
M) Gire a la izquierda para continuar por Paseo de la Castellana. 78 metros.
U) Siguiente.
M) Gire a la derecha hacia Calle de San Benito. El destino está a la izquierda, 51 metros.
U) Siguiente.
M) Ha llegado a su destino.
M) Si quiere realizar otra consulta diga otra, si quiere salir diga salir.
U) Salir.
M) Gracias por usar el callejero de la ciudad de Madrid, que pase buen día.

Capítulo 4: Evaluación de la aplicación

En este capítulo se describe la evaluación realizada de la aplicación. Para llevarla a cabo se ha requerido a 15 usuarios probar la aplicación y responder a un cuestionario con un conjunto de 10 preguntas.

4.1 Metodología de evaluación

Dado que la aplicación se ha realizado para usuarios, cobra sentido que sean los propios usuarios los que la evalúen. Para ello, como hemos anticipado se ha elaborado un cuestionario para recoger la opinión subjetiva y el grado de satisfacción de los usuarios. De este modo, no solo se obtendrá la evaluación del sistema, sino que además tenemos la posibilidad de localizar errores.

Para obtener una evaluación lo más completa posible hemos analizado los siguientes factores: la experiencia previa del usuario con este tipo de sistemas, entendimiento por parte del sistema de sus elocuciones, entendimiento de los mensajes reproducidos por el sistema, como de fluida ha sido la obtención de la información requerida, la dificultad, su satisfacción y posibles problemas.

El cuestionario elaborado para este fin consta de 10 preguntas, tal y como muestra la Figura 76.

1.- ¿Cuántas veces ha interactuado oralmente con máquinas?

- Nunca
- Pocas veces
- A veces

- Bastantes veces
- Muchas veces

2.- ¿Cómo ha entendido el sistema los calles/lugares de interés que le ha solicitado?

- Muy bien
- Bien
- Regular
- Mal
- Muy mal

3.- ¿Cómo de claros considera que han sido los mensajes producidos por el sistema?

- Muy claros
- Claros
- Normales
- No han sido claros

4.- ¿Cómo ha sido la comunicación de las indicaciones para llegar de un lugar a otro?

- Muy buena
- Buena
- Regular
- Mala
- Muy mala

5.- ¿Considera que ha resultado sencillo obtener las indicaciones del sistema?

- Muy sencillo
- Sencillo
- Normal
- Difícil
- Muy difícil

6.- ¿Cómo considera que ha sido la velocidad al obtener las indicaciones del sistema?

- Muy rápido

- Rápido
- Adecuada
- Lenta
- Muy Lenta

7.- ¿Piensa que un humano hubiera realizado mejor trabajo que la maquina?

- Si
- No
- Igual

8.- En términos generales, ¿está usted satisfecho con el sistema?

- Si
- No

9.- ¿Ha tenido algún problema con el sistema?

- Si
- No

10.- En caso de haber tenido problemas, por favor describa brevemente cuales han sido.

Figura 76. Cuestionario desarrollado para la evaluación de la aplicación

4.2 Resultados de la evaluación

Se requirió a los 15 usuarios que probaran el sistema y acto seguido respondieran el test de evaluación. Para probar el sistema se les explicó la funcionalidad del callejero y las posibles opciones disponibles, animándoles a probar tanto la búsqueda de direcciones como la de lugares de interés. A continuación, se muestra de forma gráfica las respuestas dadas por los usuarios a cada una de las preguntas descritas anteriormente.

Pregunta 1.- ¿Cuántas veces ha interactuado oralmente con maquinas?

Como observamos en la Figura 77, el 60% de los usuarios encuestados nunca ha interactuado oralmente con maquinas y ninguno de ellos lo ha ha realizado muchas veces.

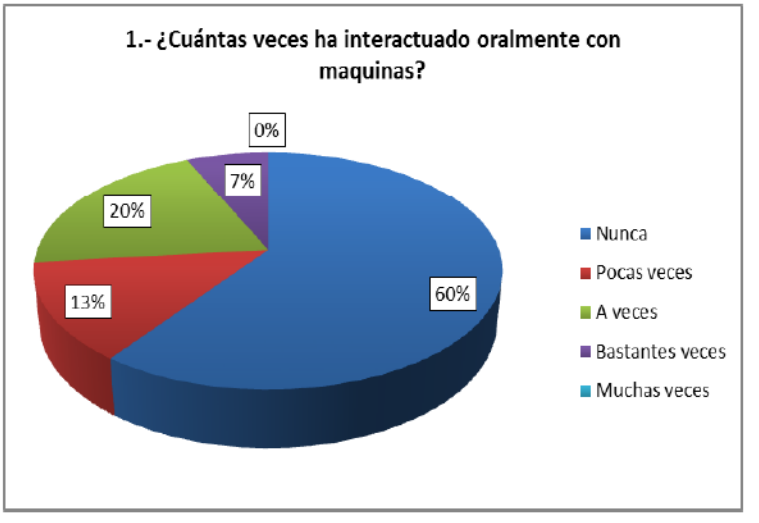

Figura 77. Test evaluación pregunta 1

Pregunta 2.- ¿Como ha entendido el sistema los calles/lugares de interés que le ha solicitado?

Como vemos en la Figura 78 solo el 13% de los usuarios encuestados consideran que el sistema ha entendido mal o muy mal los nombres de las calles/números que han informado al sistema, en la pregunta de problemas con el sistema entraremos con más detalle en este asunto.

