Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Metodología para el Desarrollo de Interfaces de Usuario Vocales

Tesis que presenta:

David Augusto Céspedes Hernández

Para obtener el título de:

Licenciatura en Ingeniería en Ciencias de la Computación

Asesores:

Dr. Juan Manuel González Calleros

Dra. Josefina Guerrero García

Jurado:

Dr. Manuel Isidro Martín Ortiz

Dr. Ivo Humberto Pineda Torres

Puebla, Pue. Julio 2013

Agradecimientos

Para la realización de este trabajo de tesis, agradezco a la Facultad de Ciencias de la Computación de la Benemérita Universidad Autónoma de Puebla por su valiosa participación en mi formación tanto en el ámbito profesional como en el personal. Así como a todos los profesores que compartieron conmigo sus conocimientos y me apoyaron para llegar al término de esta etapa de mi vida.

De manera particular, agradezco al Dr. Juan Manuel González Calleros y a la Dra. Josefina Guerrero García por ser mis asesores en este trabajo pero además por el apoyo que me han brindado en todo momento. Ustedes han sido parte muy importante de mis inicios en la investigación y deseo que podamos seguir colaborando y conviviendo ya que han pasado de ser mis profesores para convertirse también en amigos a los que estimo.

Al Dr. Manuel Martín Ortiz y al Dr. Ivo Pineda Torres, por aceptar ser parte del jurado en la presentación de mi examen profesional, por sus comentarios y correcciones a este trabajo y por apoyarme en esta etapa de mi vida profesional.

A mis padres, David Céspedes García y Patricia Hernández Rivera, por luchar siempre por darme lo mejor en todos los aspectos y por ser quienes siempre me han guiado y alentado para llegar más alto. A mi hermana Stephanny por estar siempre ahí, por soportar algunas veces mi luz encendida hasta tarde y el sonido de las teclas de la computadora que sé que no le gusta, ya pronto llegarás tu a esta etapa y espero poder compartir ese momento contigo.

A Liliana Rodríguez Vizzuett (Lili), por estar siempre a mi lado y apoyarme en todo momento incondicionalmente, porque este trabajo en gran medida fue impulsado por ti, por haber sido mi compañera en una buena parte de esta carrera y de mi vida, y sobre todo por soportar los momentos malos y compartir los buenos. Espero que podamos seguir cosechando el fruto del trabajo que hemos realizado hasta hoy y que también sigamos sembrando, porque el camino a penas comienza para nosotros y aún me falta escribir tu nombre en los agradecimientos de una tesis más.

A mi familia y amigos por haber creído siempre en mí y ayudarme a conseguir este primer logro, Muchas gracias.

Tabla de Contenido

Tabla de Figuras

Tabla de Tablas

1. Introducción

A lo largo de la historia de la computación, la manera en la que se lleva a cabo la interacción con sistemas informáticos ha evolucionado de acuerdo a los nuevos requisitos que surgen, por un lado al avance de la tecnología y por otro al progreso en los intereses por parte de los usuarios. Una Interfaz de Usuario (IU) es un conjunto de comandos o menús a través de los cuales un usuario se comunica con un programa o dispositivo [Ayob09].

En un inicio, la interacción se realizaba completamente por medio de la línea de comandos, después a través de menús desplegables, evolucionando hacia las IUs gráficas que son soportadas por el teclado y el mouse. Durante los últimos años, los esfuerzos se han encaminado a reemplazar la forma en la que se interactúa con los sistemas informáticos con la interacción natural soportada por Interfaces de Usuario Naturales (NUI por sus siglas en inglés) [Cama11]. Las NUIs son interfaces cuya utilización resulta natural para los usuarios, es decir, que pueden ser usadas sin recibir o con muy poco entrenamiento [Stei12], por ejemplo, el lenguaje utilizado para ordenar una pizza o los gestos para seleccionar una película en una SmartTV[1].

Un tipo especial de NUI es la interacción vocal, la cual ofrece sistemas cuya meta es emular el diálogo que una persona puede tener con otra. La importancia de las IUs vocales se enfatiza cuando se considera el contexto de uso de un sistema, por ejemplo, cuando el usuario no puede utilizar un sistema con interfaz gráfica por problemas de visión o por que se encuentre llevando a cabo alguna otra tarea al mismo tiempo como manejar un auto o realizar ejercicio físico.

En la década de los noventa, se elaboraron algunos sistemas Web soportados por comandos vocales, esto causó que se crearan lenguajes de programación que le dieran a los desarrolladores las herramientas para realizar implementaciones que interactúen de manera oral con los usuarios. Actualmente, algunas empresas e instituciones usan éste tipo de sistemas para dar información y para automatizar servicios como la realización de reservaciones para vuelos, ordenar comida y servicio al cliente.

Existen entonces, múltiples lenguajes para desarrollar sistemas informáticos con interacción de tipo vocal pero algunos son específicos a una plataforma o dispositivo, otros fueron creados como una solución para ambientes Web y existen otros más que dependen para su funcionamiento de la incorporación de diversas librerías y utilidades. Las características de estos lenguajes de programación dificultan la adquisición de experiencia en su uso y no permiten que se elaboren sistemas funcionales para todas las plataformas y contextos.

[1] La televisión inteligente (traducido del inglés "SmartTV") describe la integración de Internet y de las características Web 2.0 a la televisión digital (en especial, a la televisión 3D).

De esto último, se puede observar la necesidad de contar con una metodología, es decir, un método, una técnica y una herramienta para desarrollar sistemas de interacción vocal para múltiples plataformas sin necesidad de repetir en su totalidad el proceso de diseño y desarrollo de los mismos. La idea básica de esta metodología, consiste en que el desarrollador o diseñador de IUs vocales elabore modelos de las tareas que desea que el sistema realice para mediante procedimientos provistos por la misma, generar código en múltiples lenguajes de programación.

Una opción para el desarrollo de metodologías, es la propuesta por el Grupo de Administración de Objetos (OMG por sus siglas en inglés) (http://www.omg.org) bajo el concepto de arquitectura basada en modelos (MDA por sus siglas en inglés). MDA consiste en desarrollar aplicaciones y escribir especificaciones partiendo de modelos independientes de la plataforma y llevándolos a modelos específicos a una plataforma al tiempo que se realizan definiciones de IUs. Una aplicación MDA completa consiste en: un modelo independiente de la plataforma, uno o más modelos específicos a la plataforma e implementaciones completas, una para cada plataforma que el desarrollador de la aplicación desee utilizar.

El objetivo general del presente trabajo de tesis, es:

Elaborar una metodología que siga los principios del desarrollo basado en modelos para soportar la implementación de IUs vocales.

De este objetivo general, se desprenden como objetivos específicos los siguientes:

- Realizar un análisis de las herramientas, lenguajes, marcos de trabajo y plataformas existentes con el fin de que la metodología que se proponga sea extensible a diversos lenguajes, plataformas y dispositivos.
- Elaborar un modelo de interacción con base en los componentes que se utilizan en los diversos lenguajes con la finalidad de permitir el modelado de sistemas de interacción vocal en un nivel de abstracción ajeno a la plataforma final.
- Que la solución que se plantee considere el contexto y al usuario para permitir que los diseñadores o desarrolladores que hagan uso de esta metodología puedan realizar implementaciones centradas en el usuario.

Este trabajo de tesis se estructura en capítulos que a su vez contienen secciones. Este primer capítulo, la Introducción, da un breve panorama para entender conceptos importantes que se abordarán en los siguientes al tiempo que se definen los objetivos que se pretenden lograr.

El segundo capítulo presenta el estado del arte, que es el resultado de la investigación realizada con el fin de conocer los lenguajes, herramientas, marcos de trabajo y plataformas que existen para desarrollar IUs vocales. En el estado del arte además se realizan comparaciones entre los conceptos. La lectura de las secciones pertenecientes al capítulo 2, brindará una idea más clara de lo que se ha hecho hasta hoy en cuanto a la interacción vocal se refiere y de lo que queda aún por hacer.

En el tercer capítulo, se presenta la metodología para desarrollar IUs vocales, para esto, se eligen algunos lenguajes, herramientas y plataformas, así como un marco de trabajo y con base en éstos, se lleva a cabo un análisis que deriva en un modelo concreto de interacción vocal para desarrollo de interfaces. Así mismo, se proponen representaciones gráficas para los componentes que forman parte del modelo y se dan reglas para generar código a partir de modelos expresados en sus términos.

El capítulo cuatro contiene la validación realizada tanto para el modelo concreto de interacción vocal como para las representaciones gráficas de sus componentes y las reglas de transformación asociadas a los mismos. Para esto, se plantea una evaluación en la que partiendo del modelado de casos de estudio, se genera código para interacción vocal apoyándose en los aportes realizados por este trabajo de tesis.

En el capítulo 5 se dan las conclusiones y se muestran los resultados obtenidos después de la realización de este proyecto, se presentan los objetivos alcanzados, las principales aportaciones realizadas y se menciona el trabajo futuro.

2. Estado del Arte

2.1 Introducción

Éste capítulo presenta el resultado de la investigación de la literatura en el desarrollo de IUs, en especial las IUs vocales, y la descripción de un panorama general de los antecedentes a este trabajo. Se incluyen lenguajes de descripción de interfaces de usuario (UIDL), metodologías, técnicas y herramientas cuya comprensión resulta imprescindible para el desarrollo de este proyecto, además de ser necesarios para el establecimiento de una metodología como la que se desea proponer bajo el enfoque MDA descrito en el capítulo anterior.

Los conceptos incluidos en éste capítulo se presentan en secciones, como se muestra en la Figura 2.1, en las cuales se hacen descripciones generales para posteriormente proporcionar detalles sobre temas específicos. En la sección 2.2 se mencionan y describenmarcos de trabajo teóricos que se consideran de importancia para este trabajo ya que permiten manejar distintos tipos de interacción entre el usuario y el sistema. En la sección 2.3 se realiza la exposición de UIDLs. Un análisis de las herramientas de desarrollo de interfaces de usuario se presenta en la sección 2.4 con el fin de determinar cuáles de éstas contienen componentes que puedan resultar de utilidad para este trabajo. Finalmente la sección 2.5 presenta un resumen del capítulo con el fin de establecer las ventajas y desventajas de las diferentes herramientas de desarrollo y de los lenguajes analizados y así, poder seleccionar aquello que permita alcanzar los objetivos de este trabajo, mismos que se presentaron en el Capítulo 1.

2.2 Marcos de Trabajo Teóricos

En esta sección se presentan las características de los marcos de trabajo que se consideran importantes para el desarrollo de este proyecto, ya que permiten el manejo de distintos tipos de interacción (modalidades) entre los usuarios y los sistemas, específicamente el manejo de interacción vocal.

2.2.1 W3C Multimodal Interaction Framework

El marco de trabajo de interacción multimodal [W3C03] tiene como objetivo identificar y relacionar lenguajes de etiquetado para sistemas de interacción multimodal. Descrito de manera simple, el marco de trabajo, analiza los sistemas multimodales, identificando las partes principales de éstos. La Figura 2.2 muestra la estructura de éste marco de trabajo. Cada una de las partes o elementos, representa una serie de funciones relacionadas. Posteriormente, se determinan los lenguajes de etiquetado utilizados para representar la información requerida por los componentes y para permitir el flujo de datos entre ellos.

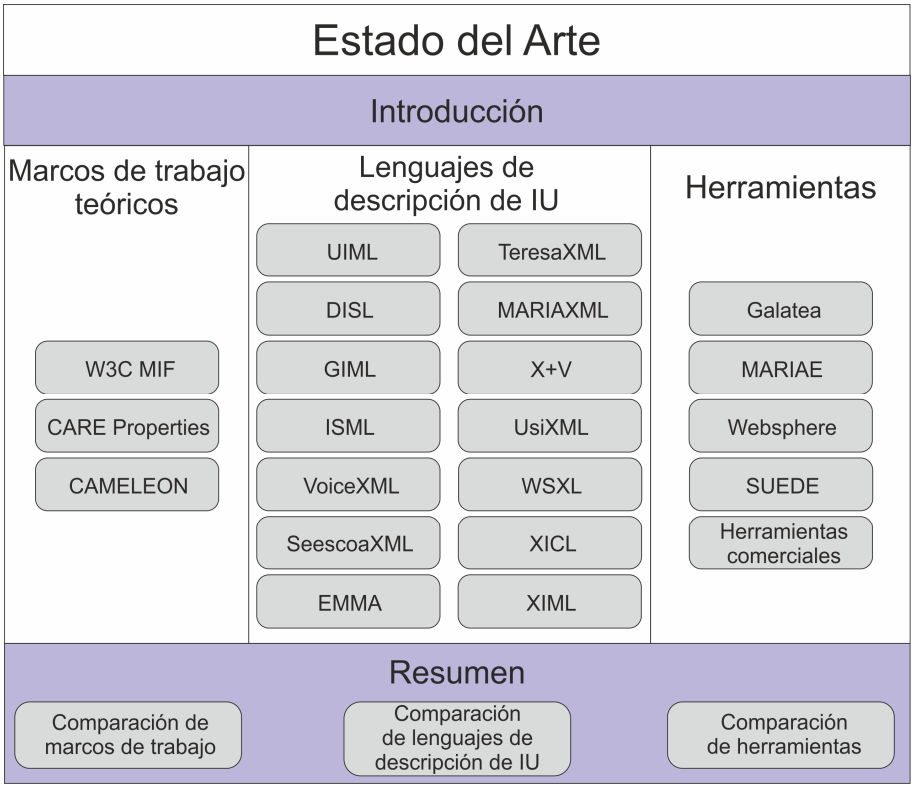

Figura 2.1 Estructura del Estado del Arte.

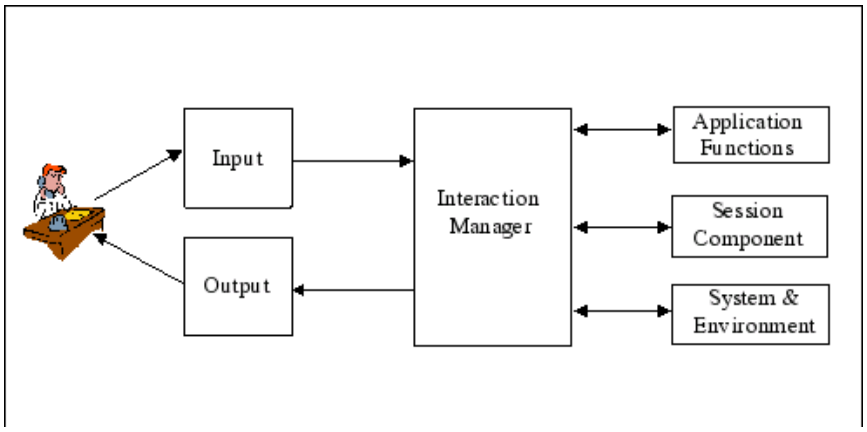

Figura 2.2 Elementos que conforman el marco de trabajo de W3C.

Los componentes básicos que el marco de trabajo identifica en una interacción multimodal son:

- Usuario: Es aquel que ingresa datos a un sistema y obtiene la información que éste proporciona, es decir, realiza una interacción con él. El usuario puede, o no, ser humano, ya que se pueden utilizar autómatas para hacer pruebas de sistemas.
- Entradas: En un sistema multimodal se consideran varios tipos de entradas tales como audio o voz, escritura a mano y desde teclado, etc. Las entradas vocales son las de interés para el desarrollo del proyecto.
- Salidas: Un sistema multimodal se apoya en uno o varios modos de presentación de la información tales como voz, imágenes, animación y texto por mencionar algunos. De igual manera que con las entradas, de los tipos de salida con los que se cuenta interesan para nuestro fin las vocales.
- Manejador de interacción: El manejador de interacción tiene como función regular el intercambio de información entre el usuario y las funciones de aplicación. Debe dar soporte a varios estilos de interacción, entre ellos:
    - Diálogo dirigido por el sistema – El sistema formula una pregunta al usuario y éste le responde.
    - Diálogo dirigido por el usuario – El usuario guía a la computadora a realizar una acción y ésta le responde mostrándole los resultados de dicha acción.
    - Diálogo mixto – Es una mezcla de diálogo dirigido por el sistema y de diálogo dirigido por el usuario, en la cual, el usuario y el sistema toman por turnos el control del diálogo.
- Funciones de aplicación: Varias aplicaciones contienen funciones que realizan operaciones de acceso a bases de datos, procesamiento de transacciones y cálculos, la forma específica en que éstas se realizan es dependiente de la aplicación y por lo tanto no es objeto de estudio de éste marco de trabajo.
- Componente de sesión: Provee una interfaz al manejador de interacción para el manejo de estados, así como sesiones temporales y permanentes en aplicaciones multimodales.
- El sistema y el entorno: Permite al manejador de interacción notar y responder a cambios en las capacidades de los dispositivos, preferencias del usuario y condiciones del entorno.

