UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SISTEMA DE VIGILANCIA DE PERSONAS MAYORES O CON INCAPACIDAD USANDO SISTEMAS MULTIAGENTE Y ROBOTS BÍPEDOS CONTROLADOS MEDIANTE VOZ

Autor: Alex Yanahuaya Arce

Revisor: Vicente J. Julián Inglada

Valencia - España - 2015

Agradecimientos

A Dios y a mi familia por su apoyo incondicional.

A mi tutor de tesis por su ayuda y sabios consejos.

A mis profesores que me han enseñado y formado durante sus clases.

A mis compañeros y amigos por ayudarme a resolver mis dudas.

Al proyecto BABEL por financiarme la beca.

Resumen del Trabajo

Los robots humanoides hoy en día están siendo utilizados en diversas áreas de investigación, con este trabajo se pretende desarrollar un prototipo para vigilar personas mayores o con incapacidad en un edificio usando el robot humanoide NAO, el cual hace uso de un sistema multiagente sobre la plataforma SPADE e incluye tecnologías lingüísticas.

Un sistema multiagente se compone de agentes autónomos que trabajan juntos para resolver un problema dado, donde cada agente es un una entidad computacional independiente. En nuestro trabajo se presentan 2 tipos de agentes:

- Agente gestor, encargado de reportar las tareas realizadas por los agentes revisores.
- Agente revisor, encargado de gestionar un robot NAO.

Otro componente del sistema, es el Sistema de dialogo Voxeo, el cual a través de VoiceXML nos permite interactuar mediante voz (Skype) con el agente gestor.

Para el desarrollo del prototipo también se emplean otras tecnologías, como el software Choregraphe que es un simulador del robot NAO desarrollado por Aldebaran Robotics, un servidor web Apache, así también el uso de diferentes lenguajes como XML (lenguaje el extensible de marcas) y Python (lenguaje de programación interpretado).

De esta forma este trabajo integra diversas tecnologías relacionadas con la Inteligencia Artificial y las Tecnologías Lingüísticas, para poder desarrollar un prototipo de sistema de vigilancia empleando robots bípedos.

Summary

Today humanoid robots are being used in various areas of research; this work is pretending develop a prototype to monitor older people or disabled in a building using the humanoid robot NAO, which makes use of a multi-agent system on the SPADE platform and it includes linguistic technologies.

A multi-agent system consists of autonomous agents that work together to solve a given problem, where each agent is an independent software entity. In our work two types of agents are presented:

- Manager Agent, responsible to report the work done by the reviewers agents.
- Reviewer Agent, responsible of managing a robot NAO.

Another component of the system, is the dialogue system Voxeo, which through VoiceXML it allows us to interact by voice using Skype with the agent manager.

To develop the prototype also we employ other technologies such as he software Choreographer, which is the NAO robot simulator developed by Aldebaran Robotics, an Apache Web server, as well as the use of different languages like XML (eXtensible Markup Language) and Python (interpreted programming language).

Thus, this work integrates various technologies related to Artificial Intelligence and Language Technology, to develop a prototype of surveillance system using biped robots.

Índice general

Índice de figuras

Capítulo 1

Introducción

La robótica es la ciencia o rama de la tecnología que se dedica al diseño, construcción, operación, disposición estructural, manufactura y aplicación de los robots. Las ramas que coadyuvan principalmente en la robótica son: Ingeniería mecánica, Ingeniería electrónica y Ciencias de la computación.

Los robots son agentes físicos capaces de realizar tareas de forma automática o manual dependiendo del tipo de control, llegando a realizar tareas complejas y/o peligrosas para los seres humanos de forma eficiente. Los robots que trabajan con control automático hacen uso de la Inteligencia Artificial para *pensar* y satisfacer algún trabajo en concreto, en cambio los robots con control manual o teleoperados ofrecen interfaces donde el usuario controla directamente al agente físico.

Un robot humanoide (cuerpo antropomorfo) es un robot construido de modo que su estructura se asemeja a la del cuerpo humano. Típicamente, estos robots se componen de un tronco, cabeza, dos brazos y dos piernas, pero en algunos casos sólo se modelan algunas partes del cuerpo humano. La diferencia sustancial con los androides es que estos últimos están diseñados con la intención de parecerse estéticamente a los seres humanos, tales como los producidos por Hiroshi Ishiguro de la Universidad de Osaka.

Las líneas de investigación en robots humanoides están experimentando un cambio de paradigma de una investigación independiente hacia un entorno social como la robótica asistencial.

En este trabajo se utiliza el robot humanoide NAO el cual es programable y autónomo fabricado por la empresa francesa Aldebaran Robotics (ver figura 1), el cual utiliza una plataforma de sistemas multiagente como SPADE para realizar la revisión del estado de salud de los pacientes en un edificio. Esta plataforma de sistema multiagente está conformada por dos agentes, un agente gestor que recibe órdenes de un usuario a través de un sistema de dialogo como Voxeo

y el otro es el agente revisor conformado por el robot NAO encargado de cumplir las tareas del usuario.

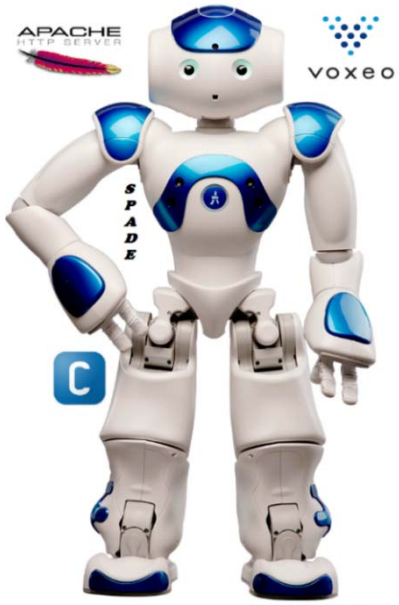


Fig. 1. *Robot Nao con las distintas tecnologías usadas*

1.1 MOTIVACIÓN

A continuación se describe de forma detallada, las distintas motivaciones que nos llevaron a la realización de este trabajo final de master:

- El crecimiento tecnológico ha dado lugar al empleo de la robótica en distintas áreas, una de estas es el servicio social, siendo factible la vigilancia con robots bípedos de las personas mayores o incapacitadas.
- La tecnología de sistemas multiagente (SMA) está siendo un gran aporte a la resolución de problemas distribuidos de diverso índole, entre estas tenemos: comercio electrónico, medicina, industria, robótica, etc. Siendo importante la integración con los robots bípedos, llegando a ser el robot un agente inteligente que trabaja con otros agentes para poder resolver problemas difíciles o imposibles de resolver por un agente individual.
- El Reconocimiento automático del habla, es otra de las disciplinas más interesantes de la Inteligencia Artificial, en este trabajo se hace uso de un sistema de dialogo entre el agente y el usuario. También se incluye el reconocimiento de voz por parte del robot hacia el paciente.
- Las líneas de investigación de la robótica están centradas en conseguir un comportamiento inteligente, esto es posible gracias al uso de la inteligencia artificial. De ahí nace otra motivación e interés sobre los robots humanoides para: programar, controlar, simular y planificar rutas.

1.2 OBJETIVOS

1.2.1 OBJETIVO GENERAL

El principal objetivo de este trabajo es desarrollar un prototipo para vigilancia de personas mayores o con incapacidad usando sistemas multiagente y robots bípedos controlados mediante un sistema de dialogo (donde el usuario interactúa por voz con el robot).

El robot bipedo utilizado será el robot NAO, el cual presenta muchas características, entre las más notables tenemos reconocimiento de marcadores (Landmark), reconocimiento vocal, sistema multiplataforma y soporte de programación en diferentes lenguajes.

1.2.2 OBJETIVOS ESPECÍFICOS

Este objetivo general se divide en los siguientes objetivos específicos:

- Estudio de los diferentes sistemas de dialogo, desde la interfaz oral a su arquitectura de módulos. Esto servirá de base para elegir el sistema que más se adecue al trabajo a desarrollar, donde la información de entrada será la voz mediante el móvil o la computadora y la salida será el acceso a la información requerida.
- Diseño e implementación del algoritmo para recepción de órdenes en el sistema de reconocimiento elegido. Una vez ya elegido el sistema de reconocimiento de voz, se procede a elaborar el código que ejecuta el reconocimiento de voz del usuario (sistema de dialogo) que interactuará con el control del agente revisor.
- Estudio y análisis de plataformas de sistemas multiagente. Es necesario hacer un estudio de las diferentes plataformas que hacen uso de sistemas multiagente para resolver problemas de diverso índole, con el objetivo de seleccionar la más apropiada para la interacción con el robot bípedo.
- Diseño e implementación de un sistema multiagente. Una vez seleccionada la plataforma se procede a diseñar el código de los agentes necesarios para el desarrollo del prototipo planteado.

- Planificación de las rutas del robot bípedo, es necesario planificar la ruta de los robots para poder alcanzar las posiciones previstas en cada habitación donde se encuentre un paciente.
- Diseño del sistema de reconocimiento de voz del robot bípedo, una vez ya planificado las rutas se procede a elaborar el programa de reconocimiento de voz del robot bípedo hacia el paciente.
- Implementación y simulación del prototipo diseñado. Con el fin de observar el comportamiento de los robots para la ejecución de las tareas, se implementa y simula el prototipo diseñado, como también se realizan las correcciones necesarias para ajustar el prototipo al objetivo deseado.

Capítulo 2

Estado del arte

En este apartado se presentan los avances de las distintas áreas que se estudian cómo son: los sistemas de dialogo (reconocimiento automático del habla), los sistemas multiagentes y la robótica móvil.

2.1 SISTEMAS DE DIÁLOGO

Los sistemas de diálogo (sistemas conversacionales) basados en procesamiento del habla son sistemas informáticos que reciben como entrada frases del lenguaje natural expresadas de forma oral y generan como salida frases del lenguaje natural expresadas asimismo de forma oral [1].

Un sistema de dialogo ideal reconocería el habla espontanea, comprendería enunciados sin restricciones de contenido, proporcionaría respuestas con sentido, gramaticalmente bien formadas y prácticamente adecuadas, respondería con voz completamente natural y seria multimodal [2].

Los sistemas de dialogo están limitados a las restricciones del reconocimiento automático del habla, la comprensión y la limitación de respuestas a dominios específicos. La aplicación es muy amplia, permitiendo el acceso a servicios y control industrial vía teléfono.

Estos sistemas posibilitan la comunicación en ciertos casos donde no sería posible hacer uso del teclado, ratón o la pantalla, como son: sillas de ruedas inteligentes controladas por voz para discapacitados, aplicaciones accesibles desde vehículos (marcación de números telefónicos, tele- servicios, información del tráfico, etc.)

2.1.1 Arquitectura de un sistema de diálogo

El esquema utilizado para el desarrollo de sistemas de dialogo suele englobar una serie de módulos genéricos que deben coordinarse para responder a los requerimientos del usuario [2,3]. La figura 2.1 muestra la arquitectura de un sistema de diálogo basado en el procesamiento del habla.

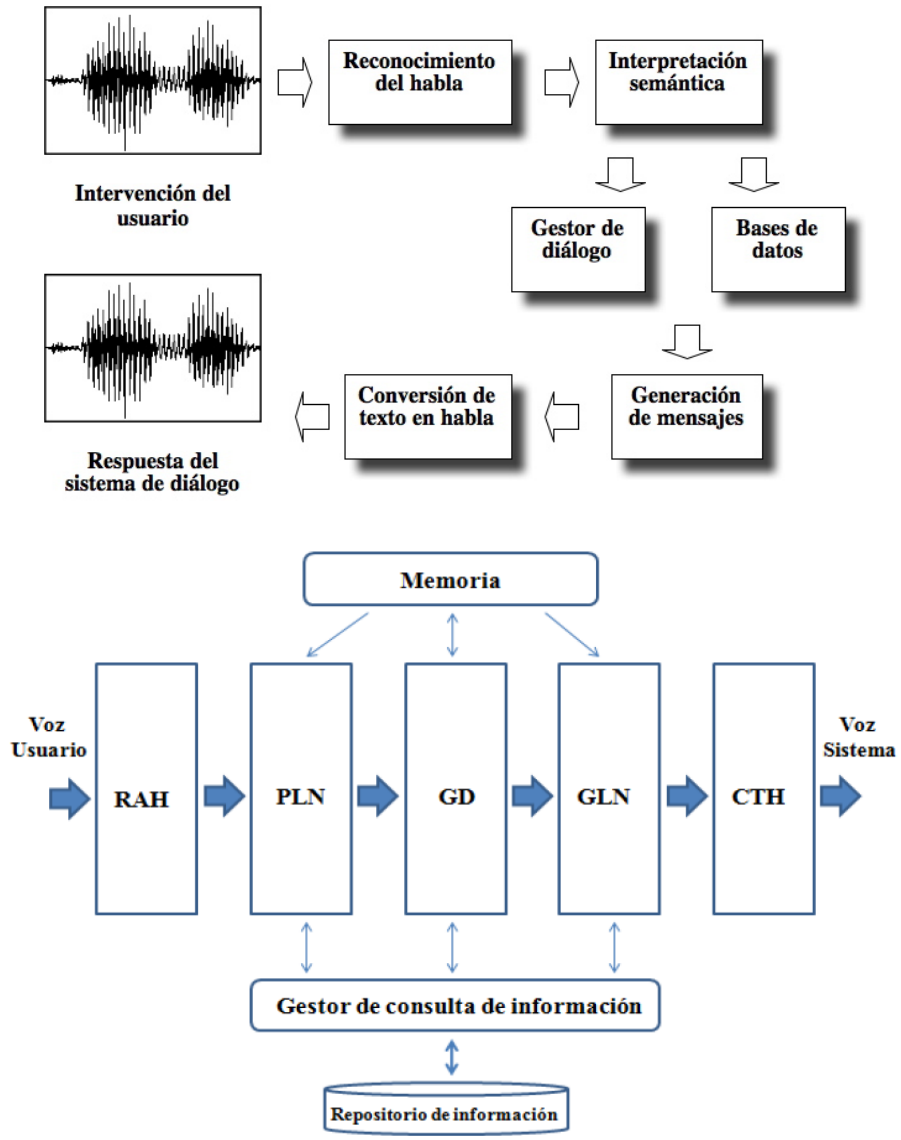


Fig. 2.1. *Arquitectura modular de un sistema de dialogo [6]*

La descripción de los módulos y sus principales características se explican a continuación:

- **Módulo de Reconocimiento Automático del Habla (RAH)**: Procesa la voz de usuario y la transforma en la secuencia de palabras reconocidas con mayor probabilidad.

- **Módulo de Procesamiento del Lenguaje Natural (PLN)**: Extrae el significado de las palabras reconocidas en el módulo anterior, obteniendo la representación semántica (significado) de la frase y expresándolo en términos de un lenguaje especificado para la tarea.

- **Gestor del diálogo (GD):** Decide qué paso debe dar el sistema tras cada intervención del usuario. Puede decirse que este es el módulo fundamental del sistema, pues su finalidad es lograr que la interacción con el usuario sea lo más cómoda e "inteligente" posible. Para ello, se basa en la interpretación semántica de la petición del usuario, la historia del proceso de diálogo, la información disponible en ese punto, el estado actual del sistema, la información obtenida de la base de datos, la estrategia definida, etc.

- **Módulo de Generación del Lenguaje Natural (GLN)**: Tiene como función la generación de una frase, gramaticalmente correcta y en un lenguaje lo más cercano posible al lenguaje natural, que transmita el mensaje generado por el gestor de diálogo.

- **Módulo de Conversión Texto Habla (CTH)**: Transforma la frase de respuesta en señal de audio.

- **Módulo de memoria:** Almacena las representaciones semánticas obtenidas a lo largo de la interacción así como las frases previamente generadas por el sistema, proporcionando esta información histórica a los módulos de procesamiento del lenguaje natural, gestor del diálogo y generación del lenguaje natural. De esta forma, el sistema puede resolver las referencias anafóricas existentes en las frases pronunciadas por los usuarios, puede conocer qué frases ha expresado el usuario previamente, y puede utilizar información contextual (mediante el uso de anáforas y elipsis) durante la generación de las frases.

- **Módulo gestor de consulta de información de la aplicación**: Se encarga de generar las consultas necesarias al repositorio de información, procesarlas y proporcionar la información obtenida al módulo gestor del diálogo.

- **Repositorio de información**: Se trata de bases de datos o páginas web donde se almacena la información necesaria para el sistema de diálogo y que será consultada por el gestor de consulta de información.

2.1.2 Clasificación de los sistemas de diálogo

Los sistemas de dialogo se pueden clasificar según distintos criterios [6]:

- **Iniciativa del diálogo**

Sistemas de diálogo guiados por el sistema: La interacción se realiza mediante turnos concretos y fijados entre pregunta y respuesta. Se restringen las iniciativas del usuario.

Sistemas de iniciativa mixta: Aceptan las interrupciones y negociaciones por parte del usuario, reparten de forma equilibrada el turno de palabra e incorporan mecanismos de detección de incoherencias gramaticales.

Sistemas de diálogo guiados por el usuario: Es el usuario el que lleva la iniciativa de diálogo en cualquier instante de la interacción, permitiéndose al usuario que utilice lenguaje natural y no delimitando los mensajes del sistema (por ejemplo, ¿en qué puedo ayudarle?).

- **Dependencia del hablante**

Dependiente del hablante: Un sistema dependiente del hablante está desarrollado para funcionar con un sólo hablante. Estos sistemas, normalmente, son más fáciles de implementar, más baratos y más precisos, pero no tan flexibles como los sistemas adaptables al hablante o los sistemas independientes del hablante.

Independiente del hablante: Un sistema independiente del hablante está desarrollado para funcionar para cualquier hablante de un determinado tipo (por ejemplo, inglés americano). Estos sistemas son los más complicados de desarrollar, los más caros y la precisión es menor que la de los sistemas dependientes del hablante. Sin embargo, son más flexibles.

- **Tipo de comunicación**

Unimodal (oral): Sólo se utiliza el habla.
Multimodal: Se utilizan varios canales de comunicación. Por ejemplo, dispositivos de entrada como el habla, el teclado, el ratón, el micrófono, la cámara, una pantalla o un PDA y canales de salida para proporcionar información como voz, texto, gráficos o imágenes.

- **Tipo de idioma**

Monolingüe: Permiten interacción sólo mediante un idioma.
Multilingüe: Permiten interacción mediante varios idiomas. Por ejemplo, castellano e inglés.

- **Tipo de discurso**

Reconocimiento de palabras aisladas: En el reconocimiento de palabras aisladas el patrón acústico de cada palabra del vocabulario se almacena como una secuencia temporal de características derivadas, usando LPC, análisis de banco de filtros o alguna otra técnica de análisis del lenguaje. El reconocimiento se lleva a cabo comparando el patrón acústico de la palabra a reconocer con los patrones almacenados, seleccionando la palabra que mejor encaje con la palabra desconocida.

Reconocimiento de palabras conectadas: En el reconocimiento de palabras conectadas, la entrada hablada es una secuencia de palabras de un vocabulario específico, y el reconocimiento se lleva a cabo basándose en la coincidencia de palabras de referencia aisladas. Ejemplos de esto son las cadenas de dígitos conectados donde el vocabulario es un conjunto de 10 dígitos, o el reconocimiento de letras conectadas, donde el vocabulario es el conjunto formado por el abecedario. No hay que confundir esto con el reconocimiento continuo de voz, donde el reconocimiento se basa en unidades lingüísticas denominadas fonemas, sílabas, etc…, lo que supone separar la voz en estas unidades y etiquetarlas subsecuentemente.

Reconocimiento continuo: Un sistema de reconocimiento continuo funciona sobre un lenguaje en el que las palabras están conectadas, es decir, no están separadas por pausas. El lenguaje continuo es más difícil de tratar debido a la variedad de efectos.
Primero, es difícil encontrar el comienzo y el final de las palabras. Otro problema es la coarticulación. La generación de cada fonema se ve afectada por la generación de los fonemas adyacentes, y de modo parecido, el comienzo y final de las palabras se ven afectados por las palabras que les preceden y suceden. El reconocimiento del lenguaje continuo también se ve condicionado por la frecuencia de habla (un discurso rápido suele ser más difícil).

Reconocimiento discreto: Un sistema de reconocimiento discreto funciona con palabras simples, necesitando de una pausa entre la dicción de cada palabra. Esta es la forma más sencilla de reconocimiento para llevar a cabo ya que los puntos de finalización son más fáciles de encontrar y la pronunciación de una palabra no afecta a las demás. De este modo, y ya que las ocurrencias de las palabras son más consistentes, es más fácil reconocerlas.

