Procesamiento del Lenguaje Natural, núm. 35 (2005), pp. 285-292 recibido 29-04-2005; aceptado 01-06-2005

Un Sistema de Diálogo Multicanal para Acceder a la Información y Servicios de las Administraciones Públicas

Meritxell González
TALP Research Center
Software Department
Universitat Politècnica de Catalunya
mgonzalez@lsi.upc.edu

Marta Gatius
TALP Research Center
Software Department
Universitat Politècnica de Catalunya
gatius@lsi.upc.edu

Resumen: En este artículo se presenta un sistema de diálogo desarrollado para el proyecto HOPS. El proyecto HOPS tiene como objetivo facilitar el acceso a la información y servicios de las administraciones locales en los que el conocimiento de la aplicación está representado en una ontología. Esta representación permite gestionar la interacción con el usuario en modo oral o textual en diferentes lenguas. El *gestor de diálogo* utiliza las ontologías para decidir cuál será la siguiente interacción con el usuario, así como para generar en las diferentes lenguas las gramáticas, léxicos y mensajes que intervendrán en cada interacción. Esta forma de representar el conocimiento implicado en la comunicación permite la reutilización de los recursos desarrollados en diferentes aplicaciones.
Palabras clave: Sistema de diálogo, ontologías, procesamiento semántico, sistema multicanal, administración pública.

Abstract: This article presents a dialogue system developed for the European project HOPS. Hops project focuses on facilitating the access to the information and services of local administrations using ontologies to represent knowledge. This representation allows managing the user interaction in textual and vocal mode in different languages. The dialogue controller uses ontologies both for managing user interactions and for generating grammars, lexicons and messages implied in communication. This way of representing knowledge implied in communication allows reusing developed resources in future applications.
Keywords: Dialogue system, ontologies, semantic processing, multichannel system, public administration.

1 Introducción

Los recientes avances en las tecnologías de voz y lenguaje natural, juntamente con la creciente importancia del uso de Internet, favorecen el desarrollo de nuevas aplicaciones que integran las tecnologías de la lengua y de la web semántica. En este contexto nace el proyecto europeo HOPS (2), el principal objetivo del cual es la integración de las tecnologías de voz, lenguaje natural y web semántica para desarrollar una plataforma que permita acceder a los servicios públicos a través de diferentes canales y en diferentes lenguas.

La mayoría de los servicios públicos están presentes hoy en día en la web. La integración de las tecnologías del lenguaje y la web semántica permitirá su explotación a través del teléfono o la web. En el proyecto HOPS estamos desarrollando una plataforma que permita el acceso a las aplicaciones de tres entidades públicas (el Ayuntamiento de Barcelona, el Ayuntamiento de Torino y el London Borough of Camden) en cuatro idiomas (catalán, castellano, inglés e italiano). Uno de los objetivos del proyecto es desarrollar un sistema de diálogo que pueda adaptarse fácilmente a otros servicios. En este ámbito se han realizado diferentes trabajos, entre los que destaca el realizado en el proyecto GEMINI,

ISSN: 1135-5948

descrito en (D'Haro et al, 2004) y en el proyecto SmartKom (Wahlster , 2003).

Este artículo se centra en describir cómo el *módulo de voz* del sistema desarrollado se adapta a nuevos servicios[1]. La sección 2 describe la arquitectura general del sistema. En la sección 3 se presenta el funcionamiento del sistema de diálogo para el *módulo de voz* y la adaptación a un servicio concreto. En la última sección se plantean las conclusiones y el trabajo futuro.

2 Arquitectura del sistema

Uno de los objetivos del proyecto HOPS es desarrollar una plataforma en la que la incorporación de nuevos servicios no resulte muy costosa. La plataforma facilitará la adaptación a un nuevo servicio de las diferentes bases de conocimiento implicadas en el proceso comunicativo. Para facilitar la reusabilidad de la plataforma, tanto las tecnologías de voz como de web semántica se basan en el uso de estándares desarrollados por el W3C (5), así como en el uso de software libre.

La arquitectura del sistema se muestra en la Figura 1. Como puede observarse en dicha figura, la arquitectura del sistema es modular, de forma que cada componente puede ser fácilmente sustituido por una versión mejorada. La comunicación entre módulos se realiza a través de FADA (1), una herramienta que permite la integración de sistemas heterogéneos sin requerir ningún tipo de administración. FADA permite la comunicación síncrona entre los componentes del sistema. La implantación de la tecnología FADA permite que la arquitectura del sistema sea distribuida y que los módulos desconozcan la implementación concreta del resto de módulos.

