Módulo de habla CORBA para SOUL

Mª Cristina Panizo Cano

5 de noviembre de 2005

A mi familia.

Me gustaría agradecer su ayuda y apoyo a todos los que han hecho posible la realización de este proyecto. A Ricardo Sanz por haberme propuesto un tema tan interesante, a Raquel, Carlos y todos los compañeros del ASLab por echarme una mano siempre en todo, por imposible que pareciese. A mi familia, amigos y compañeros de la escuela por estar siempre ahí. Gracias a todos!

Índice general

Índice de figuras

Capítulo 1

Introducción

En este primer capítulo introductorio se tratan temas generales relacionados con el presente proyecto fin de carrera titulado "Módulo de habla CORBA para SOUL" *(motivación, contexto general y objetivos del mismo), y se realiza una descripción de la estructura utilizada para organizar la presente memoria.*

1.1. ¿Por qué un módulo de habla?. Motivación

Actualmente son muchos los sistemas que incorporan módulos de voz o habla para interactuar con sus usuarios. Entre ellos encontramos desde sistemas de mensajería, que permiten leer la correspondencia electrónica a través del teléfono, hasta sistemas de respuesta interactiva que, para dominios limitados como es el caso de sistemas de llamadas telefónicas, gozan de un nivel de calidad de síntesis elevado y, gracias a la automatización, permiten un mayor número de peticiones atendidas [R. Sproat, 1999]. Soluciones como éstas responden a la necesidad de mejorar la interacción hombre-máquina dentro de cada sistema, proporcionando no sólo el servicio requerido de manera más eficiente sino también de una forma más adecuada a las necesidades humanas.

Las razones por las que la implantación de un módulo de voz, o habla, es necesaria dentro de un sistema electrónico, no sólo responden a factores tecnológicos ni psicológicos, relacionados con los posibles destinatarios, sino también a que este tipo de interacción a menudo es indispensable para satisfacer las necesidades de cierto grupo de la población, entre ellos los invidentes, para los que constituye su único modo de acceso a la información. La comunicación hablada es un rasgo característico de la naturaleza humana que puede ser empleada en la interacción hombre-máquina para intentar re-

ducir la reticencia que, todavía hoy, muchos individuos presentan a la hora de utilizar dispositivos electrónicos.

1.2. Contexto general del proyecto fin de carrera. El proyecto SOUL

Este trabajo se enmarca dentro del proyecto de investigación a largo plazo CS^2 — Complex Software-intensive Control Systems. Este es un proyecto de investigación en tecnologías software para sistemas complejos de control realizado por el Autonomous Systems Laboratory (ASLab) de la UPM, dentro del Departamento de Automática de la ETS de Ingenieros Industriales de la Universidad Politécnica de Madrid.

El objetivo de este proyecto investigador es la definición de arquitecturas de control integrado para la construcción de sistemas complejos de control y el desarrollo de tecnologías software para su construcción. Las líneas maestras son simples y guían todo el desarrollo del proyecto:

- Modularidad
- Seguimiento de estándares
- Reutilizabilidad
- Diseño basado en patrones
- Independencia del dominio

En este contexto, se ha definido una plataforma software genérica de base denominada **Integrated Control Architecture** (ICa) [Sanz et al., 1999c] que proporciona los criterios de diseño software centrales. La arquitectura ICa es una metaarquitectura software que ofrece una serie importante de ventajas frente a otros enfoques :

Coherente y unificada: La arquitectura ICa proporciona integración, vertical y horizontal, total y uniforme; por ejemplo permite emplear un única tecnología en todos los niveles en la verticalidad del sistema de control (desde la decisión estratégica hasta los dispositivos empotrados de campo) así como la integración horizontal de unidades de negocio o empresas extendidas.

Clara: El modelo empleado en ella es un modelo preciso: el modelo de objetos distribuidos de tiempo real que constituye la base de las plataformas estado del arte en este campo.

Flexible y extensible: Permite su adaptación a distintos contextos de ejecución y distintos dominios al realizar un mínimo de compromisos de diseño; lo que permite la reutilización modular de componentes y la incorporación de nuevos componentes en dominios concretos.

Abierta: Permite la interoperabilidad con sistemas ajenos (tanto heredados como futuros).

Portable: Gracias a basarse en estándares internacionales.

Esta arquitectura —o metaarquitectura como debe ser considerada ICa en realidad [Sanz et al., 1999a]— se ha venido desarrollando en nuestro departamento durante los últimos años y, gracias a diferentes proyectos de I+D, se ha aplicado con éxito en múltiples ámbitos de control:

- Control estratégico de procesos de fabricación de cemento [Sanz et al., 2001].
- Gestión de emergencias en plantas químicas [Sanz et al., 2000].
- Sistemas de monitorización distribuida de tiempo real de producción y distribución de energía eléctrica [Clavijo et al., 2000].
- Robots móviles cooperantes [Sanz et al., 1999b].
- Bucles de control en red [Sanz, 2003].
- Protección de subestaciones eléctricas [Sanz, 2002].
- *etc.*

La característica primaria de ICa es su enfoque modular. Los sistemas se construyen por medio de módulos reutilizables sobre *frameworks* de sistemas distribuidos de objetos de tiempo real. La organización concreta de los módulos viene dada por su arquitectura de aplicación, que se deriva de la metaarquitectura por medio del uso de patrones de diseño [Sanz and Zalewski, 2003]. Su despliegue se hace según las necesidades y restricciones de la aplicación, explotando la reubicabilidad de los componentes (ver Figura 1.2).

En este proyecto fin de carrera se aborda la construcción de un módulo reutilizable específico para una cierta clase de arquitecturas que estamos investigando en la actualidad. La clase de arquitecturas se denomina **SOUL**.

El proyecto **SOUL** intenta desarrollar una arquitectura cognitiva para el desarrollo de sistemas de control cognitivo complejo.

Las tendencias modernas en arquitecturas de control para sistemas autónomos (robots, plantas industriales, web bots, etc.) progresivamente se centran

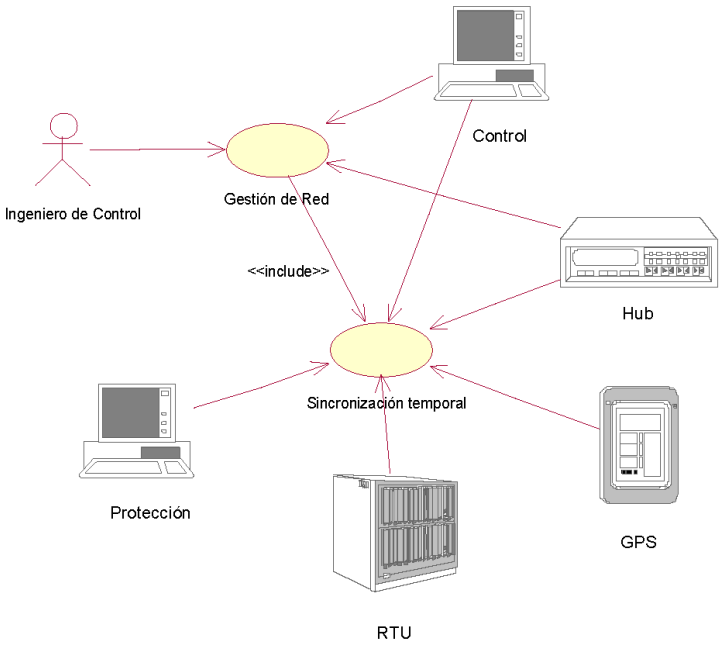

Figura 1.1: Diagrama de caso de uso para un despliegue de una aplicación de control y protección de redes.

en formulaciones explícitas de las capacidades cognitivas humanas de alto nivel. Entre ellas se considera esencial la capacidad de asignar significado a los flujos de información perceptual que el agente autónomo obtiene de su interacción con el mundo físico que le rodea para su introspección.

En este proyecto se intenta:

1. investigar la naturaleza y los mecanismos de generación de significado en sistemas cognitivos autónomos,

2. aplicar los conceptos que surjan en diversas plataformas de investigación con requisitos y contextos muy diferentes entre sí: heterogeneidad, escalabilidad y percepción visual.

En este contexto, el proyecto **SOUL** intenta construir una teoría formal del significado para ser aplicada en la definición de mecanismos de control basados en representaciones explícitas de significado. Estos mecanismos se usarán en el diseño de arquitecturas de control conscientes de sí mismas para sistemas autónomos y se implementarán mediante módulos software reutilizables usando plataformas de desarrollo software estandarizadas. Finalmente, la arquitectura y los módulos reutilizables se usarán en tres dominios de aplicación: robots autónomos, plantas industriales complejas y juegos inmersivos.

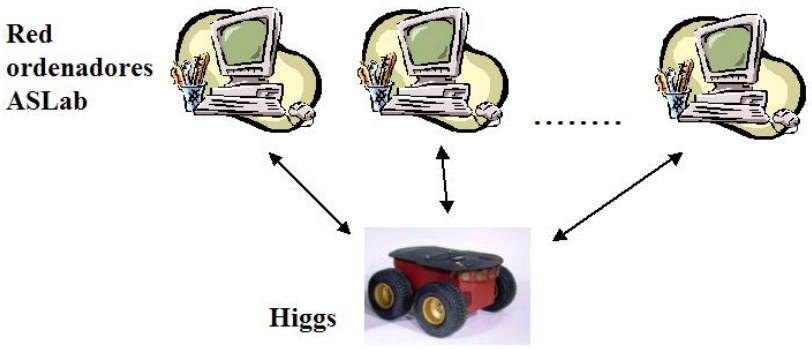

Figura 1.2: Sistema de comunicaciones.