- **Adaptación**

Sistemas adaptativos: El sistema es capaz de aprender nuevas estrategias comunicativas en función del comportamiento del usuario.

Una línea de investigación actual consiste en la incorporación de las emociones a los sistemas de diálogo. Para ello, el análisis del comportamiento del usuario no debe incluir únicamente las emociones básicas sino también todos sus estados emocionales y cambios en el comportamiento. Por ejemplo, si se detecta que el usuario puede estar enfadado, se inician estrategias de recuperación o se transfiere la llamada a un operador humano, o si se detecta algo similar a la alegría, se aprovecha para ofrecer un nuevo producto.

- **Dominio de la aplicación**

Clasificación según la tarea que realiza el sistema, actualmente existen multitud de aplicaciones de sistemas de dialogo. En este apartado se analiza aplicaciones de voz clasificándolas según su utilidad, la dificultad de la interacción y las tareas que realizan. De este modo se pueden diferenciar varios niveles de menor a mayor complejidad [4,5].

- **Centralita telefónica**

Las aplicaciones de voz más sencillas son aquellas en las que se recibe una llamada y se transfiere automáticamente esa llamada a otro número, como es el caso de las centralitas telefónicas que ponen en contacto al usuario con la persona adecuada:

  - Atos [ATOS] – Centralita telefónica. España.

- **Consulta de información**

Los sistemas de diálogo que se utilizan para obtener información son aplicaciones en las que los datos que se consultan se encuentran en un repositorio de información, como puede ser una base de datos o página web a los que accederá el sistema de diálogo. Estos sistemas se utilizan en diversos ámbitos, que podemos comprobar en los siguientes ejemplos de aplicación:

  - Júpiter [JUPITER] – Predicción meteorológica por teléfono. EE.UU.
  - Dihana [DIHANA] – Información de viajes en tren. España.
  - Pegasus [PEGASUS] – Información de viajes en avión. EE.UU.
  - Conquest [CONQUEST] – Información de horarios en congresos. EE.UU.

Además, hay sistemas que ofrecen la posibilidad de gestionar datos pertenecientes al propio usuario, como sucede en los sistemas que permiten la consulta del correo electrónico a través del teléfono:

  - Athosmail [ATHOSMAIL] – Lectura de correos electrónicos utilizando teléfonos móviles. EE.UU.

- **Reservas**

Existen otros sistemas que además de proporcionar información realizan tareas algo más complejas, como son reservas a través de la aplicación de voz:

  - DARPA Communicator [DARPA] – Interfaz conversacional inteligente para la reserva de billetes de avión. EE.UU.
  - Mercury [MERCURY] – Interfaz conversacional que proporciona información sobre vuelos y permite hacer reservas. EE.UU.

- **Transacciones**

También los sistemas conversacionales pueden llevar a cabo transacciones. Son las aplicaciones relacionadas con el comercio electrónico, como la venta de entradas o de billetes de tren y de avión, y los relacionados con la banca electrónica:

  - Cine-entradas [CINEENTRADAS] – Venta telefónica de entradas de cine. España.
  - Línea BBVA [BBVA] – Sistema que permite solicitar transferencias o domiciliaciones, vender o comprar valores, contratar seguros o realizar aportaciones a planes de pensiones o fondos de inversión. Se comprueba la identidad del cliente mediante una clave personal de acceso. España.

- **Control remoto**

La interacción oral también es útil para el control de dispositivos, especialmente en entornos inteligentes. Un ejemplo de este tipo de sistemas es:

  - MIMUS [MIMUS] – Sistema de diálogo multimodal para controlar un hogar inteligente. España.

- **Tutorización**

Los sistemas de diálogo se emplean también en el ámbito de la educación, particularmente con el fin de mejorar habilidades fonéticas y lingüísticas de los usuarios:

  - LISTEN [LISTEN] – Tutor de lectura. España.
  - VOCALIZA [VOCALIZA] – Sistema para terapias de voz. España.
  - ITSPOKE [ITSPOKE] – Tutor para dar información y corregir errores de aprendizaje. EE.UU.

- **Robótica**

Los sistemas de diálogo se integran en las aplicaciones del mundo de la robótica, como por ejemplo Cogniron [COGNIRON] (Robot cognitivo, Europa).

- **Compañeros virtuales**

Los agentes virtuales y compañeros constituyen la aplicación más compleja de los sistemas de diálogo:

- Ikea. Pregunta a Anna [IKEA] – Compañera relacionada con la tienda IKEA. España.
- Companions [COMPANIONS] – Compañero personalizado multimodal. Europa.
- Olga [OLGA] – Compañera 3D animada multimodal. Suecia.
- Collagen [COLLAGEN] – Asistentes conversacionales y agentes colaborativos. EE.UU.

2.1.3 Voice Extensible Markup Language (VoiceXML)

El lenguaje VoiceXML (Voice Extensible Markup Language) tuvo sus orígenes en 1995. Fue diseñado como un lenguaje de diálogo basado en XML (eXtensible Markup Language), que pretendía simplificar el proceso de desarrollo de aplicaciones de reconocimiento de voz dentro de un proyecto de AT&T llamado PML (Phone Markup Language).

En 1998, el W3C (Consorcio World Wide Web) organizó una conferencia sobre navegadores por voz. En este momento, AT&T y Lucent tenían diferentes variantes de su PML original, mientras que Motorola e IBM habían desarrollado sus propios lenguajes VoxML y SpeechML respectivamente. Muchos otros asistentes de la conferencia también estaban diseñando lenguajes parecidos, por ejemplo, TalkML de HP y VoiceHTML de PipeBeach.

Debido a esta situación, AT&T, IBM, Lucent y Motorola decidieron formar el Foro de VoiceXML para aunar sus esfuerzos. La misión del Foro VoiceXML era definir un lenguaje estándar que los desarrolladores pudieran utilizar para crear aplicaciones de voz. Eligieron XML como base para ello.

En 2000, el Foro de VoiceXML lanzó VoiceXML 1.0 al público. Poco después, VoiceXML 1.0 se presentó al W3C como base para la creación de un nuevo estándar internacional.

Finalmente, VoiceXML 2.0 es el resultado de ese trabajo basado en la colaboración conjunta de las empresas miembros de W3C, otros grupos de trabajo del W3C y el público. La versión 2.1 aparece como recomendación del W3C en junio de 2007. El working draft de VoiceXML 3.0 fue aprobado en diciembre de 2010.

VoiceXML es un lenguaje diseñado para crear sistemas de diálogo mediante voz (tecnología RAH) que incluye reconocimiento de entradas de voz y de entradas DTMF, funciones de telefonía, salida de voz sintetizada y ficheros de audio, grabación de diálogos y control del flujo del diálogo (tecnología de conversión texto-habla Text-To-Spech(TTS)).

Su objetivo es aprovechar las ventajas de las tecnologías Web para el desarrollo de aplicaciones controladas mediante la voz.

Un sencillo ejemplo de código VoiceXML se muestra en la Figura 2.2.

```
<?xml version="1.0" xml:lang="es-ES" encoding="UTF-8"?>

      <vxml version="2.0">
      <form>
            <block>
                  <prompt>
                        ¡Hola Mundo!
                  </prompt>
            </block>
      </form>
      </vxml>
```

Fig. 2.2. *Ejemplo de "hola mundo" en VoiceXML*

Este ejemplo consta de un formulario simple con un bloque que sintetiza y reproduce al usuario la frase "¡Hola Mundo!".

La arquitectura definida para una aplicación desarrollada en VoiceXML contiene los componentes que se muestran en la Figura 2.3.

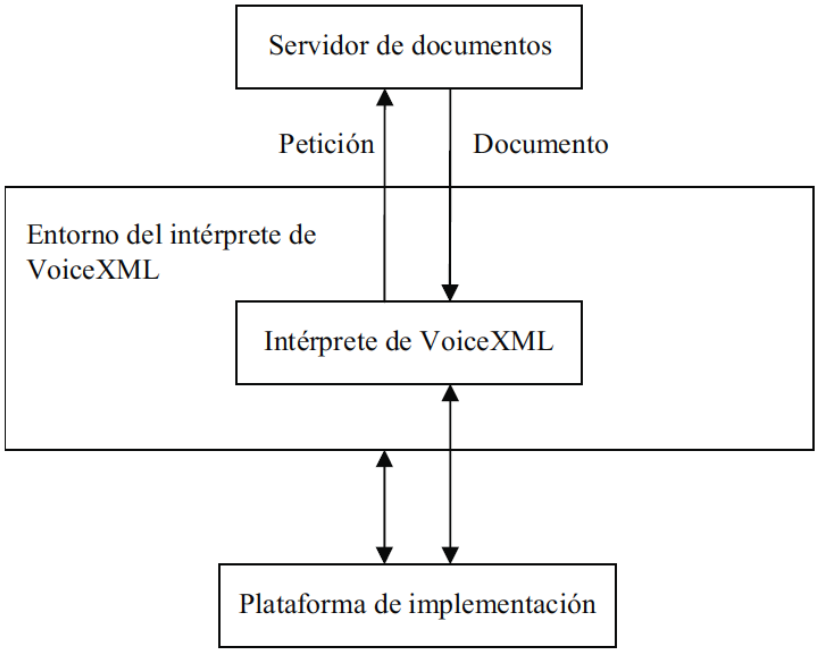


Fig. 2.3. *Arquitectura VoiceXML [7]*

El servidor de documentos (p.e. servidor de páginas web) procesa las peticiones enviadas por la aplicación cliente (intérprete de VoiceXML) a través del intérprete de VoiceXML y proporciona como respuestas documentos VoiceXML que son procesados posteriormente por el intérprete de VoiceXML.

El entorno del intérprete VoiceXML puede monitorizar entradas de usuario en paralelo con el intérprete; por ejemplo, puede detectar una frase especial que inicie la ejecución de un agente de ayuda de alto nivel, o bien, una frase predeterminada para cambiar las preferencias del usuario (p. e. características de la conversión texto-habla).

La plataforma de implementación contiene el hardware telefónico y otros recursos relacionados con el mismo, generando eventos en respuesta a las acciones del usuario (p. e. pulsaciones de botones o comandos orales). Además, es función de la plataforma ejecutar eventos del sistema (p. e. expiración de temporizadores) [7].

El objetivo principal de VoiceXML es dar toda la potencia del desarrollo Web y de la entrega de contenido a los sistemas de diálogo y, de esta forma, liberar a los autores de estas aplicaciones de la programación a bajo nivel y de la administración de recursos.

VoiceXML permite la integración de servicios de voz con servicios de datos mediante el modelo cliente-servidor. Un servicio de voz es visto como una secuencia de diálogos de interacción entre un usuario y una plataforma de implementación. Los diálogos son proporcionados por el servidor de documentos, que puede ser externo a la plataforma de implementación.

El servidor de documentos se encarga de mantener una completa lógica del servicio, el funcionamiento de las bases de datos y las operaciones legales del sistema, además de producir los diálogos.

La entrada de usuario afecta a la interpretación del diálogo y es recogida en *requests* enviadas al servidor de documentos. El servidor de documentos responde con documentos VoiceXML para continuar la sesión del usuario con otros diálogos.

2.1.4 Plataforma VOXEO

Voxeo Corporation (www.voxeo.com) es una compañía que ha desarrollado la plataforma *VoiceCenter*, la cual permite crear, implementar y probar aplicaciones de voz desarrolladas en VoiceXML de forma telefónica y completamente gratuita.

Además, Voxeo incluye una web estática local para el alojamiento de las aplicaciones de voz, un número de teléfono dedicado para su uso durante la prueba de las aplicaciones, registro y depuración en tiempo real de la aplicación, conectividad de voz sobre IP, foros de soporte donde se puede interactuar con la Corporación Voxeo y con otros miembros de la comunidad.

En la red de Voxeo, las aplicaciones de voz siguen el mismo modelo de navegador que las páginas web estándares. La diferencia es que, en vez de HTML, se utilizan lenguajes de marcado para voz (VoiceXML, CCXML o CallXML) para escribir la página fig. 2.4. Cuando estas páginas son procesadas por la red de Voxeo, lo que se devuelve es un audio [6].

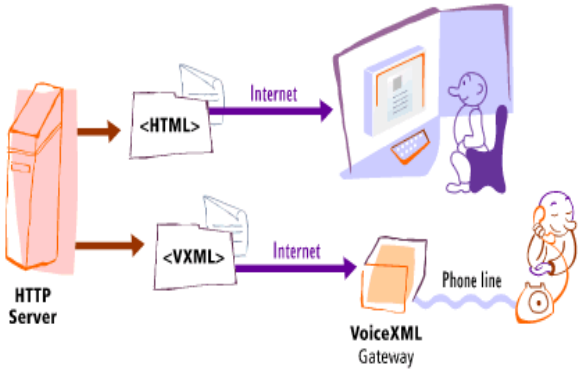


Fig. 2.4. *HTML y VoiceXML (http://philip.greenspun.com/seia/voice)*

La plataforma de sistema de dialogo elegida es Voxeo, debido a que es un estándar abierto que nos ofrece portabilidad, flexibilidad y extensibilidad, además que nos permite interactuar de forma muy amigable con Skype.

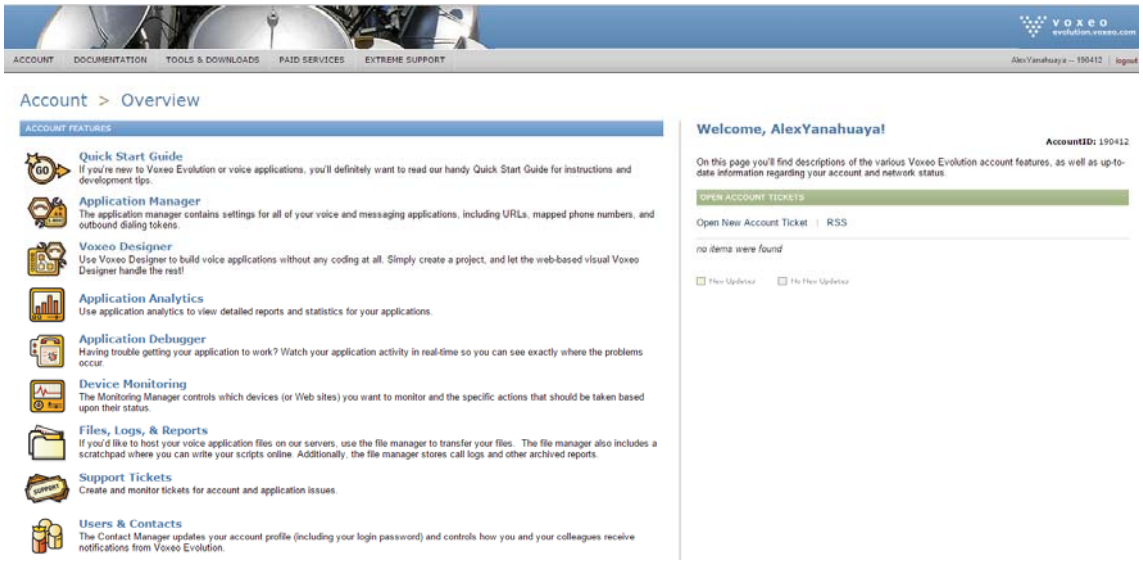


Fig. 2.5. *Plataforma de dialogo Voxeo (https://evolution.voxeo.com/account/login.jsp?action=logout)*

2.2 SISTEMAS MULTIAGENTE

Un agente según Marvin Minsky es un programa computacional con cierta inteligencia [8]; *una entidad software a la que se le puede delegar tareas.*

Un indicativo de que la tecnología de agente ha madurado dentro de una parte significativa en el área de las ciencias computacionales, es el gran número y variedad de aplicaciones que han aparecido recientemente [9]. Estas aplicaciones abarcan: interacción hombre-máquina, comercio electrónico, control industrial, robótica, acceso y recuperación de información, gestión y planificación de recursos, juegos, entornos virtuales, simulación social, interfaces inteligentes, procesos de negocios entre otras.

Como consecuencia de lo anterior, la industria ha comenzado a interesarse en adoptar esta tecnología para desarrollar sus propios productos. La introducción de la tecnología de agentes software en la aplicación de soluciones industriales, requiere de metodologías que asistan en todas las fases del ciclo de vida para desarrollar sistemas computacionales basados en agentes.

Sin técnicas adecuadas para soportar este proceso, tales sistemas no son lo suficientemente confiables, sustentables o extensibles, y serán difíciles de comprender y sus elementos, no podrán ser reutilizados.

La mayor parte de problemas que se requiere resolver utilizando múltiples agentes puede ser resuelta con un sólo agente. Sin embargo, al igual que el concepto de programación descendente, modular y orientada a objetos intenta dividir los problemas en sub-problemas cada vez más sencillos [10], en el ámbito de los agentes se renuncia a la idea de un agente único. Es decir, lo mismo que en la sociedad humana podemos encontrar las actividades claramente divididas para conseguir un elevado grado de especialización y eficiencia, en el caso de los agentes se tiende a éste planteamiento en el que múltiples agentes autónomos interactúan, se coordinan (cooperando, compitiendo o ambas cosas) y todo ello de un forma adaptativa. Por lo que se constituyen sociedades de agentes a las que se les consideran Sistemas Multiagente (SMA).

Cualquier SMA ofrece importantes ventajas sobre los sistemas de centralizados, de la misma forma que la computación distribuida mejora las prestaciones de la centralizada. Parece lógico que los agentes no actúan solitariamente, sino que se localizan en entornos o plataformas con varios agentes. Cada uno de los agentes del entorno tendrá sus propios objetivos, tomará sus propias decisiones y podrá tener la capacidad de interactuar y comunicarse con el resto de agentes [11].

Los entornos o plataformas que soportan varios agentes es lo que en la literatura se conoce con el nombre de Sistema Multiagente o agencias.

Los sistemas multiagentes constituyen un área de continuo crecimiento, con resultados promisorios en diversas áreas de aplicación cono inteligencia artificial, sistemas distribuidos, simulación, etc.

2.2.1 Características de los Sistemas Multiagentes

Un SMA tiene las siguientes características [12]:

- Cada agente tiene información o capacidad incompleta para resolver el problema.
- Cada agente tiene un comportamiento concurrente. Por ejemplo, el agente puede realizar varias tareas e interactuar con varios agentes simultáneamente.
- No hay un control global del sistema.
- Los datos son descentralizados.

De acuerdo a estas características, modelar los sistemas complejos como un SMA tiene las siguientes ventajas:

- Se evitan problemas de limitación de recursos o el riesgo de fallos de manera natural en situaciones críticas que se podrían tener al encomendar un problema complejo a un solo agente.
- Se permite la interconexión e interoperabilidad de sistemas existentes.
- Se obtiene una solución flexible a un problema en una sociedad de agentes que interactúan.
- Se tiene soluciones eficientes a problemas en los que se requiere el uso de información que esta espacialmente distribuida.
- Se permite la extensibilidad ya que el número y la capacidad de los agentes trabajando en el sistema pueden cambiar dinámicamente.

Los agentes inteligentes tienen la capacidad de tener acciones autónomas flexibles en orden para cumplir los objetivos de diseño, donde la flexibilidad significa tres cosas [13]:

Reactividad, son capaces de percibir su entorno y responder de manera oportuna a cambios que ocurren con el fin de satisfacer los objetivos de diseño.

Proactividad, son capaces de mostrar un comportamiento dirigido a un objetivo al tomar la iniciativa con el fin de satisfaces sus objetivos de diseño.

Capacidad social, son capaces de interactuar con otros agentes (posiblemente seres humanos) con el fin de satisfacer sus objetivos de diseño.

2.2.2 Colaboración, coordinación y comunicación entre agentes

Los problemas que intentan solucionar los SMA se resuelve en función del grado de cooperación o colaboración entre los agentes individuales. Es importante enfatizar en el diseño de esta faceta en los agentes, tradicionalmente se han presentado tres estrategias de interacción básicas [12]:

- *Agentes colaborativos,* trabajan juntos con la intención de resolver juntos un problema se podría decir que es un modelo adecuado para la gestión de redes. Esta estrategia es útil para el control de sistemas críticos en donde se debe conocer el estado de equilibrio del sistema. Este tipo de agentes está pensado para sacrificar la utilidad individual a cambio del bien general de todo el sistema.
- *Agentes auto-interesados,* son aquellos que intentan maximizar su propio beneficio sin preocuparse del interés general del sistema. En general, este tipo de coordinación es apropiado cuando agentes van a competir en entornos abiertos.
- *Agentes hostiles,* Este tipo de agentes no tiene una aplicación directa en los agentes que integran al sistema propuesto; ya que se basan en la idea de una utilidad que se ve incrementada con su propia ganancia y con la perdida de otros agentes competidores.