2.1 El módulo de voz

El *módulo de voz* controla la interacción a través del teléfono. Está basado en el estándar VoiceXML. Sus componentes son: el reconocedor, el intérprete VoiceXML y el sintetizador. Estos componentes se han adaptado de los de la plataforma basada en VoiceXML desarrollada por Loquendo (4). El intérprete de VoiceXML (5) es el responsable de controlar la interacción con el usuario. Las interacciones vienen definidas en una página VoiceXML. Cada interacción define el atributo sobre el que se da o pide información, el texto que representa el mensaje del sistema (*prompt*) y una gramática. La gramática es específica para reconocer la respuesta del usuario a la pregunta (o comentario) del sistema. Las gramáticas están escritas según el estándar *Speech Recognition Grammar XML Form* (GRXML) del W3C (5). Para que el reconocedor pueda interpretar la respuesta del usuario es necesario que ésta se corresponda con alguna de las posibles opciones representadas dentro de la gramática. El valor resultante del reconocimiento se asocia al atributo que representa la intervención.

El *módulo de voz* recibe del *gestor de diálogo* el documento VoiceXML que ha de procesar. Este documento contiene los mensajes que guiarán al usuario a introducir la información necesaria para el sistema. El documento contiene también las referencias a las gramáticas que deben reconocer la respuesta. Las gramáticas incorporan información semántica que facilita el procesamiento de la intervención del usuario. La información semántica resultante se pasa al *gestor de diálogo*.

2.2 El módulo de texto

El *módulo de texto* controla la interacción a través de la web. Consiste en dos componentes: un servidor de texto y un analizador sintáctico-semántico. Una vez obtenida la interpretación semántica de la intervención del usuario, ésta se pasa al *gestor de diálogo*.

La comunicación textual no presenta las limitaciones técnicas de la interacción oral. Por tanto, el *módulo de texto* soporta un lenguaje más rico, modelado por gramáticas más complejas. Además, el *módulo de voz* debe de controlar eventos como la interrupción (*barge-in*) o la falta de fiabilidad de la entrada reconocida, que no ocurren en la comunicación textual.

El analizador utilizado realiza el procesamiento semántico guiado por la sintaxis. La interpretación semántica de la sentencia introducida por el usuario se obtiene a partir de la información incorporada en las reglas gramaticales y en las entradas léxicas. Esta información se basa en el *lambda calculus*. Una descripción más detallada de las técnicas

[1] "Este trabajo ha sido subvencionado por el proyecto europeo HOPS (IST-2002-507967) y la Acción Complementaria del CICyT TIN2004-0169-E"

utilizadas en el procesamiento de la entrada textual puede encontrase en (Gatius et al, 2005).

2.3 El gestor de diálogo

El *gestor de diálogo* es el responsable de controlar la comunicación a través de los diferentes canales. El *gestor de diálogo* es de propósito general, independiente del servicio concreto.

La intervención del usuario se realiza a través del teléfono o de la web. Los respectivos *módulos de voz y texto* procesan esta intervención y obtienen la información semántica. Esta información se pasa al *gestor de diálogo* que es quien controla el flujo de interacciones. El *gestor de diálogo* usa el conocimiento representado en las ontologías para decidir qué va a hacer en la siguiente intervención. El *gestor de ontologías* envía al *gestor de diálogo* una representación del servicio basada en precondiciones. En cada intervención el *gestor de diálogo* evalúa las precondiciones de los estados activados y selecciona el que tiene una precondición más estricta. Esta nueva intervención puede ser una consulta o una nueva interacción con el usuario.

En el caso de la nueva interacción con el usuario, la información necesaria se obtiene a partir de la ontología que representa a la aplicación. En el caso de la consulta a la aplicación, se mostrarán los resultados de la consulta al usuario. El *gestor de diálogo* delega en el *gestor de acciones y consultas* la realización de las consultas. Este módulo es el encargado de acceder a las ontologías y a las bases de datos de las aplicaciones (*backend*) de forma transparente para el *gestor de diálogo*, devolviendo exactamente la información que necesita.