2.2.2 CARE Properties

Las propiedades CARE(Complementariedad, Asignación, Redundancia y Equivalencia) [Cout95], ofrecen una manera formal para la definición y caracterización de aspectos en las IUs multimodales. Para entender las propiedades CARE, se deben tener en cuenta las siguientes definiciones y nociones:

- Goal (Meta) es un estado al que un agente intenta llegar aplicando expresiones. Una expresión de entrada es producida por el usuario que desea alcanzar una meta particular, dicha expresión es procesada por el sistema, y éste a su vez produce una expresión de salida.

- State (Estado) es un conjunto de propiedades que pueden ser medidas o determinadas para describir una situación en un tiempo en particular.
- Agent (Agente) es una entidad capaz de realizar acciones.
- Modality (Modalidad) es el método de interacción que un agente utiliza para llegar a su meta. Se describe en términos de un dispositivo físico *d* y un lenguaje de interacción *L*: <*d, L*>.
- Temporal window (Ventana temporal) es un intervalo de tiempo que se usa como restricción para la producción de expresiones de entrada y salida.
- Temporal relationship (Relación temporal) representa el uso de un conjunto de modalidades en cuanto al tiempo. El uso de esas modalidades puede ocurrir de manera simultánea o en secuencia dentro de una ventana temporal.
- Las modalidades de un conjunto M se utilizan de manera simultánea o paralela (Parallel), si dentro de una ventana temporal, parecen estar activas al mismo tiempo. De manera formal, si se tiene que *Active(m, t)* es un predicado que expresa que la modalidad *m* se utiliza en un momento *t*, el uso simultáneo de las modalidades de un conjunto *M* sobre una ventana temporal finita *tw* se describe de la siguiente manera:

$$Parallel(M, tw) \leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \infty) \wedge (\exists t \in tw, \forall m \in M, Active(m, t))$$

  Donde *Card(M)* es el número de modalidades en el conjunto *M* y *Duration(tw)* es la duración del intervalo de tiempo *tw*.
- Las Modalidades de *M* son usadas de manera secuencial (Sequential) en una ventana temporal *tw* si hay como máximo una modalidad activa en un momento determinado, y si todas las modalidades del conjunto se utilizan en tw:

$$Sequential(M, tw) \leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \infty) \wedge (\forall t \in tw, (\forall m, m' \in M, Active(m, t) \rightarrow \neg Active(m', t)) \wedge (\forall m \in M, \exists t \in tw, Active(m, t))$$

- La función *Reach(s, m, s')* modela la potencia expresiva de una modalidad m, lo cual es, la capacidad de m de permitir a un agente llegar al estado s' desde el estado s en un paso. A la sucesión de pasos o estados se le llama trayectoria de interacción.

Una vez que estos conceptos se han mencionado, se pueden especificar las siguientes definiciones:

(1) Complementariedad(Complementarity). Las modalidades de un conjunto M son usadas de forma complementaria para alcanzar el estado s' desde el estado s dentro de una ventana temporal, si todas ellas deben ser utilizadas para alcanzar dicho estado, es decir, si ninguna de éstas de manera independiente puede lograrlo. De manera formal, la complementariedad se define como se muestra a continuación:

$$Complementarity\ (s, M, s', tw) \Leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \infty) \wedge$$

$$(\forall M' \in PM\ (M' \neq M \Rightarrow \neg REACH(s, M', s'))) \wedge REACH(s, M, s') \wedge (Sequential(M, tw) \vee Parallel(M, tw)).$$

Donde *PM* denota las partes del conjunto *M*.

La complementariedad puede ocurrir tanto de manera simultánea como secuencial dentro de una ventana temporal, estos tipos de complementariedad se conocen como *complementariedad sinérgica* (Synergistic Complementarity) y *complementariedad alterna* (Alternate Complementarity) respectivamente. La definición formal para ambas clases de complementariedad se muestra a continuación:

$$Synergistic\text{-}Complementarity(s, M, s', tw) \leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \infty) \wedge (\forall M' \in PM, (M' \neq M \rightarrow \neg Reach(s, M', s'))) \wedge Reach(s, M, s') \wedge Parallel(M, tw)$$

$$Alternate\text{-}Complementarity(s, M, s', tw) \leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \infty) \wedge (\forall M' \in PM, (M' \neq M \rightarrow \neg Reach(s, M', s'))) \wedge Reach(s, M, s') \wedge Sequential(M, tw)$$

(2) Asignación (Assignment). Se dice que una modalidad *m* tiene asignado el alcanzar un estado s' desde el estado s, si ninguna otra modalidad es utilizada para dicho propósito:

$$Assignment(s, m, s') \Leftrightarrow Reach(s, m, s') \wedge (\forall m' \in M.\ Reach(s, m', s') \Rightarrow m' = m)$$

La asignación expresa que no existen opciones, es decir, que no se puede elegir otra modalidad para llegar de un estado a otro o que si hay opciones pero que el agente siempre opta por la misma modalidad para llegar al estado siguiente. De esto se desprende que hay dos tipos de asignación: *Asignación estricta* (StrictAssignment) y *Asignación por agente* (Agent Assingment), las cuales se expresan de la siguiente manera:

$$StrictAssignment(s, m, s') \leftrightarrow Reach(s, m, s') \wedge (\forall m' \in M, Reach(s, m', s') \rightarrow m' = m)$$

$$AgentAssignment(s, m, M, s') \Leftrightarrow (Card(M) > 1) \wedge (\forall m' \in M, (Reach(s, m', s') \wedge (Pick(s, m', s')) \Rightarrow m' = m))$$

Donde *Pick(s, m, s')* representa la selección de una modalidad *m* de un conjunto de modalidades para alcanzar s' a partir del estado s.

(3) Redundancia (Redundancy). Las modalidades de un conjunto *M* se utilizan de manera redundante para alcanzar un estado s' desde el estado s, si son equivalentes y se usan dentro de la misma ventana temporal *tw*. Dicho de otra forma, cuando el usuario muestra un comportamiento repetitivo:

$$Redundancy(s,M,s',tw) \leftrightarrow Equivalence(s,M,s') \wedge (Sequential(M,tw) \vee Parallel(M, tw))$$

En esta propiedad, se pueden encontrar ambos tipos de relaciones temporales (Secuencialidad y paralelismo), las cuáles tienen implicaciones particulares en la usabilidad y la implementación. El paralelismo coloca restricciones en los tipos de modalidad que pueden ser usados de manera simultánea: Las modalidades que compiten por un agente no pueden ser activadas de forma paralela. En este caso, el agente debe actuar de manera secuencial, cuidando cumplir con las restricciones de tiempo. Cuando el paralelismo se permite, se tiene *redundancia concurrente* (Concurrent redundancy) y se tiene *redundancia exclusiva* (Exclusive redundancy) cuando el comportamiento es secuencial:

$$Concurrent\text{-}Redundancy(s, M, s', tw) \leftrightarrow Equivalence(s, M, s') \nwedge Parallel(M, tw)$$

$$Exclusive\text{-}Redundancy(s, M, s', tw) \Leftrightarrow Equivalence(s, M, s') \wedge Sequential\ (M, tw)$$

(4) Equivalencia (Equivalence). Las modalidades en un conjunto *M* son equivalentes para alcanzar un estado s' desde un estado s, si es necesaria y suficiente la aplicación de cualquiera de ellas. Se asume que *M* contiene al menos dos modalidades. La descripción formal de ésta propiedad se muestra a continuación:

$$Equivalence(s, M, s') \Leftrightarrow (Card(M) > 1) \wedge (\forall m \in M, Reach\ (s, m, s'))$$

La equivalencia expresa la posibilidad de elegir entre varias modalidades sin imponer alguna restricción temporal entre ellas.

2.2.3 CAMELEON Reference Framework

El marco de trabajo teórico CAMELEON (Context Aware Modelling for Enabling and Leveraging Effective interactiON o Modelado centrado en el contexto para permitir y aprovechar la interacción efectiva) [Calv03] tiene como finalidad la construcción de métodos que soporten el diseño y desarrollo de sistemas de software sensibles al contexto altamente usables.

Este marco de trabajo se basa en dos principios clave:

- Acercamiento al desarrollo basado en modelos
- Cobertura de las fases de diseño y de tiempo de ejecución de una interfaz de usuario multiplataforma.

En la Figura 2.3 se muestra de manera resumida la estructura que sigue CAMELEON para la definición de interfaces de usuario y para la transformación de las mismas desde diferentes niveles de abstracción.

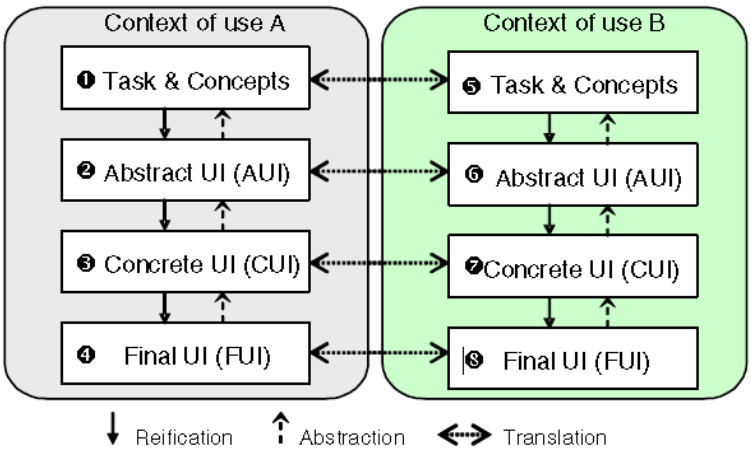

Figura 2.3 Marco de trabajo CAMELEON para IUs multiplataforma.

De esta estructura, se pueden observar cuatro elementos importantes para la definición de una IU:

1. Las tareas y los conceptos (T&C): describen las acciones a realizar por el usuario; son conceptos requeridos por los modelos de datos para la realización de las tareas.
2. Interfaz de Usuario Abstracta (AUI- Abstract User Interface): define contenedores abstractos y compontentes individuales de interacción. Las tareas son asociadas a contenedores para su ejecución o a componentes individuales para su manipulación. Una AUI es una abstracción de una IU Concreta con respecto a la modalidad de interacción. En este nivel, la IU se compone principalmente por la definición de entradas y salidas del sistema pero no define la modalidad que será utilizada (vocal, gráfica, táctil).
3. Interfaz de Usuario Concreta (CUI- Concrete User Interface): define la modalidad de interacción y se compone de objetos que la describen, Objetos Concretos de Interacción (CIOs- Concrete Interaction Objects) para la definición de los *widgets* de diseño y la navegación por la interfaz. La CUI es independiente de la plataforma de cómputo y aunque hace explícito el aspecto y comportamiento de la IU, es sólo un modelo para un ambiente en particular. Una CUI puede ser considerada como la concretización de una AUI y como una abstracción de la IU Final con respecto a la plataforma de cómputo.
4. Interfaz de Usuario Final (FUI- Final User Interface): corresponde a los elementos operacionales, es decir, la IU de usuario implementada para una plataforma de cómputo determinada.

Éste marco de trabajo se considera de utilidad para el desarrollo del presente trabajo ya que contempla la posibilidad de transformar una interfaz creada de manera específica para una plataforma en una interfaz para una distinta plataforma realizando procesos de abstracción y aplicando después transformaciones en cualquiera de los niveles mostrados en la Figura 2.3

que al concretizarse nuevamente da como resultado una nueva implementación para una plataforma distinta.

2.3 Lenguajes de Descripción de Interfaces de Usuario

Un UIDL es un lenguaje de especificación que describe varios aspectos de una interfaz de usuario en desarrollo. En la literatura se ha realizado un análisis minucioso sobre los principales UIDLs identificando las ventajas de éstos y comparando sus características de acuerdo con el fin que se desea alcanzar. Una revisión en extenso puede encontrarse en [Guer09]. A continuación se muestran, de manera resumida, características de UIDLs resaltando al final aquellas que resultan de utilidad para este trabajo.

2.3.1 User Interface Markup Language (UIML)

El lenguaje de etiquetado para interfaces de usuario [Abra04] es un meta-lenguaje compatible con XML (eXtensible Markup Language) [W3C11] para la descripción de interfaces de usuario. UIML permite implementar interfaces de usuario para cualquier dispositivo sin la necesidad de aprender lenguajes o interfaces para programación de aplicaciones (APIs) específicos al dispositivo. UIML pretende también reducir el tiempo de desarrollo de IUs para una familia de dispositivos.

Éste lenguaje provee una separación entre el código de la IU y el de la lógica de la aplicación permitiendo un prototipado rápido de interfaces entre otras características.

UIML se basa en 4 principios clave:

1. Es un meta-lenguaje: UIML define un conjunto pequeño de etiquetas independientes tanto a la modalidad como a la plataforma y al lenguaje. La especificación de una interfaz de usuario se realiza a través de un vocabulario que establece un conjunto de clases y de propiedades.
2. Separa los elementos de una IU e identifica: las partes que componen la IU en sí, el contenido de cada una de las partes, el comportamiento de esas partes expresado como un conjunto de reglas con condiciones y acciones, así como la definición del vocabulario de las clases que les corresponden.
3. Agrupa lógicamente la IU en un árbol de partes que cambian con respecto al tiempo de vida de la interfaz. Durante éste tiempo de vida, el árbol inicialmente definido sufre modificaciones de manera dinámica cambiando su forma y agregando o eliminando partes. Se proveen elementos para describir la estructura inicial del árbol así como las modificaciones que sufrirá de manera dinámica en su estructura.
4. Permite empaquetar las partes y los árboles de manera que puedan ser reutilizados en otros diseños de interfaz.

2.3.2 Dialog and Interface Specification Language (DISL)

El Lenguaje de Especificación de Diálogo e Interfaz [Scha06] consiste en un lenguaje que permite descripciones de diálogo independientes de la plataforma. DISL hereda la mayoría de su estructura del UIML. Las modificaciones que se realizan a UIML para obtener DISL se ven principalmente en la incorporación de widgets genéricos así como en cambios realizados en la sección comportamental.

La relevancia que DISL representa para este trabajo reside en el hecho de que al implementar widgets genéricos, se permite asignar a cada uno de los widgets con un tipo de funcionalidad (comando, campo de texto, etc.), facilitando así que un motor de renderización utilice esta información para crear los componentes apropiados para la modalidad en la que dicho widget trabajará.

La estrucutra de DISL consiste de un componente opcional *head* para meta información y una colección de plantillas e interfaces desde la que una interfaz se considera activa en un momento dado. Las interfaces son usadas para describir estructuras de diálogo, estilos y comportamientos mientras que las plantillas sólo describen estructuras y estilos de manera que sean utilizables para otros componentes de diálogo.