Los agentes que integran a un SMA deben *colaborar* entre ellos para desempeñar sus tareas, esta colaboración suele realizarse mediante un lenguaje de comunicación comprensible por todos los agentes que integran el sistema, como también por otros programas si es necesario.

La diferencia entre un *agente* y un *objeto* (dentro la programación orientada a objetos) es que el ultimo ejecuta métodos de otro objeto siempre que tenga permiso, un agente puede rechazar una petición de otro agente, por lo que deben ser capaces de comunicarse entre sí, para decidir qué acción ejecutar o que datos obtener [11]. El mayor inconveniente para conseguir la comunicación está en el hecho de que existen múltiples lenguajes propios de cada fabricante, lo que no facilita la compatibilidad en sistemas heterogéneos. Los agentes que integran un SMA deben coordinar sus actividades para alcanzar sus objetivos y ser capaces de negociar con otros agentes cuando aparecen problemas como la escasez de recursos.

La *coordinación*, es necesaria para determinar la organización estructural entre un grupo de agentes fig. 2.6, la técnica de coordinación más exitosa para la asignación de recursos y tareas entre una sociedad de agentes es Contract-Net Protocol (CNP) adoptado en las especificaciones

de Foundation for Intelligent Physical Agents (FIPA). En CNP se aplica una estructura de mercado donde los agentes toman dos roles principales, director y contratista [12].

El principio básico en la coordinación es; si un agente no puede resolver la acción que tiene asignada usando su conocimiento local, entonces lo que requiere hacer es descomponer el problema en sub-problemas y buscar a otro agente con recursos suficientes que sean capaces de solucionar los sub-problemas. La asignación de los sub-problemas se soluciona mediante un mecanismo de contratación donde un agente director crea un contrato que luego difunde a otros agentes del sistema que pueden aceptarlo o no.

La *comunicación*, los agentes pueden pasar información a otros agentes para compartir entre ellos los cambios producidos en el sistema o bien para informarse de los cambios previstos en el estado del sistema. Esto generalmente se realiza en los SMA en los niveles de abstracción muy altos, en lugar de usar los tradicionales métodos remotos de paso de mensajes [14].

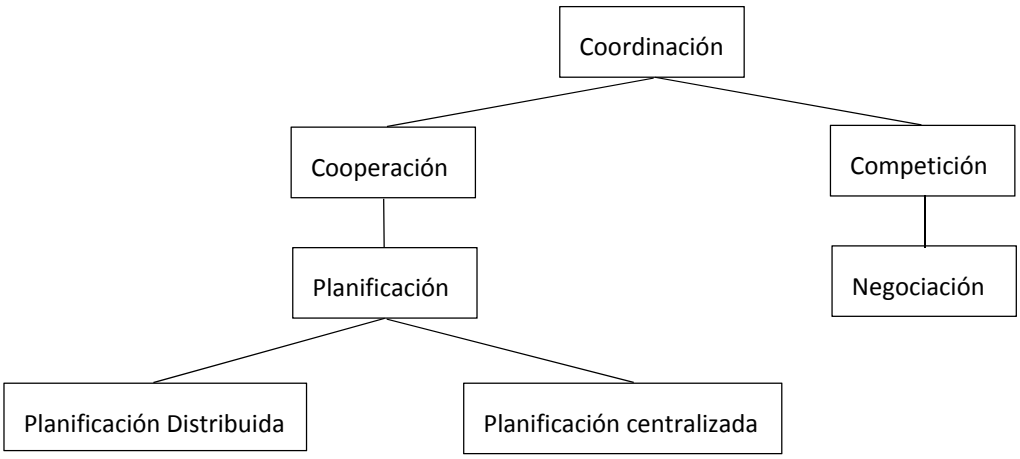


Fig. 2.6. *Una taxonomía de algunas de las diferentes formas en que los agentes pueden coordinar sus comportamientos y actividades [13]*

2.2.3 FIPA (Foundation for Intelligent Physical Agents)

La Fundación para Agentes Físicos Inteligentes (FIPA) es una organización de estándares IEEE Computer Society, se formó con el fin de evitar la proliferación de sistemas de agentes incompatibles y para promover la tecnología basada en agentes y la interoperabilidad de sus normas con otras tecnologías.

FIPA ha producido un conjunto de documentos (llamados “especificaciones del FIPA”) que en conjunto conforman el estándar moderno (o HOW-TO) en términos de tecnología de agentes.

FIPA define un modelo para una plataforma de agente y un lenguaje de comunicación para agentes. Todas las plataformas que quieren estar conformes a FIPA deben seguir este modelo y entender el idioma. El cumplimiento requiere básicamente cuatro características:

- ACC (Agent Comunnication Channel), Canal de comunicación de agente, mecanismo que permite que los agentes (y la plataforma en sí) puedan comunicarse con otros.
- AMS (Agent Management System), Sistema de Gestión de Agente, vía para que los agentes sean registrados en la plataforma y sea accesible para el contacto parecido al servicio de páginas blancas.
- DF (Directory Facilitator), Director facilitador, es una clase de servicio público en el que los agentes publican los servicios que ofrecen, algo parecido al servicio de páginas amarillas.

Soporte por el FIPA ACL (Agent Communication Language) que es un lenguaje común para todos los agentes que se comunican, en esencia son dos: un basado en un entorno de sintaxis Lisp-like (usando una cantidad obscena de paréntesis) y una clara, pura, practica y hermosa basada en la sintaxis XML.

2.2.4 Mensajería Instantánea (IM)

Mensajeria instantánea (IM) es una forma de comunicación de tiempo real entre dos o más personas basado en texto escrito. El texto se transmite a través de ordenadores conectados en red como internet. Permite una fácil colaboración, lo que podría ser considerado más propio de una verdadera conversación que el formato “carta” del correo electrónico. En contraste con el correo electrónico, las partes saben si el interlocutor está disponible.

La mensajería instantánea permite la comunicación instantánea entre varias partes al mismo tiempo, mediante la transmisión de información de forma rápida y eficaz, con la recepción inmediata de reconocimiento o respuesta.

2.2.5 XMPP y Jabber

XMPP (Extensible Messaging and Presence Protocol), es un protocolo abierto inspirado en XML para NRT (Near-real-time), mensajería instantánea extensible (IM) e información de presencia. Es el protocolo núcleo de mensajería instantánea Jabber y tecnología de presencia. El protocolo basado en estándares abiertos, está diseñado para ser extensible y proveer otras características como Voice IP y señalización de transferencia de archivos.

Jabber es un protocolo abierto basado sobre XML para mensajería instantánea y notificación de presencia. Proyecto iniciado en 1998 por Jeremie Miller, lanzado el año 2000, estandarizado por IETF y W3C para mensajería instantánea a través de internet.

Entre sus características tenemos: protocolo abierto y libre, asincrónico, descentralizado, extensible, seguro, flexible, multiredes y multiusuario (salas de conversación).

2.2.6 Plataformas de Sistemas Multiagentes

2.2.6.1 MADKit (Multi-Agent Development Kit) [15,16]

Propuesta por Jacques Ferber y Gutkenecht, es una plataforma multiagente para desarrollar y ejecutar aplicaciones basadas en un paradigma orientado a la organización. Este paradigma multiagente utiliza *agentes*, *grupos* y *roles* como base para construir aplicaciones complejas desarrollado en el contexto AALAADIN. MADKit no fuerza ninguna consideración acerca de la estructura interna de los agentes, de esta manera permite a los desarrolladores implementar libremente sus propias arquitecturas de agentes.

MADKit permite el desarrollo de aplicaciones distribuidas de manera muy simple. Para los programadores, consideraciones acerca de componentes distribuidos básicos (como "sockets" o "ports") son totalmente transparentes. Una aplicación desarrollada en MADKit puede ser ejecutada en forma distribuida sin cambiar ninguna línea de código. Los mecanismos de distribución de MADKit no utilizan las técnicas de RMI o CORBA de acceso remoto, lo cual brinda un modo más eficiente de comunicación [15].

A continuación se presentan algunas características generales de este modelo:

Agente, especificado como una entidad activa que se comunica, la cual posee roles dentro de los grupos.

Grupos, son definidos como conjuntos atómicos de agregaciones de agentes.

Rol, representación abstracta de la función de un agente, servicio o identificación dentro de un grupo.

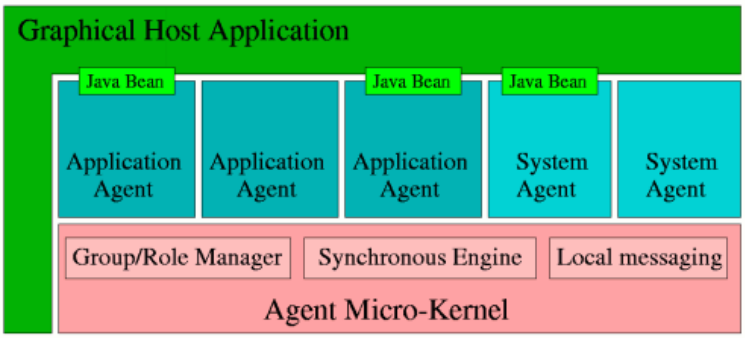


Fig. 2.7. *Arquitectura MADKit [15]*

2.2.6.2 JADE (Java Agent Development Framework) [17,18]

Es un framework de software implementado en el lenguaje java. Es un entorno que simplifica la implementación de sistemas multiagentes mediante una capa de soporte (middle-ware) que respeta las especificaciones FIPA y con un conjunto de herramientas

para el desarrollo y debugging. La plataforma de agentes puede ser distribuida en varias máquinas (las cuales no necesitan compartir el mismo sistema operativo) y la configuración puede ser controlada mediante una interface gráfica remota. La configuración puede incluso ser cambiada en tiempo de ejecución por la creación de nuevos agentes y moviendo agentes de una maquina a otra, cuando es necesario. El sistema solo requiere Java Run Time versión 5 o superior.

La arquitectura de comunicación ofrece mensajes flexibles y eficientes, donde JADE crea y administra una cola de mensajes ACL entrantes. Los agentes pueden acceder su cola mediante una combinación de varios modos: blocking, polling, timeout y pattern matching. El modelo de comunicación FIPA ha sido implementado completo y sus componentes han sido claramente distinguidas y completamente integradas: protocolos de interacción, desarrollo, ACL, lenguaje de contenido, esquemas de codificación, ontologías y finalmente protocolos de transporte. El mecanismo de transporte, en particular, es como un camaleón debido a que se adapta a cada situación, seleccionando transparentemente el mejor protocolo disponible. Otros protocolos pueden ser fácilmente agregados. Integraciones de SMTP, HTTP y WAP ya están planificadas para ser incorporadas. Muchos de los protocolos de interacción de FIPA ya están disponibles y pueden ser instanciados después de definir la dependencia de la aplicación de cada estado del protocolo. SL y ontologías de manejo de agentes están implementados, así como el soporte para lenguajes de contenido y ontologías definidos por el usuario que pueden ser implementadas, registradas con los agentes y automáticamente utilizadas por el framework.

JADE ha sido usado por numerosas compañías y grupos académicos, tales como BT, Telefonica, CNET, NHK, Imperial College, IRST, KPN, University of Helsinky, INRIA, ATOS, y muchos otros. JADE se distribuye en código abierto bajo la licencia LGPL.

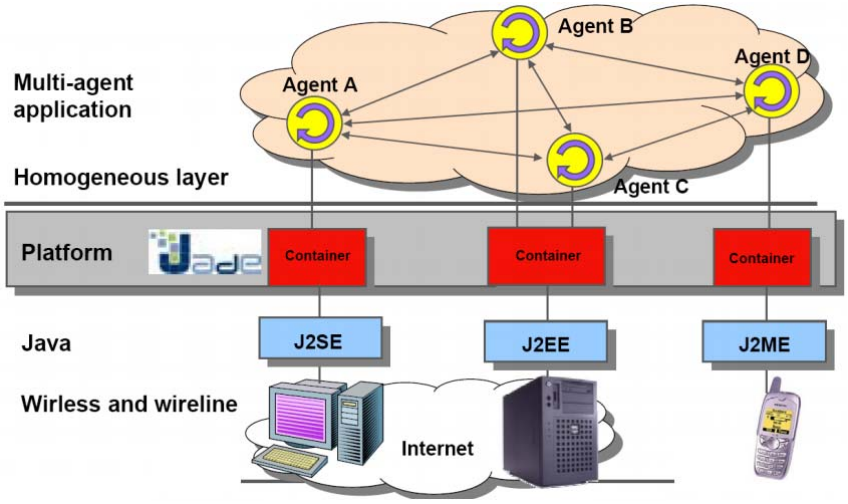


Fig. 2.8. *Arquitectura JADE (*http://webdiis.unizar.es/asignaturas/ISBC/lecciones/11.JADE.pdf*)*

2.2.6.3 SPADE (*Smart Python multi-Agent Development Environment*)

Es una plataforma libre de sistemas multiagente desarrollada en Python el 2005 en la Universidad Politécnica de Valencia por J. Palanca y G. Aranda. La plataforma nació como una prueba de concepto para probar la tecnología de la mensajería instantánea XMPP como protocolo de transporte para los agentes inteligentes. Desde ese momento el proyecto ha seguido creciendo y añadiendo nuevas características aprovechando la flexibilidad del protocolo de mensajería instantánea basado en XML y la cantidad de extensiones desarrolladas para el mismo que son aprovechables dentro del marco de los sistemas multiagente. La plataforma SPADE está basada en un conjunto de estándares, siendo los más destacables FIPA y XMPP/Jabber [20], en la figura 2.9 se puede ver la arquitectura de SPADE.

Las principales características de la plataforma SPADE son:

- Soporte del estándar FIPA mediante el protocolo de mensajería instantánea XMPP (Agentes AMS y DF incluidos)
- Notificación de presencia entre agentes.
- Organizaciones Virtuales basadas en el protocolo de multiconferencia MUC.
- Comunicación P2P entre agentes.
- Invocación remota de servicios entre agentes usando el estándar XML-RPC.
- Procesadores de lenguajes de contenido en SL0 y RDF.
- Modelo de agente BDI basado en Conocimiento, Deseos e Intenciones.
- Modelo de comportamientos: Cíclicos, Periódicos, Timeout, una ejecución, máquina de estados finito y basado en eventos.
- Soporte de comunicación con otras plataformas mediante diversos protocolos de transporte: JADE (vía HTTP o XMPP) y SIMBA.
- Publicación y subscripción de eventos mediante el estándar PubSub.
- Interfaz gráfica basada en web.

Librería de agentes SPADE, es un módulo para el lenguaje de programación Python, donde se tiene una colección de clases, funciones y herramientas para la creación de nuevos agentes SPADE, como se muestra en la figura 2.10.

Mencionar que la plataforma multiagente elegida para el sistema de vigilancia es SPADE, debido a las características que presenta y la programación desarrollada en Python, lo que facilita la interacción con otras tecnologías y la integración con el robot nao.

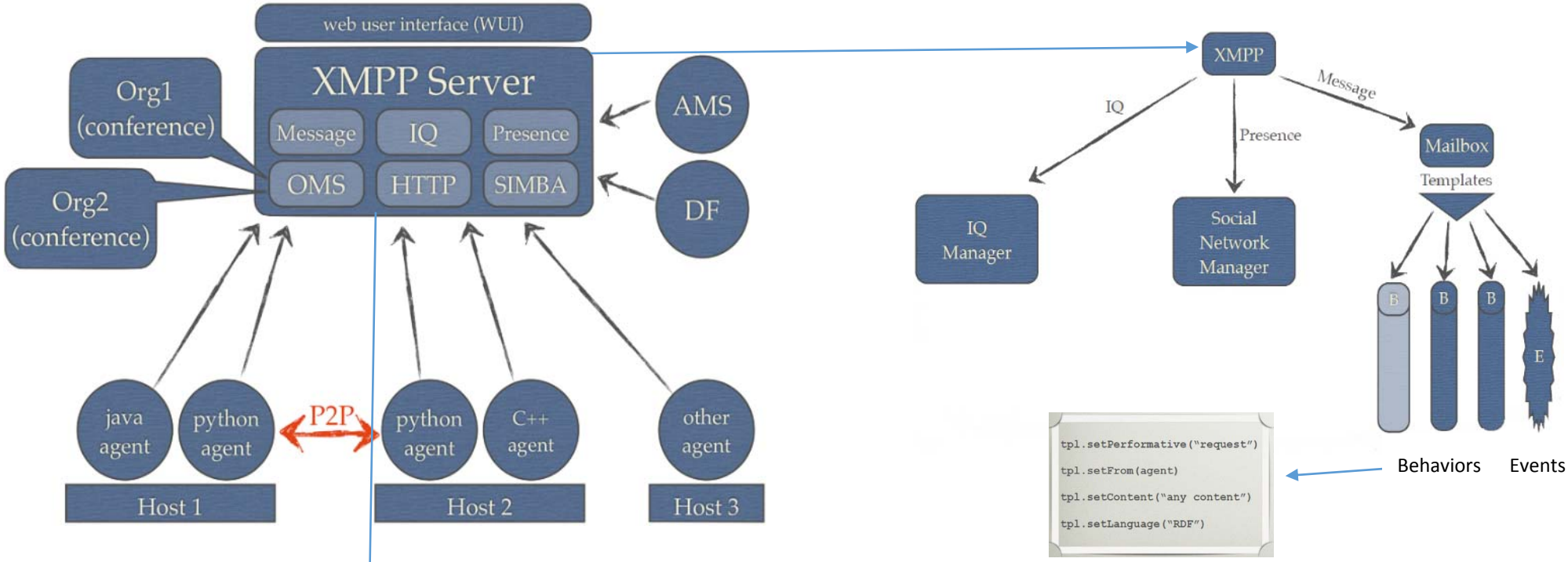


Fig. 2.9. *Arquitectura de SPADE [20]*

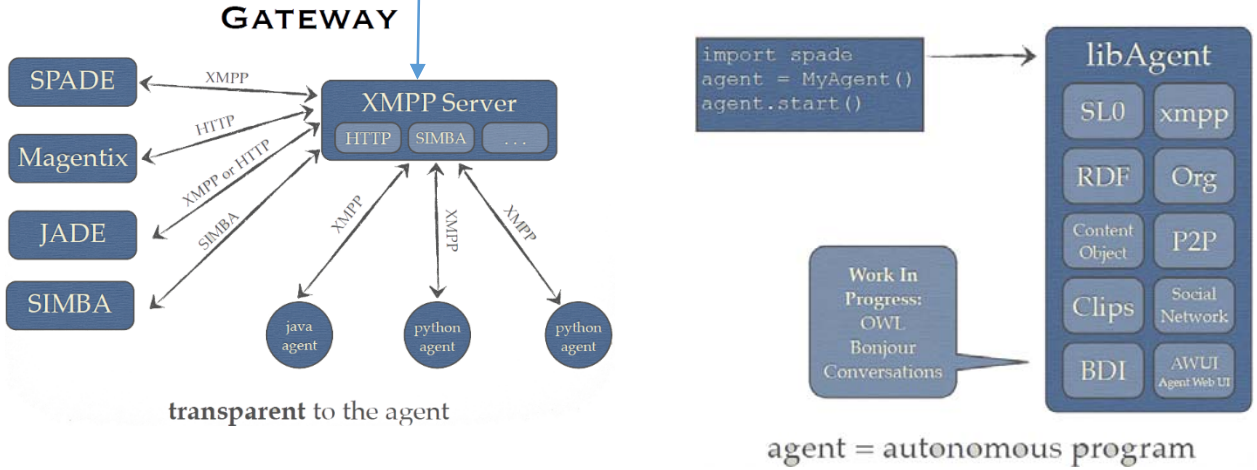


Fig. 2.10. *Librería de agentes SPADE [20]*

2.3 ROBÓTICA

2.3.1 Historia de la robótica

A los largo de la historia el ser humano ha demostrado un interés constante por diseñar y construir artefactos o sistemas que sean capaces de realizar parte del trabajo diario.

Desde la antigua Grecia, los seres humanos han intentado simular las acciones realizadas por los seres vivos, mediante la creación de sistemas automáticos denominados *autómata*, derivando en los que actualmente conocemos como autómatas o robots.