El *gestor de diálogo* se encarga de generar los documentos VoiceXML y XML para los *módulos de voz y texto* con la información necesaria para llevar a cabo la interacción. La gestión de la interacción con el usuario se delega a los respectivos *módulos de voz y texto*. Cada interacción se describe por la información sobre el atributo concreto que se debe preguntar o dar al usuario. Estos atributos se definen en la ontología que representa toda la información relacionada con el servicio que puede aparecer en la comunicación. El *gestor de diálogo* genera los mensajes que deben ser mostrados al usuario y los incorpora en los documentos que se van a enviar al *módulo de texto o voz*, junto con las referencias a las gramáticas que deben reconocer las respuestas del usuario.

El módulo *gestor de aplicaciones* se ha incorporado a la arquitectura del sistema para controlar los recursos que requiere cada servicio implementado en el sistema. La plataforma puede soportar más de un servicio y usuario a la vez.

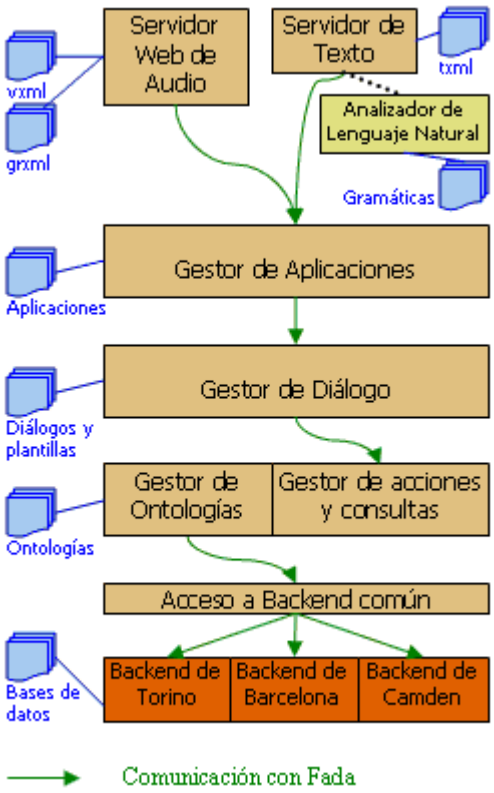

Figura 1: Arquitectura del sistema

2.4 Las ontologías

El proyecto HOPS incorpora las tecnologías de la web semántica para representar el conocimiento relativo a los servicios implementados en la plataforma. Las ontologías usadas en la plataforma HOPS incluyen una ontología de alto nivel que describe de forma general los distintos servicios, un conjunto de ontologías genéricas que contienen términos generales utilizados en todos los servicios y dominios, y un conjunto de ontologías que contienen los términos y servicios específicos para cada aplicación.

El conocimiento del servicio que interviene en la comunicación se representa en una ontología. En ella, toda la información que el servicio necesita del usuario se representa

mediante conceptos. Estos conceptos se describen mediante atributos. Los atributos representan la información específica sobre el concepto que se le pregunta o da al usuario.

Los atributos también se usan para generar precondiciones. Cada concepto tiene una precondición. En cada interacción el *gestor de diálogo* selecciona todos los conceptos cuya precondición se cumple y selecciona aquél que tiene la precondición más estricta (con más términos).

También se usan ontologías de diferentes dominios disponibles en sitios Web. Por ejemplo, hemos construido una ontología de muebles extraída del sitio Web IKEA (3). A partir de esta ontología hemos creado el léxico que usa la gramática.

Las ontologías se han implementado usando el lenguaje OWL. Como ya se ha comentado anteriormente, el módulo llamado *gestor de acciones y consultas* es el encargado de realizar las consultas sobre las ontologías. El *gestor de diálogo* utiliza este módulo para obtener la información, ya sea de las ontologías que representan el conocimiento del dominio, o de las aplicaciones de las entidades públicas.

3 Adaptación del módulo de voz a un nuevo servicio

El *módulo de voz* se basa, como ya se ha indicado anteriormente, en el formalismo VoiceXML. En esta sección se describen los mecanismos generales utilizados para generar los documentos VoiceXML y las gramáticas GRXML para un nuevo servicio.

3.1 El diálogo

Para cada aplicación definimos tres tipos de interacciones con el usuario: obtener el valor de uno o varios atributos, confirmación de los valores obtenidos, y mostrar resultados de una consulta a las bases de datos de la aplicación.

3.1.1 Obtener el valor de los atributos

En tiempo de diseño generamos los mensajes que se van a mostrar al usuario para pedir el valor de un atributo. Estos mensajes se diseñan de forma que, en tiempo de ejecución, se completan con información dinámica obtenida en las interacciones anteriores con el usuario y la información contenida en las ontologías.