2.3.3 Generalized Interface Markup Language (GIML)

El lenguaje de etiquetado de interfaz generalizada [Kost04] es un lenguaje utilizado por el paquete de herramientas Generalized Interface ToolKit (GITK) [http://gitk.sourceforge.net/] en ese contexto como un descriptor de interfaces. GIML al igual que UIML separa la funcionalidad y la presentación. Todos los objetos de interfaz en GIML son identificados por un tipo. La estructura de estos tipos es de carácter funcional y da a los mismos nombres funcionales. Entonces, un botón se llama “acción” permitiendo y facilitando así la reutilización de interfaces en distintas modalidades (Un botón no tendría sentido en un contexto vocal).

2.3.4 Interface Specification Meta-Language (ISML)

El meta lenguaje de especificación de interfaz [Crow03] fue desarrollado con el propósito de hacer explícito el diseño de los componentes de interacción de la interfaz así como el de los conceptos compartidos entre el usuario y la computadora.

ISML descompone el modelo de componentes de la interfaz de una implementación y expresa conexiones entre los conceptos compartidos entre el usuario y el sistema. Para que esto sea útil, ISML provee un marco de trabajo que soporta mapeo entre los modelos orientados al usuario y los conceptos de arquitectura de software.

2.3.5 VoiceXML

VoiceXML [W3C10] es un lenguaje estandarizado que permite la interacción vocal y es utilizado en aplicaciones industriales de manera común. Su meta principal es reforzar las

aplicaciones de voz con el desarrollo Web así como facilitar a los desarrolladores la elaboración de aplicaciones al trabajar en un nivel más alto y con menos restricciones en cuanto al manejo de recursos se refiere. Además de permitir la integración de servicios de voz y datos utilizando el paradigma cliente-servidor.

VoiceXML reduce las interacciones entre cliente y servidor al especificar interacciones múltiples por documento. Oculta a los desarrolladores detalles de programación de bajo nivel y aquellos específicos de la plataforma. La última versión de este lenguaje hasta la fecha, ha solucionado problemas de extensión y funcionalidad y será tomada como referencia en el siguiente capítulo al identificar los conceptos para la definición del modelo de soporte al proyecto.

En la literatura se pueden encontrar otras aplicaciones para éste lenguaje [Davi05] en las que se demuestra que no sólo tiene utilidad al definir IUs vocales sino para otras modalidades también.

2.3.6 Software Engineering for Embedded Systems using a Component-Oriented Approach (SeescoaXML)

SeescoaXML [Luyt04] consiste de una serie de modelos y mecanismos para producir distintas IUs en tiempo de ejecución para distintas plataformas y de manera independiente a la modalidad. Además, SeescoaXML es sensible al contexto de manera que se expresa inicialmente en una forma independiente a la modalidad y después se conecta a una especificación para cada plataforma. Para este caso, la sensibilidad al contexto se observa en la posibilidad de variación en cuanto a la plataforma de computo se refiere. Se obtiene entonces una IU abstracta, la cual contiene especificaciones para los diferentes mecanismos de renderización y los comportamientos relacionados a ellos. Dichas especificaciones se elaboran en un UIDL compatible con XML y se transforman entonces mediante reglas XSLT en especificaciones para la plataforma.

2.3.7 Extensible MultiModal Annotation markup language (EMMA)

El lenguaje de etiquetado extensible de anotación multimodal [W3C04] es un lenguaje de etiquetado desarrollado con el fin principal de ser utilizado en sistemas que provean interpretación semántica de diversas entradas incluyendo: vocal, texto, gráficos, etc.

Éste lenguaje es usado, además, para contener y anotar información extraída automáticamente de la entrada independientemente de la modalidad de ésta. EMMA es capaz de dar un significado a distintos tipos de entrada como las mencionadas en el párrafo anterior y las combinaciones entre ellas. Esto le hace compatible con el marco de trabajo W3C descrito en la sección 2.2.1 y por tanto con los lenguajes que de éste se han derivado.

El hecho de ser compatible con el marco de trabajo W3C, permite que EMMA también se pueda utilizar como un formato estándar para el intercambio de datos entre los componentes de un sistema multi-modal.

2.3.8 TeresaXML

TeresaXML [Bert03] es un UIDL para producir múltiples IUs para múltiples plataformas en tiempo de diseño. TeresaXML sugiere iniciar el diseño de un sistema con el modelado de tareas, para después identificar las especificaciones de la interfaz abstracta en términos de su estructura estática y comportamiento dinámico. Dichas especificaciones abstractas conducen a la implementación final. El cambio entre un contexto y otro para este sistema sólo puede permitirse en el nivel de abstracción más alto del marco de trabajo CAMELEON (Tareas y conceptos) mostrado en la Figura 2.3, permitiendo máxima flexibilidad en cuanto a las restricciones propias del contexto de uso.

TeresaXML tiene como soporte una herramienta de software para el desarrollo de IUs: TERESA [Bert03].

2.3.9 MARIAXML

MARIAXML [Pate09] es el UIDL sucesor de TeresaXML conservando sus ventajas y añadiendo soporte a los comportamientos dinámicos, eventos, aplicaciones de Internet, IUs multi-plataforma y en particular aquello que se relacione con servicios Web. MARIAXML es además compatible con el marco de trabajo CAMELEON (sección 2.2.3).

Este UIDL tiene como soporte una herramienta de software para el desarrollo de IUs: MARIAE la cual será revisada en la sección 2.4.

2.3.10 XHTML+Voice (X+V)

X+V [W3C01] es un lenguaje basado en Web estandarizado, permite la combinación de la interacción gráfica con la vocal y la táctil. Este lenguaje se basa en XHTML, el cual soporta la interacción gráfica, un subconjunto de VoiceXML para la interacción vocal y XML para sincronizar ambas partes. La posibilidad de elegir entre los distintos tipos de interacción les permite a los usuarios elegir aquella que resulte más conveniente para la tarea que se desea realizar tomando en cuenta el contexto de uso. Esta característica de elegir la modalidad que se desea utilizar se debe a que X+V sigue algunas de las propiedades CARE descritas en la sección 2.2.2.

La especificación que es resultado de la utilización de este lenguaje, consiste de: (1) elementos gráficos especificando la presentación y el comportamiento de la interfaz gráfica, (2) elementos vocales especificando el intercambio de información vocal entre usuario y sistema, y (3) elementos de sincronización entre ambos. El navegador multimodal se encargará de realizar la interpretación de éstos elementos de manera separada.

2.3.11 User Interface eXtensible Markup Language (UsiXML)

UsiXML [Vand04] es un lenguaje que permite a los diseñadores aplicar desarrollo multidireccional de IUs en diferentes niveles de independencia. Así, una interfaz puede ser especificada desde diferentes o incluso múltiples niveles de abstracción manteniendo la relación con otros niveles si es requerido. De esta manera, se puede comenzar el proceso de diseño desde cualquier nivel y llegar a varias implementaciones finales para varios contextos de uso.

UsiXML se caracteriza por los siguientes principios:

1. Expresividad de IUs: cualquier IU es expresada dependiendo del contexto de uso gracias a modelos analizables, editables y manipulables vía software.
2. Almacenamiento central de modelos: cada modelo se almacena en un repositorio donde todos los modelos son expresados de acuerdo al mismo UIDL.
3. Enfoque transformacional: cada modelo almacenado en el repositorio de modelos puede ser sujeto a una o más transformaciones.
4. Rutas de desarrollo múltiples: Los pasos de desarrollo que se sigan pueden ser combinados para formar rutas de desarrollo compatibles con restricciones, convenciones y contextos de uso.
5. Enfoques de desarrollo flexibles: los enfoques de desarrollo son soportados al seguir de manera flexible pasos de desarrollo y permitir a los diseñadores cambiar libremente esos destinos dependiendo de cambios impuestos por el contexto de uso.

UsiXML está estructurado de acuerdo a los distintos niveles de abstracción definidos por el marco de trabajo CAMELEON, de manera que el modelo de tareas y conceptos es independiente de la tecnología computacional, el nivel abstracto es independiente de la modalidad y el nivel concreto es independiente del *toolkit*. Este lenguaje cuenta con una serie de herramientas que le dan soporte: Editores, generadores de código e intérpretes (http://www.usixml.eu/).

2.3.12 Web Service eXperience Language (WSXL)

WSXL [Arsa02] es un lenguaje diseñado para representar datos, presentación y control en un sistema. Este lenguaje se basa en otros estandarizados y compatibles con XML, además de apoyarse en servicios Web.

El principal objetivo de este lenguaje es facilitar y reducir los costos del desarrollo de aplicaciones Web, así como permitir la realización de cambios en sistemas ya implementados de manera que se ajusten a nuevos requerimientos.

2.3.13 eXtensible user-Interface Markup Language (XICL)

XICL [Gome04] conforma una manera fácil para elaborar IUs para sistemas soportados por un navegador. Componentes para la IU son creados a partir de componentes HTML y de otros propios del lenguaje.

Las descripciones realizadas para las interfaces en este lenguaje, se traducen a código DHTML. Un documento XICL consta de una descripción de IU que a su vez se forma a partir de elementos HTML y XICL así como otros componentes (estructura, propiedades, eventos y métodos). XICL trabaja en un nivel de abstracción mayor al de DHTML y permite la reutilización y extensión de los componentes de las IUs.

2.3.14 eXtensible Interface Makup Language (XIML)

XIML [Puer02] provee dentro del ciclo de desarrollo un nivel independiente de la modalidad desde el cual las implementaciones pueden ser obtenidas en distintos lenguajes. El objetivo principal de este lenguaje es facilitar un marco de trabajo para la definición y relación de los datos de interacción, los cuales son aquellos que definen y unen a los componentes de la IU. XIML utiliza las siguientes unidades de representación:

- Componentes: De manera básica, XIML se puede considerar una colección organizada de elementos de interfaz categorizados en uno o más componentes principales. Dichos componentes son aquellos que se encuentran de manera típica en un modelo: tareas, objetos del dominio, tipos de usuario, elementos de presentación y elementos de diálogo.
- Relaciones: Una relación en XIML es una definición que une elementos XIML en o entre componentes. Haciendo uso de estas relaciones, XIML permite la creación de un cuerpo (corpus) que puede soportar funciones para diseño basado en conocimiento, operaciones y evaluaciones.
- Atributos: Los atributos en XIML son características a las que se les puede asignar un valor.

XIML permite la creación de IUs que deben ser visualizadas en múltiples plataformas. Esto se debe a la separación que el lenguaje realiza entre la definición de la IU y la renderización de la misma. Existe una serie de herramientas que permiten trabajar de manera sencilla con XIML (Editor, visor, validador, etc.) así como otras que se encargan de transformar las definiciones realizadas en este lenguaje a lenguajes específicos para la realización de implementaciones (como HTML o WML).

2.4 Herramientas para Desarrollo de Interfaces de Usuario

Una herramienta para desarrollar IUs es aquella que puede utilizar objetos de un repositorio o elementos de IU con el fin de elaborar IUs. Uno o más de estos elementos pueden ser reutilizados para definir nuevas interfaces. A los elementos de una interfaz se les puede asociar un comportamiento (por ejemplo: indicar la forma en la que un elemento aparece dependiendo de las características del dispositivo en que se mostrará).

Diversas herramientas han sido elaboradas para facilitar el desarrollo de IUs, algunas de ellas hacen uso de modelos y generan código para múltiples plataformas. La importancia del análisis de ellas para este trabajo de tesis reside en el hecho de que una herramienta de software brinda soporte a una metodología como la que se desea establecer, de tal manera que al observar las características de las herramientas existentes se pueden tomar las medidas necesarias al estudiar y proponer los conceptos que serán la base para la nueva herramienta que se desee elaborar o aquella a actualizar. A continuación se mencionan algunas herramientas de desarrollo de IUs junto con sus características más relevantes.

2.4.1 Galatea Interaction Builder

El entorno de desarrollo Galatea [Kawa02] es una herramienta para prototipado rápido que soporta al lenguaje XISL. Puede trabajar con modalidades de entrada como: voz, manipulación directa y lenguaje natural escrito; así como las modalidades de salida: voz, expresión facial y gráfica.

La motivación principal en la realización de esta herramienta está en permitir el desarrollo de agentes que interactuén de manera vocal con los usuarios, dando a éstos la impresión de estar interactuando con otro humano. El *interaction builder* cuenta con una interfaz gráfica (Figura 2.4) que permite el diseño de la interfaz a la vez que facilita la personalización e inclusión de elementos a la misma. VoiceXML es tomado como referencia para esta herramienta, sin emabargo se le realizan extensiones, ya que los fines en cuanto al reconocimiento de voz sobrepasan las capacidades ofrecidas por el lenguaje.

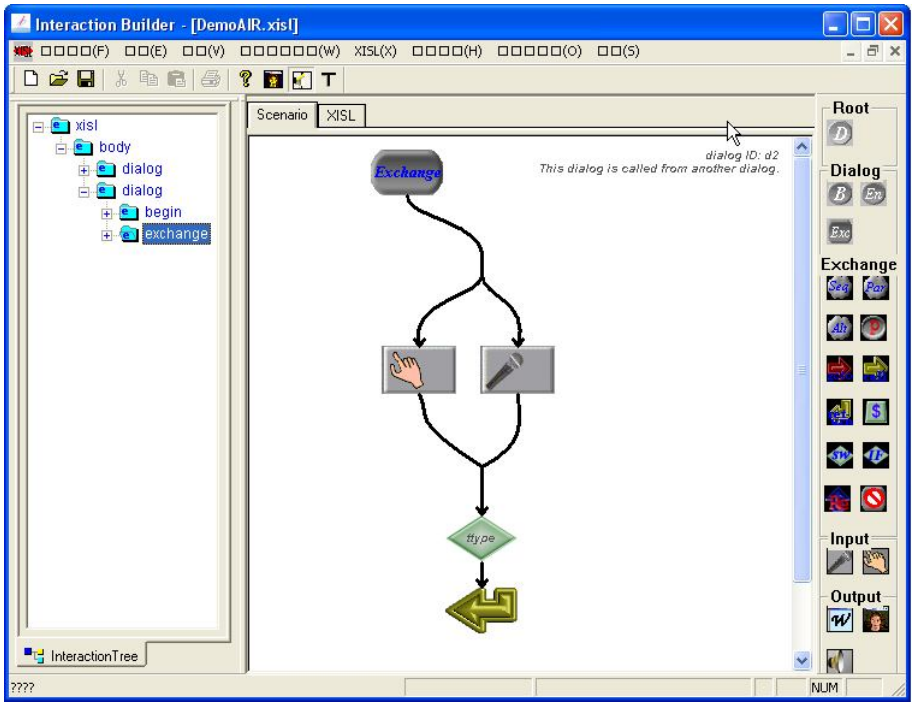

Figura 2.4 Interfaz gráfica de Galatea Interaction Builder.

2.4.2 MARIAE

El entorno de desarrollo MARIAE [Pate09] (Figura 2.5) provee una solución para diseñar y desarrollar interfaces multimodales para aplicaciones interactivas basadas en servicios Web para diversos tipos de plataforma a partir de modelos de tareas (siguiendo la notación ConcurTaskTrees [Pate03]) y modelos de interfaz de usuario realizados en el lenguaje MARIAXML.

La herramienta es capaz de importar de manera automática descripciones de servicios y anotaciones y de soportar la asociación interactiva entre sistemas de tareas básicas y servicios Web.

