Integración de un Sistema de Reconocimiento de Voz al Sistema de Navegación de un Robot Móvil

Paulino Ochoa Villafuerte

Tesis de Licenciatura
Ingeniería en Computación

FI IIMAS UNAM

Julio 2004

Agredecimientos:

A mis padres, Chentis y Pau, por todo su amor, cariño, comprensión y sacrificio. En todo momento están conmigo.

A Miguel, Luis, Alicia, Martha y Rome, por su compañia y confianza.

A Nayelly y Uriel, por permitirme terminar esta aventura.

A mis amigos de la Facultad de Ingeniería, por su compañia y lealtad.

A Dios, por llenar mi vida de bendiciones.

A Miguel Salas y Manuel Romero, por su disposición, confianza y tiempo dedicado a este trabajo.

Al personal del IIMAS, Luis, Esmeralda, Iván, Toño, Arturo, César, Roxana, por todo el apoyo recibido durante la elaboración de este trabajo.

A mis profesores, por la educación brindada.

A la Universidad Nacional Autonoma de México y en especial a la Facultad de Ingeniería, por la formación obtenida y por dejarnos ser parte de ellas.

A los profesores que complementan mi jurado de exámen profesional, Ing. Juan José Carreón, Ing. Román Osorio y M.I. Jorge Valeriano, por su valioso tiempo dedicado a este trabajo.

Al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica de la UNAM, por el apoyo brindado para el desarrollo de este trabajo.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

El presente trabajo se realiza dentro del contexto del proyecto de investigación titulado "Navegación de un robot móvil en un ambiente de oficinas por medio de visión computacional y lenguaje natural", Proyecto Golem, cuyo objetivo principal es habilitar a un robot para ubicarse y navegar en un espacio de oficinas y relacionarse con su entorno mediante información visual y de lenguaje natural hablado en español. El robot móvil que se utilizará dentro de este proyecto deberá ser capaz de dar una visita guiada a una persona dentro del Departamento de Ciencias de la Computación (DCC) del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS, UNAM).

Dentro del Proyecto Golem se deben diseñar e implementar un Módulo de Visión Computacional, un Módulo de Navegación, un Módulo de Lenguaje Natural y un Módulo de Representación e Inferencia. Los primeros tres módulos le servirán al robot para obtener información sobre su ambiente y el cuarto módulo servirá como punto de integración de la información provista por los otros tres módulos.

En específico, el diseño e implementación del Módulo de Lenguaje Natural implica, entre otras cosas, el desarrollo de herramientas que permitan analizar y comprender lenguaje hablado de manera que sea posible interactuar con el robot móvil de esta forma. Problemas de reconocimiento y síntesis de voz así como de análisis gramatical serán tratados dentro de este módulo. Concretamente, en el presente trabajo de tesis, se pretende diseñar e implementar una arquitectura que permita la integración de un Módulo de Reconocimiento de Voz y un Módulo de Navegación, el cual es uno de los problemas que necesariamente se deben solucionar para la implementación

final del Módulo de Lenguaje Natural del Proyecto Golem.

Con los cuatro módulos mencionados, el robot será provisto de mecanismos que le permitirán interactuar con su ambiente, incluyendo las habilidades, aunque limitadas, de usar y comprender lenguaje hablado, las habilidades limitadas de observar, conocer y posicionarse en su entorno. En general, estos mecanismos lo dotarán de limitada inteligencia artificial y le permitirán replicar algunos procesos y métodos que los humanos llevan a cabo de forma natural en situaciones similares. Así, este trabajo de tesis se ubica dentro del área que ocupa a la Inteligencia Artificial.

1.1. Antecedentes

La Inteligencia Artificial (IA) es una de las disciplinas más recientes en el área de la computación. Se inicia formalmente en 1956, (aunque ya se había comenzado a trabajar en ella años atrás). Esta disciplina surge de la necesidad de analizar y entender el concepto de inteligencia y de la búsqueda de técnicas que hagan posible la simulación de funciones de alto nivel del ser humano mediante programas de computadora.

Es complicado encontrar una definición de Inteligencia Artificial que abarque, aún parcialmente, las opiniones de la mayoría de los investigadores en el área. Al igual que el término Inteligencia, Inteligencia Artificial es difícil de definir. En este trabajo de tesis, se va a tomar la definición de la *American Association for Artificial Intelligence* (AAAI por sus siglas en inglés), quienes definen a la IA como *la comprensión científica de los mecanismos fundamentales del pensamiento y de la conducta inteligente y su concretización en máquinas.* Es importante resaltar que la mayoría de las definiciones formales que se pueden encontrar sobre IA están fuertemente relacionadas con la inteligencia humana. Esta definición trata de ser genérica, por lo que hace ver que la IA no sólo estudia la forma de hacer máquinas que resuelvan problemas como lo hacen los humanos, sino que *trata de resolver los problemas que tiene el mundo para la inteligencia en lugar de estudiar a los humanos o a los animales exclusivamente* [McCarthy, 2003]. Así, la IA es libre de ocupar métodos que no sean únicamente observados en las personas, es libre de ocupar métodos que incluso rebasen nuestras capacidades.

Conforme la IA madura, surgen nuevos problemas cada vez mas complejos a resolver. Técnicas como "Redes neuronales", "Búsquedas", "Algoritmos Genéticos", etc., son, algunas veces, difíciles de aplicar en problemas comple-

jos, por lo que surgen nuevas técnicas que reemplazan o complementan a las clásicas. Agentes dentro de IA es una técnica que proporciona modularidad y abstracción, características bastante útiles para tratar con problemas complejos. La abstracción y descomposición de un sistema en agentes permite implementar una solución modular y flexible, dando a cada agente una tarea específica para que estos aporten una solución a un problema particular. Así, al final, se tienen sistemas de agentes que interactúan entre sí para solucionar un problema específico y posiblemente complejo dentro del campo de la IA.

Un aspecto importante dentro del razonamiento humano es la Inferencia, por lo que la Inferencia es una de las operaciones fundamentales dentro la Inteligencia Artificial. La Inferencia Simbólica, como se le conoce dentro de IA, se refiere a la "derivación" de nuevos hechos a partir de hechos conocidos[1]. Así, lo que se pretende, es utilizar abstracciones como la de "Agentes", que se les permita obtener representaciones de su entorno, y mediante nuevas inferencias, deducir que hacer o como comportarse dentro de su entorno.

Formalmente y para fines de este trabajo, un agente se define como *un ente capaz de realizar acciones autónomas flexibles*. Los términos *acciones autónomas flexibles*, engloban tres cosas [Wooldridge, 2000]:

- *Reactividad*. El agente está habilitado para percibir su ambiente y responder oportunamente a una situación dada incluso para alterar su entorno.
- *Pro-actividad*. El agente es habilitado para tomar la iniciativa y exhibir un comportamiento orientado a la meta.
- *Habilidad social*. El agente es capaz de interactuar con otros agentes y cooperar con ellos a fin de alcanzar una meta.

Para la implementación de los agentes surge un nuevo paradigma de programación: la programación orientada a agentes, la cual se basa en programarlos directamente en términos de comportamiento en vez de considerar sólo las estructuras de datos como se hace en paradigmas tradicionales como la programación estructurada. La IA abarca una enorme cantidad de ramas y aplicaciones. Heurística, Programación Genética, Búsquedas, etc., son algunas ramas dentro de la IA; Procesamiento de Lenguaje Natural, Robótica,

[1]Free On-Line Dictionary of Computing, http://foldoc.doc.ic.ac.uk/foldoc/index.html

Reconocimiento de Voz, Sistemas Expertos, etc. son algunas de las aplicaciones más comunes de la IA[2][McCarthy, 2003]. El presente trabajo relaciona dos aplicaciones de IA: Robótica y Reconocimiento de Voz.

Por una parte, dentro de la robótica se diseñan robots para fines de todo tipo, aunque su dominio tradicional ha sido la manufactura, donde las tareas repetitivas en una línea de producción resultan tediosas para un ser humano y se prestan de manera natural para la automatización.

En los últimos años los asistentes robóticos han venido a ser comunes en entornos tales como oficinas o casas, ambientes donde necesitan tener comunicación con usuarios poco familiarizados a esta tecnología. En situaciones de este tipo, la interacción vía lenguaje natural hablado parece tener un horizonte prometedor.

Por otra parte, el avance de la tecnología en el Reconocimiento de Voz permite crear aplicaciones que pueden ser aprovechadas para hacer interfaces más flexibles. Estas interfaces pueden ser utilizadas (como en este caso) para controlar un robot. Con el Reconocimiento de Voz se puede procesar lo que el usuario dicte y generar comandos que vayan dirigidos al Sistema de Navegación de un robot móvil.

Hoy en día podemos encontrar antecedentes importantes de robots asistentes de oficina como lo son POLLY, MAIA y JIJO-2. El primero fue desarrollado en el MIT entre 1992 y 1993, aunque su interacción no empleaba reconocimiento de voz, ni mucho menos procesamiento de lenguaje natural, tenía capacidad para patrullar un piso del laboratorio de inteligencia artificial de dicho Instituto, encontrar visitantes y guiarlos en sus visitas mediante el uso de facilidades multimedia. MAIA y JIJO-2 son robots móviles que pueden navegar con lenguaje natural hablado. MAIA es un robot que lleva objetos de un lugar a otro y también es capaz de obedecer a simples comandos hablados. JIJO-2 puede transportar información, guiar gente a través de un ambiente de oficinas y puede establecer un diálogo (en japonés) acerca de su entorno usando un manejador de diálogo basado en estados.

1.2. Importancia y justificación

En el presente trabajo se pretende integrar un Reconocedor de Voz al Sistema de Navegación de un robot móvil (un robot que usa llantas para

[2]John McCarthy, Investigador pionero de IA, acuño el término "Artificial Intelligence", http://www-formal.stanford.edu/jmc/

desplazarse), con el fin de poder hacer referencia con voz a su entorno de navegación. De la importancia de poder interactuar con el robot móvil mediante la voz surgen varias ventajas importantes:

- Establecer un diálogo hablado representa una "interfaz de usuario" natural al ser humano.
- Personas sin muchos conocimientos en el área podrán interactuar con el robot, es decir, no será necesario gente experta que sepa programarlo para poder interactuar con él.
- Personas con discapacidad física podrán entrar contacto con los robots, de forma que estos les pueden ofrecer diversos servicios (como guías, como auxiliares que pueden acercar objetos o encender interruptores, etc.).
- Mediante la voz el robot podrá estár habilitado para obtener información valiosa acerca de su entorno (por ejemplo: realizar exploraciones).

El uso de voz en español representa una característica importante en el desarrollo de este trabajo, ya que los robots móviles asistentes de oficina mencionados en la sección anterior no pueden ser (directamente) aprovechados por gente de habla hispana. En nuestro caso, el robot recibirá ordenes en nuestro idioma que le permitirán navegar.

1.3. Planteamiento del problema

Para el desarrollo de este trabajo se cuenta con dos sistemas principales. Por un lado el sistema de reconocimiento de voz[3], que fue desarrollado inicialmente para el proyecto DIME [DIME, 2001]. Por otro lado se tiene el sistema de navegación del robot. La integración de ambos sistemas representa un trabajo laborioso e interesante. Es necesario proponer una arquitectura para su integración y también hacer un análisis del tipo de interacción entre el humano y el robot para obtener e implementar un modelo de lenguaje sencillo que permita la comunicación entre ellos a través de comandos dirigidos al sistema de navegación.

[3]Una breve introducción a los SRV y una descripción detallada de las características del SRV utilizado en este trabajo se presentarán en el capítulo dos.

Específicamente el problema a resolver es el de:

La integración de un módulo de reconocimiento de voz a un sistema de navegación de un robot móvil.

El robot que se utilizará en el presente trabajo es un *Magellan Pro Mobil Robot RWI* (en el capítulo dos se presenta una descripción detallada de este robot). El sistema de navegación del robot está implementado bajo la arquitectura *CORBA*[4] y el SRV esta inicialmente configurado para trabajar bajo las especificaciones del proyecto DIME [DIME, 2001]. Se requiere está integración para que la interacción con el robot sea mediante comandos de voz emitidos por un usuario.

1.4. Estrategia de solución

La solución del problema planteado puede realizarse en dos etapas como se menciona a continuación.

La primera etapa consiste en la Adaptación del SRV al contexto de navegación. Actualmente, el SRV funciona bajo los requerimientos del proyecto DIME, contiene una configuración que le permite reconocer diálogos sobre el diseño de cocinas, por lo que el vocabulario que puede reconocer esta formado por palabras de este dominio particular. Por esta razón, en esta etapa es necesario obtener una gramática como modelo de un lenguaje sencillo. Las expresiones[5] que se usarán para obtener este modelo harán referencia a comandos básicos que el robot ejecutará.

La segunda consiste en resolver el problema de la *integración* entre los dos sistemas (SRV y Sistema de Navegación). En esta etapa se debe proponer una arquitectura en donde se integre el reconocedor de voz y se resuelva su comunicación con el Sistema de Navegación. En este sentido, para la integración, es necesario una herramienta flexible, multiplataforma y que nos permita desarrollar módulos programados en diferentes lenguajes y para ambientes distribuidos. La herramienta que se utilizará es OAA *Open Agent Architecture* [Martin, 1999] debido a que cubre con los requerimientos anteriores. OAA proporciona un marco para desarrollar e integrar agentes software en ambientes distribuidos. Los agentes se pueden comunicar entre ellos, incluso

[4]Ver página web de CORBA http://www.corba.org

[5]Los términos *expresión*, *frase* y *oración* se usarán en este trabajo de forma indiferente para hacer referencia a las oraciones o frases con las que se hablará al robot.

si están en distintas máquinas. Una descripción más detallada de OAA se presenta en el capítulo tres.

1.5. Objetivos

1.5.1. Objetivo general:

El objetivo general de este trabajo de tesis es desarrollar una infraestructura que sirva como base para realizar experimentos e investigación sobre fenómenos multimodales que involucren el idioma español, específicamente sobre aspectos linguísticos y gráficos en la interacción hombre-máquina en el marco de la línea de investigación del grupo de Sistemas Multimodales Inteligentes del DCC del IIMAS.

1.5.2. Objetivos específicos:

El objetivo específico de este trabajo de tesis es proponer una arquitectura básica que permita integrar un Sistema de Reconocimiento de Voz a el Sistema de Navegación de un Robot Móvil. Para esto es necesario:

1. La *adaptación del SRV al contexto de navegación* . Como se mencionó anteriormente, el SRV esta diseñado en función del lenguaje utilizado en el proyecto DIME, específicamente, sobre un lenguaje orientado al diseño de cocinas, por lo que es necesario una fase de adaptación previa. Para ajustarlo se requiere obtener una nueva Gramática y un nuevo vocabulario, ambos orientados al dominio de la aplicación que se pretende obtener. Para lograr esto es necesario realizar un análisis del tipo de interacción, en el que se obtendrá un modelo de lenguaje que se integrará posteriormente al SRV. La adaptación del SRV consiste en:

   a) Proponer conjunto de expresiones que se van a reconocer y con las que se podrá hacer referencia al Sistema de Navegación del robot móvil.

   b) Analizar las expresiones para construir un modelo simple del lenguaje y obtener la Gramática.

   c) Incorporar el modelo obtenido al Sistema de Reconocimiento de Voz.

2. La *integración del SRV al Sistema de Navegación del robot Móvil.* Es necesario proponer e implementar una arquitectura que sirva como interfaz entre los dos sistemas, esto consiste básicamente en:

   a) Proponer una arquitectura flexible de agentes OAA para la integración de los dos sistemas.

   b) Independientemente de la arquitectura propuesta, se debe solucionar el problema de comunicación entre OAA y CORBA para la implementación de la arquitectura.

1.6. Alcances y limitaciones

El Sistema de Reconocimiento de voz que se usará en el presente trabajo es un sistema de reconocimiento de voz continua. Es importante recalcar que las frases se deben limitar al vocabulario del dominio de la aplicación. Con el vocabulario se formarán expresiones que harán referencia a cuatro comandos básicos de navegación: *avanza*, *retrocede*, *izquierda*, *derecha* y también habrá expresiones para hacer referencia a comandos combinados.

Ejemplos de expresiones haciendo referencia a este tipo de comandos son: *avanza un metro*, *avanza medio metro hacía atrás* o *gira a la izquierda 45 grados* o *camina un metro hacía adelante y gira cuarenta y cinco grados a la izquierda.* Este tipo de expresiones son suficientes para hacer pruebas a la arquitectura planteada. Es importante resaltar que las instrucciones de bajo nivel quedan bajo control exclusivo del robot (control reactivo), el usuario del sistema no tiene acceso a ellas, pues podría interferir con la autoconservación del robot[6]. El vocabulario que se abarcará será de aproximadamente 180 palabras, suficientes para formar las frases que permiten al robot navegar en su entorno.

1.7. Organización de la tesis

La tesis se organiza de la siguiente manera. En el capítulo dos se hace una revisión de los conceptos generales de Sistemas de Reconocimiento de Voz,

[6]Dentro del proyecto Golem, el Modulo de Navegación incorporará técnicas de control reactivo que le permitirán al robot detectar obstáculos oportunamente y así poder evitar colisiones. El usuario no deberá tener acceso a este tipo de funciones de bajo nivel, las cuales estarán bajo el control exclusivo del Modulo de Navegación

de Robótica y también se da una breve introducción al concepto de "Agentes de Software". En ese mismo capítulo se describen las características del SRV y del robot utilizados en la presente investigación. En el capítulo tres se presentan los fundamentos básicos de dos arquitecturas para el desarrollo de sistemas distribuidos: *OAA* y *CORBA*, las cuales son parte fundamental en la solución del problema planteado. En el capítulo cuatro se describe la metodología para la solución del problema, incluyendo una breve descripción del diseño y la implementación de los agentes de software que se desarrollaron para la integración de los módulos. En el capítulo cinco se analizan los resultados obtenidos en la evaluación del sistema implementado. Por último, en el capítulo seis se muestran las conclusiones y se describe el posible trabajo a futuro.

Capítulo 2

Conceptos Generales

En este capítulo se revisan los principales conceptos referentes a sistemas de reconocimiento de voz, robótica y agentes de software. El capítulo está conformado por tres secciones, en la primera de ellas se define un SRV, se menciona su clasificación y arquitectura y se concluye con las principales características del SRV utilizado. En la segunda sección se revisan conceptos de robótica, se mencionan algunos antecedentes de proyectos que usan robots móviles con procesamiento de lenguaje natural similares al proyecto de investigación del cual forma parte esta tesis y se listan las características del robot móvil utilizado. Finalmente, en la tercera sección, se da un breve resumen de agentes de software.

2.1. Sistemas de Reconocimiento de Voz (SRV)

2.1.1. Definición

Un sistema de reconocimiento de voz (SRV) tiene como principal objetivo el convertir una señal acústica, capturada por un micrófono o un teléfono, a un conjunto de palabras. Mediante el reconocimiento de voz se pueden desarrollar técnicas y sistemas que habiliten a las computadoras para aceptar comandos o datos de entrada para algunas aplicaciones. Formalmente, un SRV se define como sigue:

"Sea $W = \omega_1, \omega_2, ..., \omega_n$ una secuencia de n símbolos o palabras, donde n es un número natural, y sea A una señal acústica de voz.

Un SRV consiste de los siguientes elementos:

- Un canal de entrada, que puede ser un micrófono o una línea telefónica, que recibe la pronunciación de W y produce su correspondiente señal acústica A.
- Un módulo de reconocimiento de voz que al procesar A produce la secuencia de palabras reconocidas W.
- Un módulo receptor que interpreta a W como un comando de control, un dato de entrada a una aplicación, o simplemente como el texto correspondiente al conjunto de palabras reconocidas." [Uraga, 1999]

2.1.2. Clasificación

Los SRV se pueden clasificar de acuerdo a ciertos parámetros, los más importantes se muestran a continuación [Uraga, 1999]:

1. **Forma de habla**
   - *a*) *Palabras aisladas.* En este tipo de reconocimiento, se requiere que el hablante separe las palabras por medio de pausas.
   - *b*) *Voz continua.* Este tipo de reconocimiento es considerado más difícil que el reconocimiento de palabras aisladas. Los límites entre las palabras no son fáciles de detectar en el habla continua.
2. **Dependencia del hablante**
   - *a*) *Sistemas dependientes del usuario.* Este tipo de sistemas requieren ser entrenados[1] por un sólo usuario, de modo que sólo van a reconocer la voz de su entrenador.
   - *b*) *Sistemas independientes del usuario.* Este tipo de sistemas permiten reconocer la voz de un conjunto numeroso de usuarios.

[1]Entrenamiento de un SRV significa que el sistema, antes de su uso normal, se somete a una fase de adiestramiento en donde se le proporcionan muestras del habla dentro del dominio de la aplicación que le permitirán un mejor modelado del lenguaje y por consecuencia un mejor reconocimiento.

3. **Adaptación al hablante.**

   Un sistema de este tipo implementa un proceso por medio del cual permite aprender las características de la voz de un usuario y así mejorar su desempeño en la siguiente interacción con él. El proceso de adaptación puede ser estático o dinámico. En el proceso de adaptación estático es necesario una etapa de entrenamiento antes de que el sistema sea usado. En el proceso de adaptación dinámico no es necesario una etapa de entrenamiento ya que el sistema aprende las características de la voz de algún usuario específico conforme es utilizado por este.

4. **Tamaño del vocabulario.**

   La exactitud y eficiencia de un SRV están estrechamente ligadas con el tamaño del vocabulario. Entre más grande sea el número de palabras que puede reconocer, el sistema requiere de mayor tiempo de procesamiento y es más factible la posibilidad de reconocer equivocadamente las palabras. Los SRV's se clasifican generalmente como sistemas de vocabulario pequeño, mediano o grande, la forma de cuantificar estos términos varía en la literatura, en este caso particular se tomará como base la cuantificación presentada por Gibbon (tabla 2.1) [Gibbon et al., 1997]:

Vocabulario	**Rango**
Pequeño	# Palabras $\leq$ 10 palabras
Mediano	10 $<$ # palabras $\leq$ 100
Grande	100 $<$ # palabras $\leq$ 1000
Muy grande	# palabras $>$ 1000

Tabla 2.1: Clasificación de un SRV de acuerdo al tamaño del vocabulario

5. **Perplejidad**

   La perplejidad es una medida de la complejidad de la tarea de reconocimiento combinada con el tamaño del vocabulario y el modelo del lenguaje. La perplejidad es definida como un parámetro que estima el número de palabras que pueden seguir a una palabra después de aplicar

un modelo del lenguaje. Dependiendo de la medida de perplejidad la tarea de reconocimiento puede ser fácil o difícil. La tarea de reconocimiento es fácil si la perplejidad es menor que 20 y difícil si es mayor que 100 [Uraga, 1999].

6. **Tolerancia al ruido**

   Hay algunos parámetros externos que pueden afectar el desempeño de un sistema de reconocimiento de voz, como las características del entorno e inclusive el tipo y colocación del micrófono. De acuerdo a este parámetro, los SRV's pueden ser:

   a) *Sistemas no tolerantes al ruido.* Este tipo de sistemas se implementan y entrenan en laboratorios con un mínimo de ruido y con equipo de alta calidad.

   b) *Sistemas tolerantes al ruido.* Este tipo de sistemas se implementan y entrenan en un ambiente similar al entorno donde va a trabajar el reconocedor.

   c) *Sistemas con capacidad de rechazo.* Algunos sistemas son usados en entornos con mucho ruido, lo que puede ocasionar que reconozcan una palabra que aún no se ha pronunciado. Es por esto que deben implementar una especie de capacidad de rechazo de palabras fuera del vocabulario o sonidos extralingüísticos (toser, respirar, reír, etc.).

   d) *Sistemas robustos.* Son sistemas cuyo desempeño no disminuye si se presentan cambios como usar diferentes micrófonos, ruido ambiental, ruido en el canal, etc.

   Además de estos parámetros para caracterizar un SRV, se agregan dos más que propone Ronald A. Cole [Cole et al., 1995]:

7. **Modelo del lenguaje.**

   El lenguaje puede ser modelado usando gramáticas de estados finitos o n-gramas. El modelo del lenguaje puede ser especificado como una gramática de estados finitos cuando las palabras permisibles que siguen a otras palabras son especificadas explícitamente. Los modelos del lenguaje más generales, más aproximados al lenguaje natural, pueden ser

especificados por medio de n-gramas, donde la siguiente palabra dentro de la frase que se intenta reconocer es predecida (Para mayor información sobre n-gramas se puede consultar [Jurafsky, 1999]).