Módulos existentes

Actualmente se encuentran disponibles tres módulos reutilizables, desarrollados todos ellos en proyectos fin de carrera.

Módulo de comunicaciones. Desarrollado en marzo de 2004 por Iván Pareja Larios [Pareja, 2004] supone la base de todo el resto de módulos ya que, uno de los objetivos de dicho proyecto, fue dotar al robot móvil Pioneer 2AT-8 (que se utiliza como plataforma de pruebas) de comunicación inalámbrica con el resto de ordenadores de la red ASLab. Utiliza **ICa** (Arquitectura de Control Integrado) siendo este uno de los puntos a destacar, ya que, como veremos más adelante en el capítulo cuatro, este hecho no impide la elección de cualquier otra distribución de CORBA para la construcción de nuevos módulos, gracias a la interoperabilidad característica de esta tecnología.

Módulo para el mapeo facial de emociones sintéticas. Presentado en marzo de 2005 por Raquel P. Conde López, persigue la implementación de un módulo reutilizable en el contexto de **ICa** para la visualización de estados de sistemas técnicos basado en comunicación emocional mediante caras. Con este módulo se implementa un sistema metafórico-analógico de

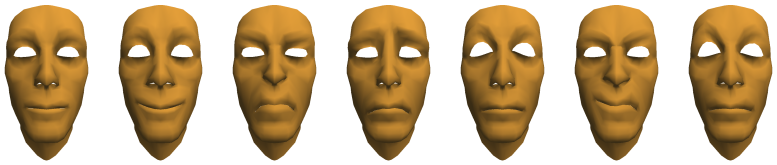

Figura 1.3: Emociones sintéticas.

visualización de estados de sistemas técnicos mediante la representación tridimensional de caras antropomórficas correspondientes a estados emocionales humanos[Conde, 2005b].

Módulo SOAR Este módulo fue presentado en el proyecto fin de carrera de Silvia Sánchez Herranz en marzo de 2005 [Sánchez, 2005]. **SOAR** es una arquitectura cognitiva para el desarrollo de sistemas que permite que éstos muestren un comportamiento inteligente. Actualmente se investiga para conseguir que esta arquitectura:

- trabaje en todo el rango de tareas en las que se supone que debería trabajar un agente inteligente, desde problemas totalmente rutinarios hasta aquellos extremadamente complicados y con soluciones abiertas,
- represente y use formas apropiadas de conocimiento, ya sean de procedimiento, declarativas o posiblemente icónicas,
- emplee todos los métodos para resolver problemas,
- interactúe con el mundo exterior y,
- aprenda todos los detalles de las tareas que desarrolla y su actuación en ellas.

El último paso en inteligencia sería la racionalidad completa que implicaría la habilidad para usar todo el conocimiento disponible para cada una de las tareas con las que se enfrentase el sistema[1].

[1]Más información sobre SOAR puede encontrarse en Internet en http://sitemaker.umich.edu/soar

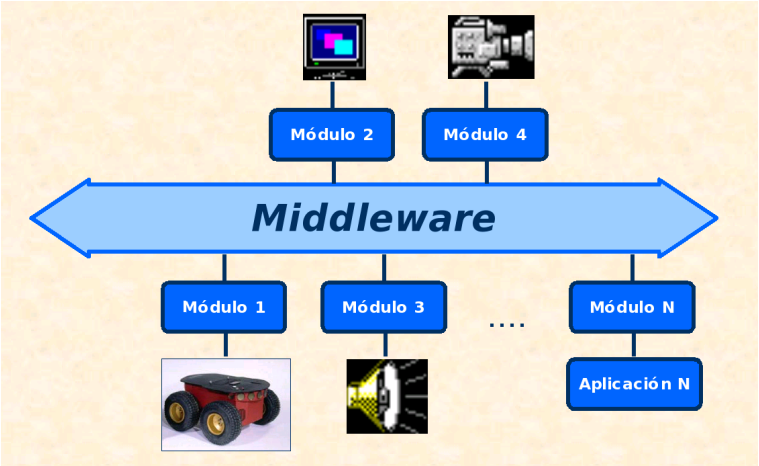

Figura 1.4: Estructura modular del sistema.

Servidor de tiempo real Teniendo en cuenta los módulos actuales, recientemente se ha incorporado un servidor de tiempo real al sistema gracias a otro proyecto fin de carrera desarrollado por Adolfo Hernando [Hernando, 2005]. En él se ha mejorado el servidor disponible añadiéndole características de tiempo real. Para ello ha sido necesaria la incorporación de un nuevo sistema operativo que implemente características de tiempo real blando con las restricciones impuestas por las comunicaciones inalámbricas del hardware existente, de forma que pueda controlar el robot móvil Pioneer2AT-8 que constituye la plataforma de pruebas común a todos los proyectos. Este nuevo servidor puede resultar una increible mejora en relación con las prestaciones de un control remoto usando el servidor CORBA no RT.

1.3. Objetivos del proyecto fin de carrera

El presente proyecto fin de carrera intenta desarrollar un nuevo módulo reutilizable que añada un nuevo componente al sistema ya existente. Este nuevo módulo constituirá una nueva interfaz entre el sistema y el operario o posible interlocutor del sistema, posibilitando la comunicación hablada por parte del sistema.

1.4. Estructura de la memoria

Esta memoria de Proyecto Fin de Carrera se estructura en siete capítulos y dos apéndices:

Capítulo 1 / Introducción: Pequeña introducción al sistema dentro del que se desarrolla el proyecto fin de carrera.

Capítulo 2 / Ordenadores parlantes: Hace un repaso a la evolución a lo largo de los años de la tecnología del habla.

Capítulo 3 / Motores de generación de habla. Festival: Recoge las distintas posibilidades de las que se dispone a la hora de intentar dotar a cualquier sistema de un modo de comunicación hablado.

Capítulo 4 / CORBA: Introduce la tecnología estándar CORBA que ha permitido el desarrollo del módulo de voz.

Capítulo 5 / Plataforma de pruebas: Higgs: Realiza una descripción de la plataforma de pruebas del sistema constituida por un robot PIONEER 2AT-8.

Capítulo 6 / Desarrollo: Recoge los distintos pasos seguidos en el desarrollo del presente proyecto, su evolución y etapas.

Capítulo 7 / Conclusiones y líneas futuras: En este capítulo se recogen por un lado las conclusiones del trabajo realizado en este proyecto fin de carrera así como las líneas de trabajo que quedan abiertas tras la finalización del mismo.

Apéndice A / Otras herramientas utilizadas: Enumera otras herramientas software utilizadas para el desarrollo de este proyecto.

Apéndice B / Manuales de referencia: Establece una lista de direcciones útiles donde conseguir toda la información necesaria para completar conocimientos sobre todo lo relacionado con el proyecto.

Capítulo 2

Ordenadores parlantes

En este capítulo haremos un repaso a la historia de la investigación sobre síntesis de habla, hablaremos de la situación actual de la tecnología, los sistemas disponibles y las distintas áreas de investigación activas.

Investigación en síntesis de voz Para poder conocer el estado del arte actual así como el futuro de la tecnología en síntesis de voz, es necesario tener una idea general de la historia en este campo. Con este fin,presentamos en esta sección un pequeño resumen del trabajo realizado sobre síntesis de voz y análisis de textos para síntesis de *text-to-speech* (TTS) o generación de habla a partir de texto. Los principales hitos a destacar dentro de la evolución de la tecnología de síntesis de habla se reflejan en la Figura 2

2.1. Síntesis de sonidos articulados

Si nos remontamos a finales del siglo XVIII encontramos un primer intento de síntesis por parte de Kratzenstein y von Kempelen, quienes construyeron un dispositivo que imitaba los sonidos producidos por el aparato vocal humano. Alterando la forma de la cámara de resonancia de la que disponían podían producir diferentes sonidos que asemejaban el habla humana. Esta primera aproximación, por tanto, utilizaba medios mecánicos para la producción de sonidos. No es hasta 1920 cuando comienza la utilización de medios electrónicos para síntesis, gracias al trabajo de J.Q.Stewart sobre producción de sonidos vocálicos estáticos. El primer dispositivo electrónico capaz de sintetizar frases inteligibles fue el Voder, desarrollado por Homer Dudley en los laboratorios Bell.

En estos mismos laboratorios a finales de la década de los cuarenta se desarrolló el *espectógrafo de sonido*, un dispositivo básico e imprescindible todavía

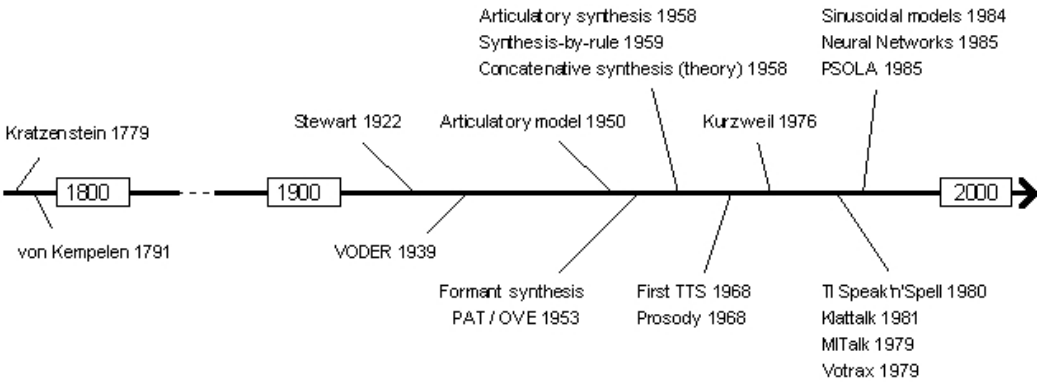

Figura 2.1: Hitos en la síntesis de voz.[Lemmetty, 1999]

Figura 2.2: Voder se exhibió en la Feria Mundial de 1939 de Nueva York (*http://en.wikipedia.org/wiki/Speech_synthesis#History*).