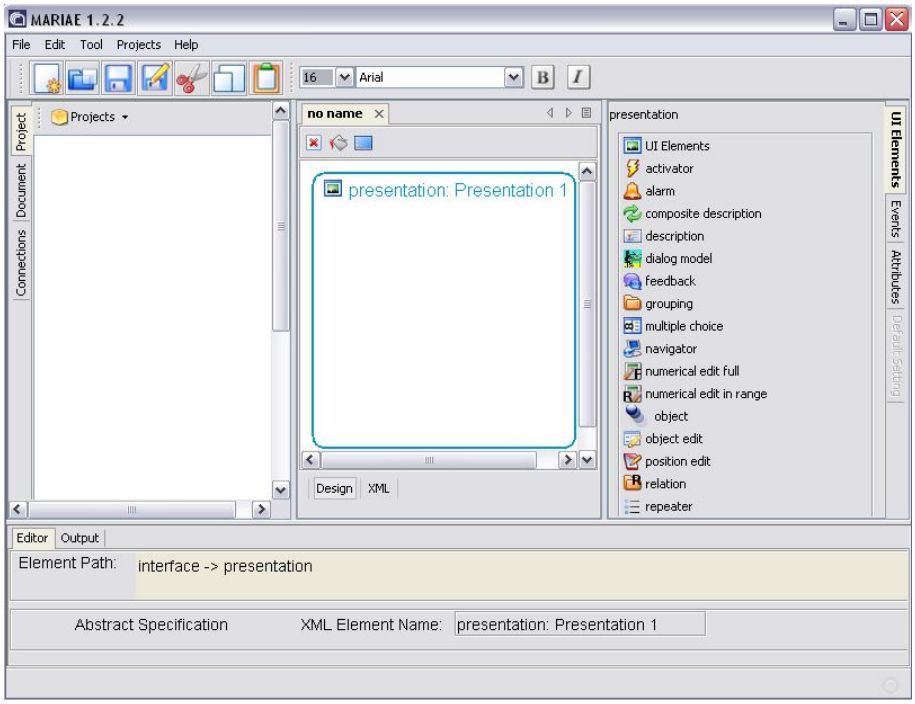

Figura 2.5 Interfaz gráfica del entorno de desarrollo MARIAE.

2.4.3 WebSphere Voice Toolkit

El *toolkit* WebSphere para voz [IBM05] soporta el lenguaje VoiceXML y ofrece una amplia gama de elementos para desarrollar aplicaciones de voz. Para utilizar WebSphere no se requiere tener un alto nivel de conocimiento en la utilización de algún lenguaje que soporte la interacción vocal. Además incluye una interfaz gráfica (Figura 2.6) en la que el desarrollador puede realizar el diseño de un diagrama de flujo del sistema a crear, a partir del cual WebSphere realiza transformaciones para generar código. Las aplicaciones elaboradas con éste *toolkit* son diseñadas para trabajar por vía telefónica.

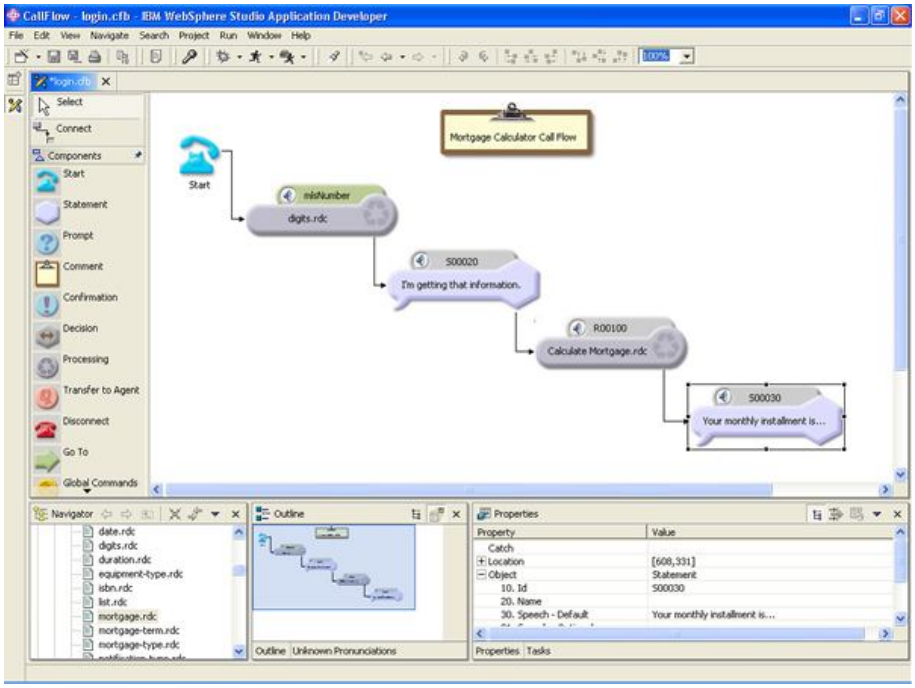

Figura 2.6 Interfaz gráfica de WebSphere Application Developer.

2.4.4 SUEDE

SUEDE [Klem00], es una herramienta para prototipado rápido de IUs vocales basada en la técnica del mago de Oz [Kell84]. El diseño de SUEDE se realizó tomando como base entrevistas realizadas a desarrolladores de interfaces vocales, comprende tres fases básicas que se ven reflejadas en su interfaz gráfica (Figura 2.7): Diseño, Pruebas y Análisis. En la fase de diseño, se crean ejemplos de conversaciones que serán la base para el diseño de la interfaz en si. En la segunda fase se realizan pruebas con usuarios reales con el fin de determinar en una tercera fase los cambios que deben realizarse en una siguiente iteración. SUEDE utiliza para la definición de las IUs cuatro elementos: *start card* (carta de inicio), *prompt card* (carta de mensaje), *response card* (carta de respuesta) y *group card* (carta de agrupamiento). El diseño de los sistemas en la interfaz gráfica se realiza al arrastrar y colocar estas tarjetas de manera que se pueda seguir un flujo de conversación. El asistente que esta herramienta provee se encarga de reconocer las respuestas del usuario de manera que no se ocupan ni un reconocedor ni un sintetizador de voz para probar los prototipos elaborados.

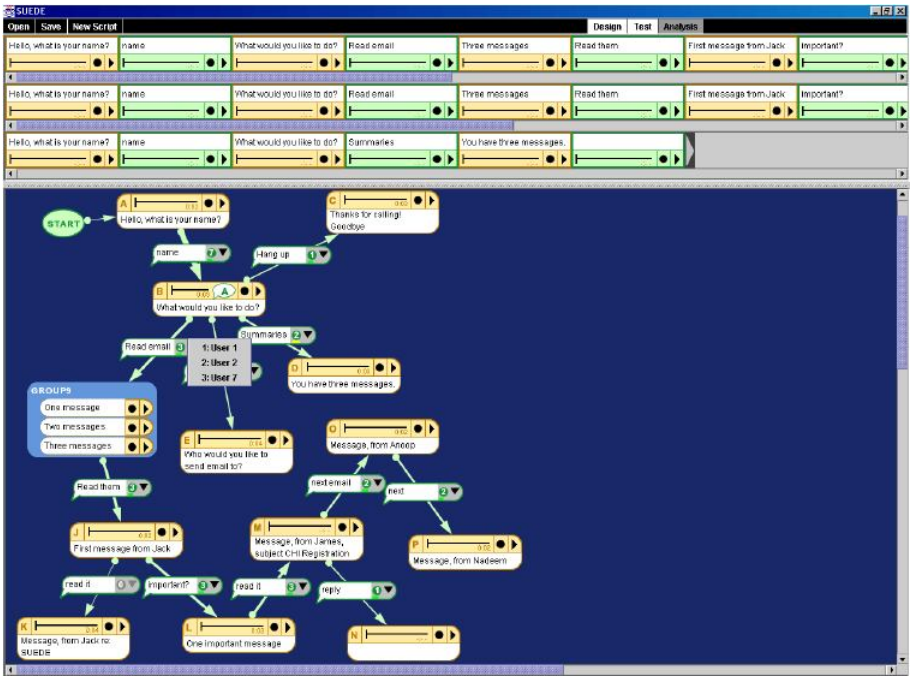

Figura 2.7 Interfaz gráfica de la herramienta de prototipado rápido SUEDE.

2.4.5 Herramientas comerciales

Existe otra gama de herramientas que dan soporte al desarrollo de IUs vocales de carácter comercial. En esta subsección se mencionarán algunas que se consideran de importancia para la realización de este trabajo debido a los elementos que utilizan para garantizar la interacción.

2.4.5.1 Android.speech y Android.speech.tts

Dentro de la API para el desarrollo de aplicaciones compatibles con el sistema operativo Android, se pueden encontrar dos paquetes para soporte a la interacción vocal. El primero de ellos, *android.speech* (disponible desde el API nivel 3) contiene clases necesarias para realizar reconocimiento de palabras mientras que *android.speech.tts* (disponible desde el API nivel 4) provee clases para sintetizar voz a partir de texto. Múltiples aplicaciones hacen uso de estos paquetes entre los que se pueden destacar aplicaciones para accesibilidad y asistentes interactivos.

La documentación de ambos paquetes así como de las clases que contienen se encuentran disponibles para su consulta dentro del sitio Web para desarrolladores de Android [Andr13].

2.4.5.2 Microsoft System.Speech.Recognition y System.Speech.Synthesis

Microsoft ofrece distintas soluciones para dar soporte a la interacción vocal, dentro de las cuales se puede destacar a dos espacios de nombres (namespace) que contienen las clases

necesarias para proporcionar reconocimiento de palabras así como aquellas requeridas para realizar síntesis de voz.

El espacio de nombres para síntesis de voz, tiene elementos que permiten configurar propiedades de la voz que se utiliza como el acento, entonación, énfasis y volumen. Dentro del reconocedor, también se proveen clases para la elaboración de gramáticas. La documentación para ambos espacios de nombres se encuentra disponible en el sitio de soporte a desarrolladores de Microsoft [Micr13].

2.4.5.3 Siri de Apple

Siri [Appl13] es una aplicación para iOS(Sistema operativo para dispositivos móviles) de Apple que soporta la interacción vocal con el usuario con el fin de facilitar la utilización de dispositivos con este sistema operativo y permitir accesibilidad a usuarios con discapacidad visual o motriz. Existe una gran cantidad de aplicaciones con las que Siri es compatible, además de estar disponible en varios lenguajes y soportar diferentes entonaciones y acentos. Esta aplicación también cuenta con una interfaz gráfica (Figura 2.8) cuya función es dar retroalimentación de la información que recibe del usuario como audio y mostrar las salidas que proporciona ya sea de forma vocal o al abrir una aplicación distinta.

A pesar de ser una de las aplicaciones más completas en cuanto a interacción vocal se refiere, Siri no cuenta con documentación suficiente para determinar el lenguaje que fue utilizado en su implementación ni las clases o componentes que le conforman.

Figura 2.8 Interfaz gráfica de la aplicación Siri para el envío de un mensaje de texto.

2.5 Resumen del Capítulo

Tomando como base la información descrita en secciones anteriores, se pueden realizar una serie de comparaciones entre los distintos marcos de trabajo, lenguajes y herramientas con el fin de destacar las ventajas de unos sobre otros así como establecer relaciones entre ellos para determinar aquellos que serán utilizados como parte de este trabajo de tesis.

2.5.1 Comparación de marcos de trabajo teóricos

La primera diferencia que se puede notar entre los primeros dos marcos de trabajo mencionados (W3C Multimodal Interaction Framework y Propiedades CARE) y el tercero (CAMELEON reference framework) es que éste último es independiente de las modalidades de interacción que se deseen utilizar mientras que en los otros si se considera como un factor importante la modalidad y, de forma particular en CARE también es importante el uso de distintas modalidades para alcanzar una meta a partir de un estado determinado.

Otra diferencia importante a considerar es que mientras el marco de trabajo de W3C se centra en el flujo de información entre el usuario y el sistema para establecer los lenguajes y modalidades que se utilizan, las Propiedades CARE se centran en la aplicación de funciones dependientes de modalidades para alcanzar una meta, y CAMELEON consiste en obtener mediante la aplicación de procesos de abstracción y concretización interfaces para diversas modalidades.

2.5.2 Comparación de UIDLs

Una vez que se han citado y descrito de forma breve algunos de los lenguajes para descripción de IUs que pueden resultar de ayuda para este trabajo, la mayoría de éstos con soporte para aplicaciones multi-modales, es posible realizar una comparación entre los mismos. En particular; interesan aquellos lenguajes que tienen dentro de sus objetivos principales la descripción de IUs con modalidad de interacción vocal por lo que la atención se centra en este punto en VoiceXML y X+V, los cuales además de ofrecer soporte para este tipo de interacción, son compatibles con XML.

Se resalta además UsiXML como un lenguaje que ofrece soporte a múltiples modalides y que al seguir el marco de trabajo CAMELEON puede resultar de utilidad para este proyecto al aportar, mediante la utilización de las herramientas disponibles para este lenguaje, abstracciones de IUs ya implementadas las cuales a través de un proceso de concretización utilizando los criterios que se proponen en el Capítulo 3 podrán transformarse en IUs vocales.

2.5.3 Comparación de herramientas de desarrollo

Al realizar el análisis de las herramientas para desarrollo de IUs fue posible observar que éstas cuentan con una interfaz gráfica que facilita el trabajo del diseñador al mostrar la estructura que tendrá la implementación. Así mismo fue posible entender que si bien existen herramientas que dan soporte a varios lenguajes, no se procura unificar los conceptos utilizados en cada una de

manera que los desarrollos realizados para una plataforma puedan ser fácilmente transformados a implementaciones para otra.

También se puede notar que varias de las herramientas referenciadas tienen la intención de trabajar con la interacción vocal pero como un refuerzo a la información que se transmite de manera gráfica, por lo que no se centran en el desarrollo de aplicaciones exclusivamente vocales. Este punto limita las opciones para el desarrollador en cuanto a sensibilidad al contexto y diseño centrado en el usuario se refiere.

El Capítulo 3 hará uso de la información presentada en éste con el fin de elegir marcos de trabajo, UIDLs y herramientas para llevar a cabo un proceso de abstracción sobre ellos y obtener así elementos básicos para la proposición de un modelo para desarrollar IUs vocales.

3. Metodología para el Desarrollo de Interfaces Vocales

3.1 Introducción

Tomando en cuenta el análisis de la literatura llevado a cabo en el estado del arte y con base en los objetivos presentados en el Capítulo 1, este capítulo presenta la definición de la metodología comenzando por la selección de aquellos conceptos y herramientas que son importantes para definir los elementos (sección 3.2) que posteriormente serán abstraídos (sección 3.3) y agrupados para conformar el modelo que se plantea en la sección 3.4.

A continuación, en la sección 3.5 se proponen representaciones gráficas para los elementos del modelo elaborado con el fin de establecer las bases de diseño de una herramienta de software que dé soporte a éste trabajo. El capítulo concluye con un resumen del trabajo realizado resaltando el aporte del mismo.

3.2 Selección de Marcos de Trabajo Teóricos, Lenguajes y Herramientas

El objetivo de esta sección es elegir los elementos que serán considerados al realizar los procesos de abstracción y planteamiento del modelo en las siguientes secciones y que serán utilizados también en la etapa de validación del proyecto. Para tal elección se han considerado diversos factores, entre ellos, la documentación disponible, el tipo de licencia, compatibilidades y dependencias, sin perder de vista que son la base para lograr los objetivos del proyecto planteados en la introducción. Destacando que si bien la selección de estos conceptos permitirá la realización del modelo para desarrollo de IUs vocales, la solución no será exclusiva a esta tecnología, si no extensible y compatible con otras.

3.2.1 Selección del marco de trabajo teórico

Los requerimientos con los que el marco de trabajo debe cumplir de acuerdo con los intereses de este trabajo de tesis son:

1. Ser independiente de la plataforma.
2. Ser independiente de la modalidad.
3. Permitir la realización de procesos de abstracción y concretización sobre componentes de IU.

Tomando en cuenta estas características, el marco de trabajo con el que se ha decidido trabajar es el CAMELEON Reference Framework (Sección 2.2.3).Ya que además de permitir la realización de procesos de abstracción con respecto a la plataforma de cómputo y en cuanto a la modalidad de interacción de IU ya implementadas, contempla la utilización de estructuras como los CIOs que permiten la descripción de elementos de una interfaz para una modalidad en particular. Para el modelo que se presenta en la sección 3.4 se considera la definición de CIOs vocales.