8. **Estilo de habla.**

   Reconocer palabras improvisadas o generadas espontáneamente es mucho más difícil que reconocer palabras que son leídas. Esto se debe a que en el habla espontánea pueden haber sonidos que no permiten un habla fluida, como los tartamudeos, las autocorrecciones que se hacen cuando no se pronuncia una palabra correctamente, los pequeños lapsos de tiempo que se hacen para pensar antes de pronunciar la siguiente palabra, etc.

2.1.3. Arquitectura de un SRV

En la figura 2.1 se muestran los componentes principales de un SRV (figura tomada de [Jurafsky, 1999]).

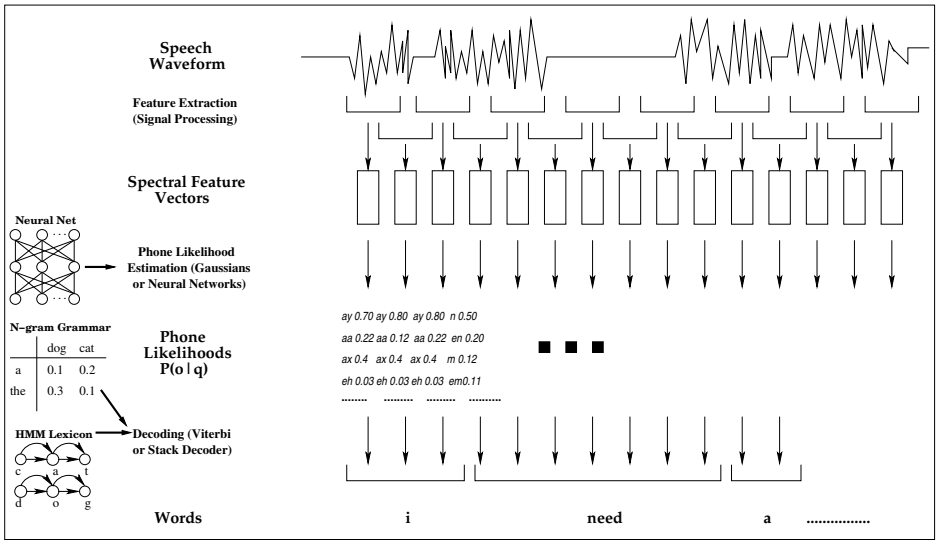

Figura 2.1: Arquitectura de un SRV

En la primera etapa, extracción de rasgos (*Feature Extraction*), la señal

acústica de voz es procesada. Mediante una segmentación se obtienen "frames"[2] usualmente de 10, 15 o 20 milisegundos. A partir de estos "frames" es posible adquirir vectores de rasgos espectrales de los cuales se consigue información sobre la cantidad de energía que hay en la señal a determinada frecuencia.

En la segunda etapa, reconocimiento de fonemas (*Phone Likelihood*), se seleccionan sonidos individuales del habla, como fonemas, bifonemas, sílabas, palabras o combinaciones de palabras.

Finalmente, en la tercera etapa, nivel de decodificado (*Decoding*), mediante un algoritmo (como el de Viterbi [Jurafsky, 2000]) en conjunción con un diccionario de pronunciación de palabras y un modelo del lenguaje, se busca la secuencia de palabras más probable dados los eventos acústicos actuales.

2.1.4. El Sistema de Reconocimiento de Voz

Basándose en los criterios para clasificar a los sistemas reconocedores de voz presentados en la sección 2.1.2, el SRV utilizado para la presente investigación tiene las características que se presentan en la tabla 2.2.

Parámetro	**Clasificación del SRV utilizado**
Forma de habla	Voz continua
Estilo de habla	Expresiones espontáneas[3]
Dependencia del hablante	Independiente del hablante
Adaptación al hablante	Sin proceso de adaptación al hablante
Tamaño de vocabulario	Grande[4]
Perplejidad	Fácil
Tolerancia al ruido	No tolerante a ruido excesivo
Modelo del lenguaje	Por medio de una gramática

Tabla 2.2: Características del sistema del SRV utilizado.

Como se mencionó en el capítulo uno, además de estas características, es importante hacer notar que el usuario debe ser claro al momento de hablar,

[2]Los "frames" son segmentos en el dominio del tiempo de la señal de voz que sirven para analizarla.

[3]Expresiones espontáneas usando un lenguaje restringido a un dominio.

[4]De acuerdo a la clasificación respecto al tamaño del vocabulario que da Gibbon, [Gibbon et al., 1997].

debe articular con claridad las palabras y además se debe restringir al vocabulario del dominio de la aplicación.

2.2. Robótica

En esta sección se define el concepto de robot, se explican brevemente las características de un sistema de navegación y finalmente se termina con una revisión de las características del robot móvil utilizado.

2.2.1. Definición

Los investigadores en el área de robótica han desarrollado muchas definiciones en torno al concepto de robot, pero la mayoría de ellas surgen de acuerdo a la funcionalidad del dispositivo, por ejemplo, un ingeniero industrial define robot como *un dispositivo capaz de realizar tareas repetitivas en una línea de producción.* Otros autores tratan de dar una definición general que abarque todos los tipos de robots que podemos encontrar, como Russell y Norvig, quienes definen robot como *un agente artificial, activo, cuyo entorno es el mundo físico.* El término agente fue definido en la sección 1.1, el término artificial descarta a los animales, el término activo descarta a las piedras y el término físico descarta a los agentes de software cuyo entorno son las redes de cómputo, las bases de datos, los sistemas de archivos, etc. [Russell y Norvig, 1996].

Para fines del presente trabajo, se definirá robot como *un dispositivo electromecánico, con cierto grado de autonomía que por medio de programas de cómputo puede simular un comportamiento inteligente siendo capaz de realizar una variedad de tareas sin tener que modificar su estructura física.*

Actualmente es posible encontrar una gran variedad de tipos de robots. Algunos son modelos comerciales hechos para automatizar la producción en fábricas, para lo cual son fijos (como brazos metálicos), otros tienen capacidad de desplazarse y sirven como exploradores. El robot que se utiliza en esta investigación es un robot móvil, que tiene la capacidad de desplazarse mediante el uso de ruedas.

2.2.2. Sistemas de navegación

En robótica móvil, el término navegación se usa para hacer referencia a un conjunto de funciones que permiten a un robot explorar y desenvolverse con cierto grado de autonomía en su entorno. Para poder realizar esta función es necesario que el robot tenga una representación de su ambiente. Con esta representación interna, el robot puede planear rutas e ir de una ubicación a otra. Esta tarea se complementa con un módulo previsor de colisiones que adiciona técnicas de control reactivo para proteger la conservación del robot. El modelo del ambiente no sólo puede servir para planeación de rutas, también se puede utilizar como una estructura de datos que incorpora información semántica a diferentes niveles de abstracción. Las tareas de alto nivel, como el procesamiento de lenguaje natural, en un sistema de control de un robot móvil pueden ser realizadas usando la representación interna que tiene el robot de su entorno, de esta forma, el sistema de navegación sirve como una interfaz entre procesos cognitivos de alto nivel y el sistema control de bajo nivel del robot [Theobalt, 2000].

De acuerdo con Theobalt, las funciones principales de un sistema de navegación son [Theobalt, 2000]:

- *Modelado del ambiente.* Para ejecutar diversas tareas, el robot debe contar con una representación de su entorno.

- *Localización y corrección de la posición.* El robot debe ser capaz de ubicarse a si mismo con respecto al modelo de ambiente previamente construido, y debe estar equipado con métodos para corregir errores que se acumulan en el tiempo.

- *Administración del mapa e integración e interpretación de los sensores.* El modelo del ambiente debe ser consistente con el estado actual del entorno. Los robots pueden venir equipados con diferentes tipos de sensores, la información que ellos entregan debe ser interpretada e integrada dentro del modelo del entorno.

- *Planeación de rutas y navegación.* El robot debe ser capaz de planear rutas para poder moverse de un lugar a otro esquivando posibles obstáculos.

- *Servir como una interfaz para funciones cognitivas de alto nivel.* El sistema de navegación del robot es un nivel básico de control y el modelo

del ambiente que es creado en él puede servir como una representación unificada para tareas de alto nivel como el procesamiento de lenguaje natural.

Esta última función ha despertado grandes expectativas en la comunidad de inteligencia artificial, dado que es de gran interés la conexión entre el lenguaje natural hablado y los sistemas no-simbólicos de control robótico. Actualmente, como ya se mencionó en capítulo anterior, son comunes los robots móviles asistentes de oficina que además implementan sistemas de navegación integrados con sistemas de procesamiento de lenguaje natural hablado y, aunque el objetivo de esta tesis no es hacer una implementación de este tipo, se pretende crear una plataforma base que posteriormente pueda extenderse y servir para realizar investigaciones en esta área.

2.2.3. Robots móviles y lenguaje natural

En el capítulo uno se mencionaron algunos antecedentes de robots asistentes de oficina, en esta sección se revisan antecedentes de robots móviles con procesamiento de lenguaje natural:

- RHINO es un robot que funciona dentro de un museo sirviendo como guía y dando exhibiciones particulares a los visitantes. Aunque no soporta diálogos reales, puede reconocer frases simples como *execute tour number 3*, "ejecuta el recorrido número tres". Los paseos que RHINO ofrece siguen rutas fijas y en ellos utiliza expresiones habladas previamente grabadas para dar explicaciones a los visitantes [Thrun, 1998].

- TJ es un robot con capacidades similares a las de POLLY (mencionado en la sección 1.1). No utiliza lenguaje natural hablado, pero puede obedecer a comandos simples como "go to the conference room" ó "go left" y contestar preguntas acerca de donde se encuentra. Ambas funciones las realiza por medio de un teclado [Torrance, 1994].

- Hygeiorobot es un robot móvil asistente para hospitales. Puede entregar medicina o llevar mensajes a un cuarto específico o a un paciente. Para realizar estas funciones utiliza un SDS[5] ("*spoken dialogue systems*") que permite a los usuarios especificar la información necesaria

[5]Un *spoken dialogue system* (SDS) es un enfoque que permite la interacción hombre-máquina basada en lenguaje natural hablado, permitiendo la comunicación por medio de diálogos hablados.

para realizar la tarea. Los usuarios también pueden preguntarle información acerca de los pacientes, como su número de cuarto o su teléfono [Spiliotopoulos, 2001].

- El Instituto de Investigación Microsoft de la Universidad de Macquarie desarrolló un robot asistente de oficina que puede entregar paquetes, guiar a los visitantes a las oficinas y ofrecer paseos por las instalaciones. El robot usa un sistema comercial para reconocimiento de voz, un manejador de diálogo basado en estados y un analizador del lenguaje basado en el trabajo de Androutsopoulos [Androutsopoulos, 1996].

2.2.4. Golem: El robot móvil utilizado

El robot usado en este trabajo de investigación es el modelo comercial MagellanPro fabricado por Real World Interfaces, dentro de este proyecto el robot es llamado "Golem". Tiene una base una base circular de 40.6 cm. de diámetro, dos ruedas principales controladas independientemente y tiene aproximadamente 24.5 cm. de altura. El robot contiene una computadora Pentium a bordo on 128 MB de RAM y un disco duro de 6.4 GB de capacidad. Utiliza el sistema operativo Linux. El robot puede alcanzar una velocidad de traslación de 2.5 m/s y una velocidad de rotación de 270 grados/s. El robot está compuesto por un chasis con 16 caras, cada una contiene un sensor infrarrojo, un sensor ultrasónico (sonar) y un sensor de contacto (*bumper*). La parte superior del chasis sirve como base para una cámara CCD y un sintetizador de voz. Para comunicarse con el robot, se utiliza una conexión de red Ethernet inalámbrica. El sistema de control provee información odométrica por medio de los *encoders* que contiene cada una de las ruedas principales. En al apéndice E se muestran con más detalle las especificaciones de Golem.

El entorno de navegación del robot será el Departamento de Ciencias de la Computación (DCC) del IIMAS (figura 2.3). El ambiente consiste en un corredor principal de aproximadamente 1.5 m de ancho. A lo largo del corredor se encuentran cubículos del personal del DCC. También se encuentran dos áreas secretariales y dos laboratorios de cómputo en los extremos opuestos del corredor.

Golem viene acompañado con un toolkit de software que implementa la arquitectura del robot llamado *Mobility Robot Integration Software*. Este toolkit consiste, entre otras cosas, de un conjunto de librerias programadas en lenguaje de programación C++. *Mobility Robot Integration Software* facilita

Figura 2.2: Golem.

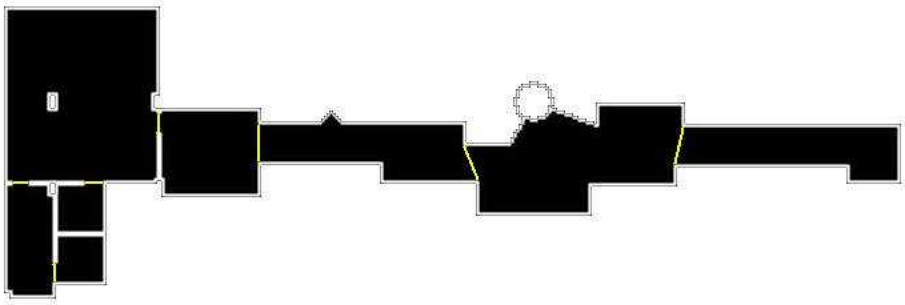

Figura 2.3: Departamento de Ciencias de la Computación del IIMAS.

el desarrollo de código de programación para controlar al robot mediante un *modelo de objetos* de sus componentes. Este modelo agrupa un conjunto de objetos en donde cada uno es una abstracción de las partes que forman al robot, como los sonares, los sensores infrarrojos, su comportamiento, las estrategias para almacenamiento de datos, etc.

En el apéndice C se proporcionan ejemplos de programación utilizando librerias de *Mobility*.

Un concepto importante dentro de Robótica Móvil es la Odometría. La Odometria se basa en el movimiento de las llantas del robot para obtener su posición. Esto trae consigo, entre otros, dos problemas principales relacionados con la estimación de la posición. El primero se refiere a que es imposible conocer la orientación exacta del robot debido a la acumulación de errores. Un pequeño error de un grado se puede convertir en un error de n centímetros conforme el robot vaya viajando determinadas distancias. El segundo problema tiene que ver con que el robot no será capaz de llevar un registro

exacto de cuanto ha viajado. Esto se debe principalmente a las propiedades del piso donde se encuentre. Cada robot se moverá diferente en una superficie lisa o en una superficie rugosa. Por ejemplo, en una superficie lisa es común que las llantas puedan resbalar [Brooks, 2002]. Actualmente existen varias técnicas que permiten reducir los errores en la Odometría, pero el estudio detallado, tanto de estas técnicas como del concepto propio de Odometría, quedan fuera del alcance de esta tesis.

2.2.5. El sistema de navegación

El sistema de navegación del robot se implementará dentro del proyecto Golem. De acuerdo a especificaciones de este proyecto, para navegar, el robot deberá representar su entorno en tres niveles:

- **Geométrico**. En este nivel, el entorno del robot es modelado como una malla formada por celdas discretas (cuadrados de aproximadamente 10 cm.) que contienen información del estado de la celda (esto es si está libre o está ocupada). Este mapa se puede ver como un plano cartesiano y es construido por exploración. En la figura 2.4 se muestra el ejemplo de un mapa métrico, las formas geométricas de color negro dentro de esta figura se forman a partir de las celdas que están ocupadas, por lo que representan obstáculos dentro del mapa del robot (paredes, muebles, botes de basura, etc.). Cada posición dentro del mapa puede ser referida mediante un punto (x, y).

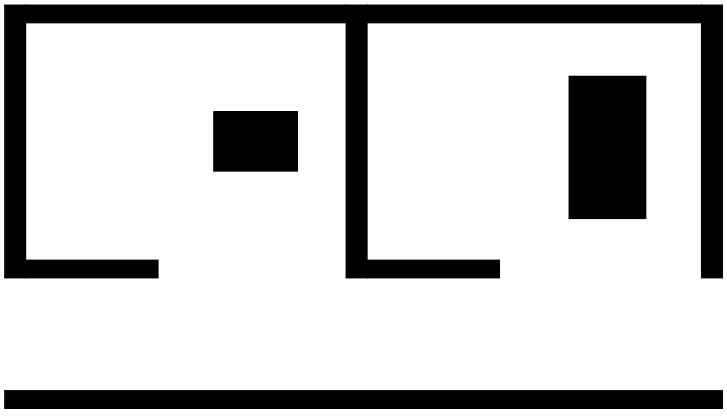

Figura 2.4: Mapa Métrico.

- **Topológico**. Esta representación se construye a partir del mapa geométrico, usando algoritmos como el de descomposición de Voronoi. Este algoritmo divide las áreas libres dentro del mapa geométrico en áreas llamadas regiones topológicas. Estas regiones se representan mediante un grafo en donde cada nodo representa una región y los arcos representan la adyacencia entre regiones [Howie, 1996].

  En la figura 2.5 se muestra, a grandes rasgos, como se construye el mapa topológico correspondiente al mapa métrico de la figura 2.4. Las regiones topológicas están representadas mediante R1, R2. R3, R4, R5, etc., y los arcos entre regiones indican que hay un espacio libre por donde el robot puede ir de una región a otra.

- **Semántico**. Este mapa se obtiene a partir del mapa topológico. En él se agregan etiquetas a las regiones topológicas del mapa anterior (topológico). De acuerdo a la figura 2.5, las regiones R1, R2, R3, etc. pueden ser etiquetadas, así por ejemplo, la región R1 puede ser etiquetada como el "pasillo", mientras que la regón R3 puede ser etiquetada como la "sala de cómputo".

Aunque estos tres niveles de representación del entorno le permitirán al robot obtener un modelo consistente de su ambiente, este modelo del entorno no juega un papel relevante dentro de este trabajo de tesis, ya que los comandos que se enviarán al robot son comandos simples pero suficientes para realizar pruebas a la arquitectura de integración que se propondrá más adelante.

2.3. Agentes de software

En esta sección se da una breve introducción a la tecnología de agentes de software, al final de la misma se exponen algunas aplicaciones que se han desarrollado utilizando como herramienta de diseño e implementación el paradigma de programación que ofrece esta tecnología.

2.3.1. Introducción

Los sistemas basados en agentes ha venido a ser un nuevo paradigma de programación con grandes expectativas dentro de la ingeniería de software. Al igual que otros enfoques, permite la conceptualización, el diseño y

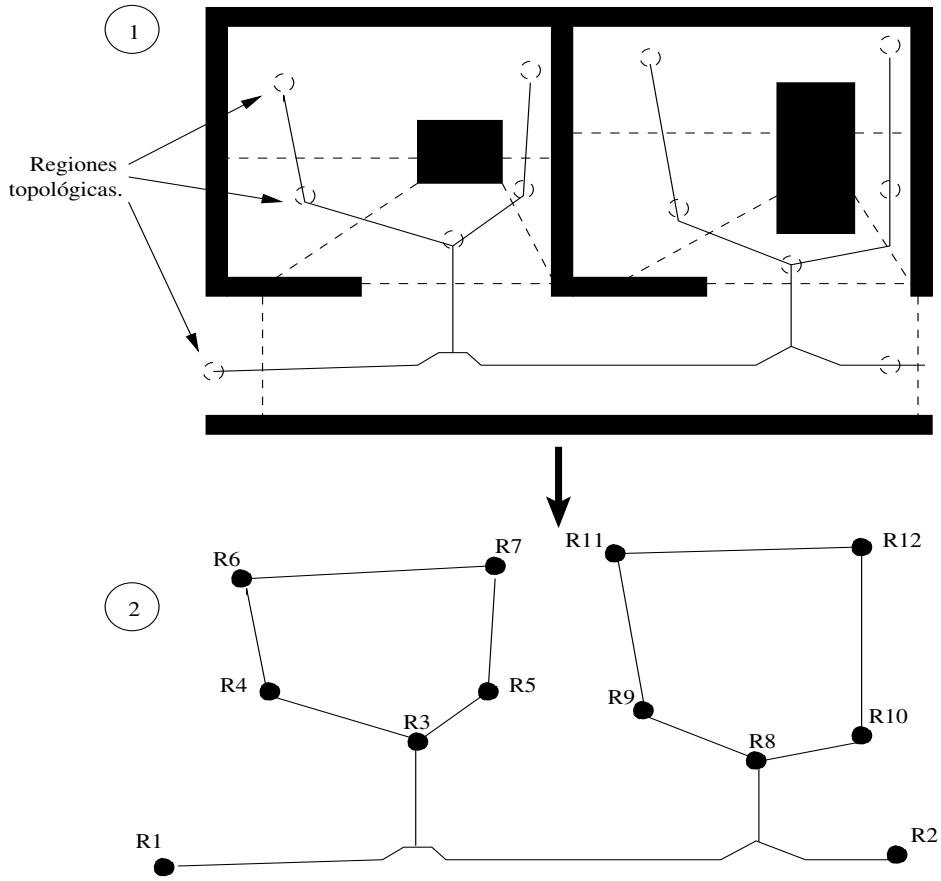

Figura 2.5: Creación de mapas topológicos, (1) Regiones topológicas, (2) Grafo topológico.

la implementación de sistemas computacionales. Este paradigma es especialmente atractivo para desarrollar software que opere en ambientes abiertos y distribuidos, y representa una alternativa a enfoques populares como la programación orientada a objetos.

Actualmente la evolución de los modelos para desarrollar aplicaciones de software se inclinan por desarrollar sistemas que operen en red, en una gran variedad de plataformas y que compartan tanto datos como recursos de procesamiento de datos. CORBA de OMG [OMG, 2002] y DCOM de Microsoft [Microsoft, 2002] son dos tecnologías de objetos distribuidos que permiten desarrollar sistemas de este tipo. Mediante estas tecnologías se pueden desarrollar aplicaciones que consisten de objetos interactivos. Estas interacciones

son predefinidas al momento de escribir el código de la aplicación por lo que la reutilización de los objetos en nuevas aplicaciones es difícil debido a las dependencias que pudieron haber tenido con otros objetos [Martin, 1999].

Un sistema distribuido de agentes de software se puede conceptualizar como una comunidad dinámica, donde varios agentes exponen sus servicios a la comunidad. De esta forma se obtienen sistemas más flexibles y se pueden agregar nuevos agentes.

La figura 2.6 muestra una arquitectura distribuida basada en agentes llamada Pleiades[6]. Esta arquitectura tiene como finalidad proporcionar una estructura de agentes que permitan la conducción, el filtrado y la fusión de información proveniente de distintas fuentes, como es el caso de Internet.

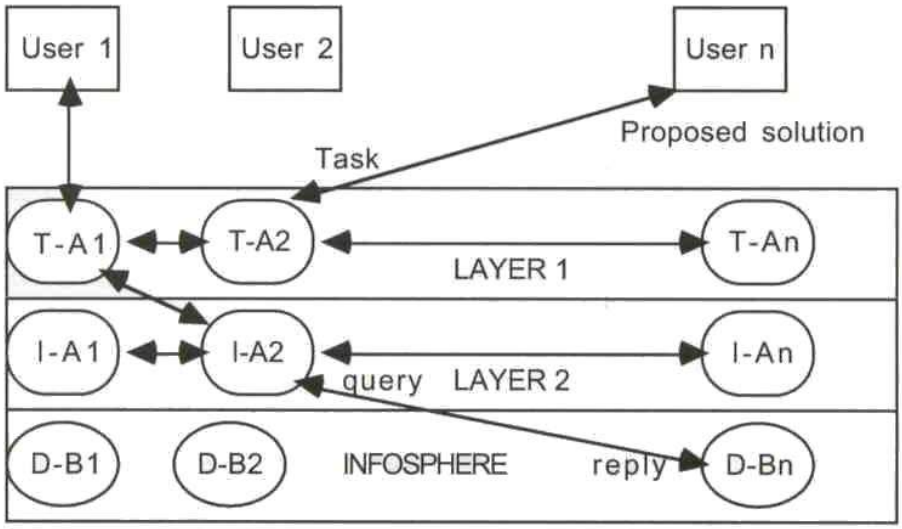

Figura 2.6: Arquitectura Pleiades.