La palabra robot aparece por primera vez en la obra teatral de ciencia ficción R.U.R. (*Rossum's Universal Robots*). Su autor es Karel Čapek (1890-1938), de nacionalidad checa. El hermano de Karel, llamado Josef, sugirió utilizar la palabra checa "robota", cuyo significado es "Trabajo forzado, servidumbre", para referirse a los androides de los que hablaba en el libro. Por otro lado, Josef ya utilizo en 1917 la palabra checa "automat" en su obra Opilec para referirse a estos androides. Los hermanos fueron los que decidieron reemplazar "automat" por "robot" en la obra R.U.R. Debido al éxito del libro de Karel, los traductores adaptaron el término "robota" a "robot" en la edición inglesa, así es como dicha palabra comenzó a ganar popularidad.

Isaac Asimov (1920-1992), escritor de renombre en el campo de la ciencia ficción y la divulgación científica, publico en 1942 el libro *Runaround* en el que se enunciaban "Las tres leyes de la robótica":

- Un robot no puede hacer daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.
- Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la primera ley.
- Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o segunda ley.

De esta forma, mediante sus libros, Asimov difundió al resto del mundo la idea de que cualquier robot se ha diseñado para cumplir con una serie de tareas y ser fiel al ser humano. Estas tres leyes fueron tan bien recibidas que la pura ficción se convirtió en realidad y Asimov ha sido referenciado en numerosos artículos científicos, películas y libros [21].

2.3.2 Clasificación de los robots

2.3.2.1 Según su cronología

Teniendo en cuenta la evolución de la robótica desde los inicios hasta la actualidad se distinguen cuatro generaciones de robots [21]:

- *Primera generación*, Robots manipuladores. Están diseñados para realizar una serie de tareas, previamente programadas por lo que no les afecta el entorno en el que se encuentren; no son sensibles a las variaciones. Los sistemas de control que llevan implantados sin muy simples y se conocen como sistemas de lazo abierto, pues en función de una misma entrada se produce una misma salida que no afectara en el futuro al resto de entradas; no existe ningún tipo de retroalimentación en el sistema. Algunos ejemplos de esta generación son los brazos robóticos de uso industrial. En la figura 2.11 se puede ver un ejemplo de un robot perteneciente a la primera generación.

Fig. 2.11. *Brazo robótico primera generación [http://johannasalguero.blogspot.com.es]*

- *Segunda generación,* Robots enfocados a la teleoperación. En esta generación los robots eran capaces de memorizar y repetir una serie de pasos realizados por un operador humano. De esta forma el robot era capaz de realizar las mismas acciones que un humano sin que este corriese ningún tipo de riesgo. El sistema de control sigue siendo de lazo abierto. La técnica en la que se trabaja con estos robots es conocida como teleoperación como se observa en la figura 2.12.

Fig. 2.12. *Brazo robótico con mando a distancia, segunda generación [http://www.bktronic.fr]*

- *Tercera generación,* Robots sensibles al entorno a través de sensores. Estos robots estaban dotados de una gran cantidad de sensores que les permitieron obtener información del entorno para así poder realizar de una forma u otra cada una de las ejecuciones a través de unos componentes llamados actuadores. Ahora los robots sí que contaban con un sistema de retroalimentación mediante el cual podían saber si su ejecución había sido exitosa o no. Estos sistemas se llaman de lazo cerrado. Las nuevas ejecuciones dependerán tanto de resultados de ejecuciones anteriores como del entorno en el que se encuentre el robot y la entrada que reciba el sistema. Se empieza a trabajar en técnicas de planificación y replanificación de tareas, un ejemplo de esto tenemos en la figura 2.13.

Fig. 2.13. *Robot de tercera generación, Lego mindstorm EV3*

[http://www.lego.com/en-us/mindstorms/build-a-robot/ev3rstorm]

- *Cuarta generación,* robots inteligentes, esta generación corresponde a la de los robots más avanzados en cuanto a tecnología. Los componentes de los robots se han mejorado, ahora tienen cámaras y se disponen de sensores muchos más complejos que permiten dotar de gran cantidad de información al robot consiguiendo que este ejecute acciones complicadas con éxito en tiempo real. Se utilizan técnicas de visión artificial, planificación, aprendizaje, toma de decisiones, etc. Todas estrechamente relacionadas con la inteligencia artificial para conseguir que el robot sea autónomo, a continuación se muestra un ejemplo de robot de cuarta generación.

Fig. 2.14. *Robot de cuarta generación, ASIMO [http://asimo.honda.com/]*

2.3.2.2 Según su estructura

Definida por la configuración general del robot, puede ser metamórfica (flexibilidad funcional de un robot a través del cambio de su configuración por el propio robot). La subdivisión de los robots se la hace según su arquitectura: poliarticulados, móviles, androides, zoomórficos e híbridos. Cabe mencionar que en este trabajo solo se tomara en cuenta los robots móviles.

2.3.2.2.1 ROBÓTICA MÓVIL

En el apartado anterior se fue viendo las generaciones de la robótica, actualmente los robots modernos tienen un mayor grado de movilidad, es decir pueden trasladarse en cualquier ambiente dado para observarlo e interactuar con él. Los robots móviles son un área de estudio de investigación por las universidades.

- **Robots rodantes**

Son aquellos que, como su nombre indica, se desplazan haciendo uso de ruedas. Podemos encontrar varias configuraciones para la posición y el número de ruedas. La primera de ellas sería la configuración de Ackerman, la cual se usa casi exclusivamente en la industria del automóvil. Es la configuración que llevan los coches: dos ruedas con tracción traseras, y dos ruedas de dirección delanteras. Esta configuración está diseñada para que la rueda delantera interior en un giro tenga un ángulo ligeramente más agudo que la exterior, y evitar así el derrape de las ruedas.

Fig. 2.15. *Robot con la configuración de Ackerman*

[http://www.maxonmotor.es/maxon/view/application/PIPE-INSPECTION-ROBOT-AB]

También es frecuente encontrar distribuciones de ruedas montadas en configuración diferencial (triciclo) se presenta como la más sencilla de todas. Consta de dos ruedas situadas diametralmente opuestas en un eje perpendicular a la dirección del robot más una rueda loca. Estas ruedas no llevan asociadas ningún motor, giran libremente según la velocidad del robot. Además, pueden orientarse según la dirección del movimiento.

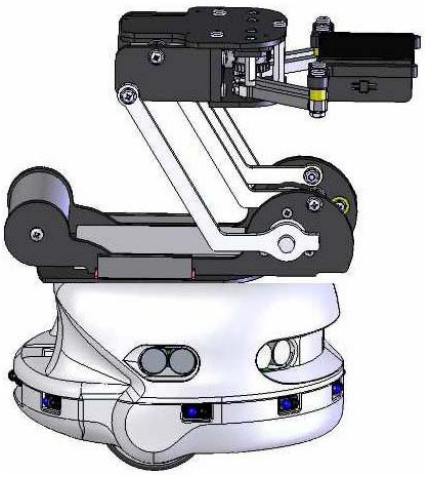


Fig. 2.16. *Robot Khepera3*

[http://www.k-team.com/mobile-robotics-products/old-products/khepera-iii]

Existen algunos casos especiales en los que se usan otras configuraciones que dotan al robot de mejor adaptación a terrenos difíciles. En estos casos los algoritmos de control de movimiento adquieren una mayor complejidad, proporcional al número de elementos direccionales.

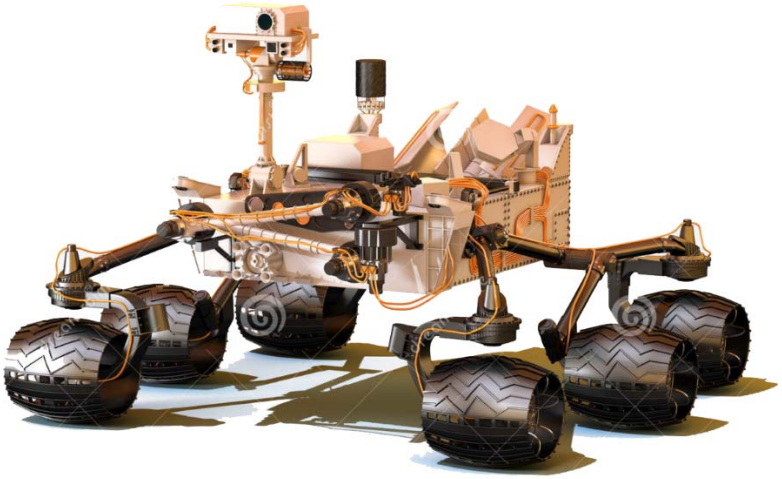


Fig. 2.17. *Robot Curiosity Mars Rover*

[http://www.nasa.gov/mission_pages/msl/index.html]

El último grupo de robots que aquí convendría nombrar es el de los robots que se desplazan mediante la utilización de cadenas. Para la realización de los giros es necesario que ambas cadenas giren en sentidos opuestos dependiendo de cuál sea el sentido de giro final del robot, de modo parecido al que se utiliza en la configuración diferencial.

Fig. 2.18. *Robot oruga Lego Mindstorm EV3*

[http://www.lego.com/en-us/mindstorms/build-a-robot/ev3rstorm]

- **Robots andantes**

Los robots andantes son aquellos que basan su movilidad en la simulación de los movimientos realizados por los seres humanos al andar. Estos robots están provistos de dos patas con varios grados de libertad, a través de las cuales son capaces de desplazarse manteniendo el equilibrio. No obstante, este tipo de robots son los más inestables que se encuentran en la actualidad, y los esfuerzos se aúnan en intentar conseguir técnicas de estabilidad y equilibro que eviten que el robot se desestabilice y vuelque en situaciones adversas, como al correr, subir gradas, evitar obstáculos, etc. Sin embargo, podemos encontrar modelos avanzados de este tipo de robot que son capaces de caminar, gatear, bailar, trotar y practicar deportes, en la figura 2.19 se puede ver un robot andante.

Fig. 2.19. *Robot Humanoide Robonova 2*

- **Robots reptadores**

Creada basándose a las serpientes, la forma de desplazarse es imitación a la usada por estos animales, formado por un número elevado de secciones que pueden cambiar de tamaño o posición de forma independiente de los demás pero coordinada. Las características que se buscan son: flexibilidad, versatilidad y adaptabilidad.

Fig. 2.20. *Robot Reptor, Lego mindstorm EV3*

[http://www.lego.com/en-us/mindstorms/build-a-robot/ev3rstorm]

- **Robots nadadores**

Estos robots son capaces de desenvolverse en el medio acuático, utilizados en áreas de exploración submarina o incluso como herramienta de rescate en lugares complicados de acceder.

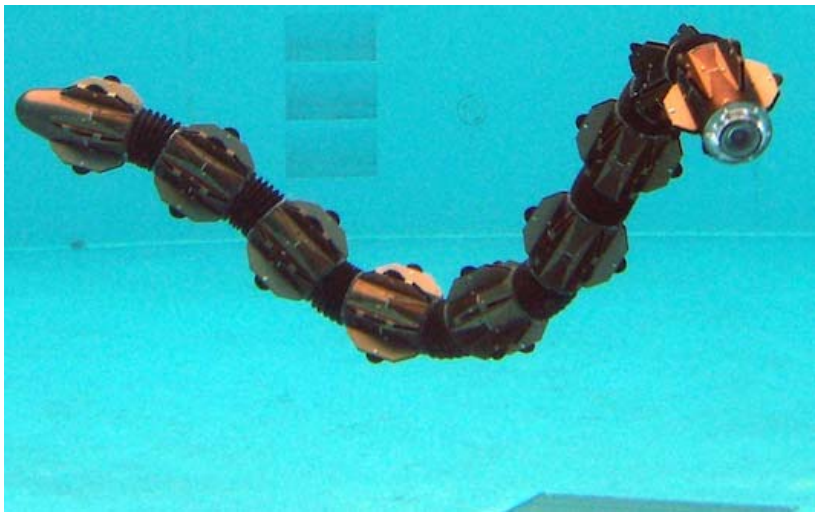


Fig. 2.21. *Robot nadador autómata ACM-R5H*

[http://www.hibot.co.jp/en/products/robots_1/acm-r5h_33]

- **Robots voladores**

Capaces de desplazarse por el aire, del mismo modo que un avión o un helicóptero. Para ello, incorporan una serie de hélices que se encargan de generar la fuerza necesaria para elevar el robot y de realizar los giros pertinentes para seguir una determinada trayectoria.

Fig. 2.22. *Drone hibrido*

[http://app.emaze.com/@AOZLRFIW/dron1#1]

Respecto a la elección del robot móvil, se optó por trabajar con los robots andantes, en especial con el robot bípedo NAO que fue adquirido por la Universidad Politécnica de Valencia.

2.3.3 Robot NAO

El robot humanoide NAO H25 dispone de múltiples sensores (micrófonos, cámaras, giroscopio, acelerómetro, etc.) y actuadores (altavoces, motores, etc.). Tiene 25 grados de libertad, por lo que posee una amplia movilidad. Sus creadores son la empresa francesa, Aldebaran Robotics. Tal es el éxito que ha obtenido este robot a nivel internacional que se ha convertido, entre otras cosas, en el robot más utilizado de la Robocup [25].

Las características más importantes del robot NAO se muestra en la tabla1.

Altura	58 cm
Peso	4,3 kg
Energia	Bateria Li-Ion, 6 celdas en serie, Vnom=21,6 V, I=2A
Autonomía	60 minutos (uso activo), 90 minutos (uso normal)
Grados de libertad	21 hasta 25
CPU	Intel Atom @ 1.6 GHz
Construido en el SO	Linux
SO compatible	Windows, Mac OS, Linux
Lenguaje de programación	C++, Python, Java, MATLAB, Urbi, C, .Net
Visión	Dos cámaras 1280x960 HD
Conectividad	Ethernet, Wi-Fi
Sensores	Giroscopio, Acelerómetro, Bumpers, Sonar, I/R

Tabla 1. *Características técnicas del robot NAO, ver apéndice 1*

El control de Nao desde el punto de vista de sus articulaciones se divide en ocho partes diferentes. A continuación se describe brevemente lo que es capaz de hacer cada una de ellas [21].

- **Head:** La cabeza del robot puede moverse de izquierda a derecha (*yaw*) y de arriba abajo (*pitch*).

- **LArm:** El *Left arm* o brazo izquierdo permite mover el hombro (*shoulder*) sobre sí mismo (*roll*) o de arriba a abajo (*pitch*) y el codo (*elbow*) sobre sí mismo (*roll*) o de izquierda a derecha (*yaw*).

- **LHand:** La *Left hand* o mano izquierda permite abrir y cerrar los dedos de la mano (*LHand*) y girar la muñeca sobre sí misma (*LWristYaw*).

- **LLeg:** La *Left leg* o pierna izquierda permite subir y bajar ambas piernas (*LHipYawPitch*), girar y mover la pierna izquierda (*LHipRoll, LHipPitch*), mover la rodilla de arriba abajo (*LKneePitch*) y mover y girar el tobillo (*Ankle*).

- **RArm:** El *Right arm* o brazo derecho permite mover el hombro (*shoulder*) sobre sí mismo (*roll*) o de arriba a abajo (pitch) y el codo (*elbow*) sobre sí mismo (*roll*) o de izquierda a derecha (*yaw*).

- **RHand:** La *Right hand* o mano derecha permite abrir y cerrar los dedos de la mano (*RHand*) y girar la muñeca sobre sí misma (*RWristYaw*).

- **RLeg:** La *Right leg* o pierna derecha permite girar y mover la pierna derecha (*RHipRoll, RHip- Pitch*), mover la rodilla de arriba a abajo (*RKneePitch*) y mover y girar el tobillo (*ankle*). El movimiento de *RHipYawPitch* es el mismo que el de *LHipYawPitch*, pues se mueven las dos piernas simultáneamente. Por lo tanto, comparten el mismo valor y podemos utilizar indistintamente uno u otro.

- **Torso:** El torso alberga los cuatro sónar, el giroscopio y el acelerómetro del robot. Estos Elementos le permiten ser capaz de prevenir el choque con obstáculos o equilibrarse entre otros. En la figura 2.23 se señalan los cuatro sónar, siendo dos emisores (*transmitter*) y dos receptores (*receiver*). Los primeros lanzan el sónar y los segundos recogen los datos obtenidos de a cuánta distancia se encuentra el objeto más cercano.

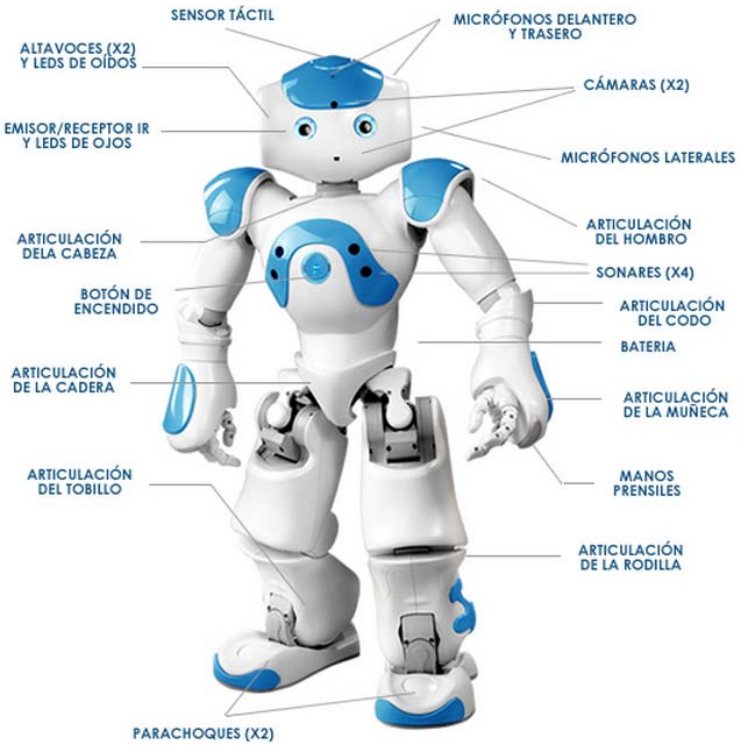


Fig. 2.23. *Sensores y actuadores del Robot NAO [25]*

2.3.4 Simulador Choregraphe [25]

Este simulador ha sido desarrollado por Aldebaran Robotics con el fin de facilitar al usuario la forma de trabajar con los robots Nao. En Choregraphe se pueden diseñar y ejecutar distintos comportamientos, probarlos sobre un robot virtual o sobre el real, acceder a la cámara del robot, etc. Es una herramienta fundamental para conocer el estado del robot y trabajar independientemente sobre cada una de sus articulaciones.

El punto fuerte de Choregraphe es su interfaz, sencilla e intuitiva. Para que el robot del simulador realice un movimiento podemos combinar los comportamientos ya desarrollados del menú de la izquierda o crear otros nuevos mediante la opción de "animación". Para ejecutarlos simplemente hay que pulsar el botón Run. Al trabajar siempre sobre diferentes ventanas todo está muy bien ordenado y la flexibilidad que ofrece es inmensa (ver fig. 2.24). A través del menú de la parte superior se puede acceder a todo el resto de información que esta herramienta proporciona. Para aprender a trabajar con Choregraphe, Aldebaran Robotics elaboró una guía de usuario con diferentes tutoriales de dificultad creciente.