3.2.2 Selección de UIDLs

Para llevar a cabo la selección de UIDLs es importante considerar:

1. Independencia de plataforma.
2. Soporte a IUs vocales.
3. Estandarización.
4. Disponibilidad de la documentación.

Con respecto a los UIDLs a utilizar en este trabajo de tesis, se seleccionan VoiceXML [W3C10] al ser el lenguaje estándar para IUs vocales, X+V [W3C01] por ser un lenguaje basado en Web orientado en parte a las IUs vocales y UsiXML [Limb05] como un lenguaje que permite la descripción de interfaces de manera independiente a la modalidad, manteniendo la compatibilidad con el marco de trabajo teórico que se eligió.

3.2.3 Selección de herramientas de desarrollo

Al elegir las herramientas de desarrollo que se utilizarán en éste trabajo, se debe tomar en cuenta:

1. Disponibilidad de la documentación.
2. Soporte a IUs vocales.
3. Aplicaciones y plataformas relacionadas.

De esta forma, las herramientas que se utilizarán son los espacios de nombres *System.Speech.Recognition* y *System.Speech.Synthesis* de Microsoft [Micr13], dada la extensa documentación de éstos se ha considerado como el principal motivo para utilizarlos, además de la facilidad que se ofrece para su uso a través de la incorporación del arreglo de micrófonos de *Microsoft Kinect* ® [Micr12] como interfaz y su *Software Development Kit* (SDK).

3.3 Abstracción de Componentes

El primer paso que se dio hacia la definición de un modelo para desarrollo de IUs vocales, consistió en realizar una comparación entre los diversos elementos que forman parte de las herramientas y los lenguajes orientados a interfaces vocales seleccionados en la sección 3.2.

Cuando alguna de las funciones consideradas como básicas (emitir un mensaje, recibir un mensaje, realizar una confirmación, etc.) no puede ser representada como una sóla función en un lenguaje o herramienta, se realizan uniones de funciones y elementos para permitir sentar las bases del modelo.

La Tabla 3.1 muestra de manera resumida la comparación realizada de los diferentes lenguajes y herramientas; la comparación completa se adjunta en el Apéndice A.

Tabla 3.1 Comparación entre lenguajes y herramientas.

Voice-XML	XHTML + Voice	Kinect con System.Speech.Synthesis
<audio>	<vxml : audio>	Speak(FilePrompt)
<prompt>	<vxml : prompt>	Speak(Prompt)
<record>	<vxml : record>	Start() + SetOutputToAudioStream
<field>	<vxml : field>	Start()

El segundo paso que se llevó a cabo, consistió en detectar los componentes principales y analizar su función. Enlistar estas funciones permite agrupar elementos. En la Tabla 3.2 se muestra el análisis realizado de manera resumida y en el Apéndice B se presenta el análisis completo.

Tabla 3.2 Comparación entre lenguajes y herramientas con análisis de funciones.

Voice-XML	XHTML + Voice	Kinect con System.Speech.Synthesis	Función
<audio>	<vxml : audio>	Speak(FilePrompt)	Sintetiza audio desde una fuente
<prompt>	<vxml : prompt>	Speak(Prompt)	Sintetiza un mensaje determinado
<record>	<vxml : record>	Start() + SetOutputToAudioStream	Recibe una entrada vocal
<field>	<vxml : field>	Start()	Espera una entrada de parte del usuario

A partir de las comparaciones y análisis presentados, se puede proponer un modelo que represente la forma en la que la interacción vocal puede ser aplicada y como se pueden desarrollar IUs vocales a partir de modelos de tareas, siguiendo la propuesta del marco de referencia CAMELEON [Calv03]. La siguiente sección presenta y describe el modelo planteado.

3.4 Modelo para Desarrollo de Interfaces de Usuario Vocales

Para la elaboración del modelo se tienen dos partes principales, la primera correspondiente al modelo del contexto (contextModel) [Flor04] y la segunda a la IU Concreta Vocal (VocalCUI) presentada en la Figura 3.2. El modelo del contexto mostrado en la Figura 3.1 describe los tres aspectos de un contexto de uso en el que un usuario final lleva a cabo una tarea interactiva con una plataforma específica y en un ambiente determinado: entorno, plataforma y el usuario en sí [Vand07].

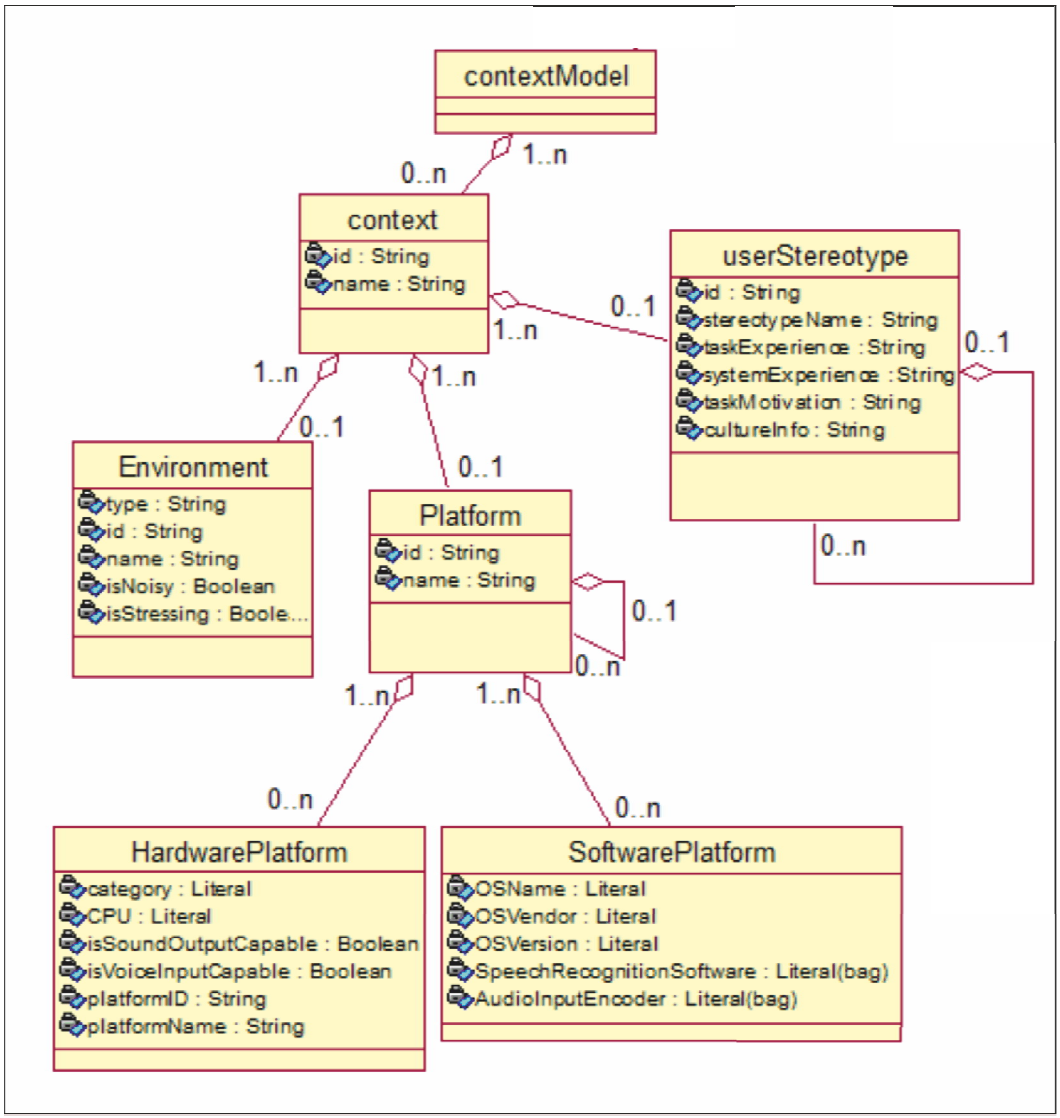

Figura 3.1 Modelo del contexto

El *modelo de usuario* consiste de estereotipos de usuario (userStereotype) [Limb04]. Un estereotipo de usuario es un conjunto de usuarios que comparten características similares. Los estereotipos pueden ser ordenados en jerarquías, de manera que también pueden ser descompuestos en sub-estereotipos. Para este modelo, una modificación fue realizada sobre el estereotipo de usuario general, al agregarle información cultural (cultureInfo), ya que influye en aspectos como el acento, la entonación, el lenguaje y la gramática.

En el *modelo de la plataforma* (Platform), se capturan atributos relevantes para los conjuntos de software y hardware que conforman una plataforma y los dispositivos relacionados con la misma que pueden influenciar de manera significativa el contexto de uso en el que el usuario lleva a cabo una tarea interactiva. Una plataforma se puede especificar en una serie de componentes de hardware, una serie de componentes de software, las características de la red a la que se conecta la plataforma, la capacidad de navegación en páginas Web y el soporte de conexión vía inalámbrica, sin embargo, para este propósito, los únicos elementos que se toman en cuenta son la plataforma de hardware (HardwarePlatform) y la plataforma de software (SoftwarePlatform).

El *modelo del entorno* (Environment), describe propiedades relevantes del ambiente físico en el que se lleva a cabo la interacción con el sistema. Estas propiedades pueden ser físicas (por ejemplo, iluminación), psicológicas (como el nivel de estrés del usuario) y organizacionales (como la localización y la definición de roles en el organigrama de una empresa). En el modelo de la Figura 3.1, el modelo del entorno se resume de manera que sólo presenta las características determinantes para la interacción vocal.

La segunda parte del modelo que se plantea, es el Vocal CUI, formado por Objetos Vocales Concretos de Interacción (VocalCIOs). A su vez, los VocalCIOs se dividen en dos grupos, Componentes Vocales Independientes (vocalIndividualComponent) y Contenedores Vocales (vocalContainer).

Los Contenedores Vocales, representan agrupaciones lógicas de otros contenedores o de componentes individuales. Dentro de estos contenedores, se tiene la propiedad *isOrderIndependent* la cual significa que las entradas en un contenedor pueden ser llenadas en cualquier orden. Al realizar una especificación sobre los contenedores se pueden encontrar:

VocalGroup (Grupo Vocal): Es un elemento raíz, actúa como contenedor principal para otros contenedores y componentes.

VocalForm (Formulario Vocal): Permite la existencia de un diálogo entre el usuario y el sistema, con el propósito de sintetizar/obtener datos tanto del sistema como del usuario.

VocalMenu (Menú Vocal): Permite la elección de uno o varios elementos.

VocalConfirmation (Confirmación Vocal): Solicita al usuario una confirmación para la entrada recién proporcionada. Se compone por al menos un elemento que pide la información al usuario y uno que obtiene la respuesta de confirmación o negación de parte de éste.

De los Componentes Individuales, se pueden separar dos grupos que engloban las funciones más generales y básicas en una interacción vocal:hablar(output) y escuchar(input).

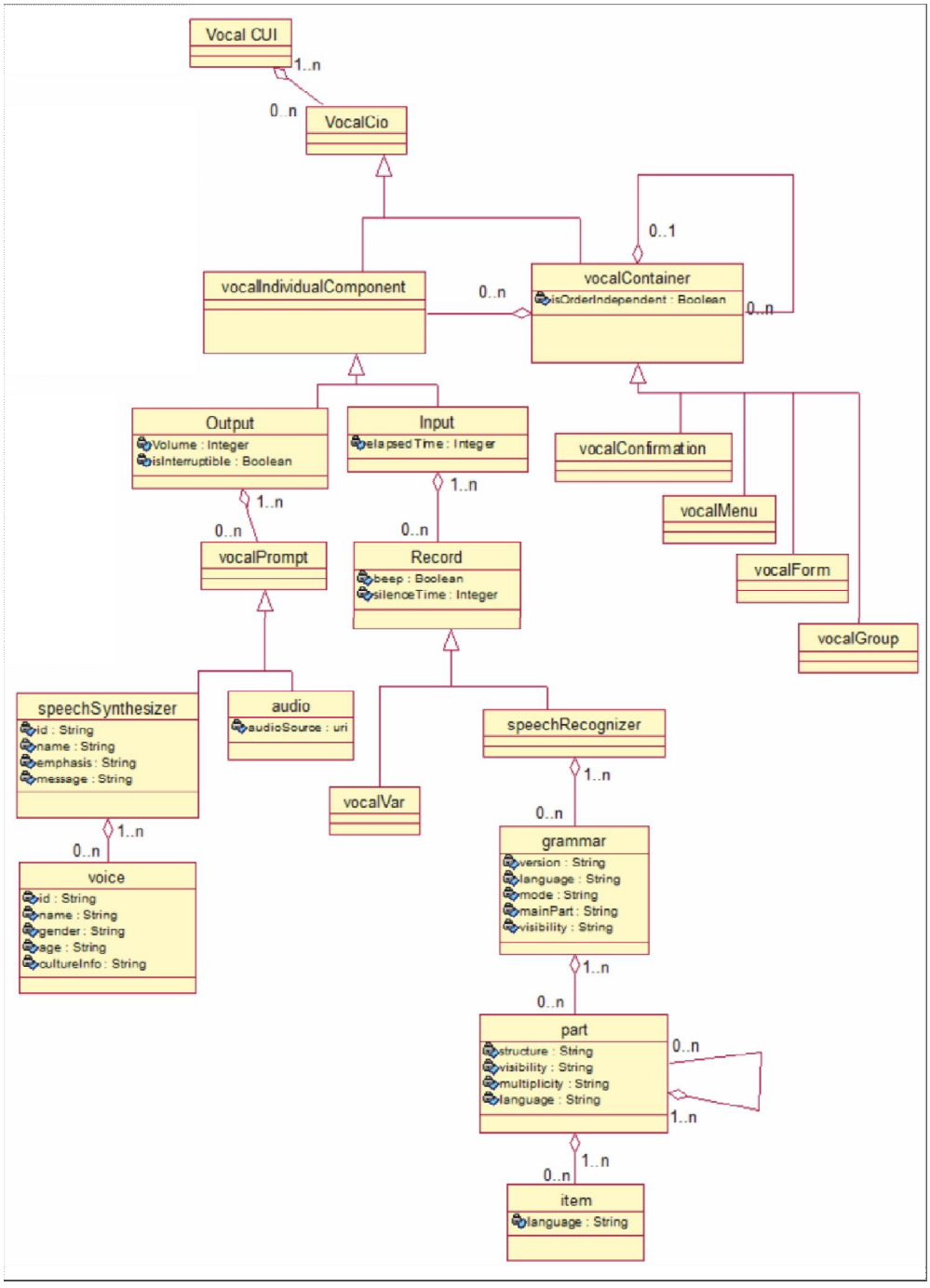

Figura 3.2 ModeloVocal CUI

Output (Salida): Es un objeto utilizado para presentar datos al usuario. El atributo *volume* (volumen), especifica el volumen de la salida auditiva en decibeles, mientras que el atributo *isInterruptible*(es interrumpible) especifica si la salida en curso puede o no ser detenida por acción del usuario.

Input (Entrada): Es un objeto utilizado para obtener datos de parte del usuario, ya sea para aplicar sobre ellos reconocimiento o para ser solamente grabados. El atributo *elapsedTime* (tiempo transcurrido) representa un fragmento de tiempo en el que al usuario se le permite ingresar datos.

Estos componentes individuales utilizan objetos para sintetizar y para procesar datos que permitan la interacción con el usuario. El objeto *Output* utiliza un *vocalPrompt* (mensaje vocal), el cual es a su vez, un objeto que prepara los datos para ser presentados al usuario; de la misma manera, este objeto puede ser dividido dependiendo de la fuente de los datos a presentar:

- *SpeechSynthesizer* (Sintetizador de voz): Es utilizado para realizar transformaciones *text-to-speech* (texto-a-voz) sobre el mensaje que se dé en el atributo *message* (mensaje). El atributo *emphasis* (énfasis) expresa el tono con el que la salida será sintetizada: positivo, negativo, interrogativo o exclamativo. El *SpeechSynthesizer* hace uso del objeto *voice* (voz) con el fin de establecer otros parámetros requeridos para completar la emisión del mensaje. El atributo *gender* (género), se utiliza para especificar si la voz que se utilizará para la síntesis corresponde a un hombre, una mujer o a un individuo neutral. *Age* (edad) es el atributo que se utiliza para determinar la edad de la fuente de voz a utilizar (niño, adolescente, adulto, adulto mayor). *CultureInfo*(información cultural) tiene como función representar el lenguaje o el nivel de gramática para la representación de datos al usuario.
- *Audio:* Es utilizado para reproducir archivos previamente grabados. El atributo *audioSource* (fuente de audio) especifica el Identificador Uniforme de Recurso (URI) del archivo de audio que se desea reproducir o la referencia a donde el archivo de audio se localiza.