La arquitectura consiste de dos niveles. El nivel uno está compuesto de agentes para *tareas especificas* (T-A1, T-A2, ...,T-An) que elaboran y coordinan planes para la solución de la tarea especificada por el usuario. Estos agentes pueden colaborar con otros agentes del mismo nivel para resolver conflictos o integrar información. En el nivel dos se encuentran los agentes de *información especifica* que tienen acceso a fuentes de información heterogéneas que pueden consultar los agentes del primer nivel. Las fuentes de

[6]Para mayor información sobre este proyecto consultar la URL: http://www2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-5/www/pleiades.html

información son representadas en la figura como D-B1, D-B2, ..., D-Bn y contienen una colección de recursos heterogéneos basados en Internet (*Infosphere*) [Nwana, 1996].

2.3.2. Definición

En el capítulo uno se mencionó la definición de agente y la de agente inteligente. En esta sección se define lo que es un agente de software. En la literatura referente al tema se puede encontrar un gran número de definiciones, pero dado el contexto del presente trabajo, se adopta la definición de agente de software de Janca: "Una entidad de software a la cual se le pueden delegar tareas" [Jennings y Wooldridge, 1996]. Así, para este trabajo, un agente es considerado como un componente de software al que se le pueden transferir diversas tareas.

Sus características principales son las siguientes:

- Un agente de software es autónomo, en el sentido de que ejecuta acciones sin la intervención del usuario.
- Un agente de software es comunicativo, en el sentido de que se comunica con el usuario, con otros agentes o con otros procesos de software.

2.3.3. Aplicaciones

Es evidente que los agentes de software tienen un gran campo de aplicación. El paradigma que ofrecen es una nueva metodología de programación, que al igual que las actuales, permite el diseño e implementación de sistemas de cómputo flexibles en los cada vez más abundantes ambientes distribuidos y heterogéneos.

Los agentes de software son, finalmente, programas de computadora, que actúan con cierta flexibilidad y autonomía. Las siguientes áreas son ejemplos de campos de aplicación:

1. *Software para interfaces de usuario.* Dentro de esta área, los agentes de software han sido utilizados para permitir la comunicación entre la aplicación y el usuario final de forma más amigable. Un ejemplo específico de trabajo sobre esta área se puede consultar en [Lieberman, 1998].

2. *Comercio electrónico.* Además de poder obtener información, en Internet se puede ofrecer y comprar una gran cantidad de artículos y servicios, se puede adquirir un libro, una pizza, un disco compacto, etc. Kasbash es un sistema que permite modelar un mercado y los usuarios pueden crear un agente al que le indican que comprar o que vender. Los agentes del sistema interactúan entre sí comprando y vendiendo entre ellos [Chavez, 1996].

3. *Sistemas de información* (obtención y recuperación). Internet representa un gran potencial para obtener información. Con los agentes de software podemos tomar ventaja de este potencial haciendo uso de un asistente entrenado que sepa que tipo de información es de nuestro interés y cual otra no lo es. Un ejemplo específico de una aplicación dentro de esta área es NEWT, que es un sistema entrenado por un usuario para que pueda seleccionar cierto tipo de artículos y noticias de varios "newsgroups", de modo que después puede ser capaz de sugerir artículos dependiendo de los intereses del usuario que lo entreno [Shet, 1996].

4. *Educación.* En educación se pueden encontrar algunas aplicaciones específicas como Coach que es un agente asistente en el aprendizaje del lenguaje Lisp [Selker, 1994]. En el CIC del IPN el proyecto EVA[7] representa una forma de educación a distancia y transmite material educativo vía Internet, utiliza agentes para ayudar en el aprendizaje de los participantes, de manera que dependiendo del perfil de la persona se prepara un plan de estudio.

Esta lista de aplicaciones de agentes de software no es definitiva, estas áreas son sólo algunas de las muchas en que los agentes de software han comenzado a usarse como una técnica alternativa de desarrollo.

[7]Información sobre este proyecto se puede encontrar en http://www.cic.ipn.mx/publicaciones/catalogo/EVA.htm

Capítulo 3

CORBA y OAA

3.1. Introducción

Los sistemas distribuidos permiten compartir recursos (hardware y software) sobre una red de computadoras de forma transparente. Todos estos recursos, como unidades de almacenamiento, archivos, bases de datos, o incluso las mismas computadoras, existen en redes que van creciendo, lo que ocasiona que se tenga que diseñar e implementar sistemas cada vez más complejos. A esto se agrega que las técnicas de desarrollo, los lenguajes de programación, los sistemas operativos, los protocolos de comunicación, los dispositivos de almacenamiento, etc., cambian constantemente. Como respuesta a este tipo de problemas, se han desarrollado diversas tecnologías que permiten la integración de aplicaciones heterogéneas, entre ellas se encuentran las siguientes [Martin, 1999]:

- Agentes distribuidos
- Objetos distribuidos

OAA, dentro del enfoque de Agentes Distribuidos y CORBA, dentro del enfoque de Objetos Distribuidos, son dos arquitecturas para el desarrollo de sistemas distribuidos, y, aunque no comparten el mismo enfoque, tienen en común el objetivo de permitir que distintos recursos puedan ser integrados o colocados en múltiples computadoras.

Estas dos arquitecturas son parte fundamental en el desarrollo de este trabajo de tesis, por lo que en este capítulo se hace una revisión de los conceptos básicos de cada una de ellas.

3.2. Características generales de CORBA

CORBA (*Common Object Request Broker Architecture*) es un estándar mantenido por el OMG (*Object Management Group*). El OMG es un grupo sin fines de lucro formado en 1989 para desarrollar, adoptar y promover estándares para el desarrollo de aplicaciones en ambientes distribuidos y heterogéneos y para promover el enfoque orientado a objetos. Dentro de CORBA existe una jerarquía cliente-servidor donde, diferentes aplicaciones, comúnmente llamados sistemas de objetos, pueden proporcionar diferentes servicios que pueden ser accedidos mediante una petición por aplicaciones llamadas clientes[1].

Uno de los primeros pasos del OMG fue la definición de OMA (*Object Management Architecture*). OMA es una arquitectura compuesta por un modelo de objetos y un modelo de referencia. A grandes rasgos, el modelo de objetos define como son descritos los objetos en un ambiente distribuido y el modelo de referencia caracteriza las interacciones entre estos objetos[2].

Dentro del modelo de objetos, un objeto es definido como una entidad que encapsula datos y que provee uno o más servicios por medio de interfaces[3] bien definidas que pueden ser requeridos por los clientes.

El componente clave de OMA es CORBA, cuyas características más importantes se listan a continuación:

- ORB (*Object Request Broker*)
- Lenguaje de definición de interfaces (*IDL*)
- OMG IDL *stubs* y OMG IDL *skeletons*
- Invocación dinámica
- Repositorio de interfaces
- Adaptadores de objetos (*Object Adapters*)
- Protocolos Inter-ORB

[1]Dentro del contexto de CORBA, un cliente es cualquier entidad capaz de requerir de algún servicio que provee un objeto.

[2]Para mayor información sobre la OMA de OMG visitar http://www.omg.org.

[3]Dentro del contexto de CORBA, una interfaz de objeto se refiere al conjunto de operaciones (o servicios) que un cliente puede invocar sobre un objeto servidor

Estas características se describen en la siguientes secciones y como a apoyo a esta descripción se presenta la figura 3.1, donde se presenta la estructura general de CORBA y los elementos involucrados en la comunicación cliente-objeto.

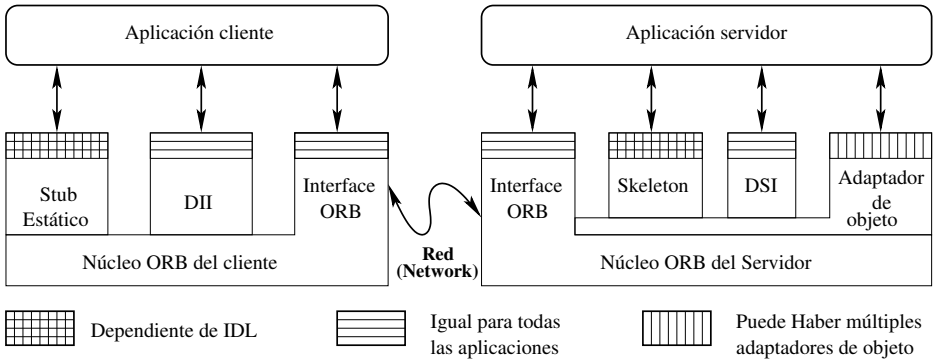

Figura 3.1: Estructura de CORBA.

3.2.1. *Object Request Broker*, ORB

El ORB, como se muestra en la figura 3.1 (núcleo ORB del cliente y núcleo ORB del servidor), es responsable de la comunicación entre los objetos servidores y los clientes. Es capaz de localizar al objeto servidor remoto sobre la red, comunicar la petición a este objeto, esperar por los resultados y cuando estén disponibles, mandarlos de regreso al cliente.

Una característica importante del ORB es que hace que la comunicación entre entidades sea fácil y de forma transparente, gracias a que una de sus capacidades es funcionar como un registro donde se almacenan las descripciones de las interfaces de los objetos disponibles en ese momento.

El ORB también oculta al cliente la ubicación y el lenguaje de implementación del objeto destino (*target object*[4]); además, el cliente no se debe preocupar en saber qué protocolo de comunicación debe usar el ORB para hacer llegar la petición, ni mucho menos debe saber el estado actual del objeto (si está ejecutándose en algún proceso y listo para recibir peticiones o si está inactivo), el ORB se encarga de activarlo.

[4]Dentro del contexto de una petición, el *target object* es el objeto CORBA "objetivo" de la petición.

En resumen, el ORB es el encargado de proveer la comunicación y de proporcionar una infraestructura de activación para aplicaciones de objetos distribuidos [Vinosky, 1996].

3.2.2. Lenguaje de Definición de Interfaces

Aunque el Lenguaje de Definición de Interfaces no se puede ver explícitamente en la figura 3.1, es una característica importante de CORBA. Para realizar una petición de un servicio, el cliente debe saber las operaciones que el objeto soporta. Mediante una *interfaz* el objeto describe las operaciones y los tipos de datos de las mismas que un cliente puede acceder. La definición de interfaces se puede hacer de dos formas, mediante una definición estática y, alternativamente, se pueden agregar nuevas interfaces mediante un servicio de CORBA llamado *Inteface Repository* que se explicará más adelante.

La definición estática de una interfaz de objeto se hace mediante el Lenguaje de Definición de Interfaces de OMG (*OMG IDL, Interface Definition Language*). Las interfaces son parecidas a las clases en C++ o a los paquetes dentro del lenguaje de programación Java.

Un ejemplo de declaración de una interfaz es la siguiente:

```
interface Manometro {
  //Leer presión
  short get_pres();
  //Actualizar presión retornando el valor previo
  short set_pres(in short new_pres);
};
```

Este ejemplo define una interfaz llamada Manometro que contiene dos operaciones llamadas *get_pres* y *set_pres*. La operación get_pres no toma argumentos de entrada y regresa una variable de tipo *short* que indica el valor de presión actual. La operación set_pres toma new_pres como argumento de entrada y regresa el valor previo de presión. Si un cliente desea acceder al objeto por medio de esta interfaz, debe invocar estas operaciones. Si el cliente desea leer la presión actual debe invocar la operación get_pres. Si desea cambiar el valor de presión debe invocar la operación set_pres. Los argumentos en IDL deben tener declarada su "dirección" para que el ORB conozca si los argumentos van del cliente al objeto o del objeto al cliente o ambos. En la operación set_pres la palabra *in* significa que el argumento new_pres es

enviado desde el cliente hacia el objeto, los argumentos pueden también ser declarados *out* para indicar que se envían del objeto al cliente o *inout*, para indicar que se envían del cliente al objeto, que a su vez, modifica el valor y lo regresa al cliente. OMG IDL permite agrupar conjuntos de interfaces que comparten un mismo propósito, a este conjunto se le llama *módulo*.

Una característica de OMG IDL es que soporta el concepto de herencia como en la programación orientada a objetos. Una interfaz se puede derivar de una o más interfaces y así se puede reutilizar código de OMG IDL.

Como se observa en el ejemplo anterior, OMG IDL es un lenguaje declarativo de alto nivel, lo que significa que no es un lenguaje para desarrollar programas de computadora (como C, C++, Java, etc.), su función es únicamente describir la interfaz de un objeto, su implementación se realiza, ahora sí, mediante un lenguaje como C o C++. Entonces, la definición de la interfaz de un objeto y su implementación son dos pasos generales en la creación de un objeto CORBA.

En resumen, una especificación en OMG IDL comprende:

- Declaraciones de módulos
- Declaraciones de interfaces (soporta herencia y permite la declaraciones de operaciones y atributos)
- Declaraciones de tipos de datos, constantes y excepciones (necesarios para definir las operaciones)

Dado que es un lenguaje para declarar interfaces, OMG IDL no provee instrucciones para control de flujo o variables, pero es a partir de la definición de su interfaz que se obtiene una plantilla genérica del objeto a la que posteriormente se le agregan líneas de código de programación para tener finalmente su implementación. Entre los diferentes lenguajes de programación para implementar objetos se tienen los siguientes: C, C++, Smalltalk, COBOL, Ada, Java, Perl, Tcl, Python, etc.

3.2.3. OMG IDL *stubs* y OMG IDL *skeletons*

Además de permitir la definición de interfaces, IDL genera *stubs* para el lado del cliente y *skeletons* para el lado del servidor durante el tiempo de compilación.

Un *stub* permite al cliente invocar operaciones sobre objetos como si fueran llamadas a funciones locales. El cliente y el *stub* correspondiente a la

interfaz definida en OMG IDL deben ser "ligados" de manera que las peticiones realizadas localmente por un cliente se conviertan en peticiones al ORB para que ejecute un método de un objeto remoto que soporta dicha interfaz.

Un *skeleton* es un mecanismo que representa la interfaz hacía los métodos que implementa cada objeto, de modo que el ORB llama a los métodos invocados por los clientes a través del *skeleton*. El *skeleton* lleva las peticiones del cliente hacía la implementación del objeto CORBA.

La *invocación estática* de operaciones sobre un objeto por un cliente es realizada a través los *stubs* y los *skeletons*. Este tipo de invocación implica que un cliente debe estar ligado mediante los *stubs* con los objetos a los que pide servicios.

Como ejemplo de invocación dinámica se tiene lo siguiente, de acuerdo al ejemplo de definición de una interfaz que se vió en la sección anterior, supóngase que se tiene un objeto cuya interfaz presenta los servicios "leer presión" (get_pres) y "fijar presión" (set_pres), si un objeto cliente desea solicitar el servicio "leer presión", debe tener bien "localizado" al objeto servidor y además debe estár ligado con el correspondiente *stub* asociado a la interfaz del objeto servidor que contiene este servicio, así, una vez que el cliente tiene bien referenciado al objeto servidor, le pasa esta "referencia" al *stub* que finalmente comienza la invocación. Del lado del servidor, la solicitud del servicio especificado en la invocación llega directamente al *skeleton*, quien se encarga de direccionarla al método apropiado dentro de la implementación del objeto servidor.

Para cada interfaz definida en OMG IDL existe su correspondiente *skeleton*, pero el hecho de que exista un *skeleton* no garantiza la existencia de su correspondiente *stub*, ya que los clientes también pueden realizar peticiones mediante *invocación dinámica*.

3.2.4. Invocación Dinámica

Además de la invocación estática, CORBA soporta invocación dinámica de operaciones sobre objetos por medio de las interfaces DII (*Dynamic Interface Invocation*) y DSI (*Dynamic Skeleton Inteface*) como se resumen a continuación.

Interfaz de invocación dinámica

La Interfaz de Comunicación Dinámica (*Dynamic Invocation Interface, DII*) permite a un cliente invocar peticiones dinámicamente sobre un objeto sin tener conocimiento de la interfaz que éste soporta. De esta forma, un cliente puede construir y emitir peticiones en tiempo de ejecución hacía métodos de objetos para los que no posee *stubs*, en lugar de construir las peticiones en tiempo de compilación, como en el enfoque estático.

Para realizar peticiones de forma dinámica, el cliente debe conocer la referencia del objeto, el método deseado y la lista de parámetros. Estas especificaciones las obtiene del Repositorio de Interfaces (*Interface Repository, IR*), una base de datos que almacena las definiciones de las interfaces de objetos en tiempo de ejecución.

Interfaz de Skeletons Dinámicos

Al igual que del lado del cliente, en el lado del servidor se encuentra la Interfaz de Skeletons Dinámicos (*Dynamic Skeleton Inteface, DSI*) que permite el control dinámico de invocaciones a objetos. Mediante esta interfaz, un objeto implementación es alcanzado sin necesidad de utilizar el *skeleton* que es específico de una operación.

3.2.5. Repositorio de Interfaces

El Repositorio de Interfaces (*Inteface Repository, IR*) es un servicio donde se almacena información que puede ser accedida por los clientes en tiempo de ejecución para construir invocaciones dinámicas. Mediante la información almacenada en el IR es posible obtener una descripción detallada de todas las operaciones que un objeto soporta.

3.2.6. Adaptadores de objeto

Un adaptador de objeto (*Object Adapter, OA*) es un objeto que se encarga de acoplar la interfaz que espera el cliente a la interfaz que provee el objeto. CORBA define dos adaptadores, el *BOA* (*Basic Object Adapter*), genérico y el *POA* (*Portable Object Adapter*), una estandarización más rígida y portable.

Entre otras cosas, el OA se encarga de:

- *Generación e interpretación de referencias de objetos.* Los OA's se encargan de producir las referencias de los objetos CORBA y también se encargan de mapear estas referencias a la implementación del objeto.
- *Invocación de métodos.* Los OA's se encargan de invocar los métodos a los que va dirigida la invocación realizada por el cliente.
- *Seguridad en las interacciones.* Los OA's son la forma primaria en que las implementaciones de los objetos acceden a los servicios que provee el ORB, por lo que deben proporcionar las garantías suficientes para llevar a cabo la interacción cliente-servidor.
- *Activación del objeto y de su implementación.* Si es necesario, los OA's activan a los objetos y a sus implementaciones cuando una petición llega hacía ellos.
- *Registro de implementaciones.* Los OA's llevan una relación de los objetos que ofrecen uno o mas servicios a las aplicaciones cliente.

3.2.7. Comunicación entre ORB's

Como ya se mencionó en secciones anteriores, el ORB la parte fundamental de CORBA, dentro de las especificaciones de esta arquitectura es posible encontrar su funcionalidad detallada, pero esta funcionalidad está solamente escrita sobre hojas de papel. La implementación del ORB se puede hacer mediante diferentes lenguajes de programación y por distintas compañías, por lo que surge la necesidad de que distintos objetos CORBA interactuando bajo un ORB puedan interactuar también con otros objetos bajo otro ORB implementado por otra compañia. Para cubrir esta necesidad, a partir de CORBA versión 2.0, se creó una arquitectura que permite la interoperatividad entre ORB´s, esta arquitectura es llamada GIOP (*General Inter-ORB Protocol*). GIOP es un protocolo que específica un conjunto de formatos de mensajes y representaciones de datos estándar para la interacción entre ORB´s. El protocolo IIOP (*Internet Inter-ORB Protocol*) específica como el protocolo abstracto GIOP es implementado sobre TCP/IP.

La interoperatividad obliga a que las referencias de objeto dentro de un ORB se conviertan en referencias de objetos universales al interactuar con otros ORB´s. Estas referencias son llamadas IOR (*Interoperable Object Refe-*

rences) y permiten a un ORB dirigir peticiones y recibir respuestas de objetos que residen en otros ORB´s.

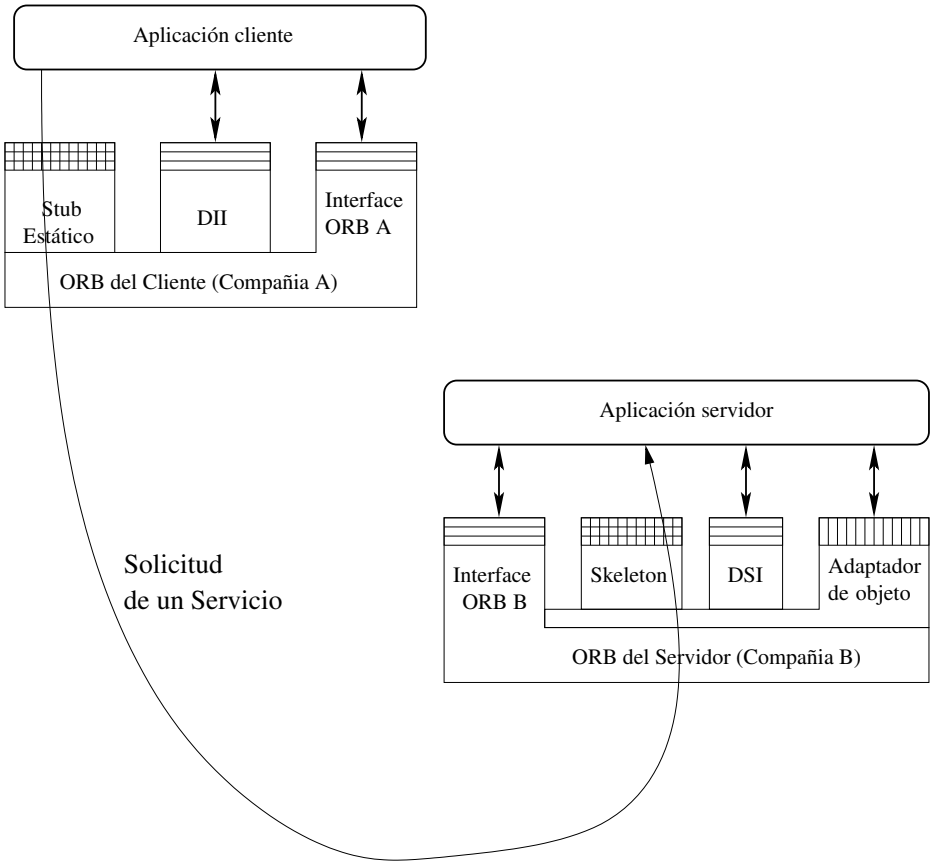

Figura 3.2: Interacción entre distintos ORB's.

La figura 3.2 se muestran dos objetos CORBA interactuando mediante invocación estática y alojados en distintos ORB's.

En el apéndice G se presenta una lista algunas implementaciones de CORBA que se pueden encontrar.

Dado que CORBA permite el desarrollo de sistemas tipo cliente-servidor, en la siguiente sección se explica, de manera general y breve, como se lleva a cabo la interacción entre clientes y objetos en el contexto de una petición[5].

[5]Una petición es una invocación de una operación sobre un objeto CORBA por un cliente.

3.2.8. Flujo general de una petición

Para acceder a un servicio proporcionado por un objeto, el cliente debe invocar una petición hacia él. Como primer paso, el cliente debe obtener una referencia de objeto[6] para poder alcanzar el servicio, un cliente no puede comunicarse, a menos que tenga esta referencia. Una vez que ya la obtuvo, puede elegir entre dos enfoques para invocar peticiones, invocación estática o invocación dinámica, de ambas formas, el cliente transmite la petición hacía el ORB, quien se encarga de guiarla hacía el *objeto implementación*[7]. Del lado del objeto implementación, el ORB "lleva" la petición hacía el adaptador de objetos, encargado de reenviarla hacía el *servant* [8] apropiado que implementa el objeto destino. Al igual que del lado del cliente, el servidor puede elegir entre un enfoque estático o dinámico para enviar la petición al objeto.

3.3. *Open Agent Architecture*, OAA

En esta sección se presentan las características más importantes de *Open Agent Architecture, OAA*. OAA es una arquitectura que, al igual que CORBA, permite el desarrollo de aplicaciones distribuidas en ambientes heterogéneos, pero, a diferencia de esta última, OAA es una arquitectura basada en agentes de software.

Dentro de OAA, las peticiones, al momento de solicitar algún servicio, se expresan en términos de *qué* es lo que se necesita realizar, no importa *quién* lo hace o *cómo* se realizará la actividad. Este modelo de *delegación* de tareas permite interacción flexible entre agentes [Martin, 1999].

3.3.1. Estructura de OAA

Un sistema basado en OAA es conceptualizado como una comunidad de agentes de software que cooperan entre sí. La coordinación de éllos se realiza a través de un *Facilitador*.

[6]Una referencia de objeto es un valor que, con seguridad, señala a un objeto particular.

[7]Un objeto implementación, o simplemente implementación para más corto, es una definición que provee la información necesaria para crear un objeto y para permitir que participe ofreciendo un conjunto de servicios.