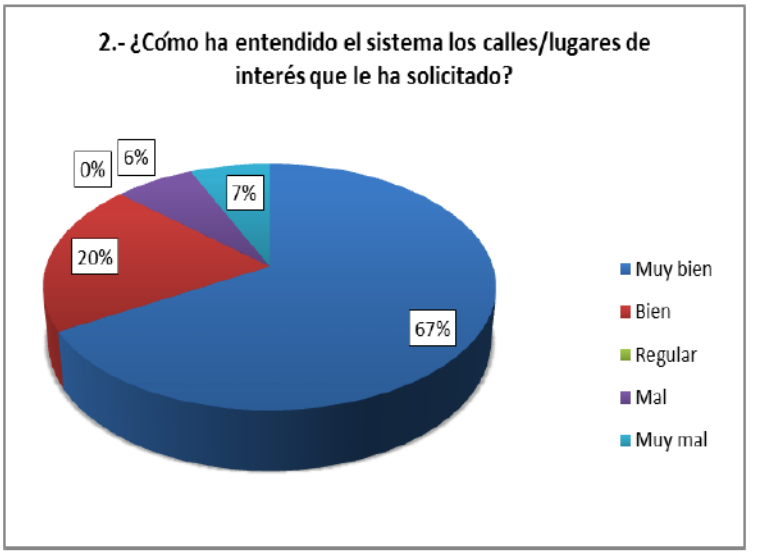

Figura 78. Test evaluación pregunta 2

Pregunta 3.- ¿como de claros considera que han sido los mensajes producidos por el sistema?

Como podemos observar en la Figura 79, los usuarios no han tenido problemas entendiendo las locuciones del sistema, ya que los usuarios consideranmayoritariamente que han sido claros o muy claros los mensajes producidos por el sistema.

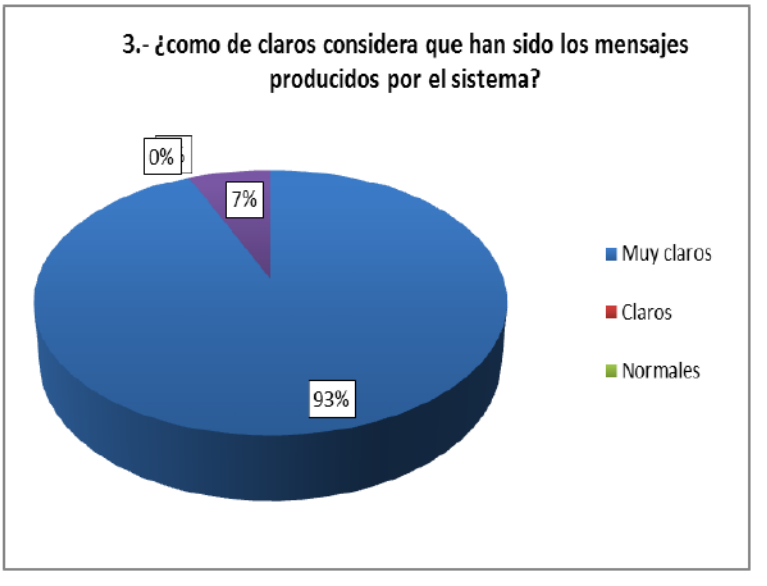

Figura 79. Test evaluación pregunta 3

Pregunta 4.- ¿Cómo ha sido la comunicación de las indicaciones para llegar de un lugar a otro?

En la Figura 80 podemos observar que la mayoria de los usuarios encuestados (93%), piensan que la comunicación de las indicaciones para llegar de un lugar a otro ha sido buena o muy buena.

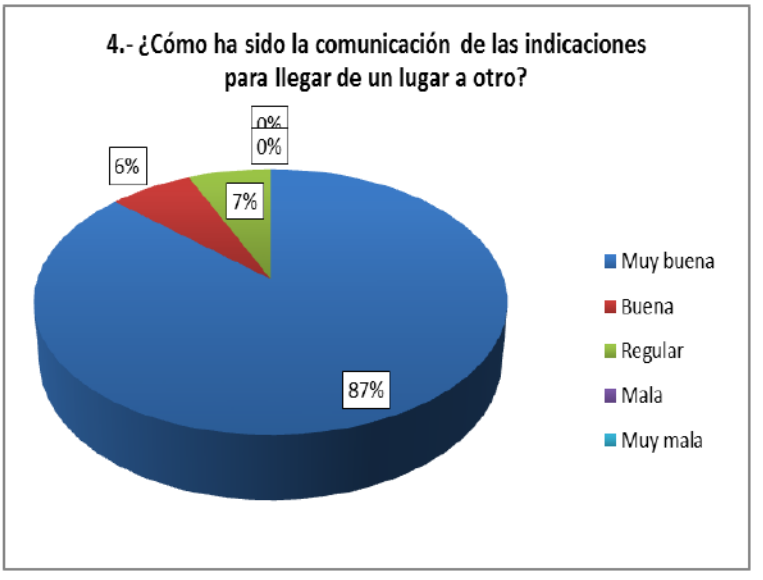

Figura 80. Test evaluación pregunta 4

Pregunta 5.- ¿Considera que ha resultado sencillo obtener las indicaciones del sistema?

Como observamos en la Figura 81, la mayoria de los usuarios piensa que ha sido sencillo o muy sencillo obtener las indicaciones del sistema, por otro lado, el resto (un 7%) piensa que no ha sido ni sencillo ni difícil.

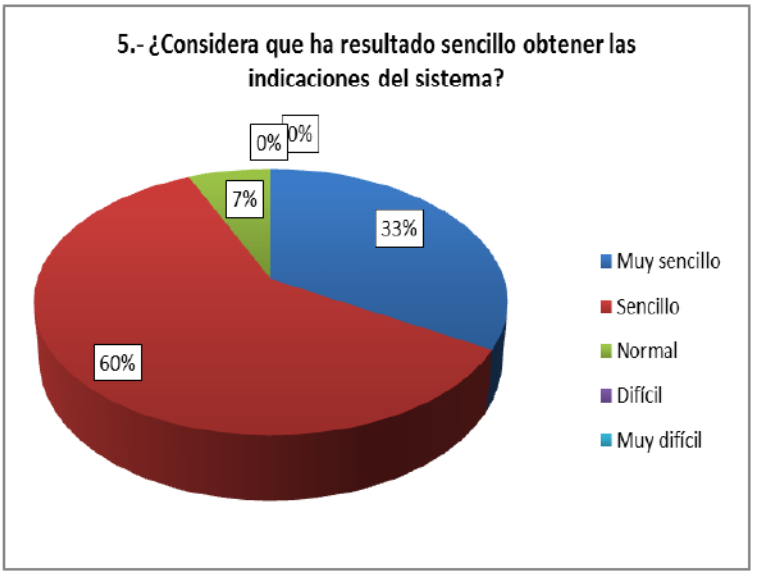

Figura 81. Test evaluación pregunta 5

Pregunta 6.- ¿Cómo considera que ha sido la velocidad al obtener las indicaciones del sistema?