El componente individual *input* utiliza un objeto llamado *record* (grabación) con el fin de recibir los datos que el usuario proporciona. Este objeto, recibe un mensaje y a su vez puede especificarse en dos: (1) *vocalVar* (variable vocal)*:* es utilizado para declarar una variable. Sobre este elemento se pueden aplicar funciones y condiciones como *setVar* (asignación), *resetVar* (reinicialización), *if*/*else* y *elsif* (condiciones) y (2) *speechRecognizer* (reconocedor de palabras), el cual es software que como su nombre lo indica, reconoce palabras a partir de la información ingresada por el usuario utilizando gramática(s), partes e ítems.

Cuando un objeto *record* tiene su atributo *beep* (bip) con un valor verdadero (TRUE), se emite un sonido alertando al usuario del momento en el que se puede comenzar a ingresar

información. Si este atributo tiene un valor falso asignado (FALSE), el usuario puede comenzar a grabar su mensaje en el momento que lo desee. El atributo *silenceTime*, especifica el periodo de tiempo a esperar desde que el usuario termina de proporcionar información hasta que el sistema detiene la grabación. Se expresa en segundos o milisegundos.

El objeto *grammar* (gramática), es una enumeración compacta y estructurada de un conjunto de declaraciones (palabras o frases) que constituyen las entradas que al usuario se le permite proporcionar. La gramática puede ser *interna* (declararse en el mismo archivo de código en el que se utiliza) o *externa* (ser especificada en un archivo diferente). El atributo *version* indica la versión de la especificación de gramática que se utiliza.

El atributo *language* (lenguaje) indica el lenguaje de acuerdo con el cual, la declaración debe ser pronunciada para ser reconocida por el sistema. La especificación del lenguaje se realiza utilizando una notación de acuerdo con la cual, se coloca el lenguaje que se desea utilizar, seguido por dos letras que identifican al país en el que se utiliza (English-UK). El atributo *mainPart* (parte principal), representa a la primera parte de la gramática que será tratada por el sistema. El atributo *mode* (modo) consiste en el tipo de interacción que se tiene disponible. El tipo por default es vocal mientras que para interacción telefónica el valor debe ser **dtmf**. El atributo *visibility* (visibilidad) especifica si la gramática puede ser visible en el documento en general o sólo en el *vocalForm* actual.

Part (parte): contiene elementos o items disponibles para las entradas de los usuarios. El atributo *structure* (estructura) especifica la forma en la que el usuario debe decir una declaración para ser reconocida por el sistema. Para esto, hay tres posibles valores: (1) elección, donde los items de la gramática aparecen de manera alternada, (2) secuencial, donde los items de la gramática aparecen de forma ordenada, y (3) asíncrona, donde los items de la gramática aparecen sin un orden en particular. En el atributo *visibility* (visibilidad) se determina si la parte es privada y sólo puede ser utilizada en la gramática que la contiene o si es pública y puede ser referenciada desde otras gramáticas. El atributo *multiplicity* (multiplicidad) indica las veces en que un ítem puede repetirse, el valor predefinido es 1. La multiplicidad se define de la siguiente manera:

- X (donde X > 0): los items se repiten exactamente X veces.
- X-Y (donde 0 <= X < Y): los items se repiten entre X y Y veces.
- X- (donde X >= 0): los items se repiten X o más veces.

El componente *language*, indica en que lenguaje deben ser pronunciados los items para poder ser reconocidos por el sistema. Este campo se expresa de la misma manera que el atributo con el mismo nombre en la gramática, definiendo el lenguaje que se desea utilizar y las letras que corresponden al país en el que se usa. Si no se especifica, toma el valor del lenguaje de la gramática que le contiene.

Item: permite especificar una entrada posible en una gramática o referenciar un objeto *part* de tipo público. Su atributo *language*, indica el lenguaje en el que debe ser pronunciado para poder ser reconocido por el sistema. La notación para definir el lenguaje de cada ítem es la misma que para definir el de cada parte y gramática, y en caso de no ser definido tomará el valor del objeto que le contiene. La definición de un lenguaje para cada ítem, parte y gramática permite la utilización de múltiples lenguajes dentro de una misma gramática.

3.5 Planteamiento de Elementos Gráficos para Representación del Modelo

Una vez que se ha establecido y definido el modelo para desarrollo de IUs vocales, es posible desarrollar una herramienta que le dé soporte. Los entornos de desarrollo que permiten a los usuarios la creación y personalización de aplicaciones son más fáciles de utilizar cuando cuentan con una interfaz gráfica principalemente cuando el diseñador o desarrollador no están familiarizados con un determinado lenguaje o con la programación para una modalidad en particular. Entonces, la herramienta de software creada a partir de el modelo realizado en este trabajo de tesis, deberá contar con representaciones gráficas para cada uno de los elementos del modelo del vocalCUI. Al observar diversas herramientas relacionadas con la interacción vocal y en general los íconos utilizados para denotar entradas de tipo vocal o salidas de tipo auditivo, se puede encontrar que la mayoría usa íconos semejantes a micrófonos para determinar las entradas al sistema e íconos similares a bocinas para las salidas que éste da. Con el fin de mantener congruencia con el trabajo realizado en el área de las IUs vocales, se decide que las representaciones gráficas de la implemenación realizada en el presente trabajo de tesis mantengan un aspecto similar al ya mencionado. En la Figura 3.3, se muestran las representaciones propuestas para cada elemento que será utilizado en la implementación.

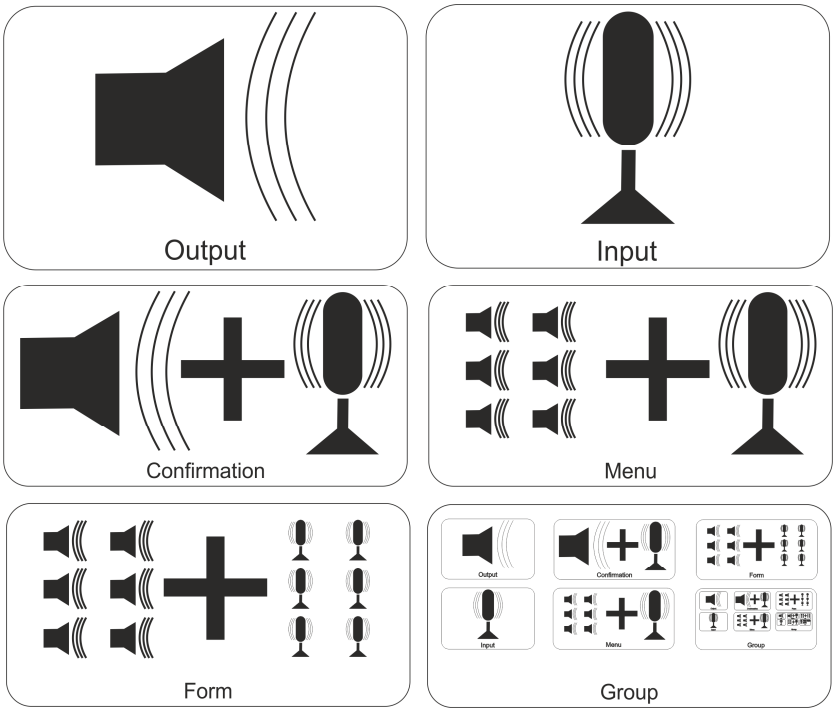

Figura 3.3 Representaciones gráficas para los elementos del modelo propuesto.

3.6 Resumen del Capítulo

En este capítulo se presentan los pasos realizados para la definición de un meta-modelo y para el establecimiento de una metodología para desarrollo de IUs vocales. El primer paso consistió en elegir los lenguajes y herramientas que servirían como base para la abstracción y análisis de funciones. El segundo paso consistió en una comparación de los componentes de cada lenguaje y herramienta con el fin de determinar cuales de estos componentes son parte importante y propia del tipo de interacción y cuales eran características propias del lenguaje. Posteriormente, en el tercer paso se realizó un análisis de las funciones de los componentes previamente comparados con el fin de agrupar aquellas funciones similares y posibilitar la expresión de todos los componentes en términos de otros más simples con el fin de determinar los elementos que deben ser parte del meta-modelo y aquellos que pueden ser obtenidos a partir de otros. El cuarto paso es el establecimiento del modelo con los componentes obtenidos en el paso anterior y que fueron identificados como funciones y elementos básicos, así como su integración con el modelo del contexto descrito en la sección 3.4.

De las tablas comparativas realizadas, se hace evidente la existencia de una serie de reglas de equivalencia que permiten pasar de definiciones de IUs realizadas con la notación del modelo a implementaciones al menos para cada uno de los lenguajes y herramientas que fueron parte del análisis:

- *Output*:<*prompt*> o <*audio*> en VoiceXML, <*vxml : prompt*> o <*vxml : audio*>en X+V y *prompt()* y/o *speak()* en System.Speech.Synthesis.
- *Input*: <*field*>o <*record*> en VoiceXML, <*vxml : field*> o <*vxml : record*>en X+V y utilizando *start()* - *stop()* apoyándose en las librerías de Kinect. Para realizar el reconocimiento de esta entrada, una gramática debe ser implementada previamente en ambos casos.
- *vocalMenu*: un menú vocal puede ser representado como una serie de opciones que se ofrecen al usuario, a través de una serie de *outputs* y una *input* para obtener el valor que el usuario elige. Para organizar, evaluar y ejecutar la selección del usuario, se pueden utilizar funciones y condiciones (if, else, elsif, oneof, foreach).

Un conjunto de reglas de equivalencia de cada componente del modelo para el lenguaje VoiceXML se presenta en el Apéndice C.

El quinto paso realizado es la proposición de elementos gráficos que representen a los componentes del meta-modelo, de manera que una herramienta de software pueda ser desarrollada y facilite la implementación de sistemas con modalidad de interacción vocal a partir de modelos de tareas.

En el siguiente capítulo, se validan mediante casos de estudio los conceptos desarrollados en este capítulo y se muestran ejemplos de la forma en la que la metodología propuesta permite trabajar.

4. Validación de la Metodología

4.1 Introducción

Con el fin de validar tanto la estructura del modelo propuesto en el capítulo anterior como las reglas de equivalencia dadas para cada elemento y su suficiencia para dar solución a los objetivos planteados en el capítulo 1, se diseñaron pruebas en las que se parte de la definición de un problema, se realizan para éste, modelos de tareas y se pasan éstos a términos de la metodología presentada en este trabajo de tesis. Posteriormente, se generan las implementaciones para cada uno de esos ejemplos basándose en las reglas de equivalencia expuestas. Finalmente se da una conclusión mostrando los resultados y objetivos alcanzados con este trabajo de tesis.

Para la elaboración de los modelos de tareas, se utiliza la notación CTTE (ConcurTaskTrees) [Pate03] en la que se pueden modelar tareas de distintas naturalezas:

1) Tareas manuales representadas por el ícono  , son aquellas que el usuario lleva a cabo por sí solo.

2) Tareas automáticas cuya representación gráfica es  , son aquellas que el sistema realiza exclusivamente.

3) Tareas interactivas representadas como  , son aquellas que realiza el usuario con ayuda del sistema.

4) Tareas abstractas representadas con el ícono  , son las tareas de más alto nivel que engloban otras tareas de distintas naturalezas.

4.2 Casos de estudio

4.2.1 Sistema para consulta de existencia de libros en una librería

Para el primer caso de estudio, se requiere la elaboración de un sistema que confirme si existe o no un libro en el inventario de una librería. Los libros que están disponibles se enlistan en una gramática con formato SRGS (Speech Recognition Grammar Specification o especificación de gramática para reconocimiento del habla) que se tendrá que incorporar al código. De manera general, el sistema debe pedir al usuario que mencione el título del libro que desea buscar,

esperar a que el usuario dé una entrada, luego, una vez que el usuario ingrese el título del libro, el sistema debe intentar reconocerlo. Si el título del libro es reconocido, el sistema debe realizar una confirmación y en caso positivo informar al usuario de la existencia del libro en el inventario.

A partir de la definición de los requerimientos del sistema, se puede llevar a cabo la definición de las tareas del mismo y con ellas un modelo que las represente. La Figura 4.1 muestra el modelo de tareas para este caso de estudio.

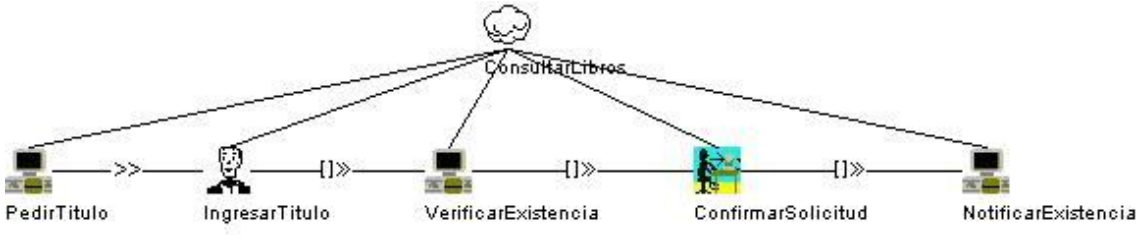

Figura 4.1 Modelo de tareas para el caso de estudio de consulta de existencia de libros

Al analizar las tareas modeladas se puede realizar un modelo más en términos del meta-modelo propuesto en el capítulo 3. La Figura 4.2 muestra el modelo realizado para satisfacer los requerimientos de este caso de estudio.