VoiceXML permite cierta iniciativa mixta en los diálogos. En una interacción podemos definir un mensaje inicial general en el que se hace una pregunta abierta al usuario sobre los atributos que se deben rellenar en esa intervención. Con la respuesta del usuario obtenemos el valor para uno o varios de los campos descritos en el documento VoiceXML. Después de esta primera interacción, el sistema pregunta el valor de aquellos campos de los que no se ha obtenido el valor. Se han incorporado mecanismos para generar interacciones iniciales en las que se pide el valor de varios atributos a la vez. Se usan plantillas VoiceXML generales, como la de la Figura 2, con marcas donde se inserta el código para cada campo de la interacción en curso. Estas marcas se completan con información específica del estado del diálogo y usando otras plantillas. Por ejemplo, la plantilla de la Figura 4 incorpora información relativa a una llamada a un subdiálogo y la asignación del resultado a un atributo. La plantilla de la Figura 3 incorpora información relativa a un atributo: el nombre, el mensaje que se muestra al usuario, la gramática que reconoce la respuesta, el tipo de la gramática, acciones a realizar una vez obtenido el valor, e información relativa a eventos que se pueden dar en cualquier momento de la conversación: i) noinput, el usuario no ha dicho nada, ii) nomatch, no se ha reconocido nada, iii) help, se ha pedido ayuda sobre el atributo.

Por ejemplo, en la Figura 2 se substituye la marca *@LANG@* por el valor "*es-ES*" (español-España), "*en-GB*" (english-Great Britain), "*ca-ES*" (catalán-España), o "*it-IT*" (italiano-Italia). La marca *@FORMID@* se substituye por el nombre del formulario. *@ATTRS_PROMPTS@* se sustituye por el resultado de completar una plantilla como la de la Figura 3 o la de la Figura 4 para cada atributo. En la Figura 3 substituimos *@ATTR_NAME@* por el nombre del atributo. *@GRAMMAR_SRC@* se sustituye por el nombre del fichero que contiene la gramática que debe reconocer la respuesta del usuario. *@PROMPT1@* se sustituye por el mensaje que se muestra al usuario. *@GRAMMAR_TYPE@* se sustituye por el nombre del formato de la gramática ("*application/srgs+xml*" para las gramáticas GRXML). Como ya se ha indicado anteriormente, en el proyecto HOPS hemos implementado las gramáticas según el formato GRXML especificado por el W3C (5).

```
<?xml version="1.0"?>
<vxml version="2.0"
      application="AppRoot.vxml"
      xml:lang="@LANG@">
    <form id="@FORMID@">
     @MIXED_PROMPT@
     @ATTRS_PROMPTS@
     <filled>
      <submit
       next="@SUBMIT_NEXT@"
       namelist="@ATTR_NAMES@"/>
     </filled>
    </form>
</vxml>
```

Figura 2: Plantilla general

```
<field name="@ATTR_NAME@">
    <grammar src="@GRAMMAR_SRC@"
     type="@GRAMMAR_TYPE@"/>
    <prompt>
        @PROMPT1@
    </prompt>
        <help count="1">
            @HELP1@
        </help>
        <noinput count="1">
            @NOINPUT1@
        </noinput>
        <nomatch count="1">
            @NOMATCH1@
        </nomatch>
</field>
```

Figura 3: Plantilla de atributos

Hemos definido gramáticas de propósito general para modelar las intervenciones del usuario que se pueden dar en cualquier momento de la comunicación con la mayoría de las aplicaciones, por ejemplo, pedir ayuda o volver al menú principal. También se han definido gramáticas de ámbito general (activas durante todo el proceso de comunicación) específicas de la aplicación. Por ejemplo, en el servicio de recogida de muebles de casas particulares, que se describe en la sección 3.3, se utilizan gramáticas generales para reconocer las preguntas del usuario que se dan comúnmente sobre si el servicio es gratuito y sobre la hora de recogida de los muebles. En estos casos se inicia un subdiálogo de clarificación. Al final el subdiálogo se continúa con el flujo normal de interacciones del diálogo.