[8]Dentro del contexto de CORBA, un *servant* es una entidad en un lenguaje de programación que implementa uno o más objetos CORBA.

La figura 3.3 muestra la arquitectura típica de un sistema implementado bajo OAA. Los agentes que se pueden apreciar (Clientes, Servidores, meta-agentes) no son formalmente parte de la arquitectura pero muestran las categorías generales bajo las cuales es posible clasificarlos.

- Los *meta-agentes* tienen la capacidad de ayudar, si así lo requiere, al Facilitador en la coordinación de actividades de los demás agentes. El Facilitador posee estrategias de coordinación independientes del dominio mientras que los meta-agentes complementan estas estrategias incorporando conocimiento específico de la aplicación y específico del dominio.

- Los *agentes clientes* no proveen servicios, en su lugar, sólo actúan como agentes que hacen peticiones a la comunidad. Con este tipo de agentes se puede implementar, por ejemplo, sistemas que permitan entradas multimodales (vía el mouse, por voz, mediante lenguaje escrito, etc.) y los agentes cliente se encargan de "monitorear" estas entradas y de enviar la información obtenida a los agentes encargados de procesarla.

- Los *Agentes servidores* tienen una funcionalidad más compleja, en el sentido de que ellos proveen servicios específicos, como el procesamiento de lenguaje natural.

3.3.2. Características de OAA

Las principales caracaterísticas de OAA se enumeran a continuación:

1. *Abierta.* OAA es una arquitectura que permite desarrollar agentes de software en varios lenguajes de programación.

2. *Distribuida.* Los agentes de software pueden estar en distintas máquinas.

3. *Extendible.* Es posible agregar nuevos agentes OAA.

4. *Alto nivel.* OAA oculta dependencias de software y de hardware.

5. *Multimodal.* El habla, la escritura a mano, los gestos, etc., representan distintos tipos de "entradas" que se pueden combinar en una aplicación para obtener un sistema multimodal.

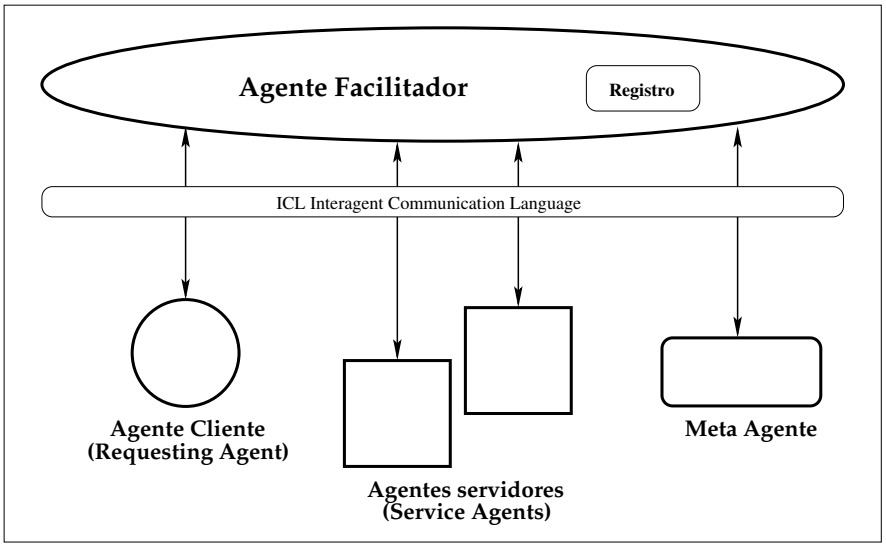

Figura 3.3: Estrucutra de OAA.

Además de estas características, OAA presenta rasgos adicionales que facilitan la implementación de agentes, estos rasgos se resumen en las siguientes subsecciones. Los agentes OAA se pueden desarrollar bajo distintos lenguajes de programación, en el apéndice H se presenta una lista de lenguajes de programación disponibles.

Protocolo de transporte

Como protocolo de transporte, OAA utiliza, al menos hasta la versión 2.1, TCP/IP. TCP/IP es un estándar que funciona en varios sistemas operativos (UNIX, Macintosh, DOS, Microsoft Windows, etc.), gracias a este protocolo es posible la interacción de agentes desarrollados bajo distintas plataformas.

El ciclo infinito

Cualquier actividad de un agente gira alrededor de un ciclo que comienza cuando se "ejecuta" el programa que lo implementa. Este ciclo le sirve al agente para "monitorear" constantemente cualquier mensaje que le pueda mandar su respectivo Facilitador. Cuando un evento (o mensaje) llega, el

agente debe "atenderlo"[9] (puede ser por medio de un *trigger*, por medio de un procedimiento, etc.).

Además de estos rasgos, el Lenguaje de Intercomunicación entre Agentes, o ICL por sus siglas, es también un rasgo distintivo de OAA, pero de esta característica se hablará en la sección 3.3.4. Para mayor detalle sobre las distintas herramientas que proporciona OAA para el desarrollo de agentes se puede consultar [Martin, 1999].

3.3.3. Facilitador

El Facilitador es parte fundamental de OAA, se encarga de coordinar la comunicación dentro de la arquitectura y guarda una base de conocimiento en donde está registrada la funcionalidad de cada agente de la comunidad. Esta última función es análoga al Repositorio de Interfaces (IR) de CORBA, en donde se almacenan las definiciones de las interfaces de los objetos para su posible uso mediante una invocación dinámica.

A excepción del Facilitador, todos los agentes son comúnmente llamados *agentes clientes*, ya que todos son, por lo menos, clientes del agente Facilitador.

3.3.4. *Interagent Communication Language*, ICL

Para la comunicación entre los agentes, OAA define un lenguaje cuya sintaxis es muy parecida a Prolog: *ICL* (*Interagent Communication Language*). Mediante ICL, los agentes pueden, entre otras cosas:

- Registrar ante el facilitador su funcionalidad. Todo agente que participe dentro un sistema OAA debe registrar su funcionalidad ante el facilitador. Esto no es más que describir mediante ICL los servicios que ofrecen a los demás agentes.

- Solicitar servicios. Un agente puede acceder a un servicio (o servicios) que provee otro agente (o más agentes) por medio de una petición descrita en ICL (y delegando la tarea o la meta a su Facilitador).

[9] El "atender" un mensaje significa que el agente proporciona código de programación por cada servicio que declaró

Estas dos funcionalidades son de las más importantes dentro de OAA, aunque no son las únicas, ICL también permite instalar *triggers*[10], intercambiar información, manipular datos, etc.

3.3.5. Creación de agentes

El desarrollo de un nuevo agente en OAA implica, como primer paso, definir su comportamiento, esto se hace declarando los servicios que va a proveer a la comunidad. Esta declaración se hace mediante ICL. Si se trata de un agente cliente, la declaración también se debe hacer, sólo que en este caso, se proporciona al Facilitador una lista de servicios vacía.

Una vez hecho lo anterior, se procede a seleccionar un lenguaje para la implementación del agente. Durante la fase de implementación, es necesario tomar una copía de las librerías que proporciona OAA ya que de ellas se obtienen los métodos (o las funciones en el caso de seleccionar un lenguaje de programación estructurada) para abrir la comunicación con el Facilitador y para registrar ante él los servicios que se van a proveer.

También es necesario que por cada servicio registrado ante el Facilitador se tenga código que implemente las acciones necesarias para atenderlo. Finalmente, para iniciar el ciclo infinito se debe llamar al procedimiento indicado dependiendo del lenguaje de programación seleccionado. Estos son los pasos básicos para crear un nuevo agente.

[10]Dentro de OAA, un *trigger* es un mecanismo mediante el cual un agente realiza una acción cuando se cumplen ciertas condiciones, esta función permite a los agentes "monitorear" ciertas aplicaciones con cierto grado de autonomía

Capítulo 4

Metodología y solución del problema

4.1. Introducción

Como se mencionó en el capítulo uno, este trabajo se desarrolla como parte del proyecto de investigación titulado "Navegación de un robot móvil en un ambiente de oficinas por medio de visión computacional y lenguaje natural", el cual tiene como objetivo principal habilitar a un robot para navegar en un espacio de oficinas e interactuar con su entorno por medio de lenguaje hablado y visión computacional.

Entre los objetivos del proyecto Golem, se deben diseñar e implementar cuatro módulos principales: un Módulo de Visión, un Módulo de Navegación, un Módulo de Lenguaje Natural y un Módulo de Representación e Inferencia.

Específicamente, el Módulo de Lenguaje Natural permitirá analizar y comprender lenguaje hablado que usarán los usuarios como una forma de interacción con el robot.

Un componente fundamental del Módulo de Lenguaje Natural es el Sistema de Reconocimiento de Voz. Mediante este sistema es posible obtener expresiones que los usuarios dictan y que se refieren a comandos que influyen en el comportamiento del robot.

Este tipo de comunicación usuario-robot es habilitada por la interacción del Módulo de Lenguaje Natural y el Módulo de Navegación, para lograr esto, el proyecto Golem se ha planteado, entre otras cosas, el diseño de un sistema de procesamiento de lenguaje natural, que a partir de voz genere los

comandos adecuados dirigidos hacía el sistema de navegación.

Una de las tareas en el desarrollo de este sistema consiste en facilitar la infraestructura necesaria para permitir la *interacción entre el SRV y el Sistema de Navegación*. Este trabajo de tesis pretende cubrir esta tarea mediante el *diseño y la implementación de una arquitectura que permita la comunicación del SRV con el Sistema de Navegación*. De esta forma, la arquitectura propuesta permitirá el acceso de los componentes del Módulo de Navegación a los componentes del Módulo de Procesamiento de Lenguaje Natural y viceversa. Con esto será posible transmitir información entre estos módulos y mediante nuevos módulos posteriormente agregados dentro del proyecto Golem será posible interactuar con el robot por medio de lenguaje hablado.

La arquitectura que se propondrá y se implementará en esta tesis deberá ser modular de forma que la disposición de su estructura permita reutilizar sus componentes parcial o totalmente dentro del Proyecto Golem. Para cubrir esta restricción es necesario una herramienta de desarrollo multiplataforma y flexible para su implementación. OAA es una herramienta de desarrollo que permite implementar sistemas de agentes de software con estas características, por lo que la arquitectura propuesta se diseñará e implementará como una colección de agentes OAA que se comunicarán recíprocamente entre ellos y con objetos del Sistema de Navegación implementados bajo la arquitectura CORBA.

El presente capítulo está organizado como sigue: en la segunda sección se describe como fue la adaptación del SRV al contexto de navegación, en la tercera sección se describe el estado inicial del Sistema de Navegación, en la cuarta sección se describe la arquitectura de integración propuesta y, finalmente, en la quinta sección, se presentan detalles de la implementación y la interfaz al usuario que presenta dicha arquitectura.

4.2. Adaptación del SRV

El SRV fue originalmente desarrollado para el proyecto DIME [DIME, 2001]. El conjunto de palabras que podía reconocer estaban orientadas al dominio del diseño de cocinas. Esto significa que solamente frases como "*pon esta mesa pegada a la pared*" o "*muéstrame el catálogo de alacenas*" podían ser reconocidas. Fue necesario adaptar el SRV al contexto de navegación, de modo que las frases que hacían referencia a éste pudieran ser reconocidas.

Para realizar la adaptación, se hizo un análisis del tipo de interacción con

el fin de caracterizar expresiones que se iban a poder usar para interactuar con el robot.

Una vez caracterizadas, es posible obtener un conjunto de ellas y generar un modelo del leguaje que posteriormente se agregará al SRV en lugar del modelo actual orientado al diseño de cocinas.

4.2.1. Análisis del Tipo de Interacción

Fue necesario definir el tipo de frases que se iban a usar para hacer referencia al Sistema de Navegación del robot.

Como punto de partida para esta definición, se propusieron tres clasificaciones de frases o expresiones posibles para interactuar con el robot. A continuación se explica en que consisten cada una de ellas:

1. **Actitud del hablante**. Esta clasificación se refiere a frases desde el punto de vista de la actitud del hablante:
   - *Imperativas.* Las frases imperativas permiten dirigirse al robot mediante una orden que indica una acción. Así, el Sistema de Navegación es capaz de generar rutas y movimientos para desplazarse de un lugar a otro dentro de su entorno.
   - *Interrogativas.* Las frases interrogativas permiten preguntar al robot por diversa información sobre él. Por ejemplo, si está en movimiento es posible conocer información sobre su entorno.
2. **Navegación Robótica**. La segunda clasificación expresiones se propuso de acuerdo a las funciones generales de un Sistema de Navegación:
   - *Generación de rutas.* Este tipo de frases hacen referencia a la generación de rutas por parte del robot. Un sistema de navegación puede planear rutas que le permiten realizar movimientos para alcanzar objetivos.
   - *Consulta sobre el plan.* Este tipo de frases permiten conocer el estado de las acciones o movimientos resultantes del plan de ruta generado.
   - *Control reactivo directo.* Las frases de control reactivo directo indican al robot ordenes como *detente* o *para*. Una frase de este tipo le puede indicar que se debe detener o suspender el plan de ruta generado.

3. **Representación del entorno**. Esta clasificación de expresiones se propuso de acuerdo a cómo el robot representa su entorno. Como se mencionó en el capítulo 2, la representación del entorno se hace en tres niveles :

   - *Geométrico*. Las frases del tipo geométrico permiten indicar al robot que ejecute un movimiento con referencia a su representación geométrica del entorno, por ejemplo: "*Ve a la posición 3,4*".
   - *Topológico*. Este tipo de frases le indican al robot que haga un movimiento respecto a su representación topológica, por ejemplo: "*Dirígete a la región 3*".
   - *Semántico*. Las frases del tipo semántico le indican al robot que ejecute movimientos de acuerdo a su representación semántica del entorno, por ejemplo: "*Camina hacía la sala de cómputo*".

Retomando los objetivos específicos del capítulo uno, no es parte fundamental crear un modelo robusto del lenguaje, es suficiente un modelo sencillo que a su vez conlleva al uso de frases simples y limitadas al dominio de la aplicación.

Las frases que se usarán dentro este trabajo de tesis son un subconjunto de las clasificaciones listadas pero limitadas al dominio de esta aplicación. Específicamente, son frases de tipo imperativo que le ordenan al robot ejecutar una acción mediante la generación de una ruta.

En la siguiente sección se explican las características de este tipo de frases y la forma en que se recolectaron para generar el modelo del lenguaje a partir de ellas.

4.2.2. Recolección de oraciones y modelo del lenguaje

En el área de IA del Departamento de Ciencias de la Computación del IIMAS se llevó a cabo un trabajo dentro del programa "Jóvenes hacía la investigación" realizado por estudiantes de nivel medio superior durante una estancia en el IIMAS [Aguilar, 2001]. Este trabajo se enfocó a aspectos de reconocimiento de voz y en él se recolectaron expresiones que indicaban movimientos al robot. A partir del conjunto de ellas obtuvieron un modelo simple definido por medio de una gramática[1].

[1]Una gramática es una definición formal de la estructura sintáctica de un lenguaje, normalmente dada en términos de reglas de producción [Free dictionary, 2004].

Para la recolección de las expresiones se basaron en una propuesta inicial de comandos primitivos a las que debían hacer referencia. Estos comandos se listan a continuación:

- *avanza n.* El robot debe avanzar n metros hacía adelante.
- *retrocede n.* El robot debe retroceder n metros.
- *izquierda n.* El robot debe girar n grados a la izquierda.
- *derecha n.* El robot debe girar n grados a la derecha.

Parte de este trabajo sirvió como base para la obtención de un modelo de lenguaje adaptado a esta tesis. Específicamente, se retomó el conjunto de expresiones y su modelo del lenguaje (que en lo subsecuente se le llamará el conjunto de expresiones original y el modelo de lenguaje original).

El modelo original tenia las siguientes características:

- *Referencia a 6 comandos.* El conjunto de expresiones hacían referencia a seis comandos, los cuatro mencionados en la especificación anterior y dos más: uno de control reactivo directo (comando *detente*) y uno que le indicaba al robot tomar una imagen (comando *foto*).
- *Reconocimiento de un sólo comando a la vez.* Con este modelo únicamente es posible hacer referencia a un sólo comando a la vez. No se permiten expresiones que se refieran a dos o más comandos al mismo tiempo.
- *Ángulos de giro específicos.* Para el caso de los comandos *izquierda* o *derecha*, estos se limitaban a dar los giros con ángulos específicos como 30, 45, 90 o 180 grados.
- *Desplazamientos específicos.* Para el caso de los comandos *avanza* o *retrocede*, estos se limitaban a desplazamientos específicos como 1, 2, 3, 4, etc., metros, no se permitían desplazamientos con cantidades decimales, como 1.5 o 2.7 metros.
- *Generación de muchas frases no usuales en el español.* Es posible simular el modelo del lenguaje con herramientas de software como el *CSLU*

toolkit[2]. Con la simulación del modelo original fue posible detectar que se generaban bastantes frases no usuales[3], como las que se muestran en la tabla 4.1, aunque esto no es una limitante ni desventaja, en este tipo de frases se puede notar que algunas presentan dos o tres palabras de sobra que hacen que la verdadera acción que expresan no sea clara.

Número	**Oración**
1	anda cuatro metros por favor
2	síguete otro metro medio metro para adelante
3	que tal si gírate giro ciento ochenta grados a la
4	síguete otro metro medio metro para adelante
5	das una lo que detectas
6	ubicarte diez metros por favor
7	avánzale hacía atrás ocho metros
8	desplazando cuatro metros por favor
9	ver ciento treinta y cinco grados hacía la derecha
10	okey puedes hacer entonces muévete cinco metros más de frente por favor

Tabla 4.1: Expresiones generadas por el modelo original.

A partir del modelo y de las expresiones originales, se obtuvo un nuevo modelo y un nuevo conjunto de expresiones (que en lo subsecuente se referirán a ellos como el *modelo mejorado* y *el conjunto de expresiones mejorado*). Las nuevas características son consecuencia de la redefinición de algunas reglas de producción y del agregado de unas cuantas más:

[2]El *CSLU Toolkit* es un conjunto de técnicas que permiten modelar lenguaje hablado para la interacción humano-computadora vía el habla. Para mayor información ver página web http://cslu.cse.ogi.edu/toolkit

[3]El SRV hace uso de probabilidad condicional para saber que palabra sigue dentro de una oración dado que anteriormente se pronunciaron otras palabras. Esta forma de trabajar, sumado a la gramática obtenida hace que el SRV tenga más opciones para reconocer la siguiente palabra por lo que se generan expresiones que no son usuales en el español, que, para el caso particular de esta tesis, son una desventaja pues es más laborioso el trabajo de análisis de frases para obtener el comando al que hacen referencia cuando hay dos o mas palabras dentro de la frase que sobran y que realmente no influyen en lo que el usuario quiere que realmente haga el robot.

- *Referencia a cuatro comandos.* En el modelo mejorado, las expresiones que se pueden usar hacen referencia únicamente a los cuatro comandos de la especificación anterior. El comando *detente* no fue tomado en cuenta por que el sistema de navegación del robot implementará técnicas de control reactivo que le permitirán autoconservarse. El comando *foto* tampoco fue tomado en cuenta por que el desarrollo de aplicaciones que involucren la cámara fotográfica montada en el robot quedan fuera del alcance de esta tesis.

- *Reconocimiento de comandos combinados.* Por medio de la reescritura de algunas reglas de producción, el nuevo modelo permite el reconocimiento de frases que hagan referencia a uno o incluso dos comandos a la ves. Hacer referencia a los comandos combinados permite al robot avanzar n metros y girar n grados a la derecha o a la izquierda, retroceder n metros y girar n grados a la izquierda o a la derecha.

  En este tipo de comandos también se permite que se pueda dar primero el giro y después la acción de avanzar, hacía adelante o hacía atrás e incluso comandos combinados que hagan referencia a una acción de doble giro o a una acción de doble avanzar (como avanza hacía adelante y después avanza hacía atrás).

- *Ángulos de giro abiertos.* Se permiten giros entre 1 y 359 grados sin punto decimal, *izquierda* o *derecha.*

- *Desplazamientos abiertos.* Se permiten desplazamientos, *avanza* o *retrocede*, con cualquier cantidad en metros, incluso con cantidades decimales.

- *Reducción de frases generadas no usuales.* El nuevo modelo mejorado, reduce el número de expresiones no usuales, lo que permite que el SRV reconozca expresiones de un conjunto reducido. Esto facilita el análisis para obtener la acción real a la que hacen referencia (el o los comandos específicos).

  En la tabla 4.2 se pueden apreciar las expresiones generadas con el modelo mejorado. En el apéndice B se puede apreciar un conjunto mayor de expresiones generadas aleatoriamente a partir de este modelo.

Concluyendo, el conjunto de expresiones mejorado propone *frases que hagan referencia únicamente a cuatro comandos básicos de navegación* (que

en lo sucesivo se referirán como *comandos simples*) o a la combinación de *dos* de ellos (que en lo sucesivo se denominarán *comandos combinados*), este tipo de expresiones *son la única forma de comunicación con el robot.*

Una vez definidas las expresiones y mejorado el modelo, el siguiente paso es la adaptación al SRV.

Número	**Oración**
1	das un giro a tu izquierda de dieciséis grados
2	te recorres ocho metros
3	gírate con dirección a la derecha setenta grados
4	avanza medio metro
5	date media vuelta a la derecha
6	avanzas hacía atrás ocho punto tres metros más por favor
7	retrocede un metro
8	regrésate uno punto cinco metros
9	golem da una vuelta hacía la izquierda de ochenta grados
10	ahora puedes regresarte cinco metros por favor

Tabla 4.2: Expresiones generadas por el modelo mejorado.

4.2.3. Adaptación del SRV

La tarea final en la fase de adaptación es obtener un modelo gramatical simple que se pueda integrar al SRV.

Para realizar esta tarea se usarán herramientas proporcionadas por HTK[4] [Young et al., 1995].

HTK provee un lenguaje de definición gramatical mediante el cual es posible obtener un modelo de las expresiones por medio de una gramática.

Una gramática se define formalmente como sigue[5]:

$$G = (N, \sum, P, S)$$

[4]HTK es un toolkit para desarrollar herramientas de procesamiento del lenguaje basado en modelos ocultos de Markov (HMM)

[5]Definición tomada de http://encyclopedia.thefreedictionary.com

Donde:

- N es un conjunto finito de símbolos no terminales
- $\sum$ es un conjunto finito de símbolos terminales
- P es un conjunto finito de *reglas de producción* escritas en la forma:

$$cadenaen(\sum \cup N)^* \rightarrow cadenaen(\sum \cup N)^*$$

- S es un símbolo dentro de N que es indicado como el símbolo inicial

La implementación del modelo mejorado es realizada por medio del conjunto de variables o símbolos no terminales (indicadas con un signo de $ antes del nombre de la variable) seguidas por una expresión regular o símbolo terminal describiendo las palabras a reconocer.

Por ejemplo, para las expresiones referentes al comando *avanza n*, se tiene la siguiente variable:

```
$NAVEGARADEL = ($ADELANTE $CANTIDAD [$FRASE_ADELANTE]) | ...
```

En este caso, la variable $NAVEGARADEL se compone a su vez de las variables $ADELANTE, $CANTIDAD y la variable opcional (como lo indican los corchetes) $FRASE_ADELANTE. Algunas de estas variables pueden ser símbolos terminales, es decir, variables cuyos componentes no incluyen a otra variable, sino símbolos que ya no son sustituibles.

La variable $ADELANTE tiene la siguiente forma:

```
$ADELANTE = AVANZA | CAMÍNALE | CAMINA | VE | DESPLÁZATE | ...
```

Como se puede apreciar, la variable $ADELANTE se compone de verbos (o símbolos terminales) cuya acción que describen es la misma o equivalente a la del verbo avanzar, por ejemplo, *camina*, *ve*, *desplázate*, etc. El símbolo | dentro la variable representa una disyunción, es decir, $ADELANTE puede tomar el valor "avanza" o "camina" o "ve", etc.