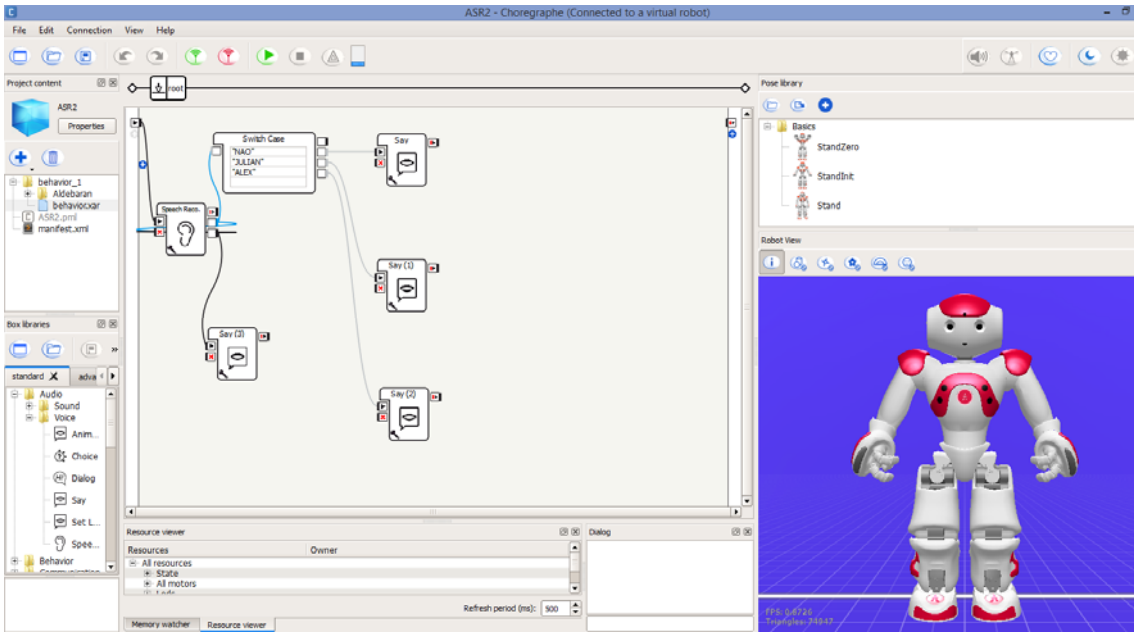


Fig. 2.24. *Simulador Choregraphe para el Robot NAO [25]*

2.3.5 NaoQi [25]

NaoQi es el framework desarrollado por Aldebaran Robotics que sirve de puente entre la computadora y NAO, controlando la ejecución de cualquier comportamiento y capturando la información de todos los sensores con el único objetivo de que todo se produzca con éxito y sin incidentes. Este framework se encuentra dentro del robot, a modo de cerebro, y también en el ordenador desde el que se envían las órdenes, dividido en seis grandes áreas claramente diferenciadas por su funcionalidad, las cuales se muestran a continuación:

- ***NAOqi Core:*** contiene un conjunto de módulos relacionados con los comportamientos genéricos del robot, la memoria del robot y los ficheros de configuración.
- ***NAOqi Motion:*** contiene funciones que permiten al robot desplazarse, controlando el equilibrio, el estado de las articulaciones y la prevención de choques o caídas.
- ***NAOqi Audio:*** contiene los módulos que le permiten al robot interactuar con el usuario mediante el habla, la reproducción de sonidos y el reconocimiento de voz.
- ***NAOqi Vision:*** contiene los módulos que permiten grabar desde las cámaras del robot y realizar reconocimiento de caras y de objetos.
- ***NAOqi Sensors:*** contiene los módulos que proporcionan información sobre el estado del robot: sensores, batería, pose en la que se encuentra, etc.
- ***NAOqi Trackers:*** contiene las funciones que permiten seguir a una persona a través de reconocimiento facial, a un objeto o a una marca.

A través del API de NaoQi se pueden desarrollar nuevos módulos en 8 lenguajes de programación Python, C++, .NET (C#, Visual Basic, F#), Java, Matlab y Urbi, en vez de utilizar la opción de Choregraphe. De esta forma, se puede conseguir una mayor flexibilidad y personalización de estos y además será posible adentrarse mucho más en el robot.
Es habitual encontrarse códigos que combinan el API de NaoQi no sólo con las librerías de Python o C++, sino incluso con paquetes de ROS (Robot Operating System).

En el sistema de vigilancia propuesto, la API de NaoQi se desarrollara en el lenguaje de programación Python, debido a que nos facilita en la interacción con el sistema multiagente SPADE (desarrollado en Python).

Capítulo 3

Diseño e Implementación del sistema

Para el diseño del prototipo se elaboraron dos versiones. La primera versión es una versión simplificada del sistema objetivo, donde se le va dando al agente revisor (robot NAO) una secuencia de órdenes básicas para el cumplimiento de la tarea a realizar. Por su parte en la segunda versión, es una versión mejorada que constituye un prototipo completo del sistema objetivo, donde el encargado simplemente le da una orden compleja al agente gestor y este se coordina con el agente revisor para el cumplimiento de la tarea a realizar.

La razón de desarrollar el sistema en dos versiones se debe a que en la primera versión se pretendía validar la interconexión de todos los componentes del sistema con un ejemplo sencillo y en el caso de la segunda versión se pretendía desarrollar un prototipo funcional y completo sobre un dominio de aplicación especifico, en este caso un sistema de vigilancia de personas mayores o con incapacidad usando sistemas multiagente y robots bípedos controlados mediante voz.

3.1 DISEÑO DE LA PRIMERA VERSIÓN DEL PROTOTIPO

En esta primera versión el usuario va interactuando con el sistema de dialogo para indicarle la secuencia de órdenes a ejecutar. El funcionamiento se explica a continuación.

El usuario llama a través de Skype, este se conecta al sistema de dialogo Voxeo e interactúa con la plataforma de sistemas multiagente SPADE, el cual está conformado por los agentes gestor y revisor. El agente gestor está encargado de habilitar el funcionamiento del agente revisor que en nuestro caso es el robot NAO, como también el agente revisor está encargado de ejecutar las instrucciones del usuario como: calcular, mover, hablar y adoptar posición.

La figura 3.1 muestra todo el proceso elaborado en el diseño de la primera versión del prototipo, indicar que se trabajó con el sistema operativo Linux para todo el desarrollo del sistema de vigilancia.

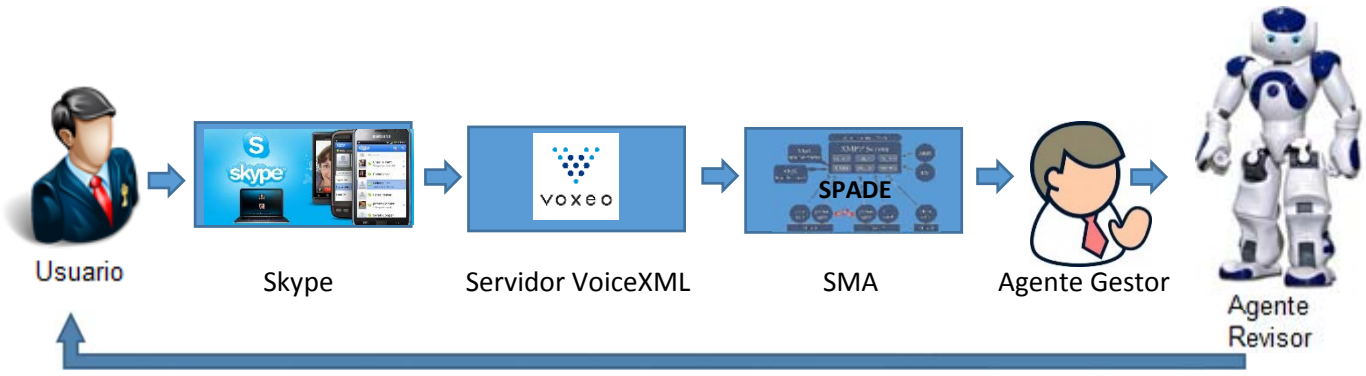


Fig. 3.1. *Diagrama de bloques utilizado en el diseño de la primera versión*

El diseño del prototipo se dividió en tres partes que se detallan a continuación:

3.1.1 Diseño del código en la plataforma de voz seleccionada

En los sistemas basados en Reconocimiento Automático del Habla (RAH), las gramáticas son el mecanismo fundamental de entrada de datos, ellos deciden que frases pronunciadas por los usuarios pueden ser reconocidas. Para esto se tiene previsto compilar gramáticas en el formato JSGF (Java Speech Grammar Format), permitiéndonos combinar los procesos de RAH y comprensión de frases. También nos indica cómo deben ser interpretadas determinadas secuencias y cuáles son las frases reconocibles.

Se trabaja con dos estrategias de interacción: Sistemas de iniciativa mixta y Sistemas de diálogo guiados por el sistema. Al inicio se trabaja con el sistema de iniciativa mixta donde ambos interlocutores (usuario y sistema) dirigen la conversación (p. e. "*Este es el servicio de manejo del robot NAO. Se tiene 4 acciones, calcular posición, moverse, hablar y adoptar una posición*").

Después se utiliza la estrategia dialogo guiado por el sistema, para detectar problemas en la interacción, por ejemplo cuando el usuario no habla (p. e. "*Yo no escucho nada. Inténtelo otra vez por favor*"), la frase pronunciada no es reconocida por ninguna gramática activa (p. e. "*Yo no reconozco la acción. Inténtelo otra vez por favor*"). En este caso el sistema toma la iniciativa en el dialogo para guiar al usuario en la interacción, solicitándole que responda de forma concreta a sus prompts.

Una vez seleccionada la plataforma de reconocimiento de voz, que en nuestro caso es **Voxeo**, empezamos a programar la secuencia de órdenes elaboradas en VXML para el agente revisor, tal como se indica en la figura 3.2.

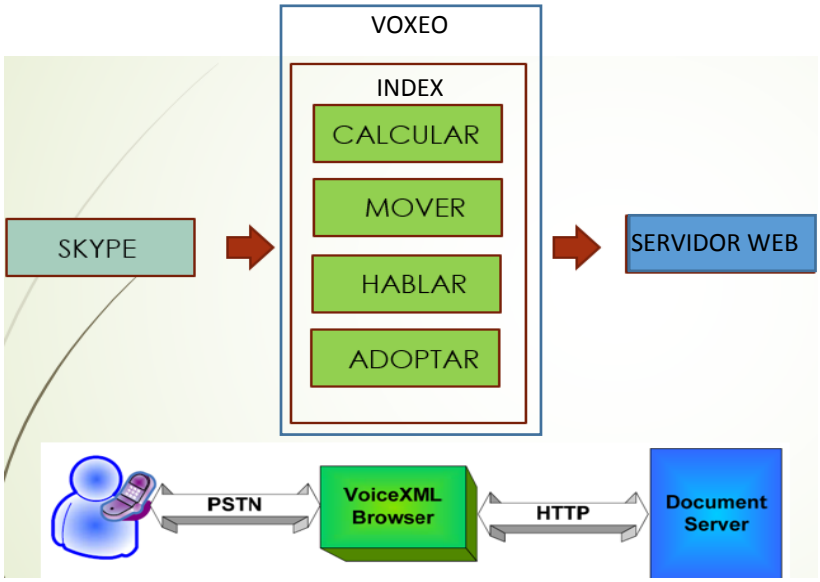


Fig. 3.2. *Diagrama de bloques utilizado para programar el código VXML del agente revisor*

Como se puede observar en este diagrama de bloques, tenemos como entrada la voz del usuario encargado de dar las ordenes al robot bípedo a través de Skype mediante un móvil o una computadora, la cual es reconocida por la plataforma de voz. Dentro de la plataforma de voz se tiene al inicio una gramática índex encargada de la interacción para reconocer frases del usuario como calcular, mover, hablar y adoptar. Las cuales nos sirven para el manejo del agente revisor. A continuación se explica el funcionamiento de cada uno de estos controles:

- *Calcular*, encargada de convertir la distancia o el ángulo de giro en movimiento (número de pasos del robot bípedo).
- *Mover*, encargado de los movimientos del robot adelante, atrás, giro derecha, izquierda y número de pasos a caminar.
- *Hablar*, se usa para la interacción del agente revisor con el usuario.

- *Adoptar*, simplemente adopta una posición el robot bípedo, como posición inicial, pararse, sentarse, etc.

3.1.2 Diseño del Sistema Multiagente

Una vez estudiada las distintas plataformas de sistemas multiagente, se optó por el uso de la plataforma de agentes SPADE tal como se mencionó en el capítulo 2 del estado del arte.
Los agentes SPADE están básicamente compuestos por un mecanismo de conexión a la plataforma, un manejador de mensajes y un conjunto de diferentes comportamientos (behaviors) que incorporan el know-how del agente. Cada agente necesita un identificador llamado Jabber ID (JID) y una contraseña válida para establecer una conexión con la plataforma. El JID (compuesto por un nombre de usuario, @, y un servidor de dominio) será el nombre que identifica a un agente en la plataforma (ej. `myagent@myprovider.com`), la dirección de agente (que es otro campo importante en el identificador de agente) sería la JID de la cuenta XMPP (account (ACC)) de la plataforma (ej. `xmpp://acc.myprovider.com`):

```
"agent1@"+robotIPSPADE,["xmpp://agent1@"+robotIPSPADE]
```

- *Conexión a la plataforma*, las comunicaciones en SPADE son manejadas internamente por medio del protocolo Jabber. Este protocolo tiene un mecanismo para registrar y autenticar a los usuarios en un servidor Jabber.
  La plataforma SPADE incluye un componente de servidor Jabber. Este proceso se activa automáticamente como parte del proceso de registro del agente.
- *Manejador de mensajes,* cada agente SPADE tiene un mensaje interno como componente manejador. Este manejador de mensajes se encarga de enviar los mensajes.
- *Comportamientos (Behaviors),* un agente puede ejecutar varios comportamientos simultáneamente. Un comportamiento es una tarea que un agente puede ejecutar utilizando patrones que se repiten. SPADE ofrece algunos comportamientos predefinidos:

  - Cíclico y Periódico: útiles para la realización de tareas repetitivas.
  - Una vez (One-Shot) y tiempo de espera (Time-Out): se puede utilizar para tareas ocasionales.
  - Máquina de estados finitos: para construir comportamientos complejos.
  - Eventos: responde a algún evento que el agente percibe.

En nuestro sistema multiagente se ha definido dos tipos de agentes:

- Agente gestor, encargado de gestionar, monitorizar y supervisar a todos los agentes revisores.
- Agente revisor, conformado en parte por el robot bípedo (Robot NAO), está encargado de ejecutar una secuencia de órdenes instruidas por el usuario como mover, hablar y adoptar posición.

En primer lugar importamos SPADE (*Import spade*), luego definimos la clase para el agente gestor (agente 2) que dará la orden agente revisor (agente 1) de empezar con la ejecución de las tareas.

Como se puede observar en el código, se extiende de la clase ***spade.Agent.Agent,*** que es la clase base para todo agente SPADE. También destacar que llamamos un método definido como ***_setup***, que es donde se coloca el código de inicialización del agente.

```
############AGENTE GESTOR ########################
        class MyAgent2(spade.Agent.Agent):
            def _setup(self):
                self.addBehaviour(self.SendMsgBehav())
                print "\nIniciando el Agente 2. . ."
```

Declaramos una clase SendMsgBehav que hereda de ***spade.Behaviour.OneShotBehaviour*** para trabajar con un comportamiento de un solo ciclo para realizar la tarea dada y luego terminar.
En el código que se expone a continuación, el agente gestor habilita el funcionamiento del agente revisor con el envío de un mensaje.

```
    class SendMsgBehav(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            msg = spade.ACLMessage.ACLMessage()    # Instanciando el mensaje
            msg.setPerformative("inform")          # Ajuste de "inform" para representacion FIPA
            msg.addReceiver(spade.AID.aid("agent1@"+robotIPSPADE,["xmpp://agent1@"+robotIPSPADE]))
            msg.setContent("Empezar con la ejecución de revisar") #Ajustamos el contenido del mensaje
            self.myAgent.send(msg)                 #Se envía el mensaje
            print "\nEl agente 2 a enviado el mensaje al agente 1: "
            print str(msg.getContent())
            print "Finalizando el Agente 2 . . .\n\n"
```

A continuación se muestra una parte del código para el agente revisor, el cual en esta primera versión solo espera la orden del agente gestor para inicializar la tarea a realizar. Se explican dos métodos utilizados:

- Método **onStart** similar al método **_setup** de la clase agente gestor, la cual se ejecuta antes de comenzar la iteración principal y se utiliza como código de inicialización.
- Método **process,** encargada de llevar a cabo todas las acciones realizadas por el agente revisor.

```
############AGENTE REVISOR#####################
class MyAgent1(spade.Agent.Agent):

    class MyBehav(spade.Behaviour.OneShotBehaviour):
        def onStart(self):
            print "\nIniciando el agente NAO (Agente 1) . . .\n"
            #print "\n Iniciando el Agente 1 . . .\n"
        def _process(self):
            ##probando el Agente 1 que recibe mensaje del Agente 2##
            msg = self._receive(block=True,timeout=10)
            print "\nEl agente NAO (Agente 1) tiene un mensaje: "
            print str(msg.getContent())
            print "\nProcesando...."
            ……
            ……
    def _setup(self):
     #######SPADE#########
          template = spade.Behaviour.ACLTemplate()
          template.setSender(spade.AID.aid("agent2@"+robotIPSPADE,["xmpp://agent2@"+robotIPSPADE]))
          t = spade.Behaviour.MessageTemplate(template)
          #self.msg = spade.ACLMessage.ACLMessage()
          #print "\n Mi agente NAO iniciando (Agente 1) . . .\n"
          b = self.MyBehav()
          self.addBehaviour(b,t)
```

Por último se dispone de la función principal donde instanciamos la clase agente, la cual contiene dos parámetros: el primero es el JID (Jabber Identification) del agente, que contiene el nombre del agente (antes de la @) y el nombre de la plataforma de agentes (después de la @), en segundo lugar está la contraseña de Jabber para este agente en particular. En el código se remarca en negrita el JID y la contraseña.

El agente se inicia con el método ***start(),*** que en realidad lo que hace es llamar al método ***_setup()****, y* finaliza con el método ***stop()***.

```
if __name__ == "__main__":
   print "Usando python almotion_moveto.py robotIP (optional default: 127.0.0.1)"
   agent1=MyAgent1("agent1@"+robotIP,"secret")
   agent1.setDebugToScreen()
   agent1.start()
   agent2=MyAgent2("agent2@"+robotIP,"secret")
   agent2.setDebugToScreen()
   agent2.start()
   ……
   ……
   agent1.stop()
   agent2.stop()
```

3.1.3 Control del Robot Bípedo

En el capítulo del estado del arte se vieron los distintos tipos de robots móviles. El robot bípedo elegido para el desarrollo de nuestro prototipo fue el robot Nao, debido a que presenta una gran variedad de características como son: sonares, cámaras 2D, micrófonos, altavoz, sintetizador de voz, etc. Dichas características pueden verse en la figura 3.3.

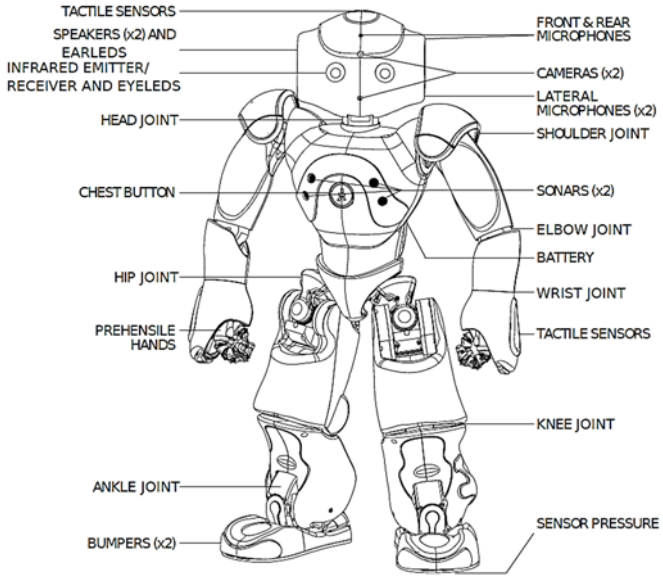


Fig. 3.3. *Sensores y actuadores presentes en el robot humanoide NAO*

El robot NAO está basado en un procesador Intel Atom de 1.6 GHz que ejecuta un kernel de Linux y un middleware de Aldebaran llamado NAOqi (software que corre sobre el robot y lo controla), adecuado para crear nuevas funcionalidades sobe el robot NAO como paralelismo, manejo de recursos, sincronización, eventos, etc. NAOqi se ejecuta en el robot NAO por medio de un broker. Un broker es un proceso que se encuentra escuchando la dirección IP y el puerto, y también contiene un conjunto de módulos que ofrecen cierta funcionalidad a través de funciones de alto nivel. Cuando se inicia el robot, se carga un archivo de preferencias almacenado en el autoload.ini que define las librerías que debe cargar, donde cada librería también presenta módulos como se muestra en la figura 3.4.

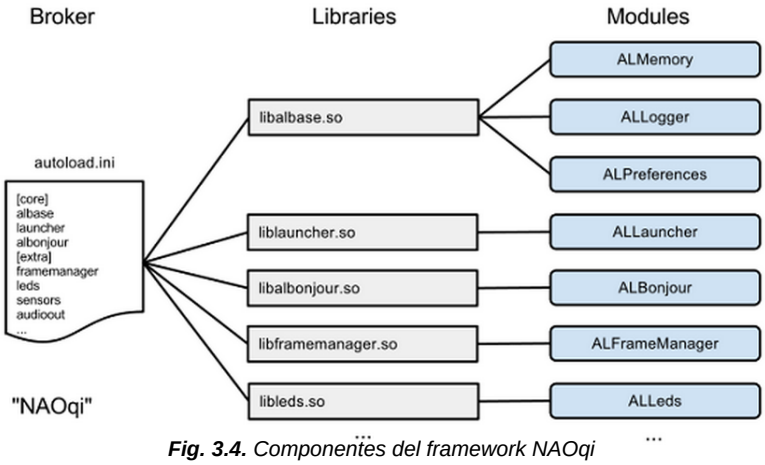


Fig. 3.4. *Componentes del framework NAOqi*

El broker también provee servicios de búsqueda, de modo que cualquier método anunciado por un módulo puede ser encontrado a través del árbol o la red, como se muestra en la figura 3.5.