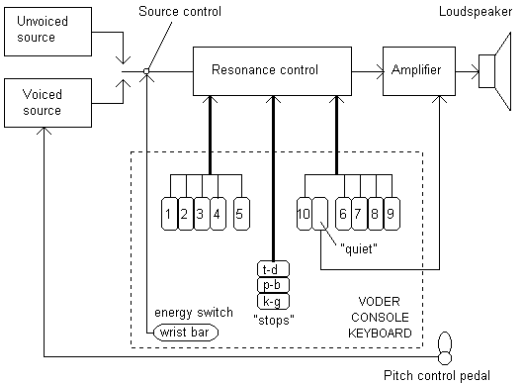

Figura 2.3: Esquema general de VODER ([Lemmetty, 1999]).

hoy dentro de este campo, ya que permitió, a partir de su creación, la representación visual del habla pudiendo mostrar claramente las diferencias entre distintos tipos de sonidos. El libro titulado *Visible Speech* [R. Potter, 1947] describe cómo los sonidos pueden distinguirse unos de otros por sus diferentes niveles de energía y diferentes frecuencias. Este aparato permitió el desarrollo de una teoría en la que se basa gran parte del trabajo moderno sobre síntesis de habla,la teoría del *filtro de fuentes*, desarrollada fundamentalmente por Gunnar Fant en la Kungliga Tekniska Högskolan (KTH), Estocolmo. Esta teoría, dicho de manera simple, establece que el habla puede ser modelada como una o más fuentes periódicas de ruido que son filtradas por un tubo acústico.

Al mismo tiempo que Walter Lawrence desarrollaba el primer sintetizador de formantes PAT (Parametric Artificial Talker) en 1953, Fant desarrolló el primer sintetizador de formantes en cascada: OVE I (Orator Verbis Electris), que fue mejorado diez años más tarde por Fant y Martony: OVE II. A éste le siguieron OVE III y GLOVE, desarrollados en la KTH, y el actual sistema comercial Infovox que originalmente desciende de ellos. John Holmes presentó su sintetizador de formantes en paralelo en 1972 después de estudiar estos sintetizadores durante varios años. Consiguió sintonizar a mano la frase sintetizada *"I enjoy the simple life"* tan bien que un oyente experto no podría distinguir entre la frase natural y la artificial.

Las aproximaciones modernas sobre síntesis pueden clasificarse según dos dimensiones parcialmente independientes. La primera hace mención al grado de parametrización de la aproximación. Puede tratarse de una aproximación altamente paramétrica, con un elevado número de parámetros independientes

a controlar, siendo éste el caso de sistemas articulatorios (se verán más adelante). También podemos encontrarnos con el caso de una aproximación con pocos o ningún parámetro variable, como el caso de algunas aproximaciones a la síntesis concatenante basadas en formas de onda. La segunda dimensión se refiere a la forma en la que se obtienen los parámetros: a partir de reglas o entrenados a partir de datos de voz; los ejemplos más comunes de éstos últimos son los sistemas concatenantes, en los que los parámetros toman la forma de segmentos procesados y almacenados a partir de voz real.

Las dos aproximaciones a la síntesis altamente paramétrica son la articulatoria y la basada en formantes. La síntesis articulatoria intenta producir voz entendiendo primeramente cómo cambia de forma el aparato vocal durante la producción de voz, para poder entender el problema acústico de cómo esos movimientos se traducen en sonidos. La síntesis por formantes intenta obtener reglas que directamente controlen la acústica, dada la cadena de sonidos a sintetizar. Para los sistemas concatenantes, donde los parámetros se encuentran en la forma de segmentos de voz almacenados, una de las principales elecciones a realizar es el tamaño de la unidad a utilizar en el inventario: en la mayor parte de los casos, la menor unidad que resulta práctica utilizar es la transición entre dos sonidos, denominada *difonema*, y ésta es la unidad que se utilizó en los primeros sistemas concatenantes [R. Sproat, 1999].

El primer sintetizador articulatorio fue presentado, en 1958, por George Rosen en el M.I.T. (Massachusetts Institute of Technology). El DAVO (Dynamic Analog of the VOcal tract) era controlado a base de la grabación en cintas de señales de control creadas a mano. A mediados de los años sesenta, se realizaron los primeros experimentos con LPC (Linear Predictive Coding). La predicción lineal fue utilizada primeramente en sistemas de bajo coste, como el TI Speakñ'Spell en 1980, y su calidad era bastante mala en comparación con los sistemas actuales. Sin embargo, con algunas modificaciones al modelo base, el método resultó muy útil, y se utiliza en muchos sistemas actuales. [Lemmetty, 1999]

2.2. Análisis de texto y prosodia

Un sintetizador *text-to-speech* no sólo debe ser capaz de sintetizar habla a partir de un conjunto de sonidos, sino también ser capaz de obtener una secuencia apropiada de sonidos, a la vez que una entonación adecuada, dado un texto determinado. El trabajo en este tema es mucho más reciente.

El primer sistema tts completo, para el idioma inglés, fue desarrollado en Japón por Noriko Umeda, entre otros, para Electrotechnical Laboratory, en 1968. Estaba basado en un modelo articulatorio e incluía un módulo de

análisis sintáctico con heurística sofisticada. Posiblemente el más sistema más conocido, y también uno de los primeros en intentar una conversión completa de texto a voz, es el sistema MITalk, desarrollado para el idioma inglés americano en el MIT Research Lab for Electronics durante la década de los 70. MITalk era capaz no sólo de pronunciar la mayor parte de las palabras corrientes de manera correcta, sino también de interpretar la puntuación, las abreviaturas y los números. Una combinación de reglas y diccionarios hacía posible manejar palabras que no estuvieran explícitamente recogidas en los mencionados diccionarios.

La primera ayuda a la lectura mediante escáner vino de la mano de Kurzweil en 1976. El sistema llamado *Kurzweil Reading Machine for the Blind* era capaz de leer bastante bien el texto escrito con varias fuentes. Sin embargo, era demasiado caro para los consumidores medios.

A finales de los ochenta surgieron numerosos ejemplos de sistemas TTS para diversos idiomas. Las aproximaciones más extendidas fueron las que utilizaban reglas *letter-to-sound*, es decir, aquellas que convertían cadenas de letras en secuencias apropiadas de sonidos,o más técnicamente *fonemas*. Sin embargo, no sólo hace falta poder predecir una serie de fonemas a partir de un texto, sino también determinar cómo "cortar" las sentencias en frases, elegir qué palabras hay que enfatizar, calcular la duración de los fonemas, la energía apropiada y la entonación asociados a cada pronunciación. Todos estos elementos del habla se recogen en el término colectivo *prosodia*. Durante siglos se han estudiado estos aspectos desde el punto de vista de la linguística y la fonética, pero su investigación en el contexto de generación de habla es, por supuesto, mucho más reciente. Por ejemplo, de acuerdo con [Klatt, 1987], el primer algoritmo implementado para determinar el contorno del tono fue realizado por Ignatius Mattingly en 1966, y fue incorporado en el sintetizador de Holmes.

2.3. La era electrónica

El primer circuito integrado para síntesis de voz fue, probablemente, el chip *Votrax*. En 1978 Richard Gagnon introdujo el sistema Type-n-Talk basado en Votrax. Dos años más tarde, en 1980, Texas Instruments presentó el sintetizador Speak-n-Spell basado en LPC y con un chip de bajo coste (TMS-5100); fue utilizado para ayudar en la lectura a los niños y se le prestó una atención importante. En 1982 Street Electronics presentó el sintetizador de bajo coste basado en difonemas *Echo*, que estaba basado en una nueva versión del mismo chip que utilizaba Speak-n-Spell Al mismo tiempo Speech Plus Inc. sacó a la luz el sistema tts Prose-2000, mientras que un año más

tarde apareció las primeras versiones comerciales de los famosos DECtalk e Infovox SA-101

2.4. Nuevos métodos

Las modernas tecnologías en síntesis de voz suponen la utilización de complicados y sofisticados métodos y algoritmos. Uno de los métodos utilizados recientemente en síntesis de habla es el de los *modelos ocultos de Markov* (HMM). Estos modelos llevan utilizándose en reconocimiento de voz desde finales de los años setenta, mientras que para síntesis se utilizan desde hace dos décadas. Un HMM es una colección de estados conectados entre sí por transiciones con dos conjuntos de probabilidades en cada una: una probabilidad de la transición que indica la probabilidad de tomar esa transición, y una función de densidad de la salida (pdf) que define la probabilidad condicionada de emitir cada símbolo de salida desde un alfabeto finito, dado que se ha tomado una transición.