Como se puede observar en la Figura 82, un 66% de los usuarios piensan que es rapido o muy rápido obtener las indicaciones del sistema, mientras que un 20% opina que es lento o muy lento.

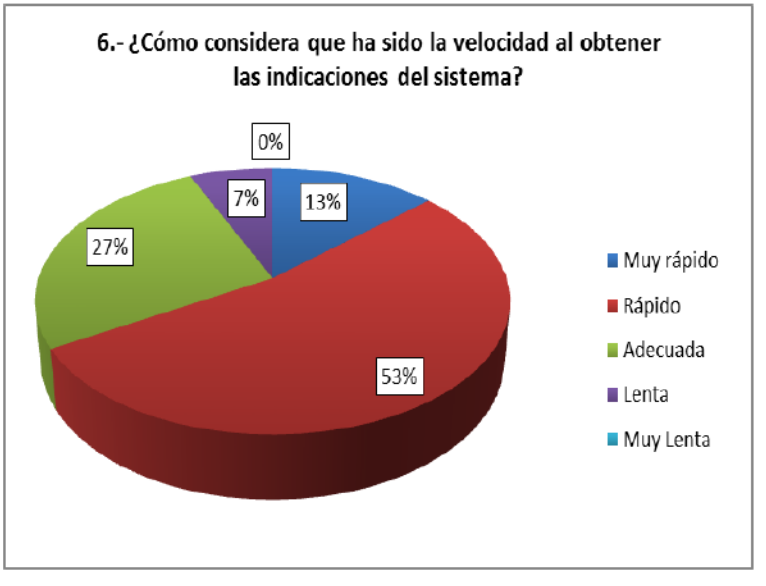

Figura 82. Test evaluación pregunta 6

Pregunta 7.- ¿Piensa que un humano hubiera realizado mejor trabajo que la maquina?

Debido al debate existente sobre sustituir a las personas por maquinas se ha incluido esta pregunta en el cuestionario. En la Figura 83 podemos ver que la mayoria de los usuarios encuestados prefiere a las personas frente a las maquinas.

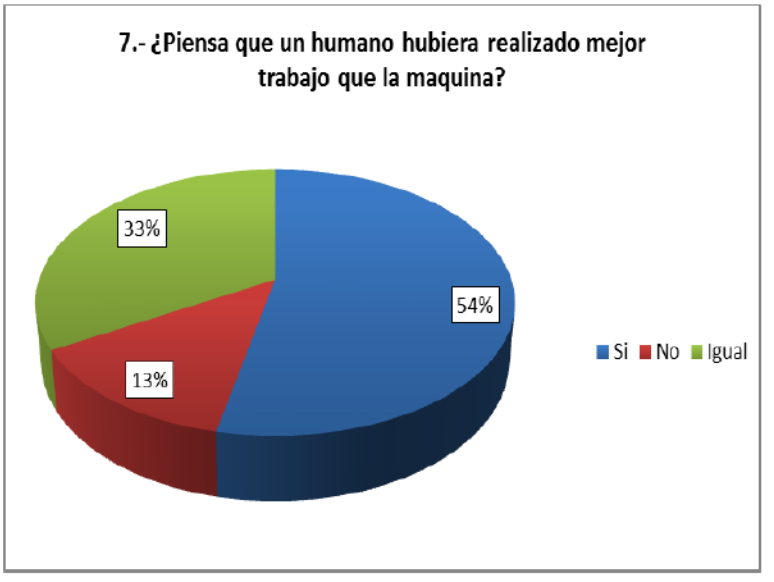

Figura 83. Test evaluación pregunta 7

Pregunta 8.- En términos generales, ¿está usted satisfecho con el sistema?

El 100% de los usuarios encuestados estan satisfechos con el sistema desarrollado, tal y como se muestra en la Figura 84.

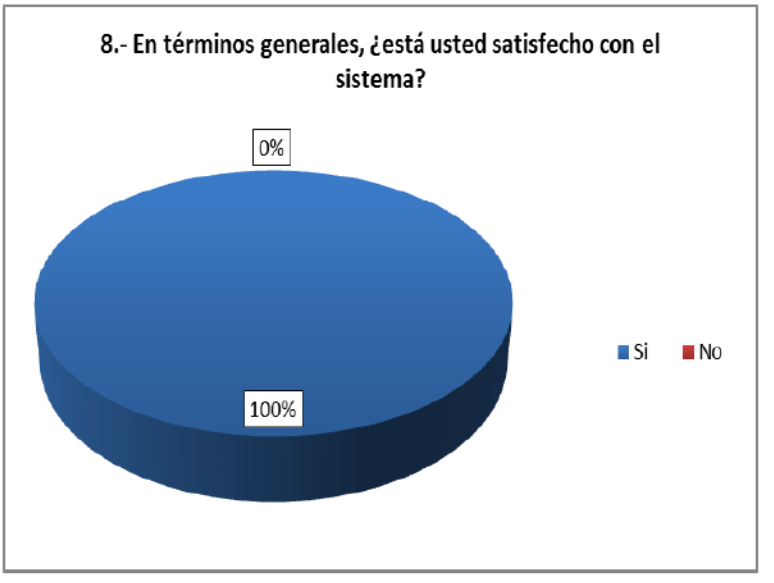

Figura 84. Test evaluación pregunta 8

Pregunta 9.- ¿Ha tenido algún problema con el sistema?

Como se observa en la Figura 85, sólo un 13% de los usuarios encuestados ha tenido algún problema con el sistema.