Los subdiálogos de clarificación son específicos para cada aplicación. Para el servicio de recogida de muebles, se ha estudiado el comportamiento de los usuarios cuando usan el servicio y se han generado varios subdiálogos que los usuarios suelen iniciar en el transcurso normal de la comunicación. Estos subdiálogos se ejecutan al reconocer la petición del usuario mediante las gramáticas de ámbito general específicas de la aplicación.

Algunas interacciones se pueden repetir en varias aplicaciones, por ejemplo, pedir el número de teléfono o la dirección. Para este tipo de interacciones se crean subdiálogos siguiendo las especificaciones del VoiceXML del W3C (5). Estos subdiálogos forman parte de los recursos generales de la plataforma, se diseñan y generan una sola vez y pueden utilizarse en varias aplicaciones. La utilización de estos subdiálogos elimina la necesidad de definir la misma interacción para varias aplicaciones. Para incorporar en una nueva aplicación un diálogo previamente definido en una aplicación ya desarrollada sólo es necesario incorporar la referencia (dirección) del documento que contiene el subdiálogo.

La Figura 4 describe la plantilla para generar una llamada a un subdiálogo y asignar el resultado a un atributo concreto. Por ejemplo, la dirección del usuario se requiere para realizar cualquier gestión en el servicio de la recogida de muebles. El subdiálogo para obtener la dirección se define en un documento VoiceXML que sigue esta plantilla. Este subdiálogo se define en un documento independiente de la aplicación, de manera que pueda ser utilizado por cualquier aplicación que necesite pedir al usuario su dirección.

```
<var name="@ATTR_NAME@"/>
<subdialog name="@ATTR_NAME@_sub"
           src="@SRC_URI@">
    <filled>
        <assign name="@ATTR_NAME@"
         expr="@ATTR_NAME@_sub"/>
    </filled>
</subdialog>
```

Figura 4: Plantilla de llamada a un subdiálogo.

3.1.2 Confirmar el valor de los atributos

Una vez el sistema ha obtenido del usuario el valor de los atributos que se utilizarán como parámetros en las operaciones que acceden al *backend* de la aplicación, es necesario confirmar que el valor obtenido es correcto. Las interacciones de confirmación también utilizan documentos VoiceXML generados de forma automática. En este tipo de interacciones se usa la misma plantilla general descrita en la Figura 2. Para completar la plantilla general usamos otra donde se insertan los nombres y valores de los atributos que queremos confirmar. En estas interacciones se usa una gramática general de confirmación, que consiste en un objeto de reconocimiento interno de la plataforma que reconoce afirmaciones y negaciones. Podemos ver una de estas plantillas en la Figura 5.

```
<?xml version="1.0"?>
<vxml version="2.0"
      application="AppRoot.vxml"
      xml:lang="@LANG@">
  <form id="confirmation">
    <field
     name="confirmationfield"
     type="boolean">
     @PROMPT1@
    </field>
    <filled>
     <if cond="confirmationfield">
      <submit next="@YES_URI@"/>
     <else/>
      <submit next="@NO_URI@"/>
     </if>
    </filled>
  </form>
</vxml>
```

Figura 5: Plantilla confirmación

En esta Figura sustituiríamos la marca @PROMP1@ por una frase que contenga todos los atributos que hay que confirmar y sus valores. El Ejemplo (1) es una frase de confirmación del servicio de la recogida de muebles:

(1) Por favor, confirme que los datos son correctos: la dirección de la recogida es avenida diagonal número 8, el teléfono de contacto es 934501234 y usted quiere tirar una mesa. ¿Es correcto?

Para reconocer la respuesta del usuario se especifica en el campo (*field*) que el atributo es de tipo booleano (*type=”boolean”*). El siguiente estado del diálogo dependerá de la respuesta del usuario, por ello se definen las dos marcas *@YES_URI@* y *@NO_URI@* donde se especifican las diferentes acciones a realizar.

3.1.3 Consultas a la base de datos de la aplicación

Una vez el sistema ha accedido al *backend* de la aplicación se pueden dar tres tipos de diferentes de interacciones: i) mostrar al usuario un listado de resultados y pedir al usuario que escoja uno, ii) mostrar un dato y pedir al usuario que responda si esta de acuerdo con el dato, iii) mostrar al usuario el resultado de la transacción y opcionalmente, permitir su modificación. Las dos primeras interacciones son el resultado de una consulta al *backend* de la aplicación y la tercera de una transacción. Para estas interacciones no se utilizan plantillas generales dado que las preguntas a realizar al usuario una vez se muestran los resultados son muy variadas y dependen del contenido específico de la base de datos. Por ello se generan plantillas específicas para cada consulta o transacción de cada aplicación. En tiempo de ejecución el *gestor de diálogo* introduce los resultados obtenidos de la base de datos en las plantillas.