Para reconocimiento de voz este método se basa en lo siguiente: sea **O** una secuencia de T medidas acústicas de la voz, **W** una secuencia de **N** palabras pertenecientes a un vocabulario fijo y conocido, $p(\boldsymbol{W/O})$ es la probabilidad de que la secuencia de palabras **W** haya sido pronunciada, dado que la secuencia **O** de medidad acústicas ha sido observada. El reconocedor decidirá a favor de la secuencia de palabras $\underline{\mathbf{W}}$ que satisfaga:

$\underline{\mathbf{W}} = argmax_W p(\boldsymbol{W/O})$

Además de para reconocimiento de voz, también se están utilizando *redes neuronales* en la síntesis de habla desde hace más de diez años y los resultados son prometedores, aunque el potencial de su uso no ha sido, todavía, suficientemente explorado.

Capítulo 3

Motores de generación de habla. Festival

En este capítulo se introducen primeramente unas pequeñas notas sobre distintos términos relacionados con la tecnología del habla y conceptos básicos de los sistemas text-to-speech, para después recoger las distintas herramientas disponibles en el mercado sobre este tema.

3.1. Algunas definiciones previas

Síntesis de habla Es la producción artificial de habla humana.

Sintetizador de habla Implementación software o hardware cuyo propósito es la síntesis de habla.

Fonema Unidad del habla que permite diferenciar palabras con distinto significado (por ejemplo, paso - vaso).

Difonema Unidad lingüística que comprende desde la mitad de un fonema hasta la mitad del siguiente (la mitad del fonema tiene una menor variación que su contorno por lo que la concatenación es mejor realizarla en ese punto).

Sílaba . Menor unidad de impulso respiratorio. Uno o más símbolos fonéticos que representan una unidad básica de una palabra fonológica. Es una unidad de pronunciación mayor que un sonido (véase fonema) y menor que una palabra.

Palabra . Conjunto de sonidos articulados que expresan una idea.

Dominio Se refiere generalmente a los campos de aplicación de los programas con capacidad lingüística, por ejemplo, banca, seguros, viajes, etc.; su significado en ingeniería lingüística es que el vocabulario de una aplicación queda restringido, delimitándose el ámbito de ésta, para reducir las necesidades de recursos lingüísticos.

Morfema Hace referencia al menor elemento significativo de la lengua.

Modelo oculto de Markov (HMM) Es una máquina de estados finitos en que las transiciones no sólo son de probabilidad sino también de producción; se utiliza actualmente en los sistemas de reconocimiento de habla para determinar las palabras que representan las formas de las ondas sonoras capturadas.

Léxicos Un léxico es un depósito de palabras y de conocimientos sobre ellas. Entre éstos se cuentan la información sobre la estructura gramatical de cada palabra (morfología), la estructura fonética (fonología) o el significado de la palabra en diferentes contextos textuales, por ejemplo, en función de la palabra o del símbolo de puntuación que esté antes o después de ella. Para ser útil, un léxico debe tener cientos de miles de entradas. Son necesarios para cada lengua de aplicación.

Sintaxis Reglas que describen la formación de las frases a partir de los elementos básicos de la lengua, es decir, los morfemas, palabras y partes de la oración.

Sistemas de vocabulario limitado e ilimitado Existen dos tipos de sistemas TTS, de vocabulario limitado y de vocabulario ilimitado. Su clasificación depende del tamaño del vocabulario que utilizan. Los sistemas limitados usan los segmentos de tipo palabra o sílaba; siendo limitados porque cuentan con un número finito o limitado de palabras o sílabas en su vocabulario.Los sistemas TTS de vocabulario ilimitado se caracterizan por poder sintetizar un número ilimitado de palabras y generalmente emplean segmentos más pequeños que la sílaba para lograr este objetivo. Para ilustrar lo mencionado anteriormente, véase la figura 3.1.

3.2. Sistemas TTS

Los sistemas de síntesis de voz, son aquellos que convierten una entrada escrita en palabras, a una salida pronunciada, simulando el proceso humano

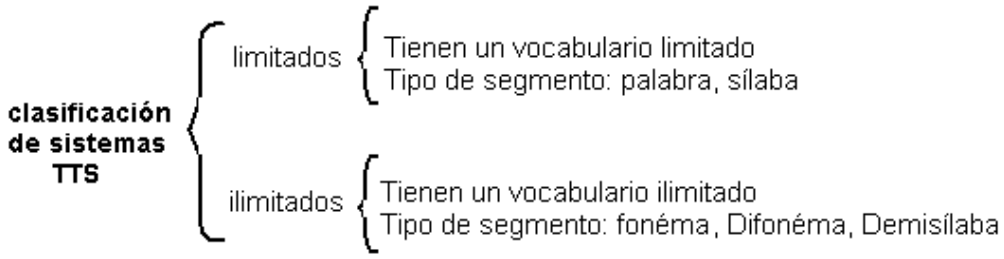

Figura 3.1: Clasificación de sistemas TTS

de leer en voz alta. Estos sistemas son también conocidos como sistemas de texto a voz o *TTS* (siglas de las palabras en inglés *Text To Speech*). Se obtienen de esta forma frases *nuevas*. Este hecho permite distinguir este tipo de sistemas respecto a los *Sistemas de respuesta por voz*, cuya única misión es la de concatenar palabras, o partes de frases, aisladas, y que sólo tienen aplicación cuando el vocabulario requerido es limitado o la estructura de las frases que deben pronunciarse respetan una estructura muy restringida, como es el caso de anuncios de llegadas en una estación de trenes, por ejemplo. Por tanto, podemos redefinir el proceso *tts* como la *producción automática de habla, mediante la transcripción grafema*[1] *- fonema de las frases a pronunciar* [Dutoit, 1997].

El concepto de síntesis tts de alta calidad(*high quality TTS synthesis*) aparece a mediados de los años ochenta, como resultado de los importantes desarrollos en síntesis de habla y técnicas de procesado de lenguaje natural, principalmente gracias a que surgen nuevas tecnologías (procesadores de inferencia lógica y señal digital).

La *naturalidad* de un sintetizador de habla suele hacer referencia a lo parecidos que resultan los sonidos producidos al habla real de una persona, mientras que el término *inteligibilidad* se refiere a la facilidad de entender la salida audible del sistema.

Arquitectura

Los sistemas TTS tienen dos módulos que interactúan entre sí para realizar la síntesis de voz y que son ilustrados en la figura 3.2 . El primer módulo

[1]La mínima unidad significativa en el plano de la lengua escrita, por analogía con fonema, que lo es en el plano de la lengua hablada.

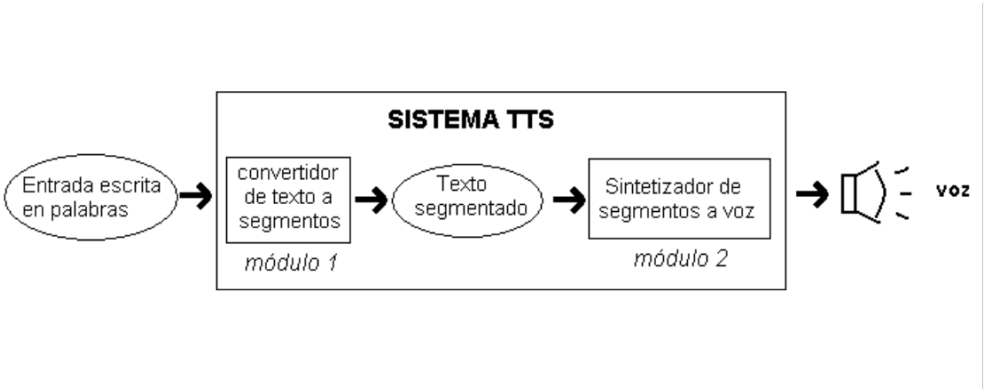

Figura 3.2: Sistema TTS (Fuente:*http://www.fismat.umich.mx/˜karina/tesisLicenciatura/tesis/capitulo1.html*).

es un convertidor de texto a segmento (NLP[2]), es decir, recibe el texto de entrada y lo separa en partes más pequeñas llamadas segmentos, produciendo la transcripción fonética del texto leído, junto con la deseada entonación y ritmo (lo que se entiende normalmente como *prosodia*) . En el segundo módulo, llamado sintetizador de segmentos a voz (DSP[3]), se convierten dichos segmentos a sonidos,transformando la información simbólica en voz.

Módulo NLP

La figura 3.3 presenta el esqueleto general de un módulo NLP. Junto con los bloques de generación de prosodia y de transformación letra-sonido, aparece un bloque que representa un analizador morfo-sintáctico, dado que existe la necesidad de procesar sintácticamente el texto introducido. El conseguir reducir las frases introducidas a una secuencia de sus partes y representarlas después en la forma de un árbol sintáctico es necesario por dos razones:

1. la prosodia natural depende en gran manera de la sintaxis,
2. una transcripción precisa sólo puede conseguirse si se sabe de qué categoría lingüística es la palabra, y son conocidas las relaciones de dependencia entre palabras sucesivas.