La variable $CANTIDAD tiene la siguiente forma:

```
$CANTIDAD = $CANTIDAD = ([OTROS] $UNIDADES METROS) | ...
```

donde $UNIDADES es, a su vez:

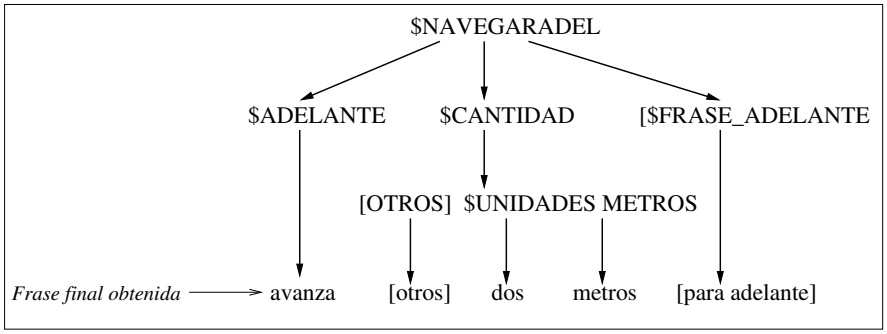

Figura 4.1: Proceso de generación de frases por la Gramática.

```
$UNIDADES = UNO | DOS | TRES | CUATRO | ...
```

y, finalmente, la variable opcional $FRASE_ADELANTE es:

```
$FRASE_ADELANTE = HACIA ADELANTE | PARA ADELANTE | ...
```

En la figura 4.1 se presenta un ejemplo de como se genera una frase que expresa un comando *avanza n*. En esta figura se aprecia como se van sustituyendo las variables por un valor o símbolo terminal para ir generando la frase que el SRV obtendrá al final del proceso de reconocimiento. Las palabras o variables entre corchetes son opcionales.

Un análisis similar se hace con las frases que hacen referencia a los demás comandos simples. Para el caso de las expresiones que hagan referencia a comandos combinados, se agregaron reglas de producción en donde se combinan cada una de las reglas de producción para los comandos simples. En el apéndice A se presenta la gramática de estados finitos que se obtuvo.

La gramática de estados finitos obtenida con HTK junto con un diccionario de pronunciación[6] son parámetros suficientes y necesarios para que el SRV este completamente adaptado a las necesidades del trabajo de esta tesis, la siguiente etapa es integrarlo al sistema de navegación del robot.

4.3. El Sistema de Navegación

Uno de los objetivos del proyecto Golem fue la creación de un sistema de navegación, por lo que inicialmente no se tenía uno. Únicamente se contaba

[6]Un *diccionario de pronunciación* contiene todas las palabras (vocabulario) que aparecen en la gramática con su respectiva pronunciación fonética.

con controladores de prueba sencillos que permitían al robot desplazarse haciendo uso de Odometría.

El software *Mobility* que acompaña al robot proporciona aplicaciones con las que es posible obtener la posición actual del robot mediante el uso de la *Odometría*. De acuerdo con Brooks, la odometría se refiere a la *estimación* de la posición (x, y, θ) del robot con respecto a lo que se ha movido [Brooks, 2002]. Los controladores de prueba hacen uso de esta aplicación.

Usando la información odométrica es posible implementar controladores de movimiento. Básicamente, estos controladores se diseñan de forma análoga a lo que es un *controlador proporcional*[7]. La ecuación formal que define a un controlador proporcional es la siguiente:

$$m(x) = ke(x)$$

en esta ecuación, $e(x)$ es la señal de error, $m(x)$ es la señal de control y k es la constante de proporcionalidad (o ganancia proporcional).

Para un movimiento de traslación, $e(x)$ es la diferencia entre la distancia que se quiere avance o retroceda el robot y la distancia que actualmente ha recorrido en su camino a la posición final, $m(x)$ es la velocidad de traslación que se enviará al robot.

Para un movimiento de rotación se hace algo similar. En este caso, $e(x)$ es la diferencia entre el valor del giro que debe dar el robot y el total de grados que ha girado en ese instante.

Como se mencionó en el capítulo dos, la Odometría es un tema por si sólo extenso y su revisión queda fuera del alcance de esta tesis.

4.4. Integración del SRV al Sistema de Navegación

Hasta este punto, se tiene un SRV completamente adaptado al contexto de navegación. La siguiente tarea es plantear una arquitectura que resuelva el problema de integración con el Sistema de Navegación.

[7]Un controlador proporcional P, a grandes rasgos, genera como salida una señal de control proporcional a la señal de error.

4.4.1. Consideraciones preliminares

Antes de trabajar sobre la arquitectura, se deben tomar en cuenta las siguientes características que debe considerar la propuesta:

- *Integración.* Es importante recalcar que ambos, el Sistema de Reconocimiento de Voz y el Sistema de Navegación han sido desarrollados por separado. El SRV fue desarrollado en lenguaje de programación Java utilizando bibliotecas de la herramienta HTK. Por otra parte, el Sistema de Navegación está actualmente en desarrollo en lenguaje de programación C++ y bajo la implementación de CORBA *omniORB*.

- *Modularidad.* La arquitectura final debe ser lo suficientemente modular para permitir la reutilización de parte de este trabajo en la implementación del sistema de propósito más general que se pretende lograr con el proyecto Golem para realizar experimentos.

- *Comunicación.* Una vez seleccionada OAA como herramienta de integración, se debe considerar y resolver el problema de comunicación entre OAA y CORBA mediante el establecimiento de un protocolo de comunicación entre ambos.

La herramienta usada para la integración de los sistemas es "Open Agent Architecture" (OAA), la cual, de acuerdo a sus características, nos permite desarrollar sistemas que cumplan con los requerimientos anteriores: integración, modularidad y suficiente flexibilidad para permitir la comunicación entre aplicaciones distribuidas.

Además de estas consideraciones, y antes de proponer la arquitectura, se hace un bosquejo sobre como sería la funcionalidad del sistema.

4.4.2. Funcionalidad

En esta sección se describe, a grandes rasgos, como debería de ser la funcionalidad de la arquitectura a proponer.

Para ejemplicar esto, se muestra la figura 4.2, por razones de legibilidad se han omitido en esta figura tanto el Facilitador de OAA como el ORB de CORBA, que ya se explicaron en el capítulo tres.

De acuerdo con la figura, los componentes que conformen la arquitectura deben de realizar la siguiente secuencia de sucesos:

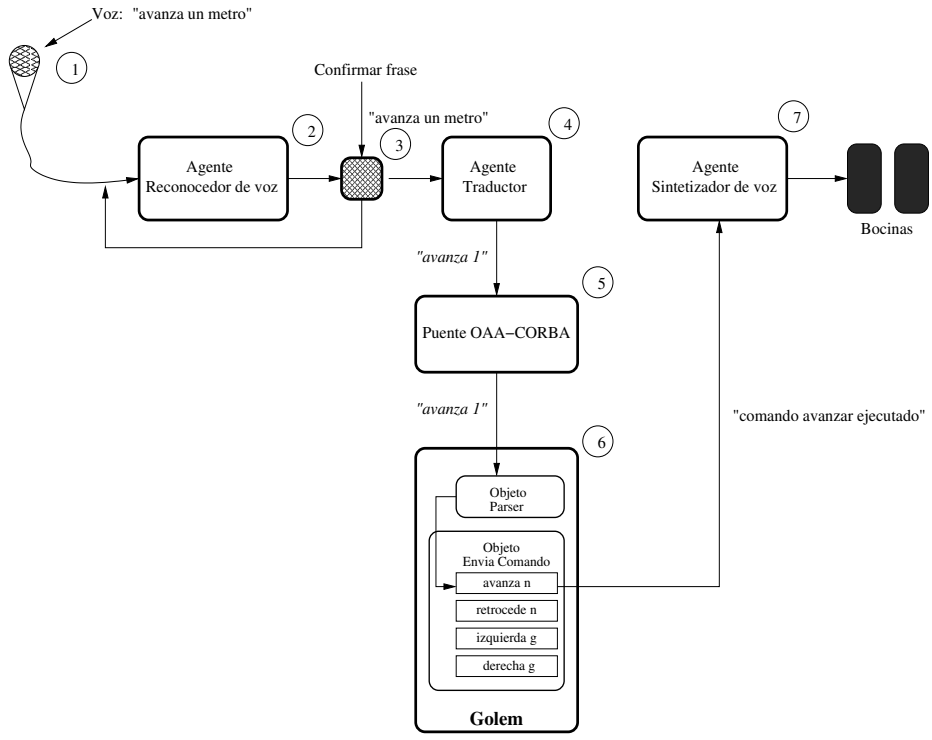

Figura 4.2: Bosquejo de la funcionalidad de la Arquitectura de integración.

1. La aplicación debe esperar por una frase dictada por el usuario, por ejemplo, el usuario expresa la frase "avanza un metro".

2. Un *Agente OAA Reconocedor de Voz* debe de obtener el texto *"avanza un metro"* correspondiente a dicha frase.

3. Antes de enviar el texto obtenido a la siguiente etapa dentro del flujo de la arquitectura, el usuario debe confirmar que lo que dijo es lo que se reconoció.

4. Una vez confirmada la frase, un *Agente Reconocedor de Voz* le envía el texto a un *Agente OAA Traductor* que se encarga de analizar el texto resultante de la expresión mediante una procesamiento orientado a obtener el comando al que hace referencia la frase, en este ejemplo particular el comando que se obtendrá es *avanza 1.*

5. Una vez obtenido el comando, el *Agente Traductor* lo envía hacía a un agente que representa la comunicación entre OAA y CORBA, un *Agente Puente OAA-CORBA*. El *Agente Puente OAA-CORBA* debe ser capaz de localizar objetos CORBA dentro del Sistema de Navegación mediante referencias de objeto para poder enviar el comando.

6. El Sistema de Navegación ejecuta el comando recibido.

7. Una vez ejecutada la acción, a través del *Agente Puente OAA-CORBA*, el sistema de Navegación envía una respuesta en una cadena a un *Agente OAA Sintetizador de Voz* y el efecto esperado es escuchar la respuesta del Sistema de Navegación a través de las bocinas de la PC.

4.4.3. Arquitectura propuesta.

De acuerdo a los requerimientos y funcionalidad mencionados en las secciones anteriores, se propone la arquitectura que se muestra en la figura 4.3. En esta figura se pueden apreciar los agentes implementados en OAA y su interacción con el sistema de navegación.

En la parte superior de la figura se presentan los agentes bajo OAA, estos agentes se comunican con el Facilitador por medio de ICL (Interagent Communication Language). ICL representa la vía de comunicación dentro de OAA[8].

En la parte inferior de la figura se encuentra el sistema de navegación del robot, el ORB (*Object Request Broker*) representa la única vía de comunicación para acceder a los objetos CORBA que conforman el sistema de navegación[9].

Para propósitos exclusivos de esta aplicación, se tuvieron que implementar dos objetos CORBA que simulan el sistema de navegación y permiten al robot ejecutar los comandos mencionados en la sección anterior.

Las características de los agentes de la arquitectura se muestran a continuación:

- *Agente Reconocedor de Voz.* El sistema de reconocimiento de voz se encapsuló como un agente de OAA, únicamente para permitirle definir una interfaz que pueda localizar a los demás componentes OAA para enviar las frases reconocidas.

[8]Véase el capítulo tres para una descripción detallada de OAA.
[9]Véase el capítulo tres para una descripción detallada de CORBA.

- *Agente Traductor.* El agente traductor se encarga de mapear las frases obtenidas por el SRV a comandos dirigidos al Sistema de Navegación.

- *Agente Sintetizador de Voz.* El agente sintetizador de voz envía los mensajes del sistema de navegación hacía las bocinas de la PC. Cuando un comando de movimiento acaba de ejecutarse, los objetos CORBA del robot envían un mensaje que se escucha a través del sonido de la PC.

- *Puente OAA - CORBA.* El agente puente provee la comunicación entre los agentes de OAA y el sistema de navegación. El problema de comunicación consistió básicamente en la comunicación entre OAA y CORBA. Por un lado se tiene a OAA como una arquitectura para desarrollar sistemas distribuidos con el enfoque orientado a agentes, en el otro extremo se encuentra CORBA, como una arquitectura también para desarrollar sistemas distribuidos, pero con un enfoque orientado a objetos. Como solución a este problema de comunicación se implemento el Agente Puente OAA-CORBA que sirve como interfaz entre estas dos arquitecturas. Esta aplicación es capaz de proporcionar comunicación en ambos sentidos, de los agentes OAA hacía el sistema de navegación y viceversa. En la figura 4.4 se muestra, como una caja negra, tanto las entradas como las salidas de dicha aplicación. Gracias a la funcionalidad del puente, los mensajes pueden ir de una arquitectura a otra por diferentes canales de comunicación.

La arquitectura permite la adición de nuevos agentes o el intercambio de agentes para obtener una arquitectura más robusta. Los agentes, por su parte, tienen la ventaja de poder ser removidos o llevados a otra arquitectura, como es el caso del Proyecto Golem.

Los objetos CORBA que simulan la navegación tienen la desventaja de ser específicos a esta aplicación, pero pueden ser sustituidos completamente por el Sistema de Navegación, esta flexibilidad la adiciona el Puente OAA-CORBA, que bajo CORBA tiene su interfaz bien definida y puede ser localizado por cualquier objeto.

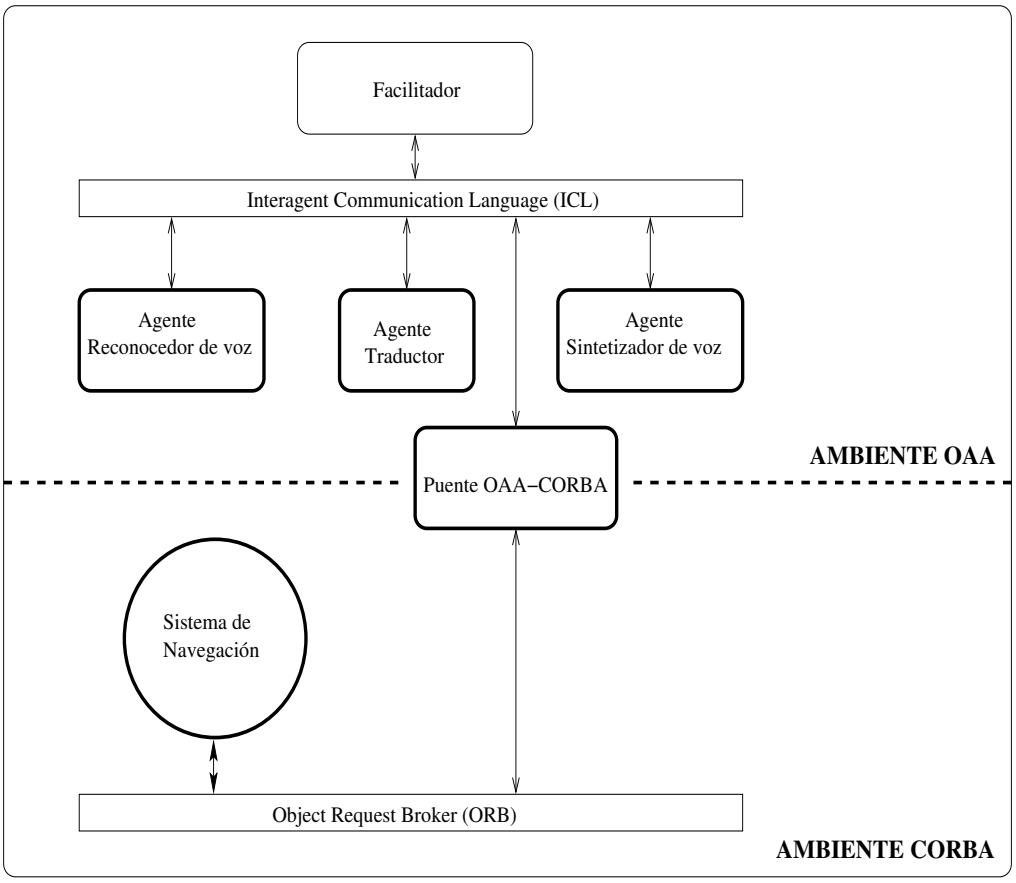

Figura 4.3: Arquitectura de integración del SRV y el Sistema de Navegación.

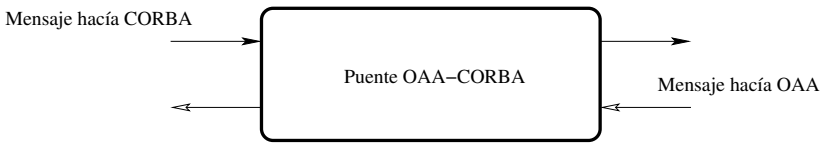

Figura 4.4: Puente OAA-CORBA.

4.5. Implementación e Interfaz de la arquitectura

En esta sección se describen detalles de implementación, funcionamiento e interfaz que presenta cada agente de la arquitectura propuesta.

Agente Reconocedor de Voz.

El sistema reconocimiento de voz se encapsuló como un agente de OAA para adaptarlo a la aplicación, pero en funcionalidad sigue siendo el mismo que actualmente se está usando en el proyecto del cual forma parte este trabajo de tesis. El SRV original presenta una interfaz gráfica que se muestra en la figura 4.5.

Figura 4.5: Interfaz gráfica del SRV original.

Esta interfaz consta de un botón con la etiqueta "hablar" que se debe presionar cuando se quiera hablar. Contiene también un campo de texto en donde se imprimirá el texto reconocido y, finalmente, un menú que contiene

un botón etiquetado con "salir" que al seleccionarlo permite abandonar la aplicación.

Es importante mencionar que el reconocedor de voz debe pasar por una etapa de calibración previa antes de comenzar a reconocer. Esta etapa se realiza cada que iniciamos el sistema. Para esto, nos presenta una ventana como la que se muestra en la figura 4.6.

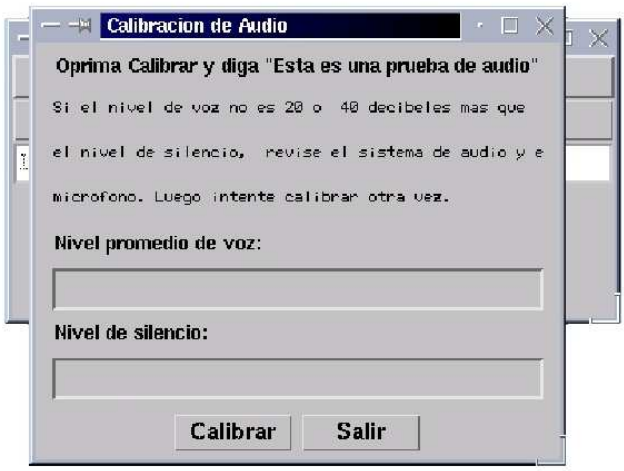

Figura 4.6: Ventana para calibrar el SRV.

Inicialmente, el SRV estaba configurado de tal forma que todo lo que reconocía se enviaba a los componentes de la arquitectura, incluso si lo que reconocía no era exactamente lo que había dicho el usuario. Para evitar este tipo de frases no deseadas y para guardar la integridad del robot, Al SRV original se le agregaron las siguientes funcionalidades:

- Un campo de texto adicional para mostrar, por escrito, el estado de la última frase que se dijo.
- Una etapa de confirmación en donde, una vez que el reconocedor obtuvó el texto de lo que se dijo por el micrófono, pregunta si es lo que se quizo decir. En caso afirmativo, se traduce la expresión en comando y se envía al robot. De lo contrario, reinicia el ciclo de reconocimiento y espera a que se presione el botón "hablar" para obtener una nueva oración.

Figura 4.7: SRV adaptado al contexto de esta aplicación.

La interfaz gráfica que presenta el SRV adaptado a esta aplicación particular es el mostrado en la figura 4.7. La etapa de calibración sigue vigente.

Dentro de OAA, el Agente Reconocedor de Voz es solamente un cliente, es decir, no presenta o registra algún servicio ante el Facilitador. Su funcionalidad como agente radica en acceder al servicio especificado por el Agente Traductor, enviarle la última cadena de texto obtenida y esperar por una respuesta sobre el estado actual de la última frase que se envió al robot.

La etapa de confirmación que se agregó sirve únicamente para preguntar al usuario si lo que dijo es realmente lo que obtuvo el reconocedor, en caso afirmativo, se envía la cadena al Agente Traductor, de lo contrario, el reconocedor de voz espera para escuchar la siguiente frase. La fig. 4.8 muestra la etapa de confirmación.

Agente Traductor.

El Agente Traductor tiene como finalidad obtener los comandos (presentados anteriormente) a partir del texto obtenido por el reconocedor de voz.

De este agente se tienen dos versiones, una versión implementada en Java y otra usando lenguaje de programación C. La versión original fue hecha en Java, pero debido a que este lenguaje de programación es interpretado, se observó que era lento el proceso de pasar el comando al robot para que lo comenzará a ejecutar. La segunda versión hecha en C es mucho más rápida debido a que C es un lenguaje compilado. El funcionamiento de las dos versiones es exactamente el mismo, sólo difieren en el lenguaje de imple-

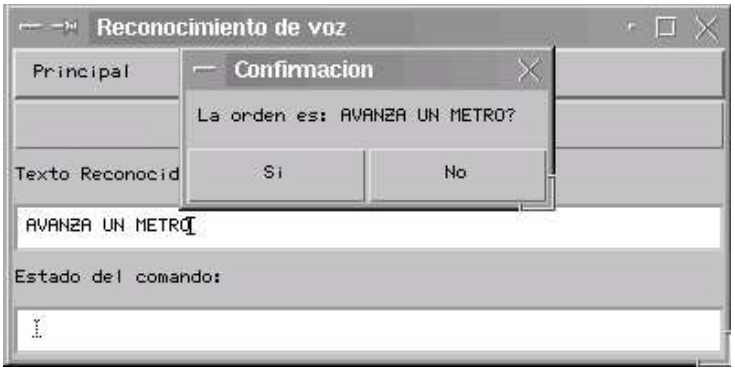

Figura 4.8: Etapa de confirmación del SRV.

mentación.

El agente Traductor de Voz hecho en Java presenta la interfaz gráfica mostrada en la figura 4.9. En ella se puede apreciar que consta de una ventana con dos campos de texto, el superior donde muestra la última frase que le enviaron e inferior, donde se puede ver el comando que obtuvo a partir de la última frase.

Figura 4.9: Interfaz Gráfica que presenta el Agente Traductor.

Además de los campos de texto, presenta un menú con dos opciones. La primera opción está etiquetada con la palabra "Principal". Esta opción contiene a su vez dos opciones: "Desconectar/Conectar OAA" y "Salir", la primera le permite conectarse o desconectarse de OAA sin necesidad de cerrar completamente la aplicación principal, y la segunda permite salir completamente de la aplicación (ver Fig. 4.10).

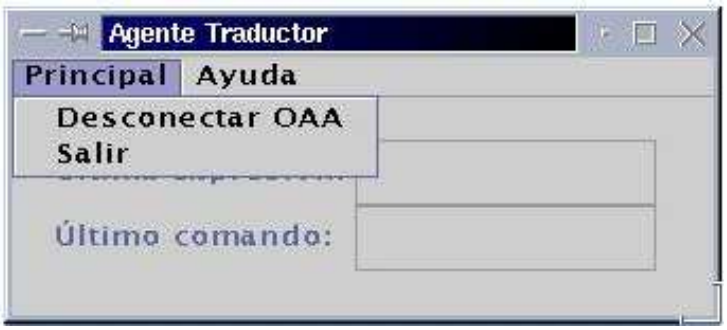

Figura 4.10: Agente Traductor con menú "Principal" desplegado.

La segunda opción del menú del Agente Traductor, "Ayuda", presenta, a su vez, dos opciones: "Descripción del agente..." y "Acerca de...". La primera muestra una descripción breve del agente y la segunda muestra la versión del Agente Traductor (ver Fig. 4.11).

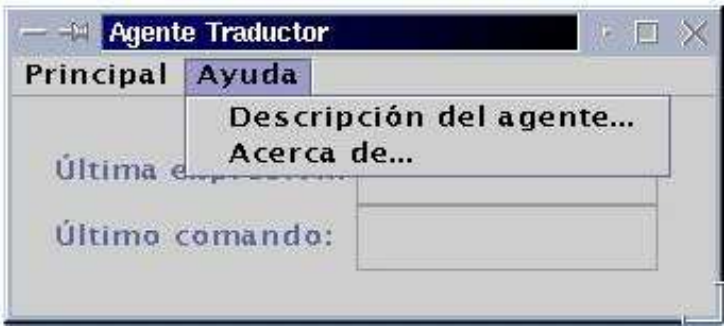

Figura 4.11: Agente Traductor con menú "Ayuda" desplegado.

La versión realizada en C no presenta una interfaz gráfica, el programa se ejecuta dentro de una terminal en modo texto (Fig. 4.12).