Actualmente, el generador de mensajes es un simulador de generador de lenguaje natural que obtiene los mensajes previamente almacenados.

3.2 Las gramáticas

Al definir la interacción con una aplicación determinada, deben incluirse los mensajes asociados a cada atributo, las gramáticas que deben reconocer las respuestas a estos mensajes y las gramáticas generales que se activan durante el transcurso del diálogo.

En las interacciones en que se pide el valor de varios atributos, se usa una gramática para cada atributo y gramáticas generales que nos sirven para relacionar atributos, de forma que podemos reconocer las entradas de iniciativa mixta. Estas gramáticas se completan con la información contenida en las ontologías. Concretamente, algunas de las palabras que incorporan las gramáticas se obtienen de forma semiautomática a partir de las ontologías de dominio. También estamos estudiando la utilización de una ontología sintáctico-semántica en la que se clasifican los atributos y

que ya se ha utilizado para la generación de gramáticas textuales (Gatius, M. y Rodríguez, H, 2002).

3.3 Los diálogos y gramáticas para el servicio de recogida de muebles.

Actualmente, hemos desarrollado un primer prototipo que permite acceder a un servicio en concreto: el servicio de recogida de muebles disponible por el Ayuntamiento de Barcelona. Este sistema soporta diálogos de voz (a través del teléfono) y diálogos de texto (a través de la web) en varias lenguas: inglés, castellano y catalán. El flujo del diálogo para este servicio incluye 36 interacciones diferentes con el usuario. Cada interacción se representa en la ontología como un concepto. La información que se intercambia con el usuario la describen 24 atributos. Cada atributo de un concepto (hay atributos que intervienen en más de un concepto) tiene asociado 4 mensajes que permiten presentar o pedir información al usuario, y se asocia con la gramática concreta que debe usarse para interpretar la respuesta del usuario en esa interacción. Se han desarrollado 6 gramáticas para la información que debe obtenerse directamente del usuario, 3 objetos de reconocimiento (boolean, integer, phone), y 4 gramáticas de ámbito general. El formalismo GRXML (5) define un tipo concreto de regla llamada "*GARBAGE*". El reconocedor del *módulo de voz* descarta cualquier parte del habla que no entienda cuando se encuentra con esta regla. Las gramáticas se simplifican considerablemente al introducir esta regla especial en posiciones estratégicas de la gramática. Para el desarrollo de las gramáticas se han usado 70 transcripciones de diálogos reales entre usuarios y operadoras del servicio.

En una parte del diálogo el sistema necesita saber los muebles que quiere tirar el usuario, así como su dirección y nombre. Después de confirmar los datos, el sistema consulta en el *backend* de la aplicación el día que el servicio de recogida de muebles puede pasar por la dirección indicada. A continuación, el sistema comunica la fecha al usuario y pide su confirmación Vemos el documento VoiceXML resultante de esta última intervención en la Figura 6. En esta Figura podemos ver, además, el uso del objeto de reconocimiento interno (*type="boolean"*). Al indicar que un campo es de tipo booleano, permitimos que la plataforma de voz use sus propios mecanismos internos de detección de afirmaciones y negaciones. El uso de los objetos de reconocimiento (descritos en el estándar VoiceXML) mejora el rendimiento de la plataforma de reconocimiento del lenguaje natural de entrada vocal. Son objetos creados a priori. Se usan sobretodo para reconocer las intervenciones más comunes, como los números, un número de teléfono, o las afirmaciones y negaciones.

```
<?xml version="1.0"?>
<vxml version="2.0"
  application="AppRoot.vxml"
  xml:lang="en-GB">

    <form id="givedateForm">

      <field name="confirmation"
            type="boolean">

        <prompt>
          The collection will be
          next wednesday. Is that OK?
        </prompt>

      </field>

      <help>
        Collection will be wednesday.
        If you agree say YES,
        otherwise say NO.
      </help>
      <noinput>
        Please, say yes or not. <reprompt/>
      </noinput>
      <nomatch>
        I'm sorry, I didn't understand you.
        <reprompt/>
      </nomatch>

    </form>
</vxml>
```

Figura 6: Documento VoiceXML para la confirmación de la fecha de recogida.