El bloque LTS (*letter-to-sound*) es el responsable de la transcripción fonética automática del texto entrante. El bloque de análisis de texto está a su vez compuesto por:

[2]Natural Language Processing module
[3]Digital Signal Processing module

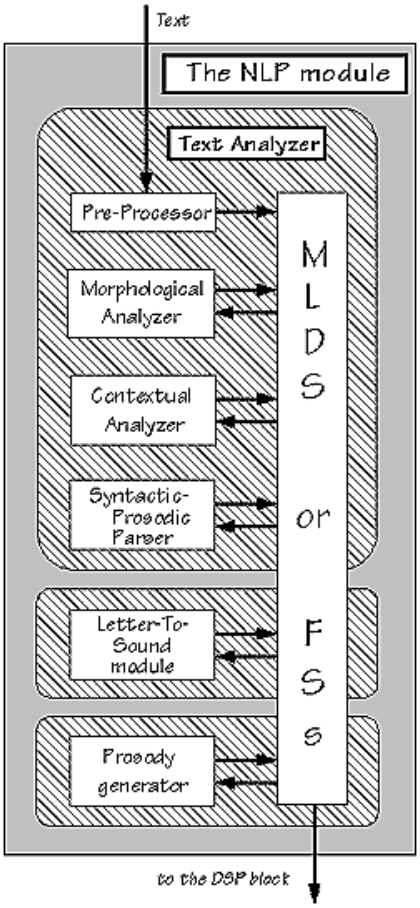

Figura 3.3: Esquema del módulo NLP.Tomado de [Dutoit, 1997]

- un módulo de pre-procesado, que organiza las frases en una lista manejable de palabras. Identifica números, abreviaturas, acrónimos y frases hechas y las transforma en texto cuando es necesario. Un problema importante a este nivel es la ambigüedad de la puntuación, pero puede resolverse, hasta cierto punto, gracias a unas básicas reglas gramaticales,

- un módulo de análisis morfológico, cuya tarea es proponer todas las categorías lingüísticas posibles para cada palabra de manera individual, en función de cómo se deletrean. Palabras compuestas o derivadas se descomponen en sus morfemas correspondientes gracias al uso de léxicos de raíces y afijos,

- un módulo de análisis de contexto, que considera las palabras en su contexto, lo que permite reducir la lista de posibles categorías lingüísticas a un número muy reducido de hipótesis altamente probables, dadas las partes correspondientes de la frase de las palabras vecinas. Esto se puede conseguir a través de *n-gramas*, que describen dependencias locales sintácticas en forma de autómatas probabilísticos de estados finitos(por ejemplo, un modelo de Markov),

- un analizador sintáctico-prosódico, que analiza el resto del espacio de búsqueda y encuentra la estructura del texto.

Como ya se ha mencionado anteriormente, el término prosodia hace referencia a ciertas propiedades de la señal de voz relacionadas con cambios audibles en el tono, sonoridad y longitud de la sílaba. El efecto más aparente de la prosodia es la enfatización. Por ejemplo, existen ciertos tonos que hacen destacar una sílaba dentro de una pronunciación, e indirectamente la palabra o grupo sintáctico a la que pertenece resaltará como un nuevo componente en el significado de la frase. Las características de la prosodia crean una segmentación de la cadena hablada en grupos de sílabas.

Módulo DSP

Las operaciones relacionadas con este módulo son una analogía al control dinámico de los músculos articulatorios y la frecuencia de vibración de los pliegues vocales consiguiendo que la señal de salida corresponda con los requisitos de la entrada. Para realizar esto de manera adecuada, el módulo DSP debería tener en cuenta, de alguna manera, las limitaciones articulatorias, ya que se conoce desde hace tiempo que las transiciones fonéticas son más importantes que los estados estables para la comprensión del habla. Por tanto esto puede realizarse de dos maneras:

- explícitamente, bajo la forma de una serie de reglas que describan formalmente la influencia de unos fonemas en los otros,
- implícitamente, acumulando ejemplos de transiciones fonéticas y coarticulaciones dentro de una base de datos de segmentos de hable, y usándolos tal como son, como unidades acústicas (en lugar de los fonemas)

Dos grandes clases de sistemas TTS han surgido a consecuencia de estas alternativas, que enseguida se han convertido en filosofías de síntesis dadas las divergencias que presentan en métodos y objetivos: *síntesis por reglas* y *síntesis por concatenación*.

En comparación con la primera, la síntesis por concatenación posee muy poco conocimiento de los datos que maneja. Su eficiencia para producir voz de alta calidad depende mayormente de el tipo de segmento elegido y el modelo de señal de voz al que se refieren el análisis y los algoritmos de síntesis.

Tipos de segmentos Los diferentes tipos de segmentos que pueden ser tomados como base son:

- *Palabra*.
- *Sílaba*.
- *CVC* Conjunto formado por la secuencia de letras: *Consonante-Vocal-Consonante*.
- *VCV* Conjunto formado por la secuencia de letras: *Vocal-Consonante-Vocal*.
- *Difonema* Sucesión transitoria de sonidos.
- *Pseudofonema*
- *Demisílaba*
- *Subfonema*

Algunas implementaciones realizadas con los diferentes tipos de segmentos hasta la fecha experimentados, se muestran en la tabla de la figura [4] 3.4.

En la tabla de la figura 3.5 se presentan algunos de los institutos de investigación que han trabajado en sistemas TTS y el idioma para el que desarrollaron sus sistemas.

[4]datos obtenidos del proyecto *Karina*: http://www.fismat.umich.mx/ karina/tesisLicenciatura/tesis/indice.html

Tipo de segmento	***Experimentos***
Palabra	Buron 1986, Chapman 1971
Sílaba	Ouh-Young 1986
CVC	Hayashi y Murakami 1992
VCV	Sato 1978
Difonema	Lefevre 1986
Pseudofonema	Mikuni y Ohta 1986
Demisílaba	Lovis y Fujimora 1976
Subfonema	El-Iman 1989, Dan y Dutta 1991

Figura 3.4: Implementaciones según segmentos

Instituto	***Idioma(s)***
YORK TALK	Inglés
University of Birmingham	Inglés británico y americano
Dec Talk	Inglés
Ipox	Alemán
Eurovocs	Japonés, Inglés, Alemán, Español y Francés

Figura 3.5: Desarrollos TTS

3.3. Herramientas disponibles

A continuación se detallan las distintas herramientas disponibles en función del sistema operativo del que se disponga. Entre todas ellas se realiza una descripción más detallada de aquellas que pueden resultar de mayor interés para nuestro actual proyecto.

- Apple Macintosh
  - BeSTspeech de Berkeley Speech Technologies, Inc., (BST)
  - Infovox Product Range
  - Macintosh Speech Output Applications
  - Macintosh Speech Synthesis Manager
  - MacYack Pro
  - MBROLA: Free Speech Synthesis Project
  - ProVoice Developer's Speech Toolkit de First Byte
  - SENSYN speech synthesizer
  - Sound Bytes DeveloperUs Kit
  - Macintosh Speech Synthesis Manager
- Windows (incluidos 95, NT, 3.1)
  - AcuVoice
  - AT&T Watson Speech Synthesis
  - BeSTspeech de Berkeley Speech Technologies, Inc., (BST)
  - Creative TextAssist and TextAssist API
  - DECtalk: Text-to-Speech de Digital
  - ETI-Eloquence
  - HADIFIX
  - Infovox Product Range
  - IPOX: All Prosodic Speech Synthesis Architecture
  - Lernout and Hauspie Text-To-Speech Windows SDK
  - Listen2 Text Reader
  - MBROLA: Free Speech Synthesis Project
  - Monologue for Windows de First Byte

- PAM - A Text-To-Speech Application
- ProVerbe Speech Engine de ELAN Informatique
- ProVoice Developer's Speech Toolkit de First Byte
- SENSYN speech synthesizer
- Sound Bytes DeveloperUs Kit
- Tinytalk
- TruVoice de Centigram
- WinSpeech
- ZMD Speech Synthesis

- DOS
  - CSRE: Computerized Speech Research Environment
  - Infovox Product Range
  - MBROLA: Free Speech Synthesis Project
  - ProVoice Developer's Speech Toolkit de First Byte
  - SENSYN speech synthesizer
  - spchsyn.exe
  - Tinytalk
  - ZMD Speech Synthesis
- OS/2
  - ProVerbe Speech Engine de ELAN Informatique
  - ProVoice Developer's Speech Toolkit de First Byte
  - Sound Bytes DeveloperUs Kit
- Unix
  - AcuVoice
  - AsTeR
  - BeSTspeech de Berkeley Speech Technologies, Inc., (BST)
  - DECtalk: Text-to-Speech de Digital
  - ETI-Eloquence
  - Emacspeak - A Speech Output Subsystem For Emacs

- Festival Speech Synthesis System
- JSRU
- Klatt-style synthesiser
- KPE80 - A Klatt Synthesiser and Parameter Editor
- "learph": Trainable text-to-phoneme software by Antonio Lucca
- Lucent Technologies Bell Labs Text-to-Speech system
- MBROLA: Free Speech Synthesis Project
- Orator de Bellcore
- ProVerbe Speech Engine de ELAN Informatique
- rsynth
- SENSYN speech synthesizer
- SGI Developers Toolbox Synthesiser
- Speak
- TrueTalk
- TruVoice de Centigram

- Circuitos integrados y hardware dedicado
  - Eurovocs
  - Infovox Product Range
  - ProVerbe Speech Engine de ELAN Informatique
  - RC Systems V8600/V8601 Text to Speech synthesizers
- Otras plataformas
  - BeSTspeech de Berkeley Speech Technologies, Inc., (BST)
  - TheBigMouth (NeXT)
  - MBROLA: Free Speech Synthesis Project
  - Narrator Translator Library (Amiga)
  - Narrator (Amiga)
  - TextToSpeech Kit (NeXT)
  - Orator from Bellcore
  - SENSYN speech synthesizer
  - WreadFiles: File reader for Commodore Amiga

3.3.1. *MBROLA*

MBROLA es un sintetizador de habla gratuito de alta calidad basado en difonemas. Se desarrolla en el laboratorio TCTS de la facultad politécnica de Mons (Belgica), cuyo objetivo es obtener un conjunto de sintetizadores para el mayor número de idiomas posible, y que pueden usarse sin ningún tipo de restricción para fines no comerciales ni militares. MBROLA 2.00 recibe como entrada una lista de fonemas, junto con información prosódica (duración de los fonemas y una descripción lineal del tono), y produce muestras de voz de 16 bits a la frecuencia de muestreo de la base de datos de los fonemas (típicamente 16kHz). No es por tanto un sistema TTS, ya que no recibe como entrada un simple texto. Las bases de datos están preparadas para diversos idiomas: inglés, español, italiano, danés y rumano. Se encuentra disponible una demostración web, donde se comparan las calidades de PSOLA, MBR-PSOLA, LPC, y sintetizadores concatenantes híbridos harmónicos/estocásticos en: http://tcts.fpms.ac.be/synthesis/modelcmp.html.