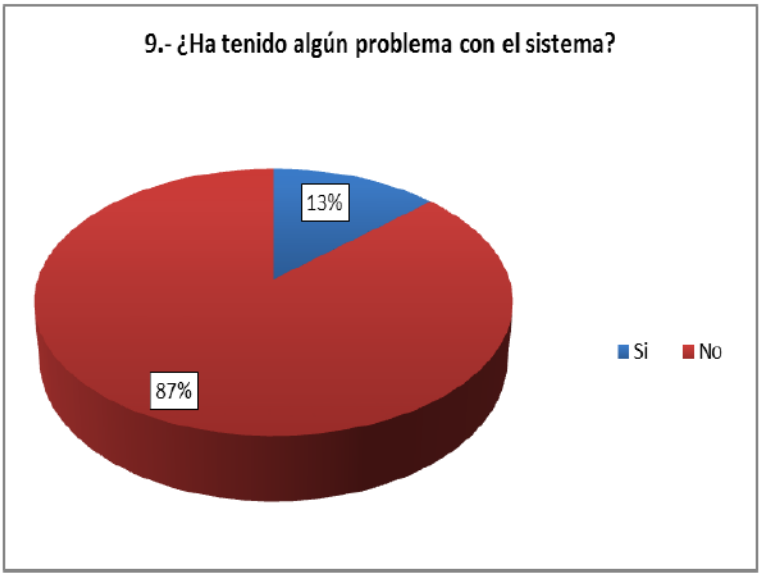

Figura 85. Test evaluación pregunta 9

Pregunta 10.- En caso de haber tenido problemas, por favor describa brevemente cuales han sido.

Sólo dos usuarios de los encuestados han tenido problemas con el sistema y en ambos casos el problema ha sido el mismo: el sistema no entendió las locuciones del usuario (ya sea calle o número). Este problema se ha debido al ruido de fondo mientras se realizaban las pruebas. Para intentar minimizar este problema se ha forzado a que en la petición de calles y números el sistema reproduzca toda la información y que no pare al escuchar ruido de fondo, así el usuario sabrá cuando tiene que hablar y se minimizará los posibles errores.

Analizando los resultados anteriores, podemos obtener las siguientes conclusiones:

- La mayoría de los usuarios a los que se les ha realizado la encuesta no tienen experiencia previa interactuando oralmente con máquinas.
- Para ningún usuario ha sido un problema entender las locuciones de la maquina, sin embargo si ha habido alguno que ha tenido problemas para hacerse entender.
- Aunque los usuarios piensan que la comunicación de las indicaciones es buena y sencilla, algunos piensan que no es rápida.
- Aunque cada vez hay más avances tecnológicos y las maquinas sustituyen más a los humanos, los usuarios todavía prefieren la comunicación con estos.
- En general los usuarios han quedado satisfechos con el sistema.

Capítulo 5: Conclusiones y trabajo futuro

5.1 Conclusiones

En este Proyecto Final de Carrera se ha desarrollado una aplicación basada en el estándar VoiceXML, siendo su principal objetivo realizar una aplicación en este lenguaje de programación. Se ha desarrollado un callejero que no sólo indica los pasos para llegar de un punto a otro, sino que permite realizar búsquedas de lugares de interés.

Se ha separado la aplicación en distintos módulos considerando las funcionalidades y las decisiones que el usuario vaya tomando durante el diálogo. Estos elementos se relacionan de tal forma que en una llamada el usuario no esté limitado a una sola acción, pudiendo realizar, dentro de las posibilidades del callejero, tantas como considere oportunas.

Para fundamentar nuestra aplicación dentro del campo de los sistemas de diálogo, se ha procedido a la realización de un estudio completo de estos sistemas, que ha quedado extensamente plasmado en el apartado relativo al estado del arte, en el que se describen sus capacidades, los módulos que los conforman y sus principales aplicaciones.

Para desarrollar la aplicación basándonos en el potencial de los sistemas de diálogo hemos usado el lenguaje de programación VoiceXML. Este lenguaje nos ha facilitado mucho el desarrollo del callejero, permitiéndonos construir diálogos de forma sencilla y simplificando el reconocimiento de voz mediante las gramáticas, la síntesis de texto a voz o el control de flujo del diálogo.

Este lenguaje de programación tiene una serie de características muy beneficiosas que conviene volver a destacar a continuación. En primer lugar, minimiza las interacciones cliente-servidor mediante la realización de una serie de interacciones múltiples por documento. En segundo lugar, permite separar el código de interacción del usuario (en VoiceXML) de la lógica de servicios. En tercer lugar, cabe destacar la promoción de la portabilidad de servicios a través de plataformas de aplicación que soportan el lenguaje. Por último, no debemos menospreciar la facilidad de uso en el ámbito de las interacciones simples, aunque ello no signifique que no dejen de estar a disposición del usuario las herramientas necesarias para soportar diálogos más complejos.

La actual proliferación y perspectivas de incremento de los sistemas de diálogo, remarcan la importancia de este lenguaje. También cabe destacar que cuenta con el apoyo de muchas empresas, entre las que cabe destacar AT&T, Lucent Technologies, Motorola, IBM, etc. Es por esta razón, que podemos predecir que VoiceXML seguirá creciendo con el tiempo y que nuestra elección de lenguaje utilizado puede considerarse completamente acertada.

La plataforma sobre la que se han implementado todos los desarrollos pertinentes a VoiceXML ha sido Voxeo Evolution. Esta plataforma nos ha proporcionado la infraestructura y los componentes de reconocimiento y síntesis de voz necesarios para nuestro proyecto.

Cabe destacar su sencillez de uso, así como la posibilidad de crear un número al que llamar a las aplicaciones desarrolladas mediante Skype (punto que en la fase de desarrollo ha resultado ser muy útil). No debemos olvidar que, además, también hace las veces de servidor, almacenando los ficheros de la aplicación y que tiene un depurador y un foro en el que profesionales responden a las dudas que

puedan ir surgiendo a medida que se desarrolla y se extiende el uso de la aplicación.