Agente Sintetizador de Voz.

El agente sintetizador de voz necesita como entrada una cadena de texto y genera la síntesis en voz de esa cadena. Este agente está desarrollado en Java

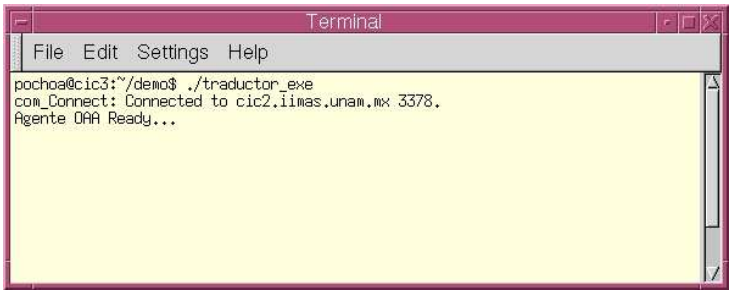

Figura 4.12: Apariencia del Agente Traductor en C.

y encapsula como agente de OAA un sintetizador de voz llamado *mbrola*[10]. Es posible obtener de Internet este sintetizador de voz y usarlo para fines no comerciales. Esta disponible para un gran conjunto de lenguajes, incluido el español.

El Agente sintetizador no presenta una interfaz gráfica, sólo se inicia en una terminal linux en modo texto. La figura 4.13 muestra al Agente Sintetizador conectado al Facilitador y a la espera de peticiones de otros agentes. En la terminal se puede apreciar el nombre de la computadora a la que se conectó, que es la misma en donde se encuentra el Facilitador (en este caso la computadora se llama deixis.iimas.unam.mx).

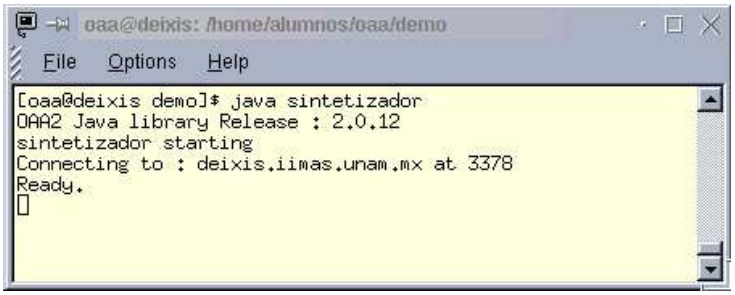

Figura 4.13: Apariencia del Agente Sintetizador.

En el apéndice C se encuentra el código fuente de este programa con el fin de poder observar el estilo de programación bajo OAA.

[10]ver página web http://tcts.fpms.ac.be/synthesis/mbrola.html

Puente OAA-CORBA.

El puente OAA-CORBA representa la interfaz entre OAA y CORBA, es el principal módulo de la arquitectura propuesta y se encarga de convertir las peticiones de agentes bajo OAA en peticiones a objetos CORBA.

El diseño de esta aplicación implicó tomar las siguientes consideraciones:

1. Modularidad. El puente no está especializado a esta aplicación en particular de modo que puede ser reutilizable.

2. Interoperabilidad. Debido a la comunicación bidireccional entre OAA y CORBA por medio de canales separados, el puente es un agente OAA y al mismo tiempo un objeto CORBA.

3. Integridad. Debido al paso constante de mensajes entre OAA y CORBA, el puente provee una estructura de datos que permite pasar los mensajes uno por uno sin riesgo a que se pierdan.

En el apéndice I se presenta detalladamente la definición en IDL del puente OAA-CORBA.

Objetos CORBA simuladores del sistema de navegación.

Actualmente el sistema de navegación del robot se está desarrollando, por lo que, para efectos de pruebas a esta arquitectura, se desarrollaron dos objetos CORBA.

La implementación de estos objetos se realizó a partir del toolkit distribuido y orientado a objetos *Mobility Robot Integration Software* el cual se mencionó en el capítulo dos de este trabajo.

El primer objeto CORBA se encarga de acceder a los métodos definidos por el objeto *controlador* de acuerdo a la cadena que llegó de CORBA. El segundo objeto se encarga de generar el movimiento de Golem.

Los componentes que conforman el sistema de navegación de Golem son bastantes complejos por lo que estos dos objetos CORBA sólo servirán para realizar pruebas a la arquitectura de integración y no forman parte del sistema de navegación que se pretende desarrollar dentro del proyecto Golem.

En al apéndice I se presentan las definiciones de las interfaces que estas aplicaciones presentan como objetos CORBA.

Finalmente, para terminar el presente capítulo, cabe mencionar que los componentes de esta arquitectura se implementaron bajo plataforma Linux

y están repartidos en tres diferentes computadoras. En la computadora uno reside el Agente Reconocedor de Voz y el Agente Sintetizador, en la computadora dos reside el Agente Traductor y el Puente OAA-CORBA y, finalmente, en la computadora a bordo de Golem (computadora tres) residen los objetos que generan el movimiento del robot. Como se menciono en el capítulo dos, la comunicación con Golem se hace vía una red Ethernet, por lo que el puente debe estar en la computadora dos que es la única por la que se puede acceder al robot.

En la figura 4.14 se presenta una aplicación llamada *monitor OAA* que viene incluida en la distribución de *Open Agent Architecture*. Esta aplicación nos permite observar, gráficamente, los agentes que en ese instante están conectados al Facilitador así como las respectivas tareas que pueden ejecutar. En esta figura se muestran los agentes que se desarrollaron en este trabajo.

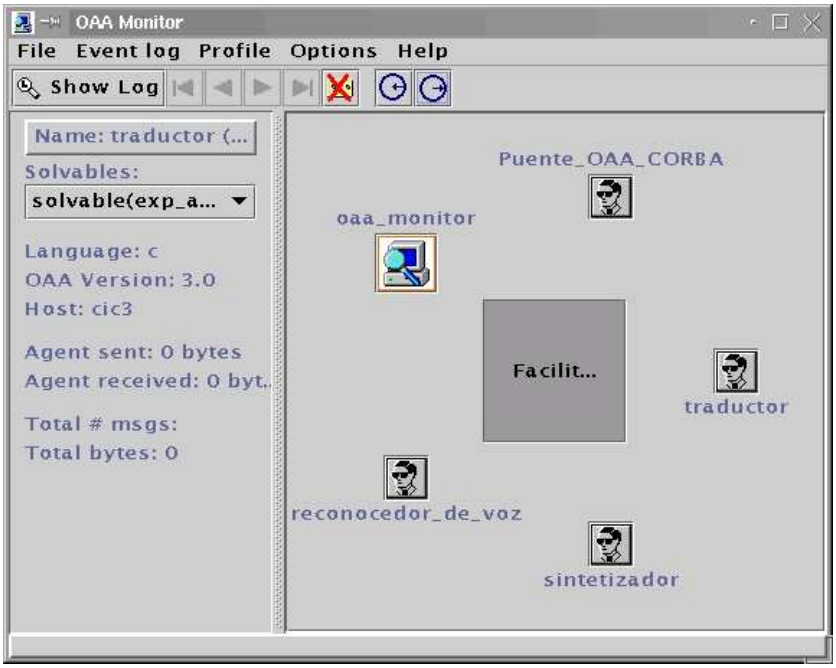

Figura 4.14: Monitor OAA.

Capítulo 5

Resultados experimentales

El propósito de este capítulo es presentar la metodología y los resultados que se obtuvieron de las pruebas a la arquitectura propuesta.

Para efectos de evaluar el sistema desarrollado se realizaron pruebas para verificar la ejecución de los comandos a los que hacían referencia un conjunto de frases generadas aleatoriamente a partir del modelo del lenguaje.

Es importante mencionar que para estos experimentos los comandos se ejecutaron de forma continua sin reinicializar el sistema.

Los experimentos se realizaron en una área de aproximadamente 2.5 m de ancho por 3.5 m de largo dentro del DCC del IIMAS. Las frases para realizar un comando de avanzar o de retroceder se restringieron a no rebasar los límites de estas medidas. Del conjunto aleatorio de frases se eliminaron aquellas en donde el comando avanzar o retroceder sobrepasaban los 3 metros por razones de seguridad del robot. Por cada experimento se generaron un total de 200 frases y se tomaron las primeras 100 que cumplian con la condición mencionada[1].

A continuación se explica en que consistieron estos experimentos.

[1]HTK provee una herramienta llamada *HSGen* con la que es posible generar frases aleatoriamente a partir de la gramática obtenida

5.1. Pruebas en la ejecución de los comandos expresados por las frases

En este experimento se generaron aleatoriamente 200 frases a partir de la gramática obtenida como modelo del lenguaje y se tomaron las 100 primeras que hacen referencia únicamente a comandos simples, *avanza n*, *retrocede n*, *izquierda g* y *derecha g*. Cada una de estas frases se dictó al reconocedor de voz y se observó si el robot ejecutaba correctamente el comando al cual hacían referencia.

La finalidad de este experimento fue evaluar la arquitectura implementada, de modo que se pueda tener una referencia númerica en donde se pueda resumir que tan confiable es el sistema para ejecutar las acciones que son expresadas mediante las frases. Es importante señalar que el objetivo de este experimento es observar que movimientos hace el robot de acuerdo a una frase proporcionada por un usuario, quién o qué usuario dicte la frase no es relevante en este experimento, ya que las evaluaciones al Sistema de Reconocimiento de Voz sí toman en cuenta el usuario que habla al Reconocedor de Voz para evaluarlo. En este caso particular, lo que se pretende es evaluar el comportamiento del robot dado que se dictó una frase que implica una serie de movimientos que el robot debe ejecutar.

Para este experimento, las frases generadas aleatoriamente a partir del modelo del lenguaje se dictarán al sistema y se pretende que el robot genere el o los movimientos expresados en dicha frase.

Los resultados reales referentes a estas frases se muestran en la tabla 5.1.

Comando al que hace referencia la frase	*bien ejecutado*	*mal ejecutado*	*total de cada tipo*
avanza	23	4	27
retrocede	32	4	36
izquierda	15	0	15
derecha	22	0	22
	total : 92	**total: 8**	**total: 100 frases**

Tabla 5.1: Resultados en la ejecución de comandos que usan frases para hacer referencia a comandos simples.

De las 100 frases que se obtuvieron, 92 se ejecutaron bien y solamente

8 finalizaron incorrectamente. De estas frases que se ejecutaron incorrectamente, 4 correspondían al comando *avanza* y 4 al comando *retrocede*. De este resultado se puede notar que el sistema, en la mayoría de los casos, ejecuta de forma correcta la acción a la que se refiere el usuario mediante la frase expresada.

Posteriormente a este experimento, se generaron aleatoriamente otras 100 frases que incluían tanto comandos simples como comandos combinados. Se realizó un experimento similar al anterior y los resultados se muestran en la tabla 5.2.

Comando al que hace referencia la frase	*bien ejecutado*	*mal ejecutado*	*total de cada tipo*
avanza	19	0	19
retrocede	22	0	22
izquierda	9	0	9
derecha	5	0	5
avanza e izquierda	8	0	8
avanza y derecha	2	0	2
retrocede e izquierda	10	1	11
retrocede y derecha	9	1	10
derecha y avanza	1	0	1
derecha y retrocede	6	0	6
izquierda y avanza	4	0	4
izquierda y retrocede	3	0	3
	total : 98	**total: 2**	**total: 100 frases**

Tabla 5.2: Resultados en la ejecución de comandos que usan frases que hacen referencia a comandos simples y combinados.

En esta última tabla se puede observar que sólo dos comandos no se ejecutaron correctamente.

5.2. Conclusiones de los resultados experimentales

De acuerdo a los resultados obtenidos en estos experimentos, generalmente el sistema ejecuta de manera correcta los comandos a los que se refieren las frases. Una característica importante del Sistema de Reconocimiento de Voz es que es independiente del hablante, por lo que quién dicte las frases no es relevante en estos experimentos.

El hecho de verificar como reacciona el SRV con distintos usuarios implica pruebas que salen del contexto de este trabajo y recaen principalmente en el marco del desarrollo de este Sistema de Reconocimiento de Voz, un tema completamente aparte.

De los experimentos realizados se puede concluir que el sistema, por cada 100 frases dictadas, más de 90 se ejecutan de forma correcta de acuerdo a la acción a la que hacen referencia, por lo que, en general, los resultados obtenidos durante estos experimentos son aceptables.

Capítulo 6

Conclusiones

En este capítulo se presentan las conclusiones finales de la presente tesis. En la primera parte se presenta un resumen de este trabajo. En la segunda parte se presentan las conclusiones de acuerdo a los objetivos planteados, en la tercera parte se describen algunas aportaciones y finalmente, en la cuarta parte el posible trabajo a futuro.

6.1. Conclusiones

Este trabajo de tesis forma parte del Proyecto de Investigación *Navegación en un robot móvil por medio lenguaje natural y visión computacional* (Proyecto Golem) que se llevó a cabo por el *Grupo de Sistemas multimodales Inteligentes* del *Departamento de Ciencias de la Computación* del *Instituto de Investigación en Matemáticas Aplicadas y en Sistemas* de la UNAM (IIMAS por sus siglas). En éste se propone una solución al problema que presenta la *integración de un Sistema de Reconocimiento de Voz al Sistema de Navegación de un robot móvil*, de forma que mediante voz se puedan dar ordenes en idioma español que hagan referencia al sistema de navegación y el robot pueda navegar en su entorno.

Inicialmente, el Sistema de Reconocimiento de Voz (SRV) fue desarrollado para operar bajo los requerimientos del proyecto DIME [DIME, 2001].

Por otra parte, no existía un Sistema de Navegación como tal, sino que éste módulo se encontraba en desarrollo de forma paralela a este trabajo. El Sistema de Navegación se diseñaba e implementaba como un conjunto de objetos bajo CORBA (*Common Object Request Broker Architecture*).

La arquitectura para la integración de estos dos sistemas debe ser modular, flexible y debe permitir comunicación de forma que los dos puedan interactuar entre sí. La herramienta seleccionada para la implementación es *Open Agent Architecture* (OAA), cuyas características cumplen con estos requerimientos.

La solución del problema se realizó en dos etapas:

- La primera etapa consistió en la *adaptación del SRV al contexto de navegación*. Inicialmente se contaba con una Gramática como un modelo del lenguaje de interacción con el Robot. A este modelo se le hicieron modificaciones que consistieron básicamente en la eliminación de algunas reglas gramaticales y en la adición de algunas más de modo que se obtuviera un nuevo modelo de lenguaje totalmente orientado a los requerimientos de este trabajo. En la tabla 6.1 se pueden apreciar ambos, ejemplos de frases obtenidas con el modelo inicial (antes de esta etapa) y ejemplos de frases obtenidas con el modelo resultante (al finalizar esta etapa). Posteriormente, este nuevo modelo fue incorporado al SRV para así tenerlo completamente adaptado al contexto de navegación.

- La segunda etapa consistió en la integración del SRV al Sistema de Navegación. En esta etapa se propuso la arquitectura de integración, distintos procesos dentro de dicha arquitectura fueron encapsulados como módulos de OAA y se implementó una interfaz que permite la comunicación de los módulos OAA a los módulos en CORBA y viceversa. La arquitectura debió ser modular, de forma que pudiera ser usada total o parcialmente dentro del Proyecto Golem según sus necesidades.

En un principio, se plantearon los siguientes objetivos:

Objetivo general: El objetivo general es *desarrollar una infraestructura que sirva como base para realizar experimentos e investigación sobre fenómenos multimodales que involucren el idioma español, específicamente sobre aspectos linguísticos y gráficos en la interacción hombre-máquina en el marco de la línea de investigación del grupo de Sistemas Multimodales Inteligentes, SMI, del DCC del IIMAS.*

Objetivos específicos: El objetivo específico es *proponer una arquitectura básica que permita integrar un Sistema de Reconocimiento de Voz a el Sistema de Navegación de un Robot Móvil.* Para esto es necesario:

Frases del modelo inicial (antes de la primera etapa)
anda cuatro metros por favor
síguete otro metro medio metro para adelante
que tal si gírate giro ciento ochenta grados a la
síguete otro metro medio metro para adelante
das una lo que detectas
Frases del modelo modificado (después de la segunda etapa)
avanzas hacía atrás ocho punto tres metros más por favor
retrocede un metro
regrésate uno punto cinco metros
golem da una vuelta hacía la izquierda de ochenta grados
ahora puedes regresarte cinco metros por favor
date media vuelta a la derecha

Tabla 6.1: Frases generadas por el modelo de lenguaje, antes y después de su revisión.

1. *Adaptación del SRV al contexto de navegación* . La adaptación del SRV consiste en :

   a) Proponer conjunto de frases que se van a reconocer y con las que se podrá hacer referencia al Sistema de Navegación del robot móvil.

   b) Analizar las frases para construir un modelo simple del lenguaje y obtener la Gramática.

   c) Incorporar el modelo obtenido al Sistema de Reconocimiento de Voz.

2. *Integración del SRV al Sistema de Navegación del robot Móvil.* Es necesario proponer e implementar una arquitectura que sirva como interfaz entre los dos sistemas, esto consiste básicamente en:

   a) Proponer una arquitectura flexible de agentes OAA para la integración de los dos sistemas.

   b) Independientemente de la arquitectura propuesta, se debe solucionar el problema de comunicación entre OAA y CORBA para la implementación de la arquitectura.

De acuerdo a estos objetivos, se tiene las siguientes conclusiones :

- *Adaptación del SRV.* Como ya se mencionó, era necesario adaptar al contexto de navegación las frases con las que se contaba inicialmente así como la incorporación de su modelo al SRV, que fue desarrollado originalmente para propósitos específicos del proyecto DIME [DIME, 2001]. Se necesitaba un conjunto de frases que pudieran ser reconocidas y se enviarán al Sistema de Navegación. Para cumplir con este objetivo, se propuso un análisis de interacción entre un usuario y el robot con el que finalmente se obtuvo un nuevo modelo del lenguaje que reconocía frases que hacían referencia a alguno de los siguientes cuatro comandos primitivos de movimiento o la combinación de dos de ellos: *avanza*, *retrocede*, *izquierda* y *derecha*. El nuevo modelo posteriormente se incorporó al SRV. Finalmente, fue posible obtener un SRV adaptado a este trabajo de tesis, con lo que este objetivo fue completamente cubierto. En la tabla 6.2 se muestran algunas frases y el efecto que producen en el sistema.

- *Integración del SRV al Sistema de Navegación.* Inicialmente no se contaba con un Sistema de navegación, solamente se tenían aplicaciones que permitían el movimiento del robot mediante el uso de Odometría. Se propuso simular el sistema de navegación con una aplicación de este tipo y diseñar e implementar una arquitectura con módulos bajo OAA que interactuará con el Sistema de Reconocimiento de Voz. Para la comunicación de las arquitecturas de OAA y CORBA, se propuso un módulo que sirviera de *puente* entre las dos permitiendo su comunicación. Finalmente, con la arquitectura propuesta e implementada fue posible dirigirse al robot mediante frases en idioma español que le indicaban ordenes que debía ejecutar.

- Como última conclusión, se obtuvó una infraestructura sobre la cual se puede realizar investigación científica. La contribución principal de este trabajo de tesis es que ha sido reutilizada parcialmente la arquitectura propuesta en el desarrollo del proyecto *Navegación en un robot móvil mediante información visual y de lenguaje natural.* Específicamente, el *Agente Sintetizador*, el *Agente Puente OAA-CORBA* y el objeto *parser* del lado de CORBA han sido reutilizados en los prototipos realizados del sistema encargado de controlar al robot para que pueda dar un recorrido en el Departamento de Ciencias de la Computación

del IIMAS (en el apéndice F se presenta la arquitectura del proyecto Golem, donde se muestran los componentes mencionados como parte de este proyecto).

Tipo de comando	**Frase**	**Efecto**
simple	*"síguete un metro de frente"*	El robot avanza un metro hacía adelante
simple	*"da media vuelta a la derecha"*	El robot gira 180 grados a la derecha
combinado	*"retrocede medio metro y gira hacía la izquierda noventa grados"*	El robot retrocede medio metro y cuando termina gira a la izquierda 90 grados
combinado	*"da una vuelta a la derecha de treinta grados y camina hacía adelante otro medio metro"*	El robot da una un giro de 30 grados a la derecha y enseguida avanza medio metro para enfrente

Tabla 6.2: Ejemplos de frases cuya acción puede realizar el robot.

La integración del SRV con el Sistema de Navegación, en conjunción con el modelo gramatical del lenguaje, permitió obtener un sistema en el que por medio de voz se puedan expresar, mediante distintas frases, ordenes que irán dirigidos al robot y que este puede ejecutar.

Como se puede apreciar en los resultados obtenidos, en este trabajo de tesis se cumplieron los objetivos planteados desde un principio.

Es importante recalcar que un resultado satisfactorio de este trabajo es que estratégicamente cumplió al realizar exploración en el terreno de la interacción humano-robot.

Una vez finalizada esta exploración, se implementará una arquitectura más completa que involucre otros módulos más complejos de un *sistema de procesamiento de lenguaje natural* como es el caso del sistema encargado de ofrecer un recorrido a los visitantes del DCC que forma parte del proyecto *Navegación en un robot móvil mediante información visual y de lenguaje natural*, actualmente desarrollado por el grupo de Sistemas Multimodales Inteligentes del DCC del IIMAS.

6.2. Aportaciones de este trabajo

La siguiente lista muestra las principales aportaciones de este trabajo de tesis:

- *Uso de voz en español.* Se obtuvo un sistema de software que funciona en base a frases en lenguaje natural hablado en español y que permite el mapeo de estas frases a comandos que el sistema debe ejecutar.
- *Sistema de software amigable.* Se obtuvo un sistema de software amigable, en el sentido de que no se necesita a personas especializadas en robótica para su uso. La interacción entre el usuario y el sistema es mediante frases habladas, naturales a cualquier ser humano, por lo que el sistema puede ser usado por cualquier persona.
- *Sistema de exploración que minimiza riesgos.* Aprovechando que el robot trae una cámara fotográfica y software para su operación, es posible tener un sistema que sirve para explorar en lugares con altos riesgos para ser revisados con una persona.
- *Infraestructura para realizar investigación sobre aspectos multimodales.* Como se mencionó en las conclusiones, se obtuvo un sistema que proporciona la infraestructura para seguir realizando investigación científica sobre cuestiones que involucren lenguaje hablado.

6.3. Trabajo a futuro

Para describir el posible trabajo a futuro es necesario mencionar las limitaciones de esta tesis:

- *Frases para interactuar con el robot.* Las frases para interactuar con el robot fueron suficientes para comprobar la arquitectura de integración, pero actualmente están limitadas a un conjunto muy restringido. Es posible enriquecer el conjunto de frases utilizadas por el reconocedor de acuerdo a los siguientes aspectos:
  - *Representación del entorno.* Las frases no explotan totalmente la forma en que el robot puede representar su entorno. En un futuro el robot contará con un *mapa métrico*, un *mapa topológico* y

un *mapa semántico*. Haciendo uso de un mapa métrico se pueden indicar frases como *Golem ve a la posición 1,2*, haciendo uso del mapa topológico es posible decirle al robot frases como *ve a la región R1* o *llévame a la región R3*, sí se refiere al mapa semántico es posible indicarle al robot frases más elaboradas como *llévame al cubículo 210* o *dirígete al laboratorio de cómputo*. El uso de los tres mapas hace posible también indicar expresiones interrogativas como preguntar por su posición con respecto a uno de sus tres mapas: posición métrica, posición topológica o posición semántica. Entonces, es posible obtener un modelo del lenguaje de un conjunto mayor de frases para que el usuario pueda tener más opciones de comunicación con el robot.

- *Riqueza de vocabulario en el idioma español*. Dentro del idioma español hay bastantes frases para hacer referencia a un comando simple o a un comando combinado del robot, muchas de ellas no están posiblemente contempladas dentro de la gramática. Dada la riqueza de nuestro idioma, es bastante complejo obtener un modelo del mismo, aún con expresiones limitadas a este tipo de comandos, seguramente habrá algún usuario que haga referencia a alguno de ellos con alguna frase que puede no estar contemplada dentro de la gramática. Aunque el objetivo principal de la tesis no era implementar una gramática lo suficientemente robusta, esto representa una limitante que se puede mejorar anexando nuevas reglas de producción y nuevo vocabulario. Además, en la etapa en la arquitectura en que se confirma lo que el usuario dice, es posible agregar un *módulo de aprendizaje* que hiciera que el robot aprendiera nuevas frases y su significado.