3.3.2. *FestVox*

Se trata de un proyecto desarrollado en la universidad Carnegie Mellon basado en el motor de síntesis de la misma universidad llamado **Flite** y en el sistema de síntesis de habla *festival* de la universidad de Edimburgo. Tanto la documentación como el software son gratuitos, independientemente del fin que se le vaya a dar, ya sea comercial o cualquier otro. Su propósito es la creación de nuevas voces sintéticas, más sistemáticas y mejor documentadas, haciendo posible a cualquier persona la construcción de una nueva voz. Suministra ayuda para la creación de voces para dominios limitados y ejemplos de bases de datos de sonidos junto con soporte para el desarrollo de voces en los idiomas soportados y en otros nuevos. Como requisitos para su utilización es necesario disponer de un sistema Linux donde se encuentren instalados *festival* y *speech_tools*(distribuidos conjuntamente) y un programa para la visualización de ondas.

3.3.3. *CHATR*

Desarrollado por *ATR Interpreting Telecommunications Laboratories*(Japón) y por el *Human Communication Research Centre* de la universidad de Edimburgo. Escrito en una mezcla de ANSI C y C++, cuenta con un intérprete de comandos de Lisp escrito en Scheme. Tiene una estructura modular que permite la selección y configuración de cada módulo en tiempo de ejecución. Permite entradas a diversos niveles: en el caso más abstracto puede recibir

descripciones lingüísticas de una frase pronunciada y de ellas generar una entonación y ritmo prosódico gracias a un proceso basado en reglas, o bien recibir como entrada la entonación y el ritmo especificando el tono, logrando de esta forma un control más explícito. Un tercer nivel de entrada especifica los fonemas individuales, las duraciones y los valores de la frecuencia fundamental F_0. El nivel más bajo permite la introducción de formas de onda para la generación de un sonido arbitrario. Gracias a los distintos niveles admitidos es posible, por tanto, especificar la forma de la expresión con el nivel de detalle que deseemos [Alan W. Black, 1994].

3.3.4. *CSLU ToolKit*

Desarrollado en el *Center for Spoken Language Understanding*(CSLU) de Oregón. Su propósito es hacer de la tecnología del habla algo accesible y fácil a todo el mundo. Incluye interfaces para telefonía estándar y dispositivos de audio junto con interfaces software para componentes de reconocimiento de voz, tts y animación. Los principales componentes que incorpora son:

- de reconocimiento de voz: soporta diversas aproximaciones al reconocimiento de voz incluyendo clasificadores de *redes neuronales artificiales*(ANN) y modelos ocultos de Markov,
- de síntesis de habla: integra el sistema de síntesis tts de *festival*. Además incorpora sintetizadores de español mejicano y alemán junto con un conversor de voz que genera automáticamente un mapeo de las voces del sintetizador a las voces grabadas de cualquier usuario,
- de herramientas de análisis de formas de onda,
- de entornos de programación,
- de animación facial: incorpora una *cabeza parlante* en tres dimensiones, llamada **Baldi**, desarrollada en la universidad de California. Baldi, controlada por los módulos de síntesis y de reconomiento de voz, es capaz de sincronizar voz sintética o natural con movimientos realistas de los labios, la lengua, la boca y la cara. La cara puede hacerse transparente revelando los movimientos de los dientes y la lengua mientras habla, con lo que se puede utilizar para el aprendizaje lingüístico. Es posible modificar la orientación de la cabeza para su análisis desde diferentes perspectivas. También incluye la posibilidad de reflejar las emociones básicas de sorpresa, felicidad, enfado, tristeza, disgusto y miedo a través de expresiones faciales.

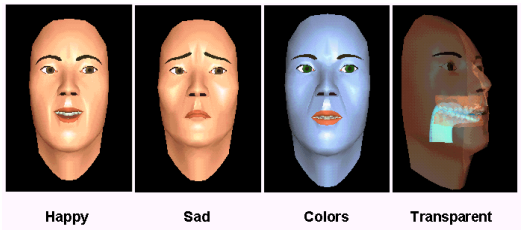

Figura 3.6: Baldi, una cabeza parlante

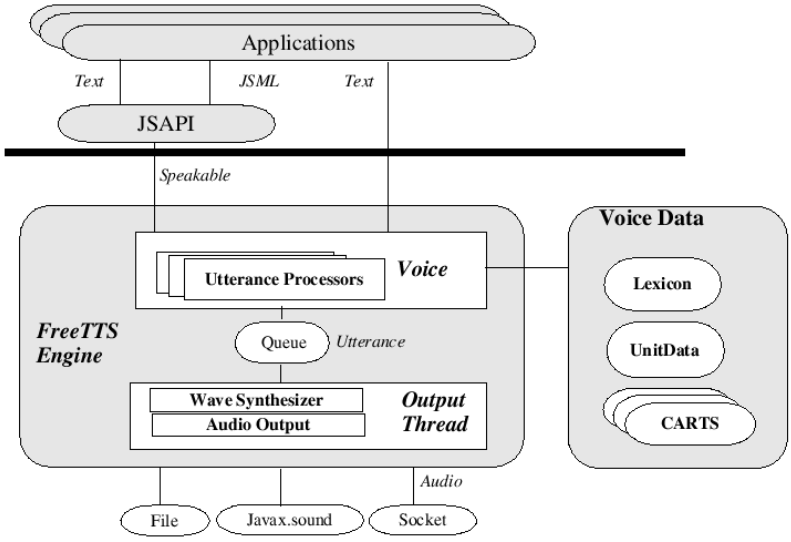

Figura 3.7: Estructura de FreeTTS[W. Walker, 2002].

El estudio de lenguaje natural se encuentra en desarrollo y está siendo integrado a través de un analizador semántico llamado PROFER(*Predictive RObust Finite-state parsER* [Sutton et al.,]

3.3.5. *FreeTTS*

Desarrollado por *Sun Microsystems Laboratories* está integramente escrito en Java, aunque se basa en dos sintetizadores: *festival* y *Flite*(Festival Lite). Debido al lenguaje de programación es totalmente portable para sistemas Windows, Linux y Solaris. Gracias a la utilización de optimizaciones resulta de dos a cuatro veces más rápido que Flite.[W. Walker, 2002]

3.3.6. *Festival Speech Synthesis System*

Desarrollado en el CSTR de la universidad de Edimburgo inicialmente por Alan W. Black, aunque actualmente son muchos los implicados en el proyecto. Festival ofrece un marco pra la construcción de sistemas de síntesis de voz a la vez que incluye ejemplos de varios módulos. En conjunto ofrece tts a través de diversas APIs (Application Programming Interface): a nivel de consola gracias a un intérprete de comandos de Scheme, como una librería de C++ y con una interfaz de Emacs. Se trata de un sistema multilingüe, estando disponible en inglés y español, aunque el inglés está más desarrollado. Está escrito en C++ y usa la librería de la universidad de Edimburgo llamada *Speech Tools* para la arquitectura de bajo nivel, y para ela control un intérprete de comandos basado en Scheme(SIOD). La documentación se da en formato FSF, el cual puede generar un manual impreso, archivos de información y HTML. Se trata de un sistema gratuito, que bajo una licencia del tipo X11 permite su uso tanto comercial como para software libre.

Scheme Para la realización de scripts usa Scheme, un lenguaje derivado de Lisp, sobre el que se pueden encontrar tutoriales y manuales en la red [5]. Este lenguaje utiliza como expresiones los átomos y las listas. Una *lista* se compone de un paréntesis izquierdo, una serie de expresiones y un paréntesis derecho. Un *átomo* puede ser un símbolo, un número, una cadena de caracteres o incluso tipos especiales como funciones, arrays, ... Todas las expresiones pueden ser evaluadas: los átomos en función de su tipo (son evaluados a sí mismos en el caso de números, cadenas, funciones, etc. mientras que los símbolos son evaluados como variables que devuelven su valor), y las listas se evalúan como llamadas a funciones en las que se evalúan todos los términos en primer lugar y entonces el primer elemento actúa como una función con el resto de elementos de la lista como argumentos. Se define el *car* de una lista como el primer elemento de la misma, mientras que se define el *cdr* como el resto de elementos.

```
festival>(set! alist '(apples pears bananas))
(apples pears bananas)
festival>(car alist)
apples
festival>(cdr alist)
(pears bananas)
```

[5] ver el Apéndice B.4

Modos de texto Festival soporta la noción de modos de texto cuando es utilizado como "caja negra" para sintetizar texto, ya que puede ser necesario que cada texto se interprete de una manera distinta. Por ejemplo, para la lectura de cualquier archivo de LaTeXde la presente memoria no nos gustaría que se pronunciaran los comandos de formato, sino simplemente el texto incluido en él. El tipo de modo deseado se indica mediante el tercer argumento de la función `tts`, es decir introduciendo por consola:

```
festival --tts example.html
```

Actualmente sólo se consideran estables los modos "STML" y "raw" pero se está trabajando en otros: email,HTML, LaTeX, etc. También existe la posibilidad de que el usuario creo su propio modo de texto.