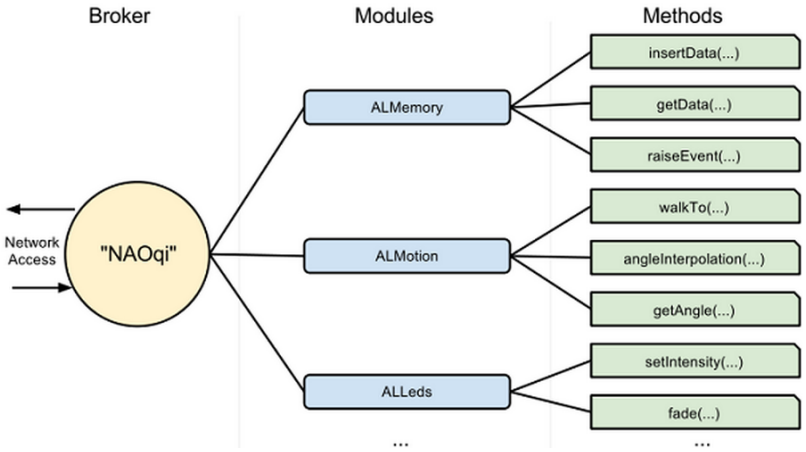


Fig. 3.5. *Árbol de conexión entre el broker, módulos y métodos.*

El broker más importante es el MainBroker (puerto 9559) el cual contiene los módulos que nos permiten acceder a los sensores y actuadores del robot NAO.

A continuación se muestra la configuración de la red que permite interactuar la plataforma SPADE y el broker del robot NAO.

```
####configuración para el robot NAO#######
robotIPSPADE = "127.0.0.1"  #Usado para el uso de la plataforma SPADE
#Ips locales
#robotIP = "127.0.0.1" #IP usado para el simulador
#robotPORT = 53151   #PORT usado para el simulador
#Ips UPV
robotIP = "192.168.1.11" #IP       usado para el robot real
robotPORT = 9559           #PORT usado para el robot real
```

Una vez configurada la red, procedemos a elaborar el movimiento del robot bípedo en Python, para esto primeramente declaramos la configuración de los módulos:

```
##########Configuracion del Robot NAO##################
motionProxy = ALProxy("ALMotion",robotIP,robotPORT)
postureProxy = ALProxy("ALRobotPosture",robotIP, robotPORT)
tts = ALProxy("ALTextToSpeech",robotIP, robotPORT)
```

- ***ALMotion,*** provee los métodos que facilitan el movimiento del robot, también implementa mecanismos de seguridad en el movimiento, como evitar colisiones y manejo de caídas. Presenta tres ejes, el eje X para movimientos adelante y atrás, el eje Y para movimientos de derecha e izquierda, y el eje Z que solo es vertical (ver figura 3.6).

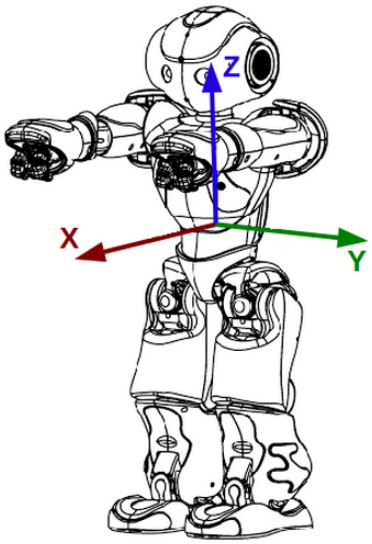


Fig. 3.6. *Ejes de robot NAO [25]*

- ***ALRobotPosture,*** hace que el robot realice diferentes posturas predefinidas (ver figura 3.7). El robot detecta en que postura está, y calcula la postura destino en base a la postura actual. Es posible seleccionar la velocidad de la postura que debe aplicarse (ej. *postureProxy.goToPosture ("StandInit",1.0)*).

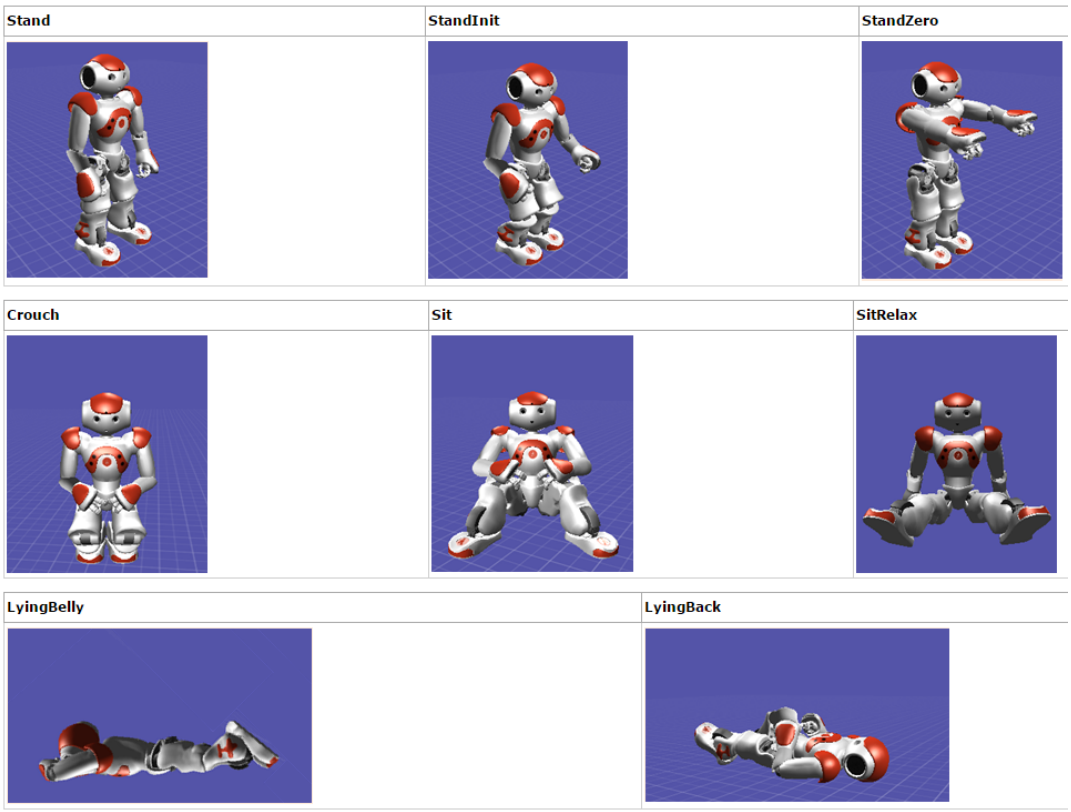


Fig. 3.7. *Posturas del robot NAO [25]*

- ***ALTextToSpeech,*** es el módulo que permite al robot hablar a través de un motor de texto-voz (text-to-speech). El resultado de la síntesis se envía a los altavoces del robot NAO.

Por último se van calculando las posiciones puntuales (figura 3.8) del robot tal y como se muestra a continuación:

```
# Enviamos el robot a la posición Inicial
postureProxy.goToPosture("StandInit", 0.5)
# La unidades de este comando son metros y radianes [-3.1415 to 3.1415]
#CODIGO PARA EL AVANCE DEL ROBOT
x  = 0.2# va la posicion X=0.2m
y  = 0.0
theta  = 0.0
motionProxy.moveTo(x, y, theta)
tts.say("LLEGUE A LA POSICION 1")
#CODIGO PARA EL GIRO DEL ROBOT
x  = 0.0
y  = 0.0
theta  = -math.pi/2#gira -90 grados
motionProxy.moveTo(x, y, theta)
.....
.....
```

En la figura 3.8 se puede observar las posiciones puntuales (P0-P3) que sigue el robot NAO, tras recibir las órdenes de girar y mover por parte del agente revisor en coordinación con el usuario.

Cabe destacar que en esta primera versión se trabajó más en el sistema de dialogo VOXEO y la plataforma de agentes SPADE, no se dio mucho interés a la planificación de la ruta y reconocimiento de voz por parte del robot.

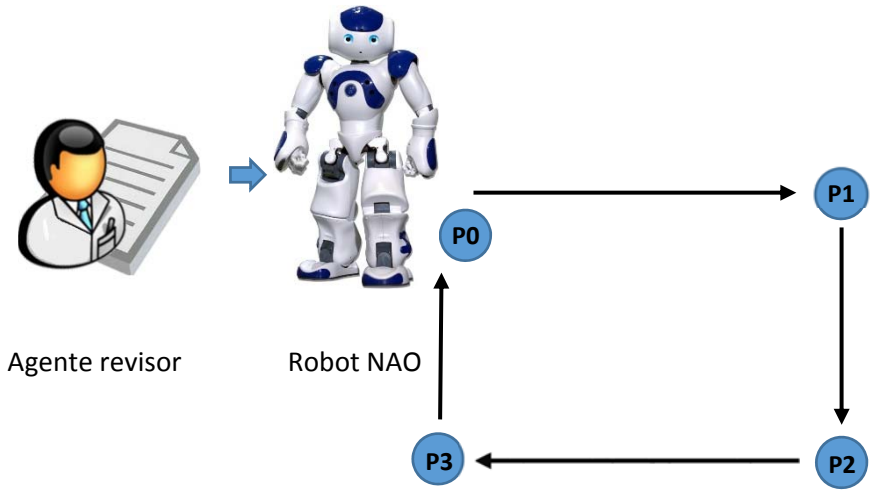


Fig. 3.8. *Movimiento del robot NAO para posiciones puntuales*

Tras realizar esta primera versión, se pudo validar las tecnologías seleccionadas así como también su interconexión. Se realizaron diversas pruebas que permitieron facilitar la transición a una segunda versión del sistema más complejo y real. Cabe destacar que se detectaron ciertas carencias (por ejemplo que el robot trabaje con distancias seguras al hacer los movimientos) que serán mejoradas en la segunda versión.

3.2 DISEÑO DE LA SEGUNDA VERSIÓN DEL PROTOTIPO

En esta segunda versión se trabaja con algo más real, se plantea el uso del sistema en un edificio que tiene varias plantas, donde cada planta (piso) está compuesto por su cocina, baños y dormitorios, los cuales estarán ocupados por personas mayores o con incapacidad como se muestra en la figura 3.9 y 3.10.

Para el diseño del sistema solo se toma en cuenta una planta, donde un agente revisor (robot NAO) se encuentra ubicado en la posición P0, que se encarga de recorrer todos los dormitorios ubicados en las posiciones (P1-P3) e interactúa con las personas mayores o con incapacidad preguntándoles su estado de salud, la información obtenida será gestionada por el agente gestor (encargado de supervisar a los agentes revisores) enviando un informe del estado de salud de los pacientes por correo electrónico al supervisor.

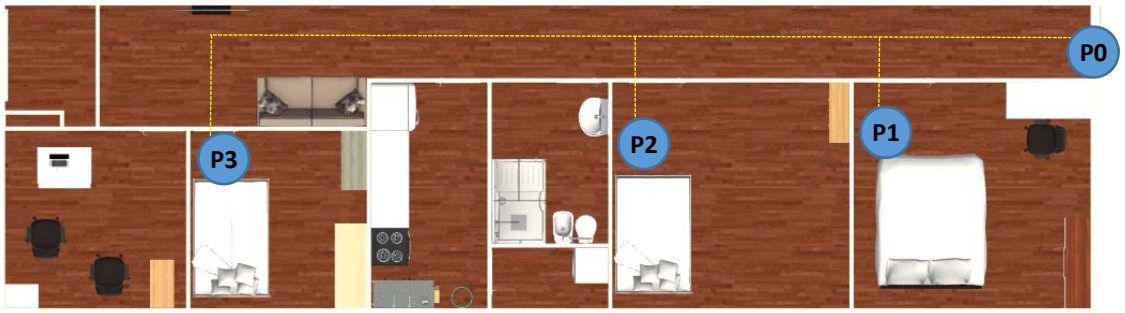


Fig. 3.9. *Vista superior del plano de recorrido*

Fig. 3.10. *Vista ortogonal del plano de recorrido*

Existe un encargado de vigilancia que se encarga de todo el funcionamiento del sistema. El encargado da una orden concreta a través de Skype, este se conecta al sistema de dialogo Voxeo y el servidor web Apache e interactúa con la plataforma de sistemas multiagente SPADE, el cual está conformado por los agentes gestor y revisor. El agente gestor reconoce la orden y habilita el funcionamiento del agente revisor, el cual ejecutara la orden del encargado, en la figura 3.11 se muestra un diagrama de bloques a seguir para el desarrollo del prototipo.

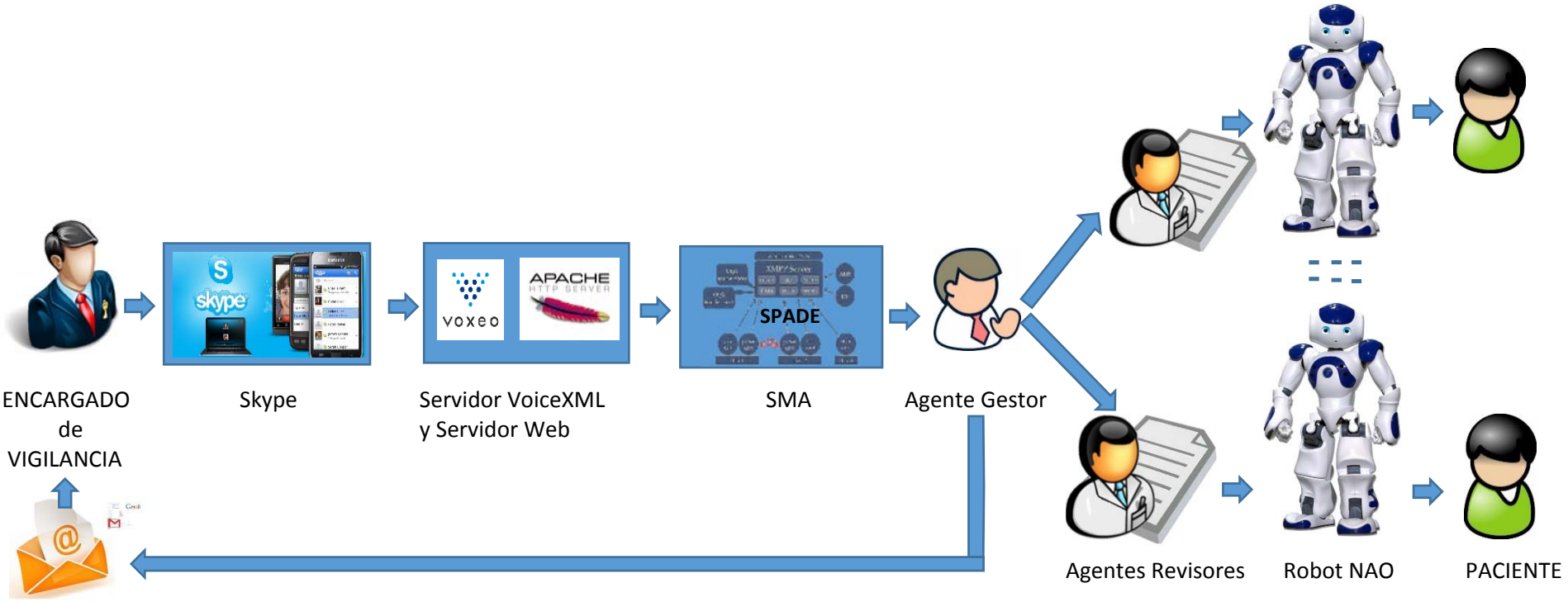


Fig. 3.11. *Diagrama de bloques utilizado en el diseño de la segunda versión*

En el diseño del sistema se incorporan los siguientes aspectos:

- Hay un encargado de vigilancia, el cual se encarga de supervisar a los agentes revisores.
- El encargado sólo da una orden concreta al agente gestor a través del sistema de dialogo, y este (agente gestor) coordina con los agentes revisores para cumplir las tareas programadas.
- Se incorpora un servidor web APACHE para una mejor interacción entre Voicexml y SPADE.
- Se modifica el código de la plataforma SPADE para que trabaje en el sistema operativo Windows.
- Se trabaja con distancias de seguridad y detección de marcadores (NAOmark) para el robot.
- Se diseña una planificación de trayectorias para el robot bípedo (figura 3.12).
- Se implementa en el agente gestor el código para mensajería por correo electrónico, para la gestión de errores e informe del trabajo desarrollado por parte del robot NAO.
- Se implementa el código para reconocimiento de voz por parte del robot hacia él paciente.

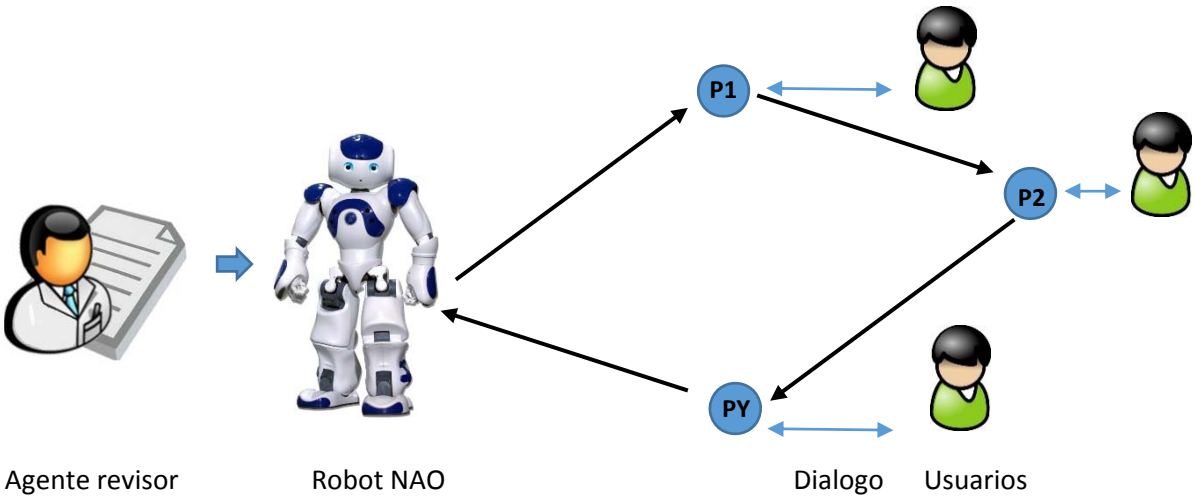


Fig. 3.12. *Planificación de rutas y dialogo con el usuario por parte del robot NAO*

Este segundo diseño del prototipo se dividió en cuatro partes que se detallan a continuación:

3.2.1 Diseño del código para el sistema de diálogo seleccionado e integración con el servidor Apache

En este apartado se sigue trabajando con las dos estrategias de interacción planteadas en la primera versión, se trabaja con órdenes elaboradas en VXML para el agente revisor y un servidor WEB (Apache) para una mejor interacción con la plataforma SPADE, tal y como se observa en la figura 3.13.