Como principal objetivo del PFC definíamos el estudio y desarrollo de un sistema de diálogo usando el lenguaje de programación VoiceXML, procediendo al estudio de la cohesión que este lenguaje tiene con otras tecnologías. Este objetivo se ha cumplido satisfactoriamente, puesto que se ha realizado un estudio intensivo de estas tecnologías y ha sido desarrollada convenientemente la cohesión con las APIs de Google, pudiendo concluir que el resultado del proyecto queda plasmado en la presente memoria y, de manera primordial, en la aplicación desarrollada.

Por último, se concluye también que se han cumplido los objetivos secundarios remarcados en el apartado correspondiente, habiéndose satisfecho las metas que recordamos a continuación:

- **Accesibilidad:** Al ser una aplicación telefónica, sólo se necesita un teléfono para poder acceder a ella, aunque también se puede usar un ordenador con conexión a Internet.

- **Eliminar barreras de acceso:** Al ser una aplicación accesible exclusivamente por voz, hemos conseguido que un usuario con discapacidades visuales o motoras pueda acceder al callejero, consiguiendo también ofrecer información de calles y lugares sobre una interfaz nueva.

- **Simplicidad:** Se ha conseguido acceder a información que se encuentra en la red a través del teléfono, diseñando una aplicación con una interacción sencilla, donde es fácil navegar y obtener la información que se necesita.

5.2 Conclusiones Personales

Al proceder a la realización del presente Proyecto Fin de Carrera, uno de los principales retos a los que se ha tenido que hacer frente es a la dificultad en la profundización de los conocimientos necesarios para el desarrollo de la aplicación.

El alumno, a pesar de haber adquirido una sólida preparación a lo largo de los años de Universidad, toma conciencia de la necesidad del análisis autónomo y de la profundidad conceptual que necesita adquirir para poder proceder al desarrollo de una aplicación que sea útil.

Aunque se conocían con anterioridad los rudimentos básicos de los elementos utilizados, la necesidad de un análisis pormenorizado del funcionamiento de los sistemas de diálogo y de las APIs de Google - uno de los gigantes informáticos de nuestro tiempo - ha requerido un tiempo considerable, puesto que sin un dominio conceptual y pragmático de estos utensilios, habría sido imposible avanzar en la realización del proyecto.

Asimismo, el empleo y aprendizaje de una de las tecnologías en auge de nuestro tiempo, como es VoiceXML, asumiendo las dificultades que entrañan su comprensión y el esfuerzo que requiere comprender sus componentes y su funcionamiento global, ha supuesto una enorme gratificación personal. Satisfacción que ha de entenderse con la adquisición de una habilidad, de una capacidad a futuro para responder a retos tecnológicos que pudieren aparecer en el marco de la propia actividad laboral.

Por otra parte, y en lo que respecta ya a la pura realización del Proyecto Fin de Carrera, he comprendido la trascendencia del mismo, como un elemento integrador de los conocimientos y capacidades adquiridos con anterioridad. La necesidad de una dedicación constante,

la agotadora actividad de documentación, la complicada tarea de plasmación de los desarrollos conseguidos en la presente memoria, el deseo de perfeccionamiento de las propias expectativas y la dificultad de compaginar la realización del PFC con la actividad laboral, han sido retos continuos que, no sin la comprensión del tutor, cuya paciencia con el retraso constante de las entregas de los hitos preestablecidos, han demostrado dar un fruto provechoso.

Por todo lo dicho, y a la espera de descubrir si el diseño implementado recibe el beneplácito y el interés del público al que va dirigido, puede concluirse que el presente trabajo no es, únicamente, un éxito personal. Detrás de él se encuentra el esfuerzo del alumno, pero también de los docentes, de los investigadores que desarrollaron sus proyectos antes que él y del propio tutor, que se ha esforzado diligentemente y con la mayor destreza en contribuir a una dirección no exenta de complicaciones que, ahora por fin, han sido superadas y solventadas. Es solamente así que hemos conseguido que esta idea original, innovadora e interesante que hemos desarrollado haya podido ver la luz.

5.3 Trabajo Futuro

El presente proyecto ha servido para la presentación de un diseño completo en sí mismo. Pero no debemos olvidarnos, que en toda creación cabe siempre considerar que no estamos más que ante una plataforma inicial, un punto de partida a partir del cual proceder a una mejora constante, a una investigación ilimitada de las posibilidades que se abren ante nuestros ojos.

En este sentido, proceder al desarrollo de innovaciones constantes que puedan adherirse a nuestro diseño ha de ser uno de los aspectos centrales a tener en cuenta. El avance científico nos impulsa a

defendernos de la, cada vez más rápida e inesperada, obsolescencia tecnológica.

Por eso, si con este proyecto hemos desarrollado las bases para el diseño de un callejero por voz, no debemos obviar las infinitas posibilidades de mejora que éste puede ir incorporando a medida que vayamos profundizando en él. Elementos novedosos podrán irse adhiriendo a la creación matriz, para dar lugar a nuevas versiones de nuestro callejero, facilitando la obtención de información, ampliando las posibilidades de búsqueda y perfeccionando la localización de lugares, el perfeccionamiento de la información obtenida y la capacidad del sistema en su conjunto para entender peticiones y procesarlas de manera adecuada. En definitiva, las posibilidades pueden ser tantas como las que actualmente sólo somos capaces de abarcar con nuestra imaginación. A partir de ahí, es nuestra tarea el convertirlas en realidades palpables al servicio de la sociedad en la que vivimos.

Es por estos motivos por los que, una vez concluida la presentación de nuestro proyecto fin de carrera, parece conveniente hacer una reflexión, esto es, un estudio sobre los puntos que podrían ser mejorados o ampliados con relación al diseño que hemos ofrecido.

Con esta ilusión puesta en el futuro, hacemos referencia a los siguientes aspectos:

1.- Mejoras relativas a la ampliación de los contenidos de carácter geográfico.