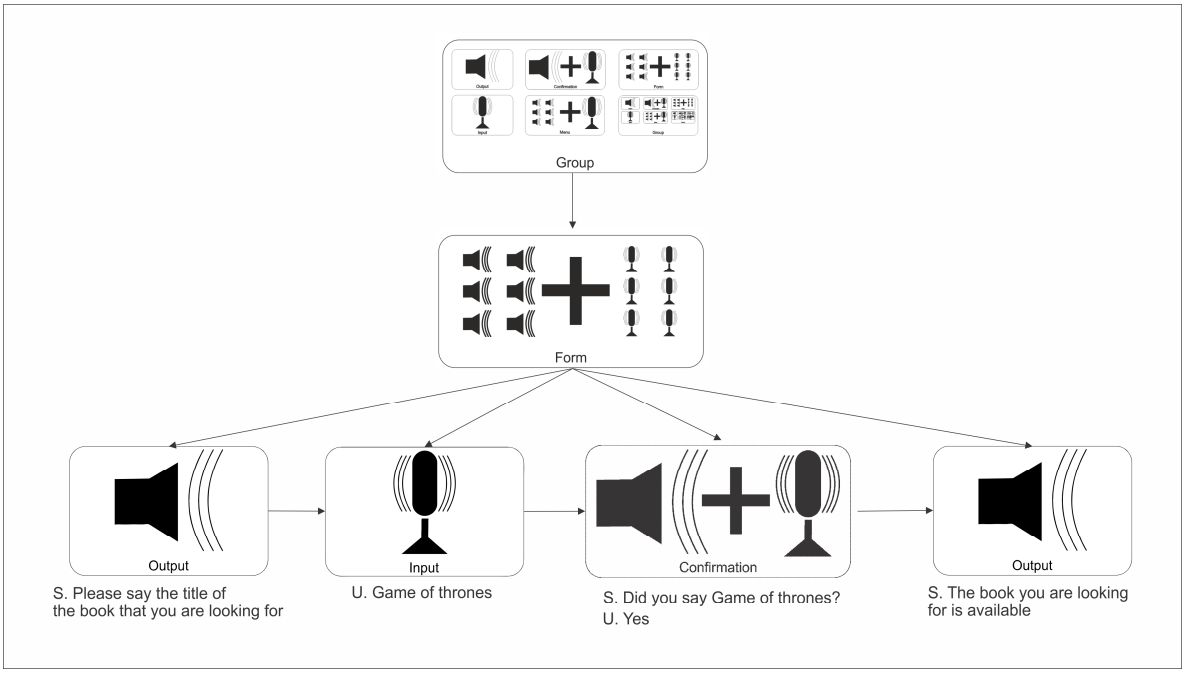

Figura 4.2 Sistema de consulta de existencia de libros modelado en terminos del meta-modelo de interacción vocal

Al realizar el modelado del sistema en términos del meta-modelo propuesto para interacción vocal, se pueden aplicar las reglas en el Apéndice C de manera que se obtenga el código específico a un lenguaje y/o plataforma. La transformación del modelo realizado a código debe realizarse tomando en cuenta el flujo del sistema, de arriba hacia abajo y de izquierda a derecha. Una relación entre componentes colocados en distintos niveles jerárquicos representa *contenido*, mientras que una relación entre elementos de un mismo nivel significa *secuenciación*, es decir, para este caso en particular, todos los elementos son contenidos en un componente *vocalGroup* y todos los componentes colocados debajo del elemento *vocalForm* deben ser implementados a un mismo nivel. Aplicando entonces las reglas de equivalencia sobre el modelo se obtiene el siguiente código como solución al problema planteado en esta subsección:

```
<vxml version="2.1">
<form id= "BookOrdering">
      <field name = "book"><vxml version="2.1">
            <prompt>Please say the title of the book that you are looking for
            </prompt>
            <grammar xml:lang="en-us" version="1.0" root="Books" mode = "voice">
                  <rule id="Books">
                        <one-of>
                              <item> game of thrones </item>
                              <item> the hunger games </item>
                        </one-of>
                  </rule>
            </grammar>
      </field>
      <field name = "confirmation_book" type = "boolean">
            <prompt>Did you say <value expr = "book"/>?
</prompt>
```

```
            <filled>
                  <if cond = "confirmation_book == false">
                        <goto nextitem="book"/>
                  <elseif cond= "confirmation_book == true"/>
                        <prompt>The book you are looking for is available
                        </prompt>
                  <else/>
                        <prompt>Did not understand your input
                        </prompt>
                        <goto nextitem="confirmation_book"/>
                  </if>
            </filled>
      </field>
</form>
</vxml>
```

[Código VoiceXML generado a partir delas reglas de equivalencia aplicadas al modelado del caso de estudio planteado en esta subsección]

4.2.2 Sistema de atención a clientes para una sala de cine

Para el segundo caso de estudio se requiere un sistema que permita la realización de un menú para las distintas tareas que se pueden realizar en un sistema de atención a clientes de una sala de cine. De manera similar a como se hizo en el caso anterior, se parte de la definición de los requerimientos para después modelar las tareas que se identifiquen y posteriormente modelar éstas en términos del meta-modelo de interacción vocal. Así mismo, se aplicaran las reglas de equivalencia del Apéndice C para generar código que brinde una solución al problema expuesto.

El sistema primero debe identificar al usuario, para este propósito le pregunta su nombre. La lista de nombres de usuarios registrados se encuentra en una gramática que se debe agregar al código. Una vez que el nombre es reconocido, se dan al usuario las opciones que éste puede elegir, consulta de cartelera, precio de entradas y la opción de encontrar una sala de cine en particular por ejemplo. La elección del usuario se debe confirmar y en caso positivo redirigirle a una parte distinta del sistema que para fines de este caso de estudio queda fuera del alcance. De esta definición de requerimientos, se obtiene el modelo de tareas mostrado en la Figura 4.3.

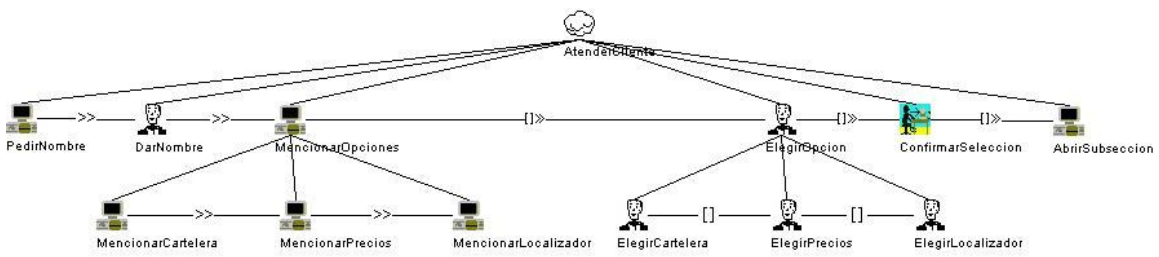

Figura 4.3 Modelo de tareas para el caso de estudio de atención a clientes de una sala de cine

El tercer paso a realizar consiste en modelar en términos del modelo de interacción vocal la solución a los requerimientos encontrados. La Figura 4.4 muestra el modelado realizado para este propósito.

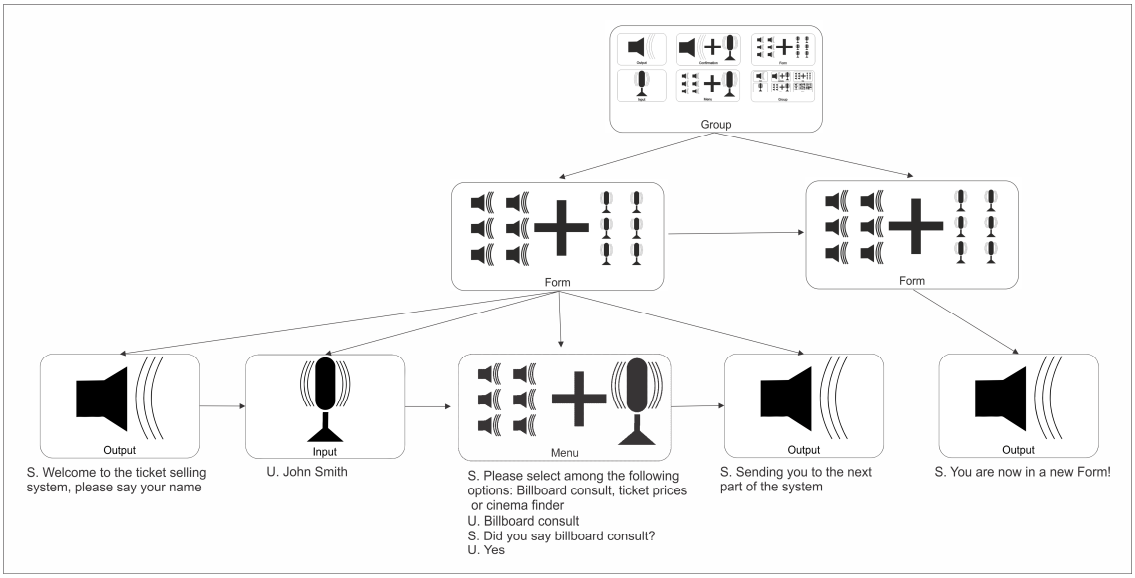

Figura 4.4 Sistema de atención a clientes de una sala de cine modelado en términos del modelo de interacción vocal

Con base en los modelados realizados y apoyándose en las reglas de equivalencia, se genera el siguiente código como solución al problema presentado en este caso de estudio:

```
<vxml version="2.1">
<form id= "TicketSelling">
      <field name = "name">
            <prompt>Welcome to the ticket selling system. Please say your name
</prompt>
            <grammar xml:lang="en-us" version="1.0" root="Names" mode = "voice">
            <rule id="Names">
                  <one-of>
                        <item> John Smith </item>
                        <item> Sheldon Cooper </item>
                        <item> Mike Tyson </item>
                        <item> Wayne Rooney </item>
                        <item> Michael Jordan </item>
                  </one-of>
            </rule>
            </grammar>
            <nomatch>Sorry, that name was not recognized</nomatch>
      </field>
      <field name = "menu">
            <prompt>Please select among the following options:
                  Billboard consult, Ticket prices or Cinema finder
            </prompt>
```

```
            <grammar xml:lang="en-us" version="1.0" root="Options" mode="voice">
            <rule id="Options">
                  <one-of>
                        <item> Billboard consult </item>
                        <item> Ticket prices </item>
                        <item> Cinema finder </item>
                  </one-of>
            </rule>
            </grammar>
            <nomatch>Sorry, the selected option was not recognized</nomatch>
      </field>
      <field name = "menu_confirm" type = "boolean">
            <prompt>Did you say <value expr = "menu"/>?</prompt>
            <filled>
                  <if cond = "menu_confirm == false">
                        <goto nextitem="menu"/>
                  <elseif cond= "menu_confirm == true"/>
                        <prompt>Sending you to next part of the system</prompt>
                        <goto next = "#form2"/>
                  <else />
                        <prompt>Did not understand your input</prompt>
                        <goto nextitem="menu_confirm"/>
                  </if>
            </filled>
      </field>
</form>
<form id= "form2">
      <block>
            <prompt>You are now in another form!</prompt>
      </block>
</form>
</vxml>
```

[Código VoiceXML generado a partir delas reglas de equivalencia aplicadas al modelado del caso de estudio planteado en esta subsección]

4.3 Conclusión del Capítulo

A partir de los modelos de tareas realizados para satisfacer los requerimientos de un sistema, se pueden elaborar otros modelos para interacción vocal con los componentes propuestos en el capítulo 3. En esta validación, además de realizar esto, se utilizaron las reglas propuestas en el Apéndice C que permiten llegar a implementaciones específicas en un lenguaje.

Llevando a cabo el proceso completo, se observa que además de que los componentes del modelo son suficientes para la representación de sistemas de interacción vocal en un nivel concreto, la metodología presentada en este trabajo de tesis se ajusta a los principios de MDA descritos en el capítulo 1 ya que para la generación de soluciones específicas a una plataforma se realiza un modelo independiente de la misma (modelo de tareas), se realiza al menos uno específico a ésta (en términos del meta-modelo de interacción vocal) y por último se elabora código en un lenguaje determinado. El hecho de generar primero un modelo concreto permite que al tener reglas de equivalencia para otros lenguajes, las soluciones puedan ser extendidas sin necesidad de repetir el proceso de diseño o análisis de requerimientos.

5. Conclusiones y trabajo futuro

Para concluir este trabajo de tesis, se realiza un breve resumen de los objetivos y la manera en la que se cumplió con cada uno de ellos. En este capítulo, se mencionan los principales aportes de este proyecto al área a la que pertenece y por último se habla sobre el trabajo a realizar en el futuro a partir de lo aquí planteado.

En el capítulo 1, además de darse una breve introducción a las NUI y más específicamente a los sistemas de interacción vocal, se plantean los objetivos generales y específicos de este trabajo de tesis. De aquí se obtiene que el objetivo general sea elaborar una metodología que siga los principios del desarrollo basado en modelos para soportar la implementación de IUs vocales y como objetivos específicos se identifican: 1) Realizar un análisis de las herramientas, lenguajes, marcos de trabajo y plataformas existentes con el fin de que la metodología que se proponga sea extensible a diversos lenguajes, plataformas y dispositivos, 2) Elaborar un modelo de interacción con base en los componentes que se utilizan en los diversos lenguajes con la finalidad de permitir el modelado de sistemas de interacción vocal en un nivel de abstracción ajeno a la plataforma final, y 3) Que la solución que se plantee considere el contexto y al usuario para permitir que los diseñadores o desarrolladores que hagan uso de esta metodología puedan realizar implementaciones centradas en el usuario.

El primero de los objetivos específicos se atiende en el capítulo 2, en el cuál se realizó la investigación sobre marcos de trabajo, UIDLs y herramientas para desarrollo de interfaces, permitiendo comparar y elegir entre ellos los que se consideran más adecuados para la posterior definición del modelo concreto de interacción.

El modelo de interacción requerido en el segundo objetivo específico, se presenta en el capítulo 3. Para su planteamiento, se lleva a cabo en primera instancia la selección de herramientas, marcos de trabajo y UIDLs, después se realiza una comparación entre los elementos de cada uno de los lenguajes seleccionados junto con un análisis de las funciones que los mismos efectúan y por último, se definen los componentes que forman parte de este modelo al tiempo que se determinan las relaciones que existen entre ellos. Con el fin de atender el tercer objetivo específico, se enriquece el modelo concreto de interacción con un modelo de contexto que contempla dentro de él al estereotipo de usuario.

La evaluación del objetivo general se realiza en el capítulo 4, en el que se puede observar que los componentes que el modelo aporta para la elaboración de interfaces vocales son suficientes y que apoyándose en las reglas de equivalencia propuestas en el Apéndice C, permiten llevar a cabo la generación de código. También se puede observar que la metodología planteada sigue los lineamientos de MDA descritos en el capítulo 1 al contemplar la realización de un modelo independiente a la plataforma (modelo de tareas), al menos un modelo específico a la plataforma (en términos del modelo concreto de interacción) y código específico para una

plataforma. De esto, se puede observar entonces que tanto el objetivo general como los específicos se cumplieron en este trabajo de tesis.

Además de la evaluación del trabajo de tesis realizada en el capítulo 4, fragmentos de él han sido sometidos a procesos de revisión en eventos como la Feria de Proyectos de la Facultad de Ciencias de la Computación de la Benemérita Universidad Autónoma de Puebla y el Cuarto Congreso Mexicano de Interacción Humano Computadora, en los cuales, expertos en las áreas de Computación, Interacción Humano-Computadora e Ingeniería de Software, así como representantes de empresas privadas lo han aceptado y enriquecido con sus observaciones. Como resultado de estas evaluaciones, un artículo que contiene de manera resumida la metodología aquí presentada se encuentra publicado en la biblioteca digital de la ACM (http://dl.acm.org/ft_gateway.cfm?id=2382184&type=pdf).

El trabajo futuro consiste en elaborar reglas de transformación para los distintos lenguajes que soportan la interacción vocal, así como desarrollar una herramienta de software que dé soporte a esta metodología, es decir, un entorno de desarrollo que permita crear modelos en términos del meta-modelo de interacción vocal y que a partir de él genere código para distintas plataformas. Una evaluación de usabilidad deberá ser realizada sobre las implementaciones que sean resultado de esta herramienta generadora de código con el fin de comprobar que los aspectos relacionados con el usuario y el contexto son atendidos correctamente.

Referencias

A

[Abra04]
Abrams, M. and Helms, J. *User Interface Markup Language (UIML) Specification: working draft 3.1*, Marzo 2004, http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf

[Andr13]
Android API , *Speech package reference*, Especificación de paquete Enero 2013. Disponible en: http://developer.android.com/reference/android/speech/package-summary.html

[Appl13]
Apple Inc., *Siri Beta Overview*, Descripción de productos de Apple Enero 2013. Disponible en: http://www.apple.com/ios/siri/

[Arsa02]
Arsanjani, A., Chamberlain, D., et al., *(WSXL) Web Service Experience Language Version 2*, IBM Note Abril 10, 2002.

[Ayob09]
Ayob, N., Hussin, A., Dahlan, H., *Three Layers Design Guideline for Mobile Application*, Abril 2009. ICIME'09 International Conference on Information Management and Engineering.

B

[Bert04]
Berti, S., Correani, F., et al., *The Teresa XML Language for the Description of Interactive Systems at Multiple Abstraction Levels*, en Proceedings of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, Mayo 2004, pp. 103-110.

C

[Calv03]
Camara, A. *Natural User Interfaces*. En: 13th IFIP TC 13 International Conference, Lisbon, Portugal. Septiembre 2011.

[Cama11]
Calvary, G., Coutaz, J., et al., A Unifying Reference Framework for Multi-Target User Interfaces, Interacting with Computers, 15(3), June 2003, pp. 289–308.