- *Controlador de movimiento*. Actualmente, el controlador que simula el sistema de navegación desarrollado no es lo suficientemente robusto para generar los movimientos de Golem, presenta problemas de odometría para recorridos de distancias grandes. Este controlador se puede mejorar para permitir que los movimientos de traslación y de rotación sean cada vez más exactos. De esta forma, la arquitectura propuesta puede ser usada tanto con un controlador que proporcione movimientos primitivos cada vez más precisos como con un Sistema de Navegación completo. El desarrollo de un controlador queda fuera del alcance de esta tesis. El problema de la odometría forma parte de un

problema mayor dentro de la navegación robótica: *localización y estimación de la posición*, que es una de las funciones más importantes dentro de un sistema de navegación, un robot móvil no puede ubicarse en su entorno si no tiene una representación de éste lo suficientemente consistente.

Finalmente, es importante recalcar que este trabajo de tesis ha servido como fase de exploración y ha aportado una infraestructura que actualmente está siendo utilizada en el DCC del IIMAS por el grupo de Sistemas Multimodales Inteligentes dentro del Proyecto Golem.

Apéndice A

La gramática de estados finitos

```
$DECENAS1 =  DIEZ | ONCE | DOCE | TRECE | CATORCE | QUINCE |
             DIECISEIS | DIECISIETE | DIECIOCHO | DIECINUEVE |
             VEINTE;

$DECENAS2 =  TREINTA | CUARENTA | CINCUENTA | SESENTA | SETENTA |
             OCHENTA | NOVENTA;

$VEINTES =   VEINTIUN | VEINTIDOS | VEINTITRES | VEINTICUATRO |
             VEINTICINCO | VEINTISEIS | VEINTISIETE |
             VEINTIOCHO | VEINTINUEVE;

$UNIDADES =  UNO | UN | DOS | TRES | CUATRO | CINCO | SEIS |
             SIETE | OCHO | NUEVE;

$GRADOS =    ($DECENAS1 | $DECENAS2 [Y $UNIDADES] |
             CIENTO TREINTA Y CINCO | CIENTO $DECENAS2 [Y $UNIDADES] |
             CIENTO $VEINTES | DOS CIENTOS [$DECENAS2 [Y $UNIDADES]] |
             DOS CIENTOS $DECENAS1 | DOS CIENTOS $VEINTES |
             CIENTO $DECENAS2 [Y $UNIDADES] | CIENTO $DECENAS1 |
             $VEINTES) GRADOS;

$ADELANTE =  AVANZA | AVÁNZALE | AVANZAS | CAMINA | CAMÍNALE | CAMINAS |
             SIGUE | SÍGUETE | TE  SIGUES | VE | VETE | TE VAS  |
             DESPLÁZATE | TE  DESPLAZAS | DIRÍGETE | TE  DIRIGES | ANDA |
             ANDALE | ANDAS | MUÉVETE | TE  MUEVES  | TE COLOCAS |
             COLÓCATE  | TE PONES | UBÍCATE | TE  UBICAS | RECORRE |
             RECÓRRETE | TE  RECORRES | SITÚATE | TE SITÚAS;

$ADELANTE2 = DESPLAZAR | MOVER | DIRIGIR | RECORRER | UBICAR | SITUAR;
```

```
$INFINITIVOS_NAVEGAR = IR | AVANZAR | CAMINAR | ANDAR;

$CANTIDAD = ([OTROS] $UNIDADES METROS | UNO PUNTO $UNIDADES METROS |
            OTRO METRO | MEDIO METRO | OTROMEDIO  METRO | UN  METRO |
            $UNIDADES PUNTO $UNIDADES METROS | [CERO] PUNTO $UNIDADES
            METROS) [MÁS];

$FRASE_ADELANTE = HACIA ADELANTE | PARA ADELANTE | DE FRENTE |
                  PARA ENFRENTE;

$ATRAS = RETROCEDE | RETROCÉDETE | RETROCEDES | REGRESA | REGRÉSATE |
         TE REGRESAS;

$INFINITIVOS_ATRAS = RETROCEDER | REGRESAR;

$INFINITIVOS_GIRAR = GIRAR | ROTAR | VIRAR | VOLTEAR;

$GIRAR1 = VOLTEARTE | GIRARTE | ROTARTE | DARTE | VIRARTE;

$FRASE_ATRAS = HACIA ATRÁS | PARA ATRÁS | EN REVERSA ;

$VUELTA = GIRA | GÍRATE | [TE] GIRAS | VOLTEA | VOLTÉATE | [TE] VOLTEAS |
          ROTA | [TE] ROTAS | RÓTATE | VIRA | VÍRATE | VIRAS;

$DIRECCION = (A LA| PARA LA | HACIA LA | A TU | CON DIRECCIÓN A LA)
             (DERECHA | IZQUIERDA);

$GIRAR2 = ((DATE | DAS | DA) (UNA VUELTA | UNGIRO) DE $GRADOS $DIRECCION) |
          ((DATE | DAS | DA | DATE) (UNA VUELTA | UNGIRO) $DIRECCION DE
                  $GRADOS);

$OPCION1 = [ROBOT | GOLEM | BIEN |BUENO | OKEY] [ ENTONCES| PORQUE NO |
           AHORA];

$OPCION2 = POR FAVOR ;

$ADELANTE3 = MOVERTE | DESPLAZARTE | DIRIGIRTE | COLOCARTE |
             PONERTE | UBICARTE | RECORRERTE | SITUARTE | IRTE;

$GIROS = (($VUELTA $GRADOS $DIRECCION) | ($VUELTA $DIRECCION $GRADOS) |
         ($DIRECCION $VUELTA $GRADOS) | ((PUEDES | PODRÍAS)
         $INFINITIVOS_GIRAR $GRADOS $DIRECCION) | $GIRAR2 | ((PUEDES |
         PODRÍAS) DAR (UNA VUELTA | UNGIRO) DE $GRADOS $DIRECCION) |
         ((DA | DAS | DATE) MEDIA VUELTA $DIRECCION));
```

```
$NAVEGARATRAS = ($ATRAS $CANTIDAD) | ($ADELANTE $CANTIDAD $FRASE_ATRAS) |
                ($ADELANTE $FRASE_ATRAS $CANTIDAD) | ((PODRÍAS | PUEDES)
                $INFINITIVOS_ATRAS $CANTIDAD) | ((PODRÍAS | PUEDES)
                $INFINITIVOS_NAVEGAR $FRASE_ATRAS $CANTIDAD) | ((PODRÍAS |
                PUEDES) $INFINITIVOS_NAVEGAR $CANTIDAD $FRASE_ATRAS) |
                (TE (PODRÍAS | PUEDES) $ADELANTE2 $CANTIDAD $FRASE_ATRAS) |
                ((PODRÍAS | PUEDES) $ADELANTE3 $CANTIDAD $FRASE_ATRAS) |
                ((PUEDES | PODRÍAS) REGRESARTE $CANTIDAD);

$NAVEGARADEL  = ($ADELANTE $CANTIDAD [$FRASE_ADELANTE] ) | ($ADELANTE
                [$FRASE_ADELANTE] $CANTIDAD ) | ((PODRÍAS | PUEDES)
                $INFINITIVOS_NAVEGAR $CANTIDAD [$FRASE_ADELANTE]) |
                (TE (PODRÍAS | PUEDES) $ADELANTE2 $CANTIDAD
                [$FRASE_ADELANTE]) | ((PUEDES | PODRÍAS) $ADELANTE3
                $CANTIDAD [$FRASE_ADELANTE]);

$RETROGIRO = ($NAVEGARATRAS (Y [LUEGO] | Y [ENSEGUIDA] | [Y] DESPÚES)
             ($GIROS | $NAVEGARATRAS | $NAVEGARADEL)) | ($GIROS
             (Y [LUEGO] | Y [ENSEGUIDA] | [Y] DESPÚES) ($NAVEGARATRAS
             | $GIROS));

$AVANGIRO = ($NAVEGARADEL (Y [LUEGO] | Y [ENSEGUIDA] | [Y] DESPÚES)
            ($GIROS | $NAVEGARATRAS | $NAVEGARADEL)) | ($GIROS (Y
            [LUEGO] | Y [ENSEGUIDA] | [Y] DESPÚES) $NAVEGARADEL);

$COMBINADOS = $AVANGIRO | $RETROGIRO;

$NAVEGACION = $NAVEGARADEL | $NAVEGARATRAS | $COMBINADOS;

$GRAMATICA = [$OPCION1] ($NAVEGACION | $GIROS) [$OPCION2];

(!ENTER $GRAMATICA !EXIT )
```

Apéndice B

Tipos de frases

En esta sección se muestran algunas expresiones generadas con el comando *HSGen* que viene dentro del toolkit HTK. Este comando genera expresiones a partir de la *red de palabras* obtenida.

1. TE PODRÍAS DIRIGIR PUNTO SEIS METROS DE FRENTE POR FAVOR
2. TE DESPLAZAS OTRO MEDIO METRO POR FAVOR
3. RECÓRRETE SIETE METROS HACIA ADELANTE
4. ANDALE CUATRO METROS PARA ADELANTE POR FAVOR
5. VIRA VEINTISIETE GRADOS A TU DERECHA POR FAVOR
6. TE MUEVES PUNTO CINCO METROS PARA ATRÁS Y GIRAS A TU IZQUIERDA VEINTIOCHO GRADOS
7 .SITÚATE EN REVERSA CUATRO PUNTO UN METROS
8. ROTA A LA DERECHA CIENTO QUINCE GRADOS POR FAVOR
9. ANDALE TRES METROS MÁS Y VOLTEA HACIA LA IZQUIERDA VEINTIDOS GRADOS POR FAVOR
10. PUEDES PONERTE UNO PUNTO SEIS METROS PARA ADELANTE POR FAVOR
11. TE SITÚAS UN METRO POR FAVOR
12. TE DESPLAZAS HACIA ATRÁS SEIS PUNTO SIETE METROS POR FAVOR
13. VIRAS DIECISEIS GRADOS PARA LA IZQUIERDA Y REGRESA UNO PUNTO OCHO METROS MÁS POR FAVOR
14. DATE UNA VUELTA PARA LA DERECHA DE SESENTA GRADOS POR FAVOR
15. PUEDES COLOCARTE SEIS PUNTO SIETE METROS HACIA ATRÁS Y DATE UNA VUELTA A TU IZQUIERDA DE CATORCE GRADOS POR FAVOR
16. AVANZA CINCO METROS POR FAVOR
17. GÍRATE A LA DERECHA CIENTO VEINTIOCHO GRADOS Y TE UBICAS UN METRO DE FRENTE
18. PUEDES REGRESARTE UN METRO Y PODRÍAS DAR UNA VUELTA DE NOVENTA Y CUATRO GRADOS CON DIRECCIÓN A LA IZQUIERDA
19. MUÉVETE HACIA ATRÁS UNO PUNTO DOS METROS MÁS Y PUEDES DAR UNA

VUELTA DE DOS CIENTOS CUARENTA GRADOS PARA LA DERECHA
20. DA MEDIA VUELTA PARA LA IZQUIERDA POR FAVOR
21. COLÓCATE EN REVERSA SEIS METROS MÁS Y DATE UNA VUELTA CON DIRECCIÓN A LA DERECHA DE VEINTE GRADOS POR FAVOR
22. HACIA LA IZQUIERDA RÓTATE CIENTO OCHENTA Y SEIS GRADOS Y DIRÍGETE MEDIO METRO PARA ATRÁS
23. SITÚATE OTROMEDIO METRO EN REVERSA POR FAVOR
24. DATE UN GIRO HACIA LA IZQUIERDA DE OCHENTA GRADOS Y TE SITÚAS PARA ATRÁS SIETE PUNTO UN METROS POR FAVOR
25. TE VAS MEDIO METRO PARA ATRÁS Y DAS UNA VUELTA DE VEINTICUATRO GRADOS CON DIRECCIÓN A LA DERECHA POR FAVOR
26. RÓTATE DOCE GRADOS PARA LA DERECHA Y RETROCÉDETE CINCO METROS POR FAVOR
27. DESPLÁZATE NUEVE PUNTO SIETE METROS POR FAVOR
28. TE VAS TRES PUNTO UN METROS HACIA ADELANTE POR FAVOR
29. SÍGUETE PARA ATRÁS UN METRO
30. PODRÍAS CAMINAR UN METRO MÁS EN REVERSA
31. DAS UNA VUELTA PARA LA DERECHA DE DIECINUEVE GRADOS Y CAMÍNALE OTRO MEDIO METRO PARA ADELANTE
32. TE DIRIGES PARA ADELANTE OCHO PUNTO SEIS METROS MÁS Y ROTAS DIECINUEVE GRADOS CON DIRECCIÓN A LA IZQUIERDA POR FAVOR
33. TE PUEDES MOVER CINCO PUNTO CUATRO METROS MÁS DE FRENTE
34. DIRÍGETE DE FRENTE OTROS DOS METROS MÁS Y GÍRATE CIENTO TREINTA Y CINCO GRADOS CON DIRECCIÓN A LA IZQUIERDA POR FAVOR
35. COLÓCATE CUATRO METROS
36. GÍRATE SESENTA GRADOS CON DIRECCIÓN A LA DERECHA Y TE UBICAS PUNTO SIETE METROS MÁS PARA ATRÁS
37. GIRA DIECIOCHO GRADOS A LA IZQUIERDA Y TE SIGUES HACIA ATRÁS SEIS METROS MÁS
38. VIRAS HACIA LA IZQUIERDA CIENTO CINCUENTA GRADOS POR FAVOR
39. SÍGUETE UNO PUNTO OCHO METROS PARA ATRÁS Y VIRAS PARA LA IZQUIERDA CIENTO VEINTIOCHO GRADOS POR FAVOR
40. TE DIRIGES PARA ATRÁS MEDIO METRO
41. AVÁNZALE PARA ATRÁS UN METROS MÁS Y ENSEGUIDA PUEDES DAR UN GIRO DE SESENTA GRADOS A TU IZQUIERDA
42. TE MUEVES HACIA ADELANTE UN METROS
43. TE MUEVES DOS METROS PARA ATRÁS Y PODRÍAS DAR UNA VUELTA DE TRECE GRADOS PARA LA IZQUIERDA
44. TE PUEDES DIRIGIR TRES PUNTO NUEVE METROS MÁS Y VOLTEAS A TU DERECHA VEINTISIETE GRADOS
45. VETE HACIA ATRÁS CUATRO METROS MÁS Y VIRA SETENTA Y NUEVE GRADOS HACIA LA DERECHA POR FAVOR
46. AVANZA EN REVERSA SIETE PUNTO TRES METROS MÁS
47. AVÁNZALE NUEVE METROS MÁS HACIA ADELANTE
48. RETROCEDE OTROS SIETE METROS MÁS

49. GIRAS A TU DERECHA VEINTISIETE GRADOS Y CAMINA CERO PUNTO OCHO METROS HACIA ADELANTE
50. TE VAS TRES METROS PARA ENFRENTE
51. DATE MEDIA VUELTA A LA IZQUIERDA Y RECORRE CUATRO PUNTO CINCO METROS MÁS PARA ENFRENTE
52. ROTA PARA LA DERECHA DIECINUEVE GRADOS Y REGRÉSATE UNO PUNTO NUEVE METROS
53. RECÓRRETE PARA ATRÁS CERO PUNTO UN METROS MÁS Y ROTA VEINTIUN GRADOS A TU DERECHA
54. DESPLÁZATE OTRO METRO MÁS EN REVERSA Y ROTA ONCE GRADOS HACIA LA DERECHA
55. DATE MEDIA VUELTA HACIA LA IZQUIERDA Y MUÉVETE CERO PUNTO TRES METROS
56. REGRESA NUEVE PUNTO DOS METROS POR FAVOR
57. TE COLOCAS SEIS METROS HACIA ATRÁS
58. PODRÍAS REGRESARTE CUATRO METROS MÁS Y GIRAS QUINCE GRADOS A TU DERECHA
59. PARA LA IZQUIERDA RÓTATE VEINTICINCO GRADOS POR FAVOR
60. TE DESPLAZAS UNO PUNTO UN METROS Y GÍRATE VEINTIUN GRADOS CON DIRECCIÓN A LA IZQUIERDA POR FAVOR
61. RETROCEDES CUATRO PUNTO DOS METROS MÁS Y PODRÍAS DAR UN GIRO DE VEINTIDOS GRADOS A LA DERECHA POR FAVOR
62. TE DIRIGES NUEVE PUNTO SEIS METROS EN REVERSA Y DA UN GIRO A TU IZQUIERDA DE DOS CIENTOS VEINTISEIS GRADOS POR FAVOR
63. GIRAS CIENTO SESENTA Y DOS GRADOS A LA DERECHA Y TE MUEVES NUEVE PUNTO UN METROS POR FAVOR
64. A LA IZQUIERDA ROTAS TREINTA GRADOS Y PODRÍAS RECORRERTE MEDIO METRO
65. PARA LA IZQUIERDA VIRAS OCHENTA GRADOS POR FAVOR
66. TE PUEDES DIRIGIR TRES PUNTO NUEVE METROS MÁS Y VOLTEAS A TU DERECHA VEINTISIETE GRADOS
67. ROTAS CON DIRECCIÓN A LA DERECHA TREINTA GRADOS
68. AVANZAS UN METRO Y VOLTEA CIENTO VEINTICINCO GRADOS A TU DERECHA
69. VOLTÉATE CATORCE GRADOS A TU DERECHA Y TE DESPLAZAS CUATRO PUNTO CINCO METROS MÁS PARA ATRÁS
70. VE SEIS PUNTO SIETE METROS PARA ADELANTE Y GIRAS CINCUENTA Y CINCO GRADOS HACIA LA DERECHA POR FAVOR
71. TE UBICAS PARA ATRÁS CERO PUNTO TRES METROS POR FAVOR
72. TE DIRIGES SEIS METROS HACIA ATRÁS
73. TE SITÚAS OTRO METRO PARA ENFRENTE Y A LA IZQUIERDA VOLTÉATE VEINTITRES GRADOS POR FAVOR
74. UBÍCATE HACIA ATRÁS OCHO PUNTO CUATRO METROS Y VOLTÉATE HACIA LA DERECHA DIEZ GRADOS
75. RECÓRRETE PUNTO SIETE METROS HACIA ATRÁS POR FAVOR
76. UBÍCATE OTRO MEDIO METRO PARA ATRÁS Y GIRAS VEINTIUN GRADOS A TU IZQUIERDA POR FAVOR

77. VIRA CIENTO TREINTA Y CINCO GRADOS CON DIRECCIÓN A LA DERECHA Y VETE UNO PUNTO CINCO METROS PARA ADELANT
78. SITÚATE TRES PUNTO OCHO METROS HACIA ADELANTE POR FAVOR
79. SIGUE UN METROS MÁS Y VIRAS PARA LA IZQUIERDA CIENTO NOVENTA GRADOS
80. VE HACIA ATRÁS NUEVE PUNTO CUATRO METROS MÁS Y HACIA LA IZQUIERDA RÓTATE CIENTO TREINTA Y DOS GRADOS POR FAVOR

Apéndice C

Programación con OAA

En este apéndice se muestra el código fuente del Agente Sintetizador de Voz donde se puede notar el estilo de programación en *Open Agent Architecture*.

```
// Titulo:        sintetizador
// Autor:         Paulino Ochoa <pau8a@biwemail.com>
// Descripción:   Un ejemplo de agente (servidor) que recibe un texto
//                mandado por otro cliente,[sintetizador(Cadena)], ese
//                mensaje lo guarda en un archivo ("mensaje.txt") que
//                se genera dentro del
//                mismo programa y ejecuta el sintetizador (mbrola)

// Este programa se ejecuta de la sig. forma:
// [home]$java sintetizador <enter>
// Este agente se mantiene conectado esperando
// peticiones de otros agentes.

import com.sri.oaa2.com.*;
import com.sri.oaa2.lib.*;
import com.sri.oaa2.icl.*;
import java.io.*;

public class sintetizador
{
  public static void main(String[]args)
  {
    //creamos una instancia de la librería de OAA
    LibOaa myOaa = new LibOaa(new LibCom(new LibComTcpProtocol(), args));

    System.out.println("Sintetizador iniciando...");
```

```
//Nos conectamos al facilitador
if(!myOaa.getComLib().comConnect("parent", IclUtils.icl("tcp(A,B)"),
    (IclList) IclUtils.icl("[]")))
{
  System.out.println("No se puede conectar al Facilitador");
  return;
}

//Nos registramos y le indicamos al facilitador lo que podemos hacer
if (!myOaa.oaaRegister("parent", "sintetizador", IclUtils.icl
     ("[sintetizador(Cadena1)]"), (IclList) IclUtils.icl("[]")))
{
  System.out.println("No se puede registrar los servicios");
  return;
}

//Recibimos las posibles tareas que mande la comunidad de agentes,

myOaa.oaaRegisterCallback("oaa_AppDoEvent", new OAAEventListener()
{
  public boolean doOAAEvent(IclTerm goal, IclList param, IclList answers)
    {
      //Si la meta es "sintetizador", la tomamos
      if (goal.iclStr().toString().equals("sintetizador"))
      {
        //Tomamos las cadenas mandadas por el cliente y las guardamos
        //en las variables cadena1 y cadena2
        String Cadena1 = goal.iclNthTerm(1).iclStr();
        System.out.println("El string que mando el cliente es: " +
                           Cadena1);
        //Instanciamos la clase que va a
        //generar el archivo para el sintetizador
        //Le pasamos como argumento el texto o
        //cadena mandada por el cliente.
        new GeneraArchivo(Cadena1);
        String Respuesta="Se escucho el sintetizador!!!";
        answers.iclAddToList(new IclStruct("sintetizador",
                             new IclStr(Respuesta)));
        return true;
       }

      return false;
    }
});
```

```
    myOaa.oaaReady(true);
  }

}

// clase que genera el archivo para el sintetizador

//Este programa genera un archivo llamado "mensaje.txt"
//dentro del directorio donde se encuentre la clase
//principal, es decir, el programa sintetizador.java,
//el archivo generado contiene el texto mandado por
//el cliente, aquí mismo creamos las instancias
//necesarias para ejecutar el sintetizador mediante
//el comando "tts.spanish mensaje.txt"

class GeneraArchivo
{
  public GeneraArchivo(String mensaje)
  {
    try
    {
      String mensaje1 = mensaje;
      InputStreamReader isr = null;
      BufferedReader br = null;
      OutputStreamWriter osw = null;
      BufferedWriter bw = null;

      isr = new InputStreamReader(System.in);
      br = new BufferedReader (isr);
      osw = new OutputStreamWriter(new FileOutputStream("mensaje.txt"));
      bw = new BufferedWriter(osw);
      bw.write (mensaje1);

      bw.close();
      osw.close();
      br.close();
      isr.close();
    }
    catch(IOException e)
    {
      e.printStackTrace();
    }

    //creamos un subproceso para ejecutar el sintetizador
```

```
    try
    {
      Process proceso = Runtime.getRuntime().exec("tts.spanish mensaje.txt");
      InputStream in = proceso.getInputStream();

      //esperamos a que el subproceso creado termine
      try
      {
        proceso.waitFor();
      }
      catch(InterruptedException e)
      {
        e.printStackTrace();
      }

      //desplegando el estado de la salida del subproceso
      in.close();
      System.out.println("proceso acabado número " + proceso.exitValue());
    }
    catch(IOException e)
    {
      System.err.println(e);
    }

  }

}
```