Para el reconocimiento de la calle del usuario se han incluido en la gramática todas las calles de la ciudad de Barcelona. Estas de han extraído de forma automática de una base de datos. Por motivos de espacio no hemos incluido la gramática resultante en el artículo, pero si se puede ver una parte de ésta gramática en la Figura 7. La gramática general es "*gramaddress*". Esta gramática usa otras dos: "*gramstreetname*" y "*gramnumber*". La gramática "*gramstreetname*" devuelve el código de la calle reconocida tal como lo va a necesitar el controlador de diálogo para acceder al *backend* de la aplicación. La gramática "*gramnumber*" devuelve el número. Se está

generando un objeto de reconocimiento de calles para mejorar este proceso, ya que en la actualidad presenta algunos problemas: el elevado número de calles (Barcelona tiene más de 4000 calles) y la utilización de catalán y castellano al dar una dirección (por ejemplo, decir el nombre de la calle en catalán y el resto de la dirección en castellano).

```
<rule id="gramaddress" scope="public">
  <item repeat="0-1">
    <ruleref uri="#gramstreetname"/>
  </item>
  <item repeat="0-1">
    <ruleref uri="#gramnumber"/>
  </item>
</rule>

<rule id="gramstreetname" scope="public">
  <one-of>
    <item>
      <ruleref uri="#is_street"/>
      <tag>:ret</tag>
    </item>
    <item>
      <ruleref uri="#is_square"/>
      <tag>:ret</tag>
    </item>
    [...]
    <item>
      <ruleref uri="#is_boulevard"/>
      <tag>:ret</tag>
    </item>
  </one-of>
  <item>
  <tag>&lt;@gramstreetname $ret&gt;</tag>
</rule>
```

Figura 7: Gramática de reconocimiento de direcciones.

4 Conclusión y trabajo futuro

En este artículo hemos presentado un sistema de diálogo multicanal y multilingüe, en el que los recursos son generados semiautomáticamente a partir del conocimiento de la aplicación representado en ontologías. Este sistema de diálogo ha sido diseñado para facilitar el acceso de los ciudadanos a los servicios públicos de entidades públicas. La utilización de los estándares definidos en el W3C ha permitido la rápida integración de estas aplicaciones en el nuevo sistema. La utilización de ontologías que representen el conocimiento del servicio, así como los recursos lingüísticos asociados, nos ha permitido generar gramáticas y léxicos para diferentes modos y diferentes lenguas, que al incorporar las mismas interpretaciones semánticas facilitan el proceso comunicativo.

Actualmente, hemos finalizado el desarrollo del primer prototipo que incluye una nueva aplicación sobre un dominio más complejo, la agenda cultural de actividades de la ciudad. Esta aplicación ha sido desarrollada en inglés e italiano. Para el desarrollo de esta nueva aplicación se ha desarrollado una nueva ontología que representa al flujo del diálogo y nuevos recursos lingüísticos para el dominio concreto.

En el desarrollo del segundo prototipo esperamos poder mejorar el proceso de generación de recursos del domino, así como mejorar la representación de las interacciones con el usuario a fin de mejorar la comunicación, y esperamos poder realizar una evaluación del sistema con usuarios reales.

Bibliografía

D'Haro, L.F, De Córdoba, R., San-Segundo, R., Montero, J. M., Macías-Guarasa, J., Pardo, J. M. Strategies to reduce Design Time in Multimodal/Multilingual Dialog Applications. En las Actas INTERSPEECH 2004 - ICSLP Year 2004.

Gatius, M. y Rodríguez, H. Natural Language Guided Dialogues for Accessing the Web. En las Actas del 5th International Conference, Text, Speech and Dialogue 2002. Brno, Czech Republic, September 2002.

Marta Gatius, Meritxell González, Leonardo Lesmo, Pietro Torasso, Livio Rovaldo. (2005). D32. Natural Language Processing Technologies. HOPS internal deliverable.

Wahlster, W. SmartKom: Symmetric Multimodality in an Adaptive and Reusable Dialogue Shell. En las Actas de Human Computer Interaction Status Conference 2003.

(1) The FADA homepage http://fada.techideas.info
(2) The HOPS Project. http://www.hops-fp6.org/
(3) IKEA Web Site http://www.ikea.com
(4) The Loquendo homepage. http://www.loquendocafe.com/index.asp
(5) W3C. http://www.w3.org