Actualmente, se ha desarrollado un callejero que se limita al municipio de Madrid, destacando su importancia como capital del Estado y su relevancia en los ámbitos económico, financiero, turístico, sociocultural y político, entre otros. Sin embargo, un desarrollo posterior de nuestro diseño de callejero por voz, podría comenzar por incluir, en las primeras etapas, a los municipios limítrofes.

Posteriormente, en la medida de nuestras posibilidades, cabría ampliarlo a todos los municipios de la zona centro, a la totalidad de la Comunidad Autónoma de Madrid e, incluso, al resto del territorio nacional.

Obviamente, la codificación de los nombres de calles y vías, la existencia de diversos idiomas y otras innumerables problemáticas, habrían de ser tenidas en cuenta para evitar duplicidades, reducir errores y controlar las posibles desambiguaciones que pudieren aparecer.

2.- Mejoras relativas a un mayor contenido de puntos de interés.

En el callejero por voz que se ha presentado, los lugares de interés sobre los que nos hemos basado se limitan a la localización de paradas de metro y de restaurantes. Sin embargo, el API de Google Places nos brinda la oportunidad de localizar muchos más lugares de interés. Dadas las limitaciones de desarrollo, hemos querido acotar a unos determinados destinos, persiguiendo con ello el poder hacer que la navegabilidad por la aplicación se ágil. No obstante, un mayor desarrollo nos permitirá hacer uso del API de Google para la búsqueda de destinos de diferente naturaleza. Pueden verse algunos de estos diversos puntos de interés que API de Google Places nos permitiría localizar en el siguiente enlace:

- https://developers.google.com/maps/documentation/places/supported_types?hl=es

Como puede apreciarse, Google nos da la opción de localizar desde colegios, universidades y hospitales hasta acuarios o parques de atracciones. En suma, existen más de cien tipos distintos de destino a nuestra disposición que podríamos añadir a nuestro callejero por voz en un futuro próximo.

3.- Mejoras relativas al idioma y a la practicidad del callejero.

El idioma empleado para la realización de este proyecto ha sido el castellano. Dada la internacionalización tecnológica y la existencia de una aldea global en términos de innovación, una mejora indiscutible de este trabajo vendría representada por una posible traducción del mismo a otros idiomas.

Este extremo no sólo contribuiría a hacer posible que ciudades de otros países puedan copiar o asimilar el modelo presentado para poder contar con un callejero propio, sino que también facilitaría la utilización del callejero por voz a un mayor número potencial de usuarios, entre los que podríamos contar a turistas, hombres de negocios o trabajadores del sector de los transportes, por poner, tan sólo, algunos ejemplos. Incluso, esto nos da pie a especular con la posible creación de una guía interactiva que ofrezca trayectos turísticos optimizados para visitar una ciudad y acceder a los lugares de interés en el menor tiempo posible y escogiendo una ruta ordenada y coherente con las limitaciones temporales que suelen estar presentes en este tipo de desplazamientos.

GLOSARIO

ABNF *Forma Normal de Backus Aumentada*
API *Application Programming Interface*
ASCII *American Standard Code for Information Interchange*
ATIS *Air Travel Information Services*
BBDD *Bases de Datos*
BNF *Backus-Naur Form*
CAST *Center for Applied Special Technology*
CFG *Context-Free Grammar*
COM *Component Object Model*
CORBA *Common Object Request Broker Architecture*
CSS *Cascading Style Sheet*
DARPA *Defense Advanced Research Projects Agency*
DB *Data Base*
DOM *Document Object Model*
DTD *Definición de Tipos de Documentos*
DTMF *Dual Tone Multiple Frequency*
ECMA *European Computer Manufacturers Association*
FSM *Finite State Machine*
GLN *Generación del Lenguaje Natural*
HTML *HyperText Markup Language*
IETF *Internet Engineering Task Force*
IP *Internet Protocol*
JSGF *JSpeech Grammar Format*
MMI *Multimodal Interaction*
MySQL *My Structured Query Language*
NLSML *Natural Language Semantics Markup Language*
PC *Personal Computer*
PDA *Personal Digital Assistant*
PHP *Personal Home Page; PHP Hypertext Pre-processor*
PLN *Procesamiento de Lenguaje Natural*
PML *Phone Markup Language*
RAH *Reconocedores Automáticos del Habla*
SDO *Sistema de Diálogo Oral*
SEM *Semantic Interpretation for Speech Recognition*
SGML *Standard Generalized Markup Language*
SRGS *Speech Recognition Grammar Specification*
SUNDIAL *Speech UNderstanding and DIALog*
SVG *Scalable Vector Graphics*
TTS *Text-to-Speech*
URI *Identificador Uniforme de Recursos*

VA *Voice Applications*
VD *Voice Dictionary*
VG *Voice Google*
VL *Voice Library*
VoiceXML *Voice eXtensible Markup Language*
VP *Voice Pronunciation*
VV *Voice Videoclub*
W3C *World Wide Web Consortium*
X+V *XHTML+Voice*
XHTML *eXtensible Hypertext Markup Language*
XML *eXtensible Markup Language*

BIBLIOGRAFÍA

[AFG06]Andeani, G., D. Di Fabbrizio, M. Gilbert, D. Gillick, D. Hakkani-Tur, y O. Lemon. 2006. Let's DISCOH: Collecting an Annotated Open Corpus with Dialogue Acts and Reward Signals for Natural Language Helpdesks. En Proc. of IEEE 2006 Workshop on Spoken Language Technology (SLT).

[AJG00]Allen, James, George Ferguson, Bradford W. Miller, Eric K. Ringger, y Teresa Sikorski Zollo. 2000. Dialogue systems: From theory to practice in TRAINS-96. En Handbook of Natural Language Processing.

[APICONSOLA] Google APIs Console. Disponible en: https://code.google.com/apis/console [Consultado el 01/07/2013]

[BCA02]Bennett, Christina, Ariadna Font Llitjós, Stefanie Shriver, Alexander Rudnicky, y Alan W Black. 2002. Building VoiceXML-based applications. En Proc. of International Conference on Spoken Language Processing (ICSLP'02).