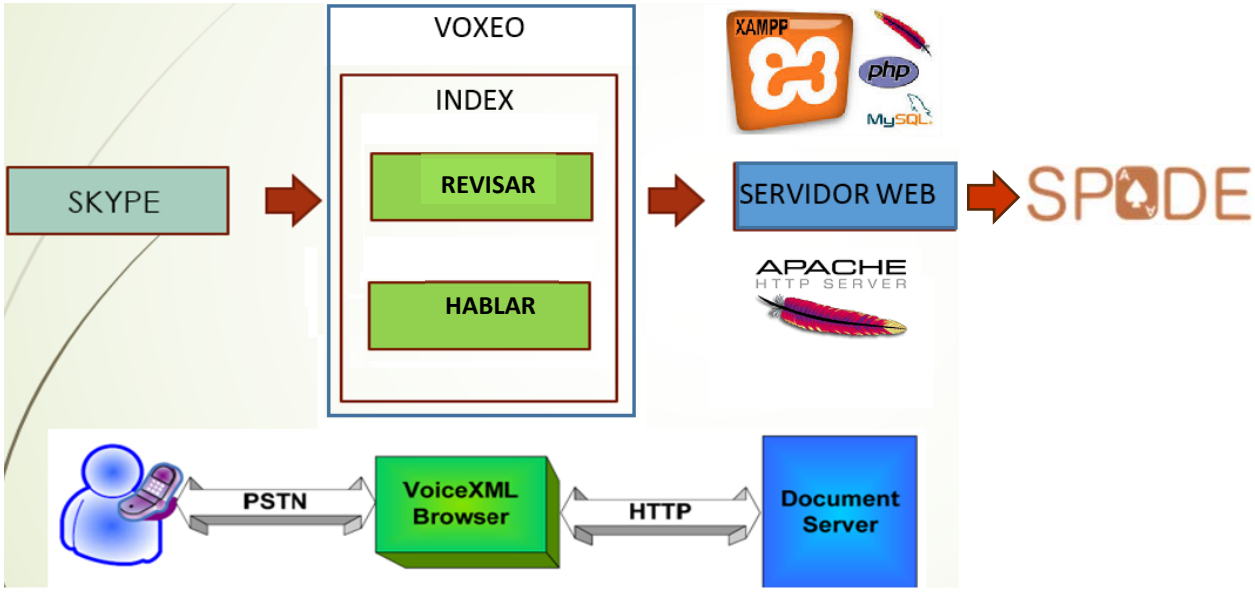


Fig. 3.13. *Diagrama de bloques utilizado en la 2ª versión para programar el código VXML del agente revisor*

Como se puede observar en este diagrama de bloques, se tiene como entrada la voz del supervisor, que será el encargado de dar las ordenes al robot bípedo a través de Skype mediante un móvil o una computadora, la cual es reconocida por la plataforma de voz. Dentro de la plataforma de voz se sigue manejando una gramática índex al inicio encargada de la interacción para reconocer frases del usuario como revisar y hablar, las cuales nos sirven para el manejo del agente revisor, a continuación se explica el funcionamiento de cada uno de estos.

- *Revisar*, encargada de dar la orden a un robot bípedo para que revise la habitación de los pacientes. El robot bípedo se encargará de llegar a las posiciones indicadas por el agente revisor e interactuará con el paciente preguntándole por su estado de salud.
- *Hablar*, se usa simplemente para verificar el estado de funcionamiento del robot.

Posteriormente elaboramos el código php para el servidor Web (Apache) para poder mejorar la interacción entre Voxeo y SPADE, considerando los siguientes puntos:

- Se direcciona la ubicación de la instalación de python en el código php.
- Todos los archivos generados en python se colocan dentro de la carpeta cgi-bin.

```
<?php
echo "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>";
?>
<vxml version="2.1" xmlns:voxeo="http://community.voxeo.com/xmlns/vxml" xml:lang="es-es">
<?php
   echo "Revisando al paciente con el robot NAO!" ;
   exec("@echo off");
   exec("C:");
   echo "Direccionamiento de la ubicación de python" ;
   exec("cd \Program Files (x86)\Aldebaran Robotics\Choregraphe Suite 2.1\lib");
   echo "Ubicación de los archivos de python";
   exec("start python C:\\xampp\\cgi-bin\\final.py");
?>
<form>
 <prompt>
  Logre concluir el trabajo de revisión
 </prompt>
 <goto next="http://webhosting.voxeo.net/190412/www/robotNAOindex2015.vxml"/>
</form>
</vxml>
```

Aclarar que se debe de tener cuidado en la interacción entre Voxeo y Apache. La dirección IP del código elaborado en voicexml (Voxeo) (ej. <goto next= "http://**79.108.179.27**/nombre.php" />) debe coincidir con la dirección IP del servidor web Apache (ver fig. 3.14).

Vxml (IP=79.108.179.27) | PHP (**IP=79.108.179.27**) | Python (IP=127.0.0.1)

Fig. 3.14. *Configuración de los IPs para la Interacción entre Voxeo, Apache y SPADE*

3.2.2 Diseño del Sistema Multiagente

En la segunda versión del diseño del sistema, se sigue manteniendo los dos agentes definidos en la primera versión, con la diferencia de las funciones que desempeñan los agentes, los cuales se explican a continuación:

- Agente gestor, encargado de gestionar, monitorizar, supervisar a todos los agentes revisores, como también el envío de mensajes electrónicos al encargado de vigilancia.
- Agente revisor, conformado en parte por el robot bípedo (Robot NAO), el cual se encarga de ejecutar las tareas de control del estado de salud del paciente.

Antes de diseñar el sistema multiagente, se tuvo que modificar parte del código de la plataforma del sistema multiagente SPADE para que trabaje en el sistema operativo Windows. Esto es debido a que se tiene más herramientas de desarrollo para el robot bípedo en dicho sistema.

En el diseño del agente gestor se integró la mensajería por correo electrónico, con el fin de enviar informes al encargado de vigilancia sobre el trabajo y posibles fallos del agente revisor (fig. 3.15). El código que se observa a continuación es utilizado para lograr el objetivo.

```
####################Configurando el correo############
to = 'ALDEBARANALEX@gmail.com'
gmail_user = 'ALDEBARANALEX@gmail.com'
gmail_pwd = 'aldebarannao'
smtpserver = smtplib.SMTP("smtp.gmail.com",587)
smtpserver.ehlo()
smtpserver.starttls()
smtpserver.ehlo
smtpserver.login(gmail_user, gmail_pwd)
```

```
header = 'To:' + to + '\n' + 'From: ' + gmail_user + '\n' + 'Subject:testing \n'
```

```
###enviando al correo########
 msg = header + '\n LLEGUE A LA POSICION FINAL\n\n'
 ……
 ……
 smtpserver.sendmail(gmail_user, to, msg)
 print 'done!'
 smtpserver.close()
```

Los agentes revisores, puede ser diseñados uno para cada planta (figura 3.9), donde todos los agentes revisores están gestionados por el agente gestor, que decide a que agente revisor le envía la orden de revisar la planta (piso).

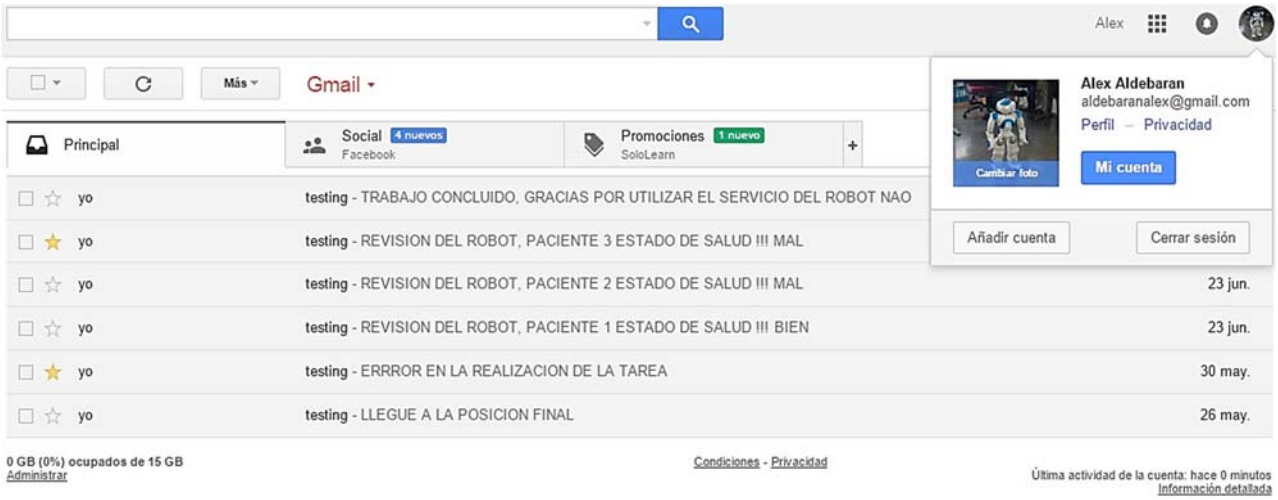


Fig. 3.15. *Mensajes enviados por el agente gestor al encargado de vigilancia*

3.2.3 Control del Robot Bípedo

En este aparatado se trabaja más sobre la configuración de red, planificación de trayectorias, detección de obstáculos y distancias seguras del robot NAO.

Las configuraciones de red adoptadas por el sistema de vigilancia, son las mismas que se planteó en la primera versión del diseño.

```
####configuración para el robot NAO#######
robotIPSPADE = "127.0.0.1"  #Usado para el uso de la plataforma SPADE
#Ips locales
robotIP = "192.168.1.11" #IP       usado para el robot real
robotPORT = 9559           #PORT usado para el robot real
```

Cuando se diseñó la primera versión del sistema solo se consideró tres módulos: ALMotion, ALRobotPosture y ALTextToSpeech, siendo necesario añadir más módulos para lograr obtener los objetivos planteados como: distancia segura, reconocimiento del habla, etc. A continuación se remarca en negrita todos los módulos adicionados.

```
##########Configuracion del Robot NAO##################
motionProxy = ALProxy("ALMotion",robotIP,robotPORT)
postureProxy = ALProxy("ALRobotPosture",robotIP, robotPORT)
tts = ALProxy("ALTextToSpeech",robotIP, robotPORT)
tts.setLanguage("Spanish")

memoryProxy = ALProxy("ALMemory", robotIP, robotPORT)
sonarProxy = ALProxy("ALSonar", robotIP, robotPORT)
navigationProxy = ALProxy("ALNavigation", robotIP, robotPORT)
landMarkProxy = ALProxy("ALLandMarkDetection",robotIP, robotPORT)
asr = ALProxy("ALSpeechRecognition",robotIP, robotPORT)
asr.setLanguage("Spanish")
```

- ***ALMemory,*** es una memoria centralizada que se utiliza para almacenar toda la información importante relacionada con la configuración del hardware del robot, necesario para almacenar la información de los sonares, marcadores y reconocimiento del habla.
- ***ALSonar,*** recupera el valor del sensor de ultrasonidos de ALMemory, lo procesa y provoca eventos de acuerdo a la situación, presenta 4 eventos: SonarLeftNothingDetected, SonarRightNothingDetected, SonarLeftDetected y SonarRightDetected.
- ***ALNavigation,*** permite al usuario realizar desplazamientos seguros al utilizar el robot. El robot no puede evitar los obstáculos, pero es capaz de moverse con precaución, deteniéndose tan pronto como un obstáculo entra en su zona de seguridad, en nuestro proyecto trabajamos con una distancia de seguridad de 30cm, para evitar que exista colisiones.

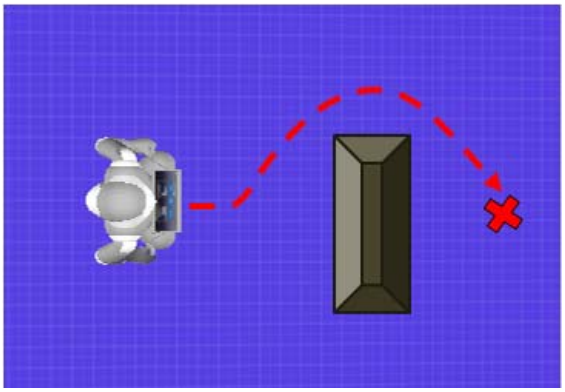


Fig. 3.16. *Distancia segura con ALNavigation*

- ***ALLandMarkDetection,*** es un módulo de visión, en el que el robot reconoce puntos de referencia especiales (landmarks) con patrones específicos en ellos. Son también conocidos estos landmarks con el nombre de Naomarks.

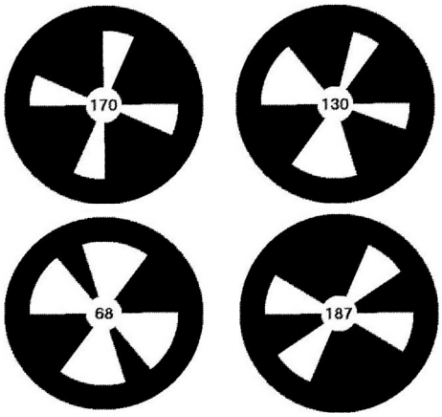


Fig. 3.17. *Naomarks utilizados para detectar localizaciones*

- ***ALSpeechRecognition,*** módulo que da al robot la capacidad de reconocer palabras o frases predefinidas en varios idiomas, en nuestro caso es utilizado para reconocimiento del habla en español del paciente.

Una vez configurado los módulos a usar en el diseño del prototipo, se trabajó sobre la planificación de las trayectorias del robot, trabajando de dos formas:

- Con detección de obstáculos, haciendo uso de marcadores (Naomarks) y sonares.
- Sin detección de obstáculos, trabajando con posiciones puntuales y sin obstáculos.

Para la detección de obstáculos se utilizó el diagrama de flujo de la figura 3.18, donde el robot NAO empieza en la posición inicial P0 con la detección del landmark de inicio, en su recorrido el robot NAO va haciendo lecturas de los sonares, se presenta con cuatro situaciones que se describen a continuación:

- Si en su recorrido se encuentra un obstáculo de frente y otro en el lado derecho gira 90° hacia la izquierda.
- Si en su recorrido se encuentra un obstáculo de frente y otro en el lado izquierdo gira 90° hacia la derecha.
- Si en su recorrido se encuentra un obstáculo de frente y otros dos ubicados en el lado derecho e izquierdo, gira 180°.
- Si en su recorrido encuentra el landmark final y la lectura de los sonares izquierdo y derecho es el máximo, acaba el recorrido.
- Sino no se presenta ninguno de los anteriores casos sigue caminando el robot NAO.

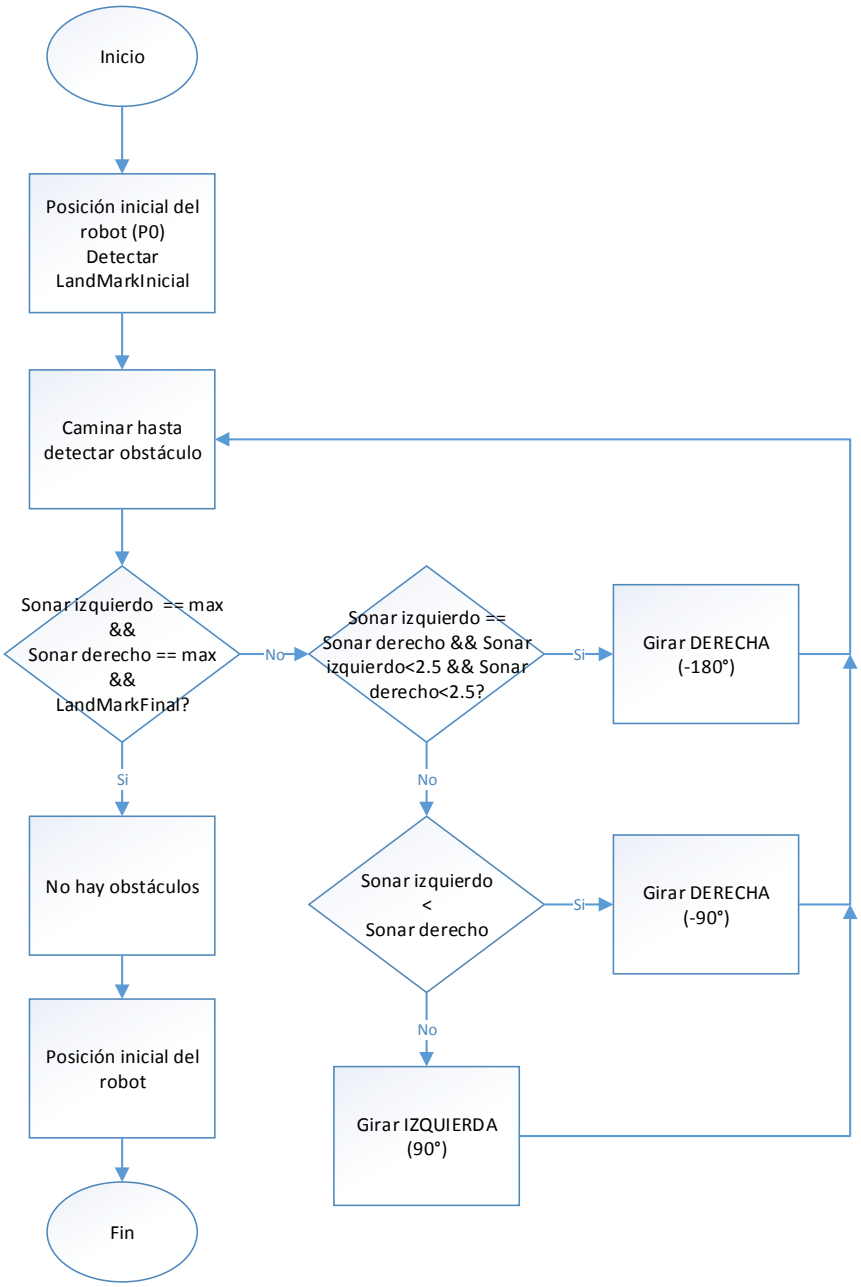


Fig. 3.18. *Diagrama de flujo elaborado para la detección de obstáculos en la trayectoria del robot NAO*

Para el segundo caso se trabaja con posiciones puntuales sin la detección de obstáculos, siendo más fácil la planificación de las rutas. El robot empieza en la posición P0 y va visitando los dormitorios ubicados en las posiciones P1-P3.

En este apartado se calculó las medidas del piso para hallar las distancias de recorrido del robot NAO, como se observa en la figura 3.19.

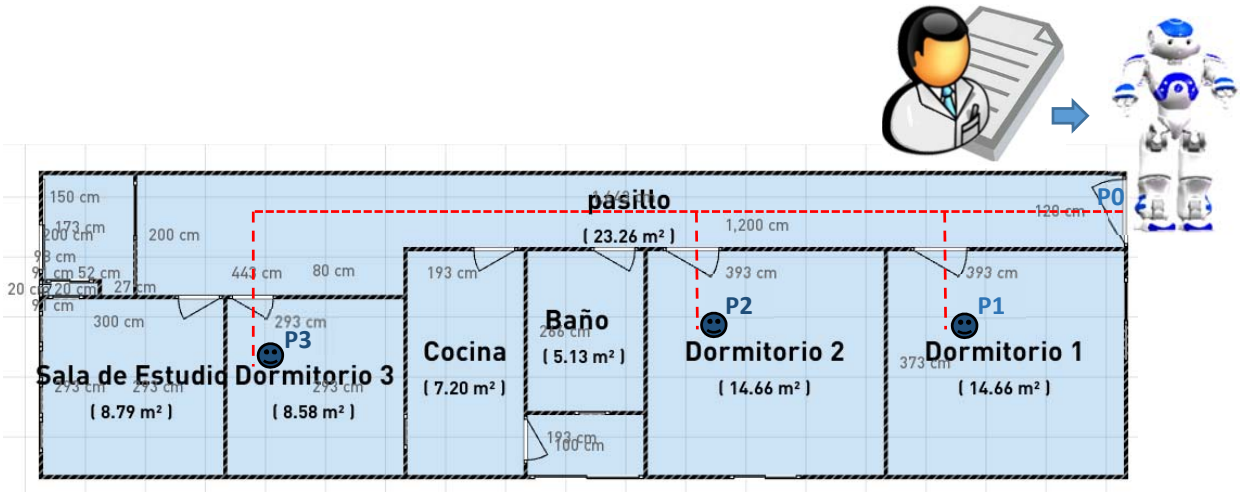


Fig. 3.19. *Plano de recorrido del robot NAO por las habitaciones (dormitorios P1, P2, P3)*

El código que se expone a continuación muestra el uso de ALNavigation para el recorrido del robot NAO a las posiciones P1-P3. En los movimientos y giros se va verificando la distancia de seguridad.

```
print "TRABAJANDO CON POSICIONES FIJAS"
iniciar(postureProxy)
################POS1###############

x  = 3     #distancia de recorrido en el eje x de 3m
y  = 0.0
theta  = 0.0
Caminar(x, y, theta, navigationProxy)

Giro_izquierda(navigationProxy)

x  = 2    #distancia de recorrido en el eje x de 2m
y  = 0.0
theta  = 0.0
Caminar(x, y, theta, navigationProxy)
.....
.....
```

Se optó por trabajar con posiciones puntuales y sin la detección de obstáculos, debido a que la lectura de landmarks no está bien implementada en el robot, existe mucha dependencia de factores externos como la luminosidad y la inclinación de la cámara. En el uso de sonares también se tuvo problemas con la detección de obstáculos ubicados en ángulos no perpendiculares.