Marcado de texto Además de STML, en versiones más recientes se incorpora la posibilidad de realizar un marcado de texto mediante SABLE basado en XML(eXtended Markup Language, un protocolo que permite la representación de objetos complicados mediante objetos de texto). Los textos con etiquetas de este tipo permiten a la herramienta recibir información sobre la forma de interpretar el texto que debe sintetizar: el idioma que se utiliza, la voz deseada (incluso cambiar de voz dentro de una misma frase), el volumen con el que se debe reproducir para enfatizar una palabra, la velocidad a utilizar, el tono, etc

Conjunto de fonemas Un conjunto de fonemas es un conjunto de símbolos que más adelante se definirán en término de características, como consonante/vocal, lugar de articulación para las consonantes, tipo de vocal, etc. Festival soporta múltiples conjuntos de fonemas de manera simultánea, siendo esto importante de cara a que tanto los léxicos como las reglas *letra-sonido* o los sintetizadores de onda requieren un conjunto de fonemas antes de poder operar. La herramienta puede realizar mapeos de un conjunto a otro de manera automática utilizando las características de cada fonema, resultando útil cuando se trata de conjuntos del mismo idioma. Mediante una serie de comandos es posible conocer los conjuntos de fonemas disponibles y el nombre, los fonemas, las características y los silencios del conjunto actual.

Léxicos Un léxico para festival es un subsistema que proporciona la pronunciación de las palabras. Las entradas están formadas por una palabra de cabecera, una parte de la frase (nombre, verbo, ...) y una pronunciación, siendo posible la creación de nuevos léxicos por parte del usuario. Es importante introducir en ellos, entre otras cosas, todas las letras del alfabeto ya que en el caso de acrónimos se deletrearán.

Duración La predicción de la duración de los segmentos se lleva a cabo a través de un módulo que llama a diferentes métodos en función del valor de un parámetro llamado `Duration_Method` que puede tomar varios valores: `Default, Averages, Klatt, Tree y Tree_ZScores` (estos dos últimos referidos a árboles CART) . Se puede modificar la duración utilizando otro parámetro llamado `Duration_Stretch`, ya que toda duración calculada por cualquier método se multiplicará por el valor del parámetro. Esto resulta útil para modificar la *rapidez* con la que se habla.

Voces Cada una de ellas se selecciona a través de una función de la forma "`voice_*`", siendo ésta la encargada de establecer el sintetizador de ondas, el conjunto de fonemas, el léxico, la duración, los modelos entonación y todo lo que sea necesario para cada locutor. Se encuentran disponibles voces británicas, americanas, escocesas y españolas, utilizando distintos sintetizadores: LPC y MBROLA. Es relativamente sencilla la creación de nuevas voces como ya se h mencionado.

Módulos Festival permite de manera sencilla la modificación y creación de nuevos módulos poniéndolos a disposición de todos los usuarios. En general es más sencillo trabajar en Scheme dejando la codificación en C++ sólo para los aspectos de más bajo nivel, como pueden ser los sintetizadores de ondas.

APIs Permiten la utilización de Festival dentro de otro tipo de aplicaciones.

API para Scheme Permite la configuración de festival a través de archivos escritos íntegramente en Scheme al inicio de la aplicación, ya sea de modo interactivo o mediante lotes.

API para la consola Simplemente permite sintetizar voz a partir de archivos de texto introducidos en la línea de comandos. Este es el uso más sencillo de la aplicación, aunque es necesario un pequeño tiempo de inicialización antes de procesar el texto.

Cliente/Servidor Se trata de una interfaz basada en sockets que permite a festival actuar como servidor a la espera de peticiones de clientes. Básicamente ofrece un nuevo intérprete de comandos para cada nuevo cliente, y resulta mucho más rápido que esperar a la inicialización como sucedía en el caso anterior. Cuenta con políticas de permiso/denegación de acceso y petición de contraseña, para garantizar la seguridad en el uso; además, entre otras medidas, permite realizar un seguimiento de las llamadas de los clientes o restringir las funciones a las que pueden tener acceso.

API para C/C++ Permite la integración de festival dentro de programas en C++, siendo necesario el enlazar las librerías adecuadas.

Java y JSAPI Es posible la utilización de Java para conectarse a un servidor remoto de festival a través de estas interfaces.

Soluciones para después de la síntesis A veces es necesario realizar diversos ajustes después de haber sintetizado el texto. Esto puede llevarse a cabo mediante ciertas reglas denominadas *hooks* en inglés , que se escriben en Scheme y que se recogen en archivos de configuración. Un ejemplo puede ser el reescalado del volumen una vez construida la forma de onda, para ello será necesario incluir en el archivo de configuración inicial la siguiente definición de la regla a aplicar:

```
(set! default_after_synth_hooks
(list (lambda (utt)
(utt.wave.rescale utt 1.0 t))))
```

3.3.7. Speech tools

Esta librería desarrollada en la universidad de Edimburgo es una colección de clases de C++, funciones y programas relacionados para la manipulación del tipo de objetos usados en el procesado de voz. Incluye soporte para la lectura y escritura de formas de onda, archivos de parámetros (LPC,F_0) en varios formatos y para la conversión entre ellos. También incluye soporte para objetos de tipo lingüístico y soporte para varios archivos etiquetados y ngramas. Se incluyen también una serie de programas. Una librería de entonación que incluye un rastreador de tonos y un sistema de etiquetado (usando el sistema *Tilt Labelling*) y un programa de construcción de árboles de clasificación y regresión(CART), llamado *wagon*. También existe un creciente avance en el soporte de clases para reconocimiento de voz, como decodificadores y HMMs.

Speech Tools no es un fin en sí misma, sino que ha sido diseñada para hacer fácil la construcción de otros sistemas de habla. Por ejemplo, suministra las clases esenciales para el sistema Festival.

Se distribuye de manera gratuita y sin ningún tipo de restricción.

3.3.8. Flite

Flite (Festival-lite) es un rápido motor de síntesis en tiempo de ejecución desarrollado en la universidad Carnegie Mellon y diseñado para pequeñas

máquinas embebidas y/o grandes servidores. Flite se ha diseñado como un motor de síntesis alternativo a Festival para las voces construidas usando las herramientas de construcción de voces de FestVox.

Se ha desarrollado enteramente en C reimplementando las partes del núcleo de la arquitectura de Festival (HRG) permitiendo así la compatibilidad entre voces construidas por cada sistema. Flite se ha escrito básicamente en sus primeras etapas de pruebas como software libre.

Capítulo 4

CORBA

La tecnología CORBA está orientada a los sistemas distribuidos heterogéneos. Un sistema distribuido heterogéneo es aquel compuesto por diferentes módulos software que interactúan entre sí sobre distintas plataformas hardware/software[1] unidas por una red de área local.

La *suite* de especificaciones CORBA *(Common Object Request Broker Architecture)* proporciona un conjunto de abstracciones flexibles y servicios concretos necesarios para posibilitar soluciones prácticas a los problemas que surgen en entornos distribuidos heterogéneos.

CORBA no es más que una especificación normativa para la tecnología de la gestión de objetos distribuidos (DOM). Esta tecnología DOM proporciona una interfaz de alto nivel situada en la cima de los servicios básicos de la programación distribuida. En los sistemas distribuidos la definición de la interfaz y ciertos servicios (como la búsqueda de módulos) son muy importantes. Proporciona un estándar para poder definir estas interfaces entre módulos, así como algunas herramientas para facilitar la implementación de dichas interfaces en el lenguaje de programación escogido.

CORBA es independiente tanto de la plataforma como del lenguaje de la aplicación. Esto significa que los objetos se pueden utilizar en cualquier plataforma que tenga una implementación del CORBA ORB (*Object Request Broker*) y que los objetos y los clientes se pueden implementar en cualquier lenguaje de programación por lo que al programador no le hará falta saber el lenguaje en que ha sido escrito otro objeto con el que se esté comunicando.

[1] Pueden tener arquitecturas diversas y soportar diferentes sistemas operativos

4.1. El grupo OMG

El OMG *(Object Management Group)* es una organización sin ánimo de lucro creada en 1989 formada con la misión de crear un mercado de programación basada en componentes, impulsando la introducción de objetos de programación estandarizada.

Su propósito principal es desarrollar una arquitectura única utilizando la tecnología de objetos para la integración de aplicaciones distribuidas garantizando la reusabilidad de componentes, la interoperabilidad y la portabilidad, basada en componentes de programación disponibles comercialmente.

El OMG es una organización de estandarización de carácter neutral e internacional, ampliamente reconocida. Hoy en día son miembros del OMG alrededor de mil distribuidores de software, desarrolladores y usuarios que trabajan en diferentes campos, incluyendo universidades e instituciones gubernamentales. Además, mantiene estrechas relaciones con otras organizaciones como ISO, W3C, etc. Sus estándares facilitan la interoperabilidad y portabilidad de aplicaciones construidas mediante objetos distribuidos. El OMG no produce implementaciones, sólo especificaciones de software que son fruto de la recopilación y elaboración de las ideas propuestas por los miembros del grupo OMG.