[Cout95]
Coutaz, J., Nigay, L., et al., *Four easy pieces for assessing the usability of multimodal interaction: the CARE properties*, Proceedings of 5th IFIP TC 13 International Conference on Human-Computer Interaction INTERACT'95, 1995. pp. 115–120.

[Crow03]
Crowle, S., Hole, L., ISML: An Interface Specification Meta-Language, en Design, Specification and Verification of Interactive Systems, Lectures in Computer Science Volume 2844, DSV-IS'2003 pp. 362-376.

D

[Davi05]
Davis, V., Gray, J., et al., *Generative Approaches for Application Tailoring of Mobile Devices*, en Proceedings of the 43rd annual Southeast regional conference - Volume 2, ACM-SE 43, 2005. pp. 237-241.

F

[Flor04]
Flor, T., *Experiences with Adaptive User and Learning Models in eLearning Systems for Higher Education*, En: Journal of Universal Computer Science, Volumen 10. 2004.

G

[Gome04]
Gomes de Sousa, L., Leite, J., *XICL- An Extensible Mark-up Language for Developing User Interface and Components*, en Computer-Aided Design of User Interfaces IV, Proceedings of the Fifth International Conference on Computer-Aided Design of User Interfaces CADUI'2004. pp. 247-258.

[Guer09]
Guerrero-García, J., González-Calleros, J.M., et al., *A Theoretical Survey of User Interface Description Languages: Preliminary Results*, en Proceedings of the 2009 Latin American Web Congress, La-Web'09. pp. 36-43.

I

[IBM05]
IBM, *WebSphere Voice Toolkit Getting Started Version 6.0.*, Second Edition, 2005. Disponibleen: http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp?topic=/com.ibm.voicetools.callflow.doc/ccfpalette.html.

K

[Kawa02]
Kawamoto, S., Shimodaira, H., et al., *Open-source Software for Developing Anthropomorphic Spoken Dialog Agents,*enPRICAI 2002: Trends in Artificial Intelligence 7th Pacific Rim International Conference on Artificial Intelligence, Tokyo, Japan. Proceedings International Workshop on Lifelike Animated Agents. pp. 64-69.

[Kell84]
Kelley, J.F., *An iterative design methodology for userfriendly natural language office information applications*, ACM Transactions on Office Information Systems, Vol. 2, No. 1, Marzo 1984, pp. 26-41.

[Klem00]
Klemmer, S., Sinha, A., Chen, J., et al., *SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces,* en CHI Letters, Proceedings of 13th Annual ACM Symposium on User Interface Software and TechnologyUIST'2000.

[Kost04]
Kost, S., *Dinamically Generated Multi-Modal Application Interfaces*, en International Conference on Advanced Visual Interfaces, AVI'2004.

L

[Limb04]
Limbourg, Q. *Multi-path Development of User Interfaces*, PhD ofUniversity of Louvain La Neuve, Bélgica, Tesis Doctoral, 2004

[Limb05]
Limbourg, Q., Vanderdonckt, J., et al: *UsiXML: a Language Supporting Multi-Path Development of User Interfaces*. En: Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCIDSVIS'2004. Springer-Verlag, Berlin, 2005.

[Luyt04]
Luyten, K., Abrams, M., et al.,*Developing User Interfaces with XML: Advances on User Interface Description Languages*, Sattelite workshop of Advanced VisualInterfaces, Gallipoli, Italy, 2004.

M

[Micr12]
Microsoft, Kinect for Windows: Learning resources and documentation, 2012, Disponible en: http://www.microsoft.com/en-us/kinectforwindows/develop/learn.aspx

[Micr13]
Microsoft, *System.Speech Namespaces*, Especificación de librerías de clases, Enero 2013, Disponible en: http://msdn.microsoft.com/en-us/library/gg145021.aspx

P

[Pate03]
Paternò, F., *ConcurTaskTrees: An Engineered Notation for Task Models*, en The Handbook of Task Analysis for Human-Computer Interaction. 2003. pp. 483-503.

[Pate09]
Paternò, F., Santoro, C., et al., *MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments*, en ACM Transactions on Computer-Human Interaction Volume 16 Issue 4 TOCHI'2009.

[Puer02]
Puerta, A., Eisenstein, J., *XIML: A Common Representation for Interaction Data*, en Proceedings of 6th International Conference on Intelligent User Interfaces IUI'2002 (San Francisco, USA, Enero 13-16, 2002), ACM Press, pp. 214-215.

S

[Scha06]
Schaefer, R., Bleul, S., et al., *Dialog Modeling for Multiple Devices and Multiple Interaction Modalities*, en Proceedings of the 5th international conference on Task models and diagrams for users

interface design, TAMODIA'06. pp.39-53.

[Stei12]
Steinberg, G. Natural User Interfaces. Disponible en línea: https://cs.auckland.ac.nz/courses/compsci705s1c/exams/SeminarReports/natural_user_interfaces_gste097.pdf USA, 2012.

V

[Vand04]
Vanderdonckt, J., Limbourgh, Q., et al., *UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces*, en Proceedings of W3C Workshop on Multimodal Interaction WMI'2004.

[Vand07]
Vanderdonckt,J., Calvary, G., Coutaz, et al., *Multimodality for Plastic User Interfaces: Models, Methods, and Principles*, en "Multimodal user interfaces: from signals to interaction", D. Tzovaras (ed.), Chap. 3, Lecture Notes in Electrical Engineering, Springer-Verlag, Berlin, 2007, pp. 79-105.

W

[W3C01]
W3C consortium, *XHTML+Voice Profile 1.0*, W3C Note Diciembre 2001. Disponible en: http://www.w3.org/TR/xhtml+voice/

[W3C03]
W3C consortium, *W3C Multimodal Interaction Framework*, W3C Note Marzo 2003. Disponible en: http://www.w3.org/TR/mmi-framework/

[W3C04]
W3C consortium, *EMMA: Extensible MultiModal Annotation markup language*, W3C Working Draft, Diciembre 2004. Disponible en: http://www.w3.org/TR/emma.

[W3C10]
W3C consortium, *Voice Extensible Markup Language (VoiceXML) Version 3.0*, W3C recommendation Diciembre 2010. Disponible en: http://www.w3.org/TR/voicexml30.

[W3C11]
W3C consortium, *Canonical XML Version 2.0*, W3C Working Draft, Abril 2011. Disponible en: http://www.w3.org/TR/2011/WD-xml-c14n2-20110421/

Apéndice A.Comparación de elementos de los lenguajes y herramientas

Tabla A.1 Comparación de los componentes de VoiceXML, XHTML+Voice y Kinect / Microsoft Speech Synthesizer

VoiceXML	XHTML+Voice	Kinect y SpeechSynthesizer
<assign>	<vxml : assign>	"="
<audio>	<vxml : audio>	Speak(FilePrompt)
<block>	<vxml : block>	
<break>	<vxml : break>	Pause()-Resume()
<catch>	<vxml : catch>	catch(){}
<choice>		case "":
<clear>	<vxml : clear>	"=null;"
<data>		
<disconnect>		form.dispose()
<else>	<else>	else{}
<elseif>	<elseif>	else if{}
<emphasis>	<vxml : emphasis>	PromptEmphasis
<enumerate>		
<error>	<vxml : error>	
<example>	<vxml : example>	
<exit>		form.dispose()
<field>	<vxml : field>	Start()
<filled>	<vxml : filled>	SpeechRecognized()
<foreach>		for(){}
<form>	<vxml : form>	form
<goto>		
<grammar>	<vxml : grammar>	choices()
<help>	<vxml : help>	
<if>	<if>	if(){}
<initial>	<vxml : initial>	
<item>	<vxml : item>	grammar.Add()
<link>		
<log>	<vxml : log>	Console.WriteLine();
<mark>	<vxml : mark>	
<media>		

<menu>		switch(){}
<meta>		
<noinput>	<vxml : noinput>	
<nomatch>	<vxml : nomatch>	
<one-of>	<vxml : one-of>	
<option>	<vxml : option>	
<paragraph>	<vxml : paragraph>	
<param>	<vxml : param>	
	<vxml : phoneme>	
<prompt>	<vxml : prompt>	Speak(Prompt)
<property>	<vxml : property>	"."
<prosody>	<vxml : prosody>	
<record>	<vxml : record>	Start(), SetOutputToAudioStream
<reprompt>	<vxml : reprompt>	Speak(Prompt)
<return>	<vxml : return>	return
<rule>	<vxml : rule>	
<ruleref>	<vxml : ruleref>	
<say-as>	<vxml : say-as>	
<script>	<script>	
<sentence>	<vxml : sentence>	
<subdialog>	<vxml : subdialog>	
<submit>		
<throw>	<vxml : throw>	try{}
<token>	<vxml : token>	
<transfer>		
<value>	<vxml : value>	Prompt()
<var>	<vxml : var>	
<vxml>		

Apéndice B.Análisis de funciones de elementos de lenguajes y herramientas

Tabla A.2 Análisis de funciones de los componentes de VoiceXML, XHTML+V y Kinect / Microsoft Speech Synthesizer

VoiceXML	XHTML+Voice	Kinect y SpeechSynthesizer	Función
<assign>	<vxml : assign>	"="	Asigna un valor a una variable de manera explícita
<audio>	<vxml : audio>	Speak(FilePrompt)	Reproduce un archivo de audio
<block>	<vxml : block>		Tipo de formulario que se reproduce si su condición es verdadera
<break>	<vxml : break>	Pause()-Resume()	Pausa en la salida del sintetizador
<catch>	<vxml : catch>	catch(){}	Uso en manejo de excepciones y errores
<choice>		case "":	Uso en menús para cada opción uso alternativo a gramáticas
<clear>	<vxml : clear>	"=null;"	Borra una variable, campo, menú, etc.
<data>			Usado para leer XMLs
<disconnect>		form.dispose()	Desconecta al usuario de la aplicación
<else>	<else>	else{}	Condicional
<elseif>	<elseif>	else if{}	Condicional
<emphasis>	<vxml : emphasis>	PromptEmphasis	Se puede utilizar para dar una pronunciación especial
<enumerate>			Se usa para dar las diferentes opciones al usuario de un sistema
<error>	<vxml : error>		Permite captar un evento de error
<example>	<vxml : example>		Es parte de la definición de una gramática para ayudar al reconocimiento de voz
<exit>		form.dispose()	Desconecta al usuario de la aplicación, permite mandar un valor al servidor que ejecutó la aplicación
<field>	<vxml : field>	Start()	Recibe una entrada por parte del usuario
<filled>	<vxml : filled>	SpeechRecognized()	Permite realizar acciones cuando se recibe una entrada por parte del

			usuario
<foreach>		for(){}	Permite recorrer los valores en un arreglo
<form>	<vxml : form>	form	Es un contenedor para los componentes de tipo campo y de control
<goto>			Sirve para realizar transiciones a un formulario en el documento actual o en uno distinto
<grammar>	<vxml : grammar>	choices()	Permite definir gramáticas en el documento actual o fuera de él
<help>	<vxml : help>		Permite captar un evento de ayuda
<if>	<if>	if(){}	Condicional
<initial>	<vxml : initial>		Permite ejecutar partes del código de acuerdo con estados dados por el usuario
<item>	<vxml : item>	grammar.Add()	Representa un valor permitido dentro de una gramática
<link>			Permite definir manejo de eventos para un proyecto
<log>	<vxml : log>	Console.WriteLine();	Permite dar mensajes de debugeo en la consola
<mark>	<vxml : mark>		Inserta marcadores para notificaciones asíncronas
<media>			Es una extensión de audio
<menu>		switch(){}	Es una abreviatura para simular un menú para el usuario
<meta>			Permite dar información sobre el documento en sí
<noinput>	<vxml : noinput>		Permite captar un evento de no entrada
<nomatch>	<vxml : nomatch>		Permite captar un evento que se produce cuando la entrada del usuario no se reconoce
<one-of>	<vxml : one-of>		Permite la creación de una gramática con una serie de oraciones o palabras
<option>	<vxml : option>		Permite dar opciones al usuario, es una alternativa a las gramáticas
<paragraph>	<vxml : paragraph>		Permite especificar párrafos para el sintetizador
<param>	<vxml : param>		Se utiliza para pasar parámetros a objetos o partes de un documento
	<vxml : phoneme>		Permite al desarrollador especificar la pronunciación del sintetizador
<prompt>	<vxml : prompt>	Speak(Prompt)	Permite dar al usuario mensajes

			utilizando el sintetizador de voz
<property>	<vxml : property>	"."	Da acceso a propiedades ya sea del documento o de un objeto en particular
<prosody>	<vxml : prosody>		Permite cambiar características del sintetizador de voz
<record>	<vxml : record>	Start(), SetOutputToAudioStream	Graba y guarda en un archivo una entrada del usuario
<reprompt>	<vxml : reprompt>	Speak(Prompt)	Repite el último Prompt realizado
<return>	<vxml : return>	return	Termina un subdiálogo y regresa el control al dialogo principal
<rule>	<vxml : rule>		Define parte de una gramática
<ruleref>	<vxml : ruleref>		Permite agregar una regla a una gramática de otro documento
<say-as>	<vxml : say-as>		Define la forma en la que se debe leer un texto
<script>	<script>		Permite importar código ECMAScript para su ejecución en un diálogo
<sentence>	<vxml : sentence>		Permite que el sintetizador lea texto como si fuera una oración
<subdialog>	<vxml : subdialog>		Permite definir sub diálogos que son ejecutados dentro de contenedores
<submit>			Pasa el control a un nuevo documento o a algún formulario en el documento actual
<throw>	<vxml : throw>	try{}	Permite lanzar un evento que puede ser manejado en alguna sección del código
<token>	<vxml : token>		Define palabras que deben ser dichas por el usuario dentro de una gramática
<transfer>			Es un elemento utilizado para enviar al usuario a un destino en el documento actual o en otro nuevo
<value>	<vxml : value>	Prompt()	Puede ser utilizado para insertar un valor para el sintetizador de voz
<var>	<vxml : var>		Se utiliza para declarar variables
<vxml>			Declaración inicial de un documento de VoiceXML

Apéndice C.Reglas de equivalencia para los componentes del meta-modelo

Para cada componente del meta-modelo presentado en el Capítulo 3, se realizan reglas de transformación para la generación de código VoiceXML. A continuación se enumeran esas reglas:

```
<prompt>Texto a ser sintetizado</prompt>
```

[Código VoiceXML para el componente Output del meta-modelo]

```
<field name = "identificadorParaElCampo">
   [Gramática]
</field>
```

[Código VoiceXML para el componente Input del meta-modelo]

```
<field name = "identificadorParaElCampo" type = "boolean">
   <prompt>Dijo<value expr="campoAConfirmar"/>?
   </prompt>
   <filled>
      <if cond = "identificadorParaElCampo == false">
         Entrada no confirmada
      <elseif cond= "identificadorParaElCampo == true"/>
         Entrada confirmada
      </if>
   </filled>
</field>
```

[Código VoiceXML para el componente vocalConfirmation del meta-modelo]

```
<field name = "nombreDelMenu">
   <prompt>Instrucciones y opciones del menú
   </prompt>
   [Gramática]
</field>
<field name = "confirmacionDelMenu" type = "boolean">
   <prompt>Dijo<value expr = "nombreDelMenu"/>?
   </prompt>
   <filled>
      <if cond = "confirmacionDelMenu == false">
         Selección no confirmada
      <elseif cond= "confirmacionDelMenu == true"/>
         Selección confirmada
      </if>
   </filled>
</field>
```

[Código VoiceXML para el componente vocalMenu del meta-modelo con confirmación de selección]

```
<form id= "identificadorDelFormulario">
   [Contenido del formulario]
</form>
```

[Código VoiceXML parael componente vocalForm del meta-modelo]

```
<vxml version="numeroDeVersion">
</vxml>
```

[Código VoiceXML para el componente vocalGroup del meta-modelo]