3.2.4 Reconocimiento de voz por parte del agente revisor (robot NAO)

En esta última parte se implementa el reconocimiento de voz por parte del agente revisor, donde coordinan el agente gestor y revisor para detallar al encargado de vigilancia un informe del estado actual del paciente (preguntándole el robot como se encuentra al paciente), como se muestra en la figura 3.20.

Fig. 3.20. *Dialogo del robot NAO con el paciente*

En esta última parte del código se expone algunas palabras claves que tiene que reconocer el robot NAO, como por ejemplo bien, excelente, mal, regular y pésimo. Mencionar que se puede ampliar más el vocabulario de reconocimiento del habla, con frases más largas como por ejemplo “me encuentro bien”.

```
  print 'Iniciando el reconocimiento de voz'
  tts.say("COMO SE ENCUENTRA EL PACIENTE")
 wordList=["bien","excelente","mal","regular","pésimo"]
 asrProxy.setWordListAsVocabulary(wordList)
asrProxy.subscribe("asr")
ALMemoryKey="WordRecognized"
memoryProxy.insertData(ALMemoryKey,["",0])
```

Tras realizar esta segunda versión se pudo concretar todos los objetivos perseguidos en el diseño del prototipo del sistema de vigilancia, también se mejoró todas las carencias presentadas en la primera versión. El uso de diversas tecnologías ayudo en la realización del trabajo del sistema multiagente. Se pudo verificar que existe una excelente coordinación entre los robots y los agentes, logrando concluir tareas complejas como la revisión del estado de salud de los pacientes ubicados en los dormitorios.

Capítulo 4

Conclusiones y líneas futuras

4.1 CONCLUSIONES

Para conseguir los objetivos planteados en este trabajo, se tuvo que estudiar diversas tecnologías relacionadas con la inteligencia artificial y la robótica. Posteriormente se fue diseñando, simulando e implementando la primera versión del prototipo del sistema de vigilancia, la cual consistía en el diseño del sistema multiagente y la validación de las diversas tecnologías seleccionadas para su interconexión entre ellos, donde se presentaron algunos problemas que fueron resueltos. En el diseño de la segunda versión del prototipo del sistema de vigilancia se mejoraron la limitaciones de la primera versión, también se añadió otras tareas al sistema como: diseño de mensajería por correo electrónico para el sistema multiagente, planificación de rutas y reconocimiento de voz del robot bípedo hacia el paciente, con el fin de revisar el estado de salud de los pacientes.

Con la realización de este trabajo se pudo observar el funcionamiento real de los sistemas de dialogo y sistemas multiagente aplicados a la robótica, donde la gran mayoría de las pruebas realizadas para lograr el comportamiento definitivo del robot NAO, se logró gracias a su simulador Choregraphe.

La plataforma SPADE utilizada nos permitió implementar aplicaciones con un alto grado de escalabilidad y de distribución en red, facilitándonos la distribución de agentes en varios computadores de una misma red o en la autonomía de los robots en un determinado entorno.

Al programar el prototipo del sistema de vigilancia en el lenguaje de programación Python, nos ofreció varias ventajas, entre las más importantes tenemos que es un lenguaje multiplataforma la cual nos ayuda en la ejecución de nuestros agentes en cualquier sistema operativo (Windows, Linux y Mac) y la extensa cantidad de librerías presentes en internet para distintas aplicaciones.

También se puede concluir que se tuvo algunos problemas en la implementación del prototipo, las cuales se mencionan a continuación:

- Falta de módulos en el sistema de dialogo VOXEO para la integración con servidores locales.
- Incompatibilidad a la hora de instalar las versiones de NAOqi, Choregraphe y Python.
- Problemas de precisión por parte del robot NAO, al caminar, hacer giros y ubicación de obstáculos con sonares.
- Problemas en el reconocimiento de marcadores por parte del robot NAO, esto debido a que el robot depende de la iluminación y del ángulo de inclinación de la cámara.

4.2 LÍNEAS FUTURAS

Por lo que puede referirse a los trabajos futuros se plantean los siguientes:

- Mejorar la colaboración entre los robots que realizan la monitorización, esto con el fin de que exista comunicación entre robots y otros agentes, para realizar trabajos más complejos.
- Implementación de máquina de estados múltiples y cadenas de Markov, esto nos ayudaría en la interacción del habla entre el agente revisor y el paciente, con el fin de tener una conversación más realista.
- Integración del sistema multiagente con el software de simulación Webots, esto para tener entornos de simulación más reales.
- Desarrollo de sistemas SCADA para dispositivos Android e IOS, esto sería de gran ayuda en la gestión y monitoreo de sistemas de vigilancia con robots.
- Seguimiento de trayectorias más precisas, con la utilización de dispositivos externos (ej. Raspberry PI), con ello se lograría mejorar el rango de visión del robot. También el uso de técnicas de Inteligencia Artificial aplicadas al campo de la visión artificial (ej. OpenCV) nos darían una mejor precisión de la trayectoria.

Bibliografía

1. López-Cózar R. and Araki M. “Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment” (Wiley 2005).

2. Llisterri, J. "Introducción a los sistemas de diálogo", in LLISTERRI J. & MACHUCA M. J. (Eds.) *Los sistemas de diálogo*. Bellaterra - Soria: Universitat Autònoma de Barcelona, Servei de Publicacions - Fundación Duques de Soria (Manuals de la Universitat Autònoma de Barcelona, Lingüística, 45), págs. 11-21, 2006.

3. Lluis Hurtado, Fernando Blat, S. Grau, David Griol, Emilio Sanchis, Encarna Segarra, F. Torres “Sistemas de dialogo para el proyecto DIHANA” Procesamiento del lenguaje natural, ISSN 1135-5948, Nº 35, 2005, págs. 453-454, 2005.

4. Griol David, “Desarrollo y evaluación de Diferentes Metodologías para la Gestión Automática del Dialogo” Tesis Doctoral. Universidad Politécnica de Valencia, 2007.

5. Callejas Zoraida “Desarrollo de Sistemas de Dialogo Oral Adaptativos y Portables: Reconocimiento de Emociones, Adaptación al idioma y Evaluación de Campo” Tesis doctoral Universidad de Granada, 2008.

6. María García Jimenez, Proyecto fin de carrera “Desarrollo de un portal de voz de atención al ciudadano mediante VoiceXML”, 2011.

7. R. López Cózar & E. Sanchís, E. Segarra, J.M. Benedí “Incorporación de VoiceXML en el Sistema de Diálogo DIHANA”, 2004.

8. Minsky M. “A framework for Representation Knowledge” Mc Hill. New York, 1975.

9. Jennings N. R. “Coordination tecniques for distributed artificial intelligence” In. O’HARE, G.M.P. JENNINGS N.R. (Eds) Foundations of distributed artificial intelligence. New Yrok: John Wiley & Sons, 1996 p. 187-210, 1996.

10. Posadas Yagë Juan L. “Arquitectura para el control de robots móviles mediante delegación de código y agentes” Tesis doctoral, Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia España, 2003.

11. Giret B., Insfrán, Pastor, Cernuzzi “Informe técnico OO.Method para el desarrollo de sistemas multiagentes”, Departamento de sistemas informáticos y computación, Universidad Politécnica de Valencia, 2000.

12. María Guadalupe Alexandres García “Arquitectura tolerante a fallos mediante un sistema multiagente para el sistema de control de un robot móvil”, 2007.

13. Gerhard Weiss “Multiagent Sytems A modern Approach to Distributed Artificial Intelligence”, 2000.

14. Giampapa J.A., Sycara K. “Conversational Case-Based Planning for Agent Team Coordination, Robotics Institute, Carnegie Mellon University”, 2003.

15. MADKit Multi-Agent Development KIT. http://www.madkit.org, 2002.

16. Ferber and Gutkenecht. “A meta-model for the analysis and design of organizations in multi-agent systems”. In Proceedings of the 3er International Conference on Multi-Agent Systems (ICMAS 98). IEEE CS Press, 1998.

17. JADE, Java agent development framework. In http://jade.tilab.com, 2014.

18. Favio Luigi Bellifemine, Giovanni Caire and Dominic Greenwood, “Developing multi-agent systems with jade”, 2007.

19. Miguel Escrivá G., Javier Palanca C., Gustavo Aranda B. “A Jabber-based Multi-Agent System Platform”, 2006.

20. http://es.wikipedia.org/wiki/SPADE

21. Nerea Luis Mingueza “Desarrollo de un sistema de juego al tres en Raya para el robot NAO H25”, Universidad Carlos III DE Madrid, 2013.

22. J. Ruiz del Solar and R. Salazar. “Introducción a la robótica.”

23. Adrián Cervera Andés “Coordinación y control de robots móviles basado en agentes”, Universidad Politecnica de Valencia, 2011.

24. http://es.wikipedia.org/wiki/Robótica

25. Aldebaran Robotics 2014, Documentación del robot NAO, http://doc.aldebaran.com/1-14/dev/naoqi/index.html

26. Jesús Bueno Gómez tesis “Optimización y modelado del movimiento de un robot bípedo NAO usando técnicas de IA”, Universidad de Castilla la Mancha.

27. Stuart Russell y Peter Norvig “Inteligencia Artificial Un enfoque moderno”, Departamento de sistemas informáticos y computación, Universidad Politécnica de Valencia, 2010.

Apéndice A

Manual de instalación

En este apéndice se describe los pasos para instalar los diferentes herramientas utilizadas para el diseño del sistema.

A.1. Python

Si se trabaja con el sistema operativo Linux (Ubuntu 14.0) se debe de tener cuidado en trabajar con la versión 2.7 de python, para esto se puede introducir este comando en una terminal:

Python --version

En la pantalla aparecerá la versión de python instalada, en caso que se instale una versión superior dará problemas de incompatibilidad con el robot NAO. El comando que permite instalar python es:

Sudo apt-get install python2.7

En el caso de Windows solo se debe de bajar el instalador de Python 2.7 (32 bits) e instalarlo.

A.2. Choregraphe

El instalador del simulador Choregraphe puede obtenerse de la siguiente dirección: https://community.aldebaran.com/en/resources/software, permitiéndonos probarlo durante 30 días, luego se debe de introducir la licencia.
Para instalarlo es necesario descargar el archivo, descomprimirlo en el directorio donde se quiere instalar. En el caso de Linux se deberá de ejecutar el siguiente comando:

./ choregraphe

Para el sistema operativo Windows simplemente es ejecutar choregraphe-xxx-win32-setup.exe.

A.2. NaoQi

Para la instalación de NaoQi se debe de comprobar que sea la versión adecuada para Choregraphe y python, solo hay que descomprimirlo en el directorio donde se desea utilizarlo.
Si se obtiene algún error relacionado con la variable de entorno PYTHONPATH a la hora de ejecutarlo, se debe de modificar la ruta asociada a la misma ruta de Python.

LINUX: export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/opt/Aldebaran/Choregraphe Suite 2.1/lib
export PYTHONPATH=$PYTHONPATH:/opt/Aldebaran/Choregraphe Suite 2.1/lib
export PYTHONUNBUFFERED=1
WINDOWS: C:\Program Files (x86)\Aldebaran Robotics\Choregraphe Suite 2.1\lib

A.3. SPADE

Para la instalación de Spade en Windows o Linux se coloca el siguiente comando:

Pip install SPADE
or
easy_install SPADE

También se lo puede descargar de https://github.com/javipalanca/spade e instalarlo utilizando el siguiente comando:

python setup.py install

A.4. XAMPP

XAMPP es una distribución de Apache completamente gratuita y fácil de instalar que contiene MySQL, PHP y Perl. El paquete de instalación de XAMPP ha sido diseñado para ser increíblemente fácil de instalar y usar.
El instalador se lo puede descargar para cualquier sistema operativo del siguiente enlace: https://www.apachefriends.org/es/index.html, la arquitectura para Linux, versión de 32-bits o 64-bits se ejecuta los siguientes comandos:
Cambiar los permisos al instalador y ejecutamos el instalador:

chmod 755 xampp-linux-*-installer.run
sudo ./xampp-linux-*-installer.run

En el sistema operativo Windows solo instalamos el instalador xampp-win32xxxx-installer.exe.

Apéndice B

Especificaciones técnicas del robot NAO H25

TECHNICAL SPECIFICATIONS

ELECTRICAL

INPUT	100 to 240 Vac - 50/60Hz - Max 1.2A	
OUTPUT	25.2 Vdc - 2A	
BATTERY	Type	Lithium-Ion
	Nominal voltage/capacity	21.6V / 2.15Ah
	Max charge voltage	24.9V
	Recommended charge current	2A
	Max charge/discharge current	3.0A / 2.0A
	Energy	27.6Wh
	Charging duration	5h
	Autonomy	60min (Active use) 90min (Normal use)

MOTHER BOARD

CONSTRUCTION

DIMENSION (H×D×W)	573×275×311mm / 22.5x10.8x12.2 inch
WEIGHT	5.2kg / 11.4 lb
CONSTRUCTION MATERIAL	ABS-PC / PA-66 / XCF-30

LANGUAGES

TEXT TO SPEECH & AUTOMATIC SPEECH RECOGNITION	Arabic, Brazilian (Portuguese), Chinese, Czech, Danish, Dutch, English, Finnish, French, German, Italian, Japanese, Korean, Polish, Portuguese, Spanish, Swedish, Russian, Turkish

VISION

CPU PROCESSOR	ATOM Z530	
	Cache memory	512KB
	Clock speed	1.6GHZ
	FSB speed	533mHz
RAM	1GB	
FLASH MEMORY	2GB	
MICRO SDHC	8GB	

CONNECTION

ETHERNET	1×RJ45 - 10/100/1000 BASE T
WIFI	IEEE 802.11b/g/n

AUDIO

LOUD SPEAKERS	×2 lateral	
	Diameter	36mm
	Impedance	8ohms
	Sp level	87dB/w +/- 3dB
	Freq range	up to ~20kHz
	Input	2W
MICROPHONE	×4 on the head	
	Sensitivity	~40 +/-3dB
	Frequency range	20Hz-20kHz
	Signal/noise ratio	58dBA

CAMERAS	×2 on front	
Sensor model	MT9M114	
Sensor type	SOC Image Sensor	
IMAGING ARRAY	Resolution	1.22MP
	Optical format	1/6inch
	Active Pixels (H×V)	1288×968
SENSITIVITY	Pixel size	1.9µm
	Dynamic range	70dB
	Signal/Noise ratio (max)	37dB
	Responsivity	2.24 V/lux-sec (960p)
		8.96 V/lux-sec (VGA)
OUTPUT	Camera output	960p@30fps
	Data Format	YUV422
	Shutter type	ERS (Electronic Rolling Shutter)
VIEW	Field of view	72.6°DFOV (60.9°HFOV, 47.6VFOV)
	Focus range	30cm ~ infinity
	Focus type	Fixed focus

FRAMERATE

Resolution	Embedded	Gigabit Ethernet	100Mb Ethernet	Wifi g
160×120px	30fps	30fps	30fps	30fps
320×240px	30fps	30fps	30fps	11fps
640×480px	30fps	30fps	12fps	2.5fps
1280×960px	29fps	10fps	3fps	0.5fps

Note: using the video stream in remote highly depends on the network and the video resolution chosen. All frame rates depend on the CPU usage. Values are calculated with a CPU fully dedicated to images gathering.

TECHNICAL SPECIFICATIONS

IR

NUMBER	×2 on front
WAVELENGTH	940nm
EMISSION ANGLE	+/-60°
POWER	8mW/sr

SONAR

EMITTERS	×2 on front
RECEIVERS	×2 on front
FREQUENCY	40kHz
SENSITIVITY	-86dB
RESOLUTION	1cm
DETECTION RANGE	0.25m to 2.55m
EFFECTIVE CONE	60°

INERTIAL UNIT

GYROMETER	×2	
	Axis	1 per gyrometer
	Precision	5%
	Angular speed	~500°/s

FSR (FORCE SENSITIVE RESISTORS)

RANGE	0 to 110N
	×4 per feet

ACCELEROMETER	×1	
	Axis	3
	Precision	1%
	Acceleration	~2g

POSITION SENSORS

	NAO HUMANOID	
MRE (Magnetic Rotary Encoder)	×36	
	Using hall effect sensor technology	
	Precision:	12bits / 0.1°

SOFTWARE

OPEN NAO	Embedded GNU/Linux Distribution based on Gentoo
ARCHITECTURE	×86
PROGRAMMING	Embedded: C++ / Python Remote: C++ / Python / .NET / Java / MatLab

LEDS

PLACEMENT	QUANTITY	DESCRIPTION
Tactile Head	×12	16 Blue levels
Eyes	2×8	RGB FullColor
Ears	2×10	16 Blue levels
Chest button	×1	RGB FullColor
Feet	2×1	RGB FullColor

CONTACT SENSOR

	NAO HUMANOID
Chest Button	✓
Foot Bumper	✓
Tactile Head	✓
Tactile Hand	✓

DEGREES OF FREEDOM

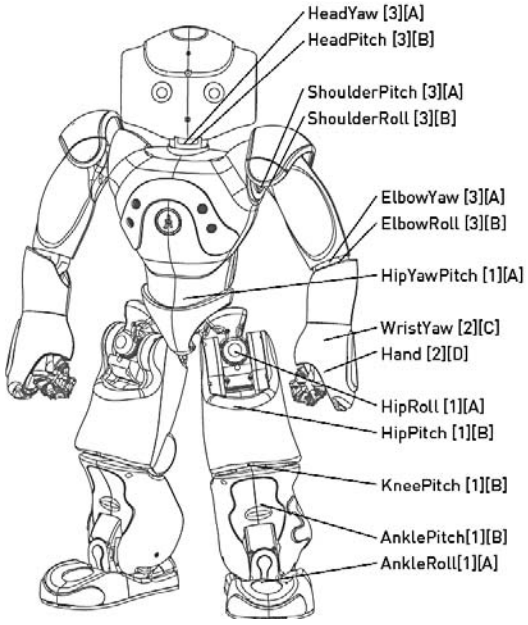
	NAO HUMANOID
HEAD	×2 dof
ARM (IN EACH)	×5 dof
PELVIS	×1 dof
LEG (IN EACH)	×5 dof
HAND (IN EACH)	×1 dof

MOTOR SPECIFICATIONS

MOTOR TYPE Brush DC Coreless

POSITION OF MOTORS

		MOTOR	REDUCTION RATIO
HEAD JOINTS	HeadYaw	Type 3	**Type A**
	HeadPitch	Type 3	**Type B**
ARM JOINTS	ShoulderPitch	Type 3	**Type A**
	ShoulderRoll	Type 3	**Type B**
	ElbowYaw	Type 3	**Type A**
	ElbowRoll	Type 3	**Type B**
	WristYaw	Type 2	**Type C**
	Hand	Type 2	**Type D**
LEG JOINTS	HipYawPitch	Type 1	**Type A**
	HipRoll	Type 1	**Type A**
	HipPitch	Type 1	**Type B**
	KneePitch	Type 1	**Type B**
	AnklePitch	Type 1	**Type B**
	AnkleRoll	Type 1	**Type A**



DESCRIPTION OF THE MOTORS

	MOTOR TYPE 1	MOTOR TYPE 2	MOTOR TYPE 3
Model	22NT82213P	17N88208E	16GT83210E
No load speed	8300rpm ±10%	8400rpm ±12%	10700rpm ±10%
Stall torque	68mNm ±8%	9.4mNm ±8%	14.3mNm ±8%
Continuous torque	16.1mNm max	4.9mNm max	6.2mNm max

Legend: Joint Name[Motor Type][Reductor Type]

SPEED REDUCTION RATIO
TYPE A

	MOTOR TYPE 1	MOTOR TYPE 3
Reduction ratio	201.3	150.27

SPEED REDUCTION RATIO
TYPE B

	MOTOR TYPE 1	MOTOR TYPE 3
Reduction ratio	130.85	173.22

SPEED REDUCTION RATIO
TYPE C

	MOTOR TYPE 2
Reduction ratio	50.61

SPEED REDUCTION RATIO
TYPE D

	MOTOR TYPE 2
Reduction ratio	36.24

CERTIFICATIONS & APPROVALS

REGION	CLASSIFICATION
Europe	CE (Declaration of Conformity)
USA	FCC

ELECTROMAGNETIC COMPATIBILITY	EN 301 489-1 / EN 301 489-17 / EN 300 328 EN 62311 : 2008 / FCC PART15, Class A
SAFETY	IEC 60950-1:2005 (2nd edition)