[BEM97]Beskow, J., K. Elenius, y S. McGlashan. 1997. Olga - A dialogue system with an animated talking agent. En Proc. of European Conference on Speech Communications and Technology (Eurospeech'97).

[CALLES] Calles de España, longitude. Disponible en: http://www.cosasdemadrid.es/madrid-6-50-calles-mas-largas-espana/ [Consultado el 01/07/2013]

[CHU96]Carlson, R. y S. Hunnicutt. 1996. Generic and domain-specific aspects of the Waxholm NLP and Dialog modules. En Proc. of International Conference on Spoken Language Processing (ICSLP'96).

[CSW03]Catizone, R., A. Setzer, y Y. Wilks. 2003. Multimodal Dialogue Management in the COMIC Project. En Proc. of EACL'03 Workshop on Dialogue Systems: interaction, adaptation, and styles of management.

[DAN08] Carlos Dan, La comunicación entre las plantas y los animales (2008). Disponible en: http://www.ojocientifico.com/2008/07/02/la-comunicacion-entre-las-plantas-y-los-animales [Consultado el 01/07/2013]

[DIRECTIONSAPI] Google Maps API Web Services. Disponible en: https://developers.google.com/maps/documentation/directions/ [Consultado el 01/07/2013]

[ECMASCRIPT] ECMAScript Documentation. Disponible en: www.ecmascript.org/docs.php [Consultado el 01/07/2013]

[JAVASCRIPT] JavaScript from FOLDOC. Disponible en: http://foldoc.org/javascript [Consultado el 01/07/2013]

[EMMA] Extensible MultiModal Annotation markup language. Disponible en: http://www.w3.org/TR/emma/ [Consultado el 01/07/2013]

[FAA98]Ferguson, George y James Allen. 1998. TRIPS: An Intelligent Integrated Problem-Solving Assistant. En Proc. of the Fifteenth National Conference on Artificial Intelligence (AAAI-98).

[FHF06]Feng, Junlan, D. Hakkani-Tur, G. Di Fabbrizio, M. Gilbert, y M. Beutnagel. 2006. WebTalk: Towards Automatically Building dialog services by exploiting the content and structure of websites. En Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'06).

[FOROVXML]: VoiceXML Forum. Disponible en: http://www.voicexml.org/ [Consultado el 01/07/2013]

[GBB00]Gustafson, J., L. Bell, J. Beskow, J. Boye, R. Carlson, J. Edlund, B. Granström, D. House, y M. Wirén. 2000. AdApt - a multimodal conversational dialogue system in an apartment domain. En Proc. of International Conference on Spoken Language Processing (ICSLP'00).

[GLL99]Gustafson, J., N. Lindberg, y M. Lundeberg. 1999. The August spoken dialogue system. En Proc. of International Conference on Spoken Language Processing (Interspeech'99).

[GOOGLEMAPS] Google Maps. Disponible en: https://maps.google.com/ [Consultado el 01/07/2013]

[GRI07] Griol, David. Desarrollo y evaluación de Diferentes Metodologías para la Gestión Automática del Diálogo. Tesis Doctoral. Universidad Politécnica de Valencia, 2007.

[GRW97]Gorin, A.L., G. Riccardi, y J.H. Wright. 1997. How may I help you? En Speech Communication, volumen 23, páginas 113-127.

[INKML] *Ink Markup Language*. Disponible en: http://www.w3.org/TR/InkML/ [Consultado el 01/07/2013]

[JOC06]Juhar, J., S. Ondas, A. Cizmar, M. Rusko, G. Rozinaj, y R. Jarina. 2006. Development of Slovak GALAXY/VoiceXML Based Spoken Language Dialogue System to Retrieve Information from Internet. En Proc. of the 9th International Conference on Spoken Language Processing (Interspeech/ICSLP).

[JUP20] JUPITER: Zue, V., S. Sene , J. Glass, J. Polifroni, C. Pao, T. Hazen, y L. Hetherington. 2000. *"IEEE Transactions on Speech and Audio processing*", volumen 8(1), páginas 85-96.

[KIS02]Kearns, Michael, Charles Isbell, Satinder Singh, Diane Litman, y Jessica Howe. 2002. CobotDS: A Spoken Dialogue System for Chat. En Proc. of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002).

[LBR97]Lamel, L., S. Bennacef, S. Rosset, L. Devillers, S. Foukia, J. Gangolf, y J. Gauvain. 1997. The LIMSI RailTel System: Field trail of a telephone service for rail travel information.

[LDS04]Litman, Diane J. y Scott Silliman. 2004. ITSPOKE: An Intelligent Tutoring Spoken Dialogue System. En Proc. of the Human Language Technology Conference: 4th Meeting of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL).

[LGC97]Lau, Raymond, Giovanni Flammia, Christine Pao, y Victor Zue. 1997. WebGalaxy. Integrating spoken language and hypertext navigation. En Proc. of the International Conference on Spoken Language Processing (InterSpeech'97).

[LGD97]López-Cózar, R., P. García, J. Díaz, y A. J. Rubio. 1997. A voice activated dialogue system for fast-food restaurant applications. En Proc. of European Conference on Speech Communications and Technology (Eurospeech'97).

[LIP02]Litman, Diane J. y Shimei Pan. 2002. Designing and Evaluating an Adaptive Spoken Dialogue System. En User Modeling and User-Adapted Interaction.

[LISTEN]: Project LISTEN. Disponible en: http://www.cs.cmu.edu/~listen/ [Consultado el 01/07/2013]

[LLI06] LLISTERRI, J.- MACHUCA, M. J. Los sistemas de diálogo. Soria: Universitat Autònoma de Barcelona, Servei de Publicacions - Fundación Duques de Soria, 2006.

[LLIAA] Joaquin Llisterri, Grupo de Fonética, departamento de Filología Espanyola, Universitat Autónoma de Barcelona. Disponible en:

http://liceu.uab.es/~joaquim/speech_technology/tecnol_parla/dialogue/diálogo_general/Sistemas_diálogo.pdf [Consultado el 01/07/2013]

[LOP04] R. López-Cózar, R.Granell (2004). Sistemas de Diálogo Basado en VoiceXML para Proporcionar Información de Viajes en Tren. Revista: Procesamiento del Lenguaje Natural nº 33.