Apéndice D

Programación con CORBA y *Mobility*

En este apéndice se muestra un ejemplo donde se puede notar el estilo de programación utilizando CORBA y *Mobility Integration Robot Software*. El ejemplo es tomado de la documentación que acompaña al robot, en específico del documento *Mobility Integration Robot Software User's Guide*.

```
/*
  This is the simple follow example program for Mobility 1.0:

  The purpose of this example is to show simple (but closed loop!)
  control of a robot using the Mobility 1.0 base servers,
  interfaces and utility classes.

  This program is a stepping stone to a full fledged Mobility program
  that provides the same functionality, but is built from
  Mobility components, rather than a simple main program
  loop.

  NOTE: This program is not an example of good robot program design. It
  is intended as a simple introduction to using some of the functions
  and interfaces in Mobility.

  Real World Interface, Inc.
  Robert Todd Pack
*/

// These includes pull in interface definitions and utilities.
```

```
#include "mobilitycomponents_i.h"
#include "mobilitydata_i.h"
#include "mobilitygeometry_i.h"
#include "mobilityactuator_i.h"
#include "mobilityutil.h"
#include <math.h>


// Start of main program.
int main (int argc, char *argv[])
{
  //
  // This framework class simplifies setup and initialization for
  // client-only programs like this one.
  //
  mbyClientHelper   *pHelper;

  // These variables are "environments" which are used to pass around
  // complex (more than success/fail) error information.
  //  CORBA::Environment  env,env2;

  // This is a generic pointer that can point to any CORBA object
  // within Mobility.
  CORBA::Object_ptr ptempObj;

  // This is a smart pointer to an object descriptor. Automacially
  // manages memory.
  // The XXX_var classes automaticall release references for
  // hassle-free memory management.
  MobilityCore::ObjectDescriptor_var pDescriptor;

  // This is a buffer for object names.
  char pathName[255];

  // Holds -robot command line option.
  char *robotName;


  // ACE_OS is a class that provides a portable wrapper around lots
  // of standard OS/C library functions like fprintf.
  fprintf(stderr,
                 "*** Mobility Simple-Follow Example ***\n");

  // Look for robot name option so we know which one to run.
  robotName = mbyUtility::get_option(argc,argv,"-robot");
```

```
if (robotName == NULL)    {
    fprintf(stderr,"Need a robot name to use.\n");
    return -1;
  }


// All Mobility servers and clients use CORBA and this initialization
// is required for the C++ language mapping of CORBA.
pHelper = new mbyClientHelper(argc,argv);


// Build a pathname to the component we want to use to get sensor data.
sprintf(pathName,"%s/Sonar/Segment",robotName); // Use robot name arg.

// Locate the component we want.
ptempObj = pHelper->find_object(pathName);

// Find the sonar data (we're going to use sensor feedback).
// The XX_var variable is a smart pointer for memory management.
MobilityGeometry::SegmentState_var pSonarSeg;
MobilityGeometry::SegmentData_var pSegData;

// Request the interface we want from the object we found
try {
  pSonarSeg = MobilityGeometry::SegmentState::_narrow(ptempObj); //,env);
}
catch (...)
  {
    return -1; // We're through if we can't use sensors.
  }

// Build pathname to the component we want to use to drive the robot.
sprintf(pathName,"%s/Drive/Command",robotName); // Use robot name arg.

// Locate object within robot.
ptempObj = pHelper->find_object(pathName);

// Find the drive command (we're going to drive the robot around).
// The XX_var is a smart pointer for memory management.
MobilityActuator::ActuatorState_var pDriveCommand;
MobilityActuator::ActuatorData      OurCommand;

// We'll send two axes of command. Axis[0] == translate, Axis[1] == rotate.
OurCommand.velocity.length(2);
```

```
// Request the interface we need from the object we found.
try {
  pDriveCommand = MobilityActuator::ActuatorState::_duplicate(
              MobilityActuator::ActuatorState::_narrow(ptempObj));
}
catch (...)
  {
    return -1;
  }

// Now, here is a loop that continually checks robot sensors,
// and sends new drive commands to make the robot follow.
// There are lots slicker ways to do this, but this is the
// simple example, so bear with us until later examples.

unsigned long index1;    // Counts through sensor readings.
float haltdist = 0.2;    // 20cm halts
float followdist = 0.9;  // 0.9m follow distance.

float mindist;          // Computed minimum sensor distance.
float minfrontdist;     // Computed minimum front distance.
float tempdist;         // Computed temp dist value
                        // compared to min/minfront.

// ACE_OS is a class that provides a portable wrapper around lots
// of standard OS/C library functions like fprintf.
fprintf(stderr,
        "********** Mobility Simple-Follow:Main Loop **********\n");

while(1)
  {
    pSegData = pSonarSeg->get_sample(0);

    // Process sonar data (you get back a set of line segments).
    // This shows how you can loop through all the segments you
    // get back from the sonar source.
    mindist = 100000.0;      // Set to "really large" distances.
    minfrontdist = 100000.0;

    // Find minimum front distance and minimum overall distance.
    for (index1 = 0; index1 < pSegData->org.length(); index1++) {
        // Compute segment lengths.
        tempdist = sqrt(
                (pSegData->org[index1].x - pSegData->end[index1].x)*
```

```
            (pSegData->org[index1].x - pSegData->end[index1].x)+
            (pSegData->org[index1].y - pSegData->end[index1].y)*
            (pSegData->org[index1].y - pSegData->end[index1].y));

    // Find the minimum length value.
    if (tempdist < mindist)
      mindist = tempdist;

    // Find the minimum front distance value. (Only things in front
    // of the robot).
    if ((pSegData->end[index1].x - pSegData->org[index1].x)> 0.2) {
        if (tempdist < minfrontdist)
          minfrontdist = tempdist;
      }
}

// Show us what was found.
fprintf(stderr,
        "Sonar Values: %ld Minimum Dist: %f Min Front Dist: %f\n",
                pSegData->org.length(),
                mindist,
                minfrontdist);

// Compute new drive command based on these distances.
if (mindist < haltdist)   {
    OurCommand.velocity[0] = 0.0;
    OurCommand.velocity[1] = 0.0;
  }
else if ((minfrontdist < (followdist + 0.2))&&
         (minfrontdist > (followdist - 0.2)))     {
    OurCommand.velocity[0] = 0.0;
    OurCommand.velocity[1] = 0.0;
  }
else if (minfrontdist > (followdist + 0.25))      {
    OurCommand.velocity[0] = 0.15;
    OurCommand.velocity[1] = 0.0;
  }
else if (minfrontdist < (followdist - 0.25))      {
    OurCommand.velocity[0] = -0.15;
    OurCommand.velocity[1] = 0.0;
  }

fprintf(stderr,"CMD: %f %f",OurCommand.velocity[0],
        OurCommand.velocity[1]);
```

```
      pDriveCommand->new_sample(OurCommand,0);

      // Was there a character pressed?
      if (mbyUtility::chars_ready() > 0)  {
        // Stop robot.
        OurCommand.velocity[0] = 0.0;
        OurCommand.velocity[1] = 0.0;
        pDriveCommand->new_sample(OurCommand,0);

        return 0;
      }
      // Wait a small time before the next loop. 0.1 second, keep going.
      else
        {
          omni_thread::sleep(0,100000000);
        }
    }

  return 0;
}
```

Apéndice E

Especificaciones del robot móvil

Las especificaciones del robot se muestran en la siguiente tabla:

Especificación	
Tamaño	Circular — 40.6 cm diametro
Altura	25.4 cm sin accesorios opcionales
Distancia del chasis al piso	5.7 cm
Peso	22.7 kg con sensores infrarrojos, 18.2 kg sin ellos
Chasis	Chasis de aluminio
Color	Rojo (estándar) o especial sobre pedido
Velocidad de traslación	2.5 m/seg
Velocidad de rotación	270 grados por segundo
Carga soportada	9.1 kg
Control	2 llantas, PWM
Dirección	diferencial
Conducción	a través de 2 llantas independientes
Llantas	2, de caucho suave
Baterías	2 , 12 V
Tiempo de uso sin recargar batería	8 - 10 horas sin accesorios

Tabla E.1: Especificaciones del robot móvil

Control de movimiento	iRobot rFLEX Control System
Motores	2, 24 V DC servo motores
Comutadora	Sistema opcional de PC
Software	infraestructura iRobot Mobility
Puertos I/O	RS-232, joystick
Seguridad	Un botón de stop de emergencia
Sensores	Sonar: 16 sonares, 16 infrarrojos y 16 bumpers

Tabla E.2: Especificaciones del robot móvil(Continuación)

Apéndice F

Arquitectura Proyecto Golem

En la siguiente figura se muestra la organización de los diferentes módulos que componen la arquitectura del Proyecto Golem, el Puente OAA-CORBA y el Agente Sintetizador de Voz fueron originalmente desarrollados para este trabajo de tesis.

A diferencia del sistema desarrollado en esta tesis, el diagrama muestra una arquitectura más robusta de un sistema que involucra procesamiento de lenguaje natural.

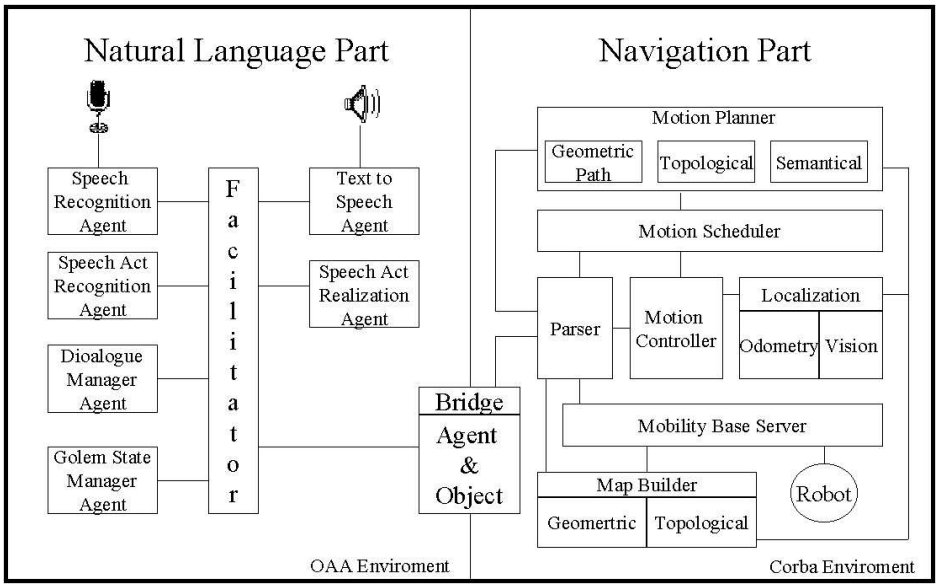

Figura F.1: Arquitectura del Proyecto Golem.

Apéndice G

Implementaciones de CORBA

Actualmente es posible encontrar muchas implementaciones de CORBA. No todas cubren completamente las características que marca el estándar emitido por el OMG, pero la mayoría implementa las características básicas de CORBA. Las siguientes son software libre que se puede obtener de Internet:

- **ORBit**. Para desarrollar aplicaciones CORBA con lenguajes de programación C y ADA [ORBit].
- **VBOrb**. Para desarrollar aplicaciones CORBA en lenguaje de programación Visual Basic [VBOrb].
- **JacORB**. Permite desarrollar aplicaciones CORBA en lenguaje de programación Java [JacORB].

Otras implementaciones son comerciales y sólo se permite evaluarlas durante un determinado tiempo. Entre ellas encontramos:

- **VisiBroker** [VisiBroker].
- **ORBacus** [ORBacus].

Una lista más completa de implementaciones de CORBA, tanto comerciales como de software libre, se puede encontrar en [FreeCORBA].

En la presente tesis se emplea *OmniORB*, una implementación de la especificación 2.0 de CORBA. OmniORB es software libre que se puede usar bajo los términos de las licencias GPL (*GNU General Public License*) y LGPL

(*GNU Library General Public License*), implementa el protocolo IIOP (*Internet Inter-ORB Protocol*) asegurando de esta forma la interoperabilidad con implementaciones de CORBA desarrolladas por otras compañías. Las aplicaciones bajo *OmniORB* se desarrollan en lenguaje de programación C++ [OmniORB].

El software que acompaña al robot fue desarrollado utilizando *OmniORB*, por lo que los diferentes componentes del robot (sonares, infrarrojos, sensores de contacto, motores que controlan el movimiento de las llantas, etc.) se pueden ver como objetos CORBA, a los cuales el sistema de navegación puede pedir información en la forma en que los objetos CORBA se comunican.

Apéndice H

Lenguajes de Programación en OAA

De acuerdo a su característica de arquitectura abierta, OAA provee bibliotecas de programación para la implementación de agentes a partir de las cuales se facilita la construcción de un sistema basado en esta arquitectura. Para la versión 1.0 de OAA, las bibliotecas están disponibles para los lenguajes de programación de la tabla H.1.

Quintus Prolog	SunOs/Solaris, Windows 9x/NT, otras plataformas que soporten Quintus Prolog
ANSI C/C++ (Unix, Microsoft, Borland)	SunOs/ Solaris, SGI IRIX, Windows 9x/NT
Common Lisp (Allegro & Lucid)	SunOs/Solaris, Linux
Java	Cualquier plataforma Java
Borland Delphi	Windows 3.1, Windows 9x/NT
Visual Basic	Windows 3.1, Windows 9x/NT
Web Lenguaje de Compaq	Cualquier plataforma Java
Perl	Unix

Tabla H.1: Lenguajes de programación disponibles para desarrollar agentes, OAA versión 1.0.

Para las versiones 2.X de OAA, las bibliotecas están disponibles para los

lenguajes de programación de la tabla H.2.

Quintus Prolog	SunOs/Solaris, Windows 9x/NT, otras plataformas que soporten Quintus Prolog
Sicstus Prolog	SunOs/Solaris, Linux, Windows 9x/NT/2000, otras plataformas que soporten Sicstus Prolog
ANSI C/C++ (Unix, Microsoft, Borland)	SunOs/ Solaris, Linux, Windows 9x/NT
Java	Cualquier plataforma Java
Web Lenguaje de Compaq	Cualquier plataforma Java

Tabla H.2: Lenguajes de programación disponibles para desarrollar agentes, OAA versión 2.X.

La versión actual de OAA es la 2.1, pero aún se siguen distribuyendo las bibliotecas para OAA versión 1.0. Los agentes desarrollados bajo la versión 1.0 funcionan bajo la versión 2.1 gracias a que el facilitador de esta versión aún guarda compatibilidad con los agentes desarrollados en lenguajes de programación cuyas bibliotecas ya no siguieron desarrollándose.

Apéndice I

Definiciones IDL

En este apéndice se presentan las interfaces que como objetos CORBA presentan el Puente OAA-CORBA y los Objetos CORBA simuladores del sistema de navegación desarrollados en este trabajo de tesis.

I.1. Puente OAA-CORBA

Como objeto CORBA el Puente define la siguiente interfaz en IDL:

```
interface bridge_agent {
        //método para enviar cadenas de OAA hacía CORBA
        void cadenaToCorba(in string cadena, out string exito);
        //método para enviar cadenas de CORBA hacía OAA
        void cadenaToOaa(in string respuesta);
};
```

El primer método, *cadenaToCorba*, tiene como finalidad, obtener la referencia al objeto CORBA indicado y enviarle la cadena. Este método es accedido por el puente como agente de OAA cada que se requiere enviar una cadena hacía CORBA. El segundo método, *cadenaToOaa*, es accedido por objetos CORBA que deseen enviar una cadena hacía OAA.

Dado que el Puente OAA-CORBA es también un agente OAA, enseguida se presenta el servicio ICL que presenta como agente OAA:

```
[stringToCorba(Cadena, Resp)]
```

De acuerdo con esta declaración el servicio es *stringToCorba*, que recibe una variable llamada *Cadena* que contiene la cadena que se pasará de OAA hacía CORBA. El argumento *Resp* es usado sólo para asegurarse que la cadena llegó a un objeto CORBA dentro del robot.

El Puente OAA-CORBA se implemento en lenguaje de programación C++ debido a que la implementación de CORBA utilizada en este trabajo nos permite desarrollar únicamente es este lenguaje de programación.

I.2. Objetos CORBA simuladores del Sistema de Navegación

El primer objeto CORBA presenta la siguiente interface:

```
//Interface IDL del objeto que accede a los módulos
//de bajo nivel de Golem y genera su movimiento
//-------------------------------------------------------------
//Los comandos contemplados en esta interfaz son:
//    - avanzar n         --> avanzar n metros
//    - retroceder n      --> retroceder n metros
//    - izquierda g       --> girar a la izquierda g grados
//    - derecha g         --> girar a la derecha g grados
//-------------------------------------------------------------

interface objetoEnviaComando {

   //avanza n metros
   void avanza(in float n);

   //retrocede n metros
   void retrocede(in float n);

   //izquierda g grados
   void izquierda(in float g);

   //derecha g grados
   void derecha(in float g);
};
```

En esta definición se exponen cuatro métodos, uno por cada comando de los mencionados en el capitulo cuatro, *avanza n, retrocede n, izquierda g, derecha g*. Este objeto accede a módulos de bajo nivel del robot y permite su movimiento.

El segundo objeto CORBA, de acuerdo al tipo de comando que le envió el puente, accede uno de los métodos definidos por el primero. Este objeto hace las veces del *parser* del lado de CORBA mencionado en el capitulo cuatro. La definición IDL de este último objeto es como sigue:

```
//-----------------------------------------------
//Interface de objeto en Golem que "interpreta"
//las cadenas enviadas desde el puente OAA-CORBA
//-----------------------------------------------

interface objetoAnalizaCadena {
   void analizaCadena(in string cadena, out string exito);
};
```

La implementación de estos objetos se realizó a partir de un toolkit distribuido y orientado a objetos llamado *Mobility Robot Integration Software* que acompaña al robot.

Bibliografía

[Aguilar, 2001] Aguilar, A., *Navegación robótica mediante información de lenguaje natural hablado*. Reporte Técnico, IIMAS, UNAM, 2001.

[Androutsopoulos, 1996] Androutsopoulos, I., *A Principled Framework for Constructing Natural Language Interfaces to Temporal Databases*. PhD Thesis, Department of Artificial Intelligence, University of Edinburgh, 1996.

[Brooks, 2002] Brooks, R. A., *Robot: The Future of Flesh and Machines*. Allen Lane The Penguin Press. 2002.

[Chavez, 1996] Chavez, A., Maes, P. *Kasbah: An agent marketplace for buying and selling goods*. In Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, 75-90, April 1996.

[Cole et al., 1995] Cole R. A., Mariani, J., Uszkoreit, H., Zaenen A. Y Zue V., *Survey of the State of the Art in Human Language Technology*, Natural Science Foundation, Directorate XIII-E of the European Commission of the European Communities, Center for Spoken Language Understanding, Oregon Graduate Institute. URL: www.cse.ogi.edu/CSLU/HLTSurvey/. 1995.

[DIME, 2001] Pineda, L., A. Massé, Meza, I., M. Salas, Uraga, E., E. Shwarz, *El proyecto DIME*, en proceedings of Second International Workshop on Spanish Language Processing and Language Technologies SLPT2, España, 2001.

[FreeCORBA] The free CORBA page. http://adams.patriot.net/~tvalesky/freecorba.html

[Gibbon et al., 1997] Gibbon, D., R. Moore, Winsky, R., *HandBook of Standards and Resources for Spoken Language Systems*, Walter de Gruyter Publishers, URL: coral.lili.uni-bielefeld.de/EAGLES, 1997.

[Haugeland, 1985] Haugeland, J., editor. *Artificial Intelligence: the very idea.* MIT Press, Cambridge, Massachusettss, 1985.

[Horswill, 1998] Horswill, Ian. Polly, a Vision-Based Artificial Agent, en *Artificial Intelligence and Mobile Robots*, editado por Kortenkamp, David, R. P. Bonasso y R. Murphy, MIT press, Cambridge, Mass. 1998.

[Howie, 1996] Howie Choset H. y Burdick J., *Sensor based motion planning: The hierarchical generalized voronoi graph.* Technical report, Carnegie Mellon University, California Institute of Technology, 1996.

[JacORB] JacORB. http://www.jacorb.org/

[Janca, 1995] Janca P. C., *Pragmatic Application of Information Agents: BIS Strategic Decisions.*

[Jennings y Wooldridge, 1996] Jennings, N., M. Wooldridge. *Software agents.* IEE Review, Pags. 17 -20, 1996.

[Jurafsky, 1999] Jurafsky, D., H. Martin. *SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* 1a edición, Prentice Hall, 2000

[Kurzweil, 1990] Kurzweil, R. *The age of Intelligent Machines.* MIT Press, Cambridge, Massachusetts, 1990.

[Lieberman, 1998] Lieberman, Henry. *Integrating User Interface Agents With Convention Applications.* Proceedings of the 1998 International Conference on Intelligent Interfaces. P.39-46.

[MAIA, 1994] Antonio, G., Caprile, B., Cimatti, A., Fiutem, R., y G. Lazzari. *Experiencing real-life interaction with the experimental platform of MAIA.* In proceedings of the 1st European Workshop on Human Comfort and Security. 1994.

[Martin, 1999] Martin L. D., A. J. Cheyer, Moran D. B. *The Open Agent Architecture: A Framework for Building Distributed Software Systems.* URL:http://www.ai.sri.com/∼oaa. 1999.

[Matsui, 1999] Matsui, T., H. Asoh, J. Fry, Y. Motomura, F. Asano, T. Kurita, I. Hara, N. Otsu, *Integrated Natural Spoken Dialogue System of Jijo-2 Mobile Robot for Office Services*, publicado por American Association for Artificial Intelligence, http://www.aaai.org. 1999.

[McCarthy, 2003] McCarthy, J. *What is AI?.* URL:http://www-formal.stanford.edu/jmc/whatisai.html. 2003.

[Microsoft, 2002] *Distributed Component Object Model Protocol DCOM/1.0.* Disponible en la URL: http://www.microsoft.com/com/tech/DCOM.asp

[Nwana, 1996] Nwana, H. S. *"Software Agents: An Overview"*, Knowledge Engineering Review, Vol. 11, No 3, 1996

[OMG, 2002] *La especificación CORBA/IIOP 2.6.1.* Disponible en la URL: http://www.omg.org/technology/documents/corba_spec_catalog.htm

[OmniORB] OmniORB. http://www.uk.research.att.com/omniORB/omniORB.html

[ORBacus] ORBacus. http://www.iona.com/products/orbacus_home.htm

[ORBit] ORBit. http://www.labs.redhat.com/orbit/

[Russell y Norvig, 1996] Russell, S., P. Norvig. *Inteligencia Artificial, un enfoque moderno.* 1ª Edición, Prentice Hall, 1996.

[Selker, 1994] Selker, T., *Coach: a Teaching Agent that Learns*, Communications of the ACM, vol.37, no.7, pp. 93-99, 1994.

[Schalkoff, 1990] Schalkoff, R. J. *Artificial Intelligence: An Engineering Approach.* McGraw-Hill, New York, 1990.

[Shet, 1996] Sheth, B., *NEWT: A learning Approach to Personalized Information Filtering.* PhD thesis, MIT Media LAb, 1996.

[Spiliotopoulos, 2001] Spiliotopoulos D., I. Androutsopoulos, Spyropoulos C. D., *Human-Robot Interaction Based on Spoken Natural Language Dialogue*. Presentado en *European Workshop on Service and Humanoid Robots* (Servicerob 2001), Santorini, Greece, 2001. Disponible en la URL:http://www.aueb.gr/users/ion/publications.html

[Free dictionary, 2004] The Free Dictionary, http://encyclopedia.thefreedictionary.com.

[Theobalt, 2000] Theobalt, C. *Navigation on a mobile robot*, University of Edinburgh, tesis de maestría, 2000.

[Thrun, 1998] W. Burgard, A. B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Shultz, W. Steiner, S. Thrun. *Experiences with an interactive museum tour-guide robot.* Technical Report CMU-CS-98-139, Carnegie Mellon University, 1998. Disponible en http://www-2.-cs.cmu.edu/afs/cs.cmu.edu/user/thrun/public_html/papers.html

[Torrance, 1994] Torrance, M. C. *Natural Communication with Mobile Robots*. Massachusetts Institute of Technology, tesis de maestria, 1994.

[Uraga, 1999] Uraga, E. *Modelado fonético para un sistema de reconocimiento de voz continua en español*, ITESM, campus Morelos, tesis de maestría, 1999.

[Vinosky, 1996] Vinosky, S. *CORBA: Integrating Diverse Applications Within Distributed Heterogenous Environments.* IEEE Communication Magazine, 14(2):37-49, February 1997.

[VisiBroker] Visibroker. http://www.borland.com/corba/index.html

[VBOrb] VBOrb. http://home.t-online.de/home/Martin.Both/vborb.html

[Winston, 1992] Winston, P. H. *Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts, 3ª edición, 1992.

[Wooldridge, 2000] Wooldridge, Michael. *Intelligent Agents*, en *Multiagent systems a modern approach to distributed artificial intelligence*, editado por Weiss, Gerhard, MIT press, Cambridge, Mass. 2000.

[Young et al., 1995] S. Young, J. Odell, D. Ollason, V. Valtchev, y P. Woodland. *The HTK Book*, Cambridge University, 1995.