[LS04] ITSPOKE: Litman, Diane J. y Scott Silliman. 2004. *"Dialogue System. En Proc. of the Human Language Technology Conference: 4th Meeting of the North American Chapter of the Association for Computational Linguistics"* (HLT/NAACL), páginas 233236, Boston (Estados Unidos).

[LSK00]Litman, Diane, Satinder Singh, Michael Kearns, y Marilyn Walker. 2000. NJFun: a reinforcement learning spoken dialogue system. En Proceedings of the ANLP/NAACL 2000 Workshop on Conversational Systems.

[MADRID] Calles de Madrid. Disponible en: http://www.madrid.org/nomecalles/ListaCalles.icm [Consultado el 01/07/2013]

[MAS05] Ramón López-Cózar, Masahiro Araki. Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assesment", John Wiley & Sons, 2005.

[MBG96]Meng, H., S. Busayapongchai, J. Glass, D. Goddeau, L. Hetherington, E. Hurley, C. Pao, J. Polifroni, S. Seneff, y V. Zue. 1996. WHEELS: A Conversational System in the Automobile Classifieds Domain. En Proc. of International Conference on Spoken Language Processing (ICSLP'96).

[MHH04]Minker, W., U. Haiber, P. Heisterkamp, y S. Scheible. 2004. The Seneca Spoken Language Dialogue System. En Speech Communication. volumem 43, Issues 1–2, páginas 89–102.

[MICH04] Michael F. McTear. Spoken Dialogue Technology: Toward the Conversational User Interface", Springer, 2004.

[MKS01]Mäkelä, Kaj, Esa Pekka Salonen, Markku Turunen, Jaakko Hakulinen, y Roope Raisamo. 2001. Conducting a Wizard of Oz Experiment on a Ubiquitous Computing System Doorman. En Proc. of the International Workshop on Information Presentation and Natural Multimodal Dialogue.

[PEG94] Zue, V., S. Sene , J. Polifroni, M. Phillips, C. Pao, D. Goodine, D. Goddeau, y J. Glass.,1994. En "*Speech Communication*", volumen 15, páginas 331-340.

[PENATES] PENATES System. Disponible en: http://groups.csail.mit.edu/sls/technologies/ [Consultado el 01/07/2013]

[PLACESAPI] Google Places API. Disponible en: https://developers.google.com/maps/documentation/places/ [Consultado el 01/07/2013]

[SCC00]Seneff, S., C. Chuu, y D.S. Cyphers. 2000. Orion: From On-line Interaction to On-line Delegation. En Proc. of International Conference on Spoken Language Processing (ICSLP'00).

[SEC06]Skantze, G., J. Edlund, y R. Carlson. 2006. Talking with Higgins: Research challenges in a spoken dialogue system. En Proc. of Perception and Interactive Technologies (PIT'06).

[SEP20]Seneff, S. y J. Polifroni. 2000. Dialogue management in the Mercury sight reservation system. En Proc. of the 1st Conference of the North American Chapter of the Association for Computational Linguistics and 6th Conference on Applied Natural Language Processing (ANLP-NAACL 2000).

[SEP96] Seneff, S. y J. Polifroni. 1996. A New Restaurant Guide Conversational System: Issues in Rapid Prototyping for Specialized Domains. En Proc. of International Conference on Spoken Language Processing (ICSLP'96).

[SISR] Semantic Interpretation for Speech Recognition. Disponible en: http://www.w3.org/TR/semantic-interpretation/ [Consultado el 01/07/2013]

[SRGS] Speech Recognition Grammar. Disponible en: http://www.w3.org/TR/speech-grammar/ [Consultado el 01/07/2013]

[SSML] Speech Synthesis Markup Language. Disponible en: http://www.w3.org/TR/speech-synthesis/ [Consultado el 01/07/2013]

[TMH00]Turunen, Markku y Jaakko Hakulinen. 2000. Mailman - a Multilingual Speech-only E-mail Client Based on an Adaptive Speech Application Framework. En Proc. of Workshop on Multi-Lingual Speech Communication (MSC'00).

[VOXEOE] Voxeo Evolution. Disponible en: https://evolution.voxeo.com/ [Consultado el 01/07/2013]

[VoxeoHIW] Voxeo Evolution how it works. Disponible en: https://evolution.voxeo.com/account/about.jsp [Consultado el 01/07/2013]

[VOXEOGUIA] Guia Voxeo. Disponible en: http://evolution.voxeo.com/docs/quickStart.jsp [Consultado el 01/07/2013]

[VOY95] VOYAGER: Glass, J., G. Flammia, D. Goodine, M. Phillips, J. Polifroni, S. Sakai, S. Sene, y V. Zue. 1995. Multilingual spoken-language understanding in the MIT Voyager system. En Speech Communication, volumen 17, páginas 1-18.

[VXML10]: W3C VoiceXML 3.0. Disponible en: http://www.w3.org/TR/voicexml10/ [Consultado el 01/07/2013]

[VXML20] W3C VoiceXML 2.0. Disponible en: http://www.w3.org/TR/voicexml20/ [Consultado el 01/07/2013]

[VXML30]: W3C VoiceXML 3.0. Disponible en: http://www.w3.org/TR/voicexml30/ [Consultado el 01/07/2013]

[W3C1] Guía Breve de Interacción Multimodal - W3C. Disponible en: http://www.w3c.es/divulgacion/guiasbreves/Multimodalidad [Consultado el 01/07/2013]

[WDG97]Walker, Marilyn, Donald Hindle, Jeanne Fromer, Giuseppe Di Fabbrizio, y Craig Mestel. 1997. Evaluating Competing Agent Strategies for a Voice Email Agent. En Proc. of European Conference on Speech Communications and Technology (Eurospeech'97).