4.2. La norma CORBA

CORBA es una especificación normativa que resulta de un consenso entre los miembros del OMG. Esta norma cubre cinco grandes ámbitos que constituyen los sistemas de objetos distribuidos:

- Un lenguaje de descripción de interfaces, llamado **IDL** *(Interface Definition Language)*, traducciones de este lenguaje de especificación IDL a lenguajes de implementación (como pueden ser C++, Java, ADA, Python, etc.) y una infraestructura de distribución de objetos llamada **ORB** *(Object Request Broker)*.
- Una descripción de servicios, conocidos con el nombre de **CorbaServices**, que complementan el funcionamiento básico de los objetos de que dan lugar a una aplicación. Cubren los servicios de nombres, de persistencia, de eventos, de transacciones, etc. El número de servicios se amplía continuamente para añadir nuevas capacidades a los sistemas desarrollados con CORBA.
- Una descripción de servicios orientados al desarrollo de aplicaciones finales, estructurados sobre los objetos y servicios CORBA. Con el nom-

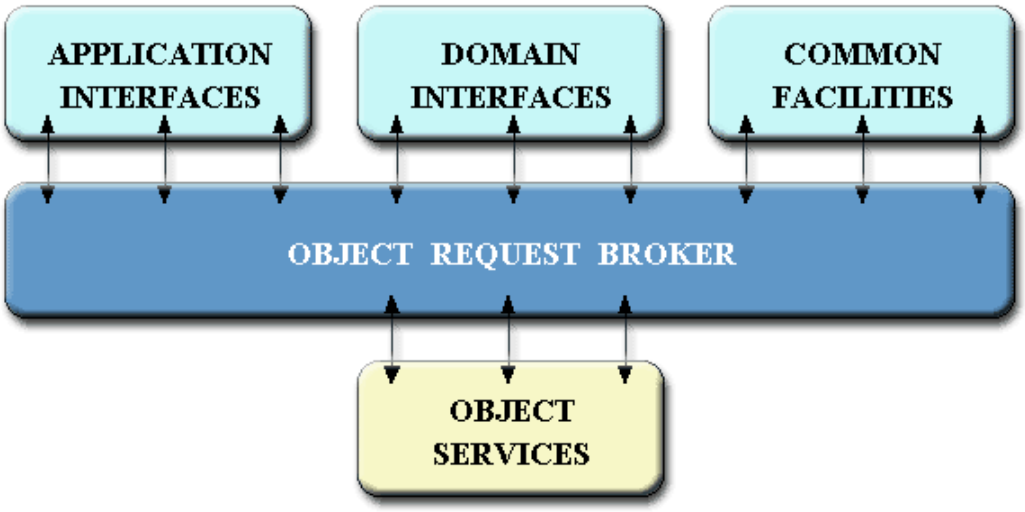

Figura 4.1: Servicio de objetos CORBA (Fuente: *http://www.cs.wustl.edu/~schmidt/corba-overview.html*)

bre de **CorbaFacilities**, estas especificaciones cubren servicios de alto nivel (como los interfaces de usuario, los documentos compuestos, la administración de sistemas y redes, etc.) Pretende definir colecciones de objetos prefabricados para aplicaciones habituales.

- Una descripción de servicios verticales denominados **CorbaDomains**, que proveen funcionalidad de interés para usuarios finales en campos de aplicación particulares. Por ejemplo, existen proyectos en curso en sectores como: telecomunicaciones, finanzas, medicina, etc.

- Un protocolo genérico de intercomunicación, llamado **GIOP** *(General Inter-ORB Protocol)*, que define los mensajes y el empaquetado de los datos que se transmiten entre los objetos. Además define implementaciones de ese protocolo genérico, sobre diferentes protocolos de transporte, lo que permite la comunicación entre los diferentes ORBs consiguiendo la interoperabilidad entre elementos de diferentes vendedores. Como por ejemplo el IIOP para redes con la capa de transporte TCP.

4.3. La Tecnología CORBA

En esta sección se hace una descripción general de la arquitectura CORBA y su funcionamiento. Para un conocimiento mas detallado es necesario recurrir a los documentos de especificación del OMG[2].

4.3.1. Common Object Request Broker Architecture (CORBA)

CORBA especifica un sistema que proporciona interoperabilidad entre sistemas que funcionan en entornos heterogéneos y distribuidos de forma transparente para el programador. Su diseño se basa en el modelo de objetos del OMG, donde se definen las características externas que deben poseer los objetos para que puedan operar de forma independiente de la implementación.

Las características más destacables de CORBA son las siguientes:

- es una tecnología no propietaria
- una base fundamental de la especificación son los requisitos reales de la industria
- es una arquitectura extensible
- es independiente de la plataforma
- es independiente del lenguaje de programación
- proporciona múltiples servicios de aplicación
- servidores y clientes (proveedores y usuarios de servicios) pueden desarrollarse de manera totalmente independiente

En una aplicación desarrollada en CORBA los objetos distribuidos se utilizan de la misma forma en que serían utilizados si fueran objetos locales, esto es, la distribución y heterogeneidad del sistema queda oculta y el proceso de comunicación entre objetos es totalmente transparente para el programador.

4.3.2. Arquitectura general

Un objeto CORBA es una entidad que proporciona uno o más servicios a través de una interfaz conocida por los clientes que requieren dichos servicios. Un objeto CORBA se define como aquel que proporciona algún tipo

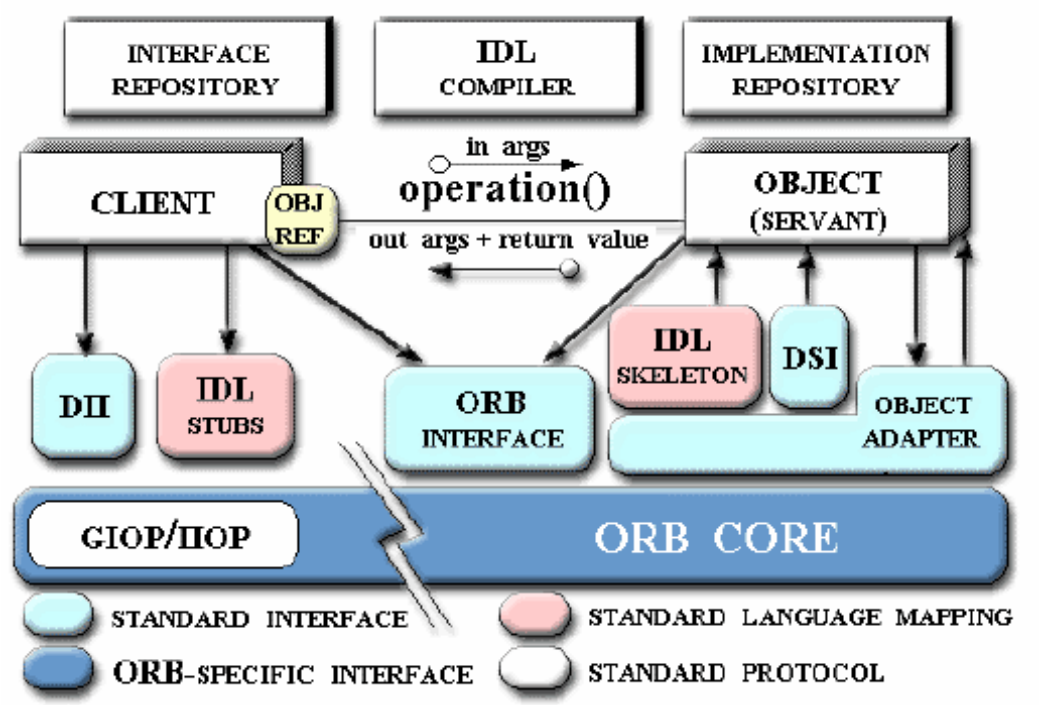

Figura 4.2: Componentes de la arquitectura CORBA (Fuente: *http://www.cs.wustl.edu/ ˜schmidt/corba-overview.html*)

de servicio, de modo que las entidades que solo los utilizan no se consideran como tales.

El componente central de CORBA es el ORB *(Object Request Broker)*, el cual proporciona la infraestructura necesaria para identificar y localizar objetos, gestionar las conexiones y transportar los datos a través de la red de comunicación. En general el ORB está formado por la unión de varios componentes, aunque es posible interaccionar con él como si fuera una única entidad gracias a sus interfaces. El núcleo del ORB *(ORB Core)* es la parte fundamental de este componente, siendo el responsable de la gestión de las peticiones de servicio. La funcionalidad básica del ORB consiste en transmitir las peticiones desde los clientes hasta las implementaciones de los objetos servidores.

Los clientes realizan peticiones de servicio a los objetos, también llamados servidores, a través de interfaces de comunicación bien definidas. Las peticiones de servicio son eventos que transportan información relativa a los objetos o entidades implicadas en dicho servicio, información sobre la ope-

[2]Más información en `www.omg.org`

ración a realizar, parámetros, etc. En la información enviada se incluye una referencia de objeto del objeto proveedor del servicio; esta referencia es un nombre complejo que identifica de forma unívoca al objeto en cuestión dentro del sistema.

Para realizar una petición un cliente se comunica primero con el ORB a través de uno de los dos mecanismos existentes a su disposición: para el caso de invocación estática el *stub* o para el caso de la invocación dinámica el DII, *Dynamic Invocation Interface*.

- Invocación estática: el *stub* es un fragmento de código encargado de mapear o traducir las peticiones del cliente, que está implementado en un lenguaje determinado, y transmitírselas al ORB. Es gracias a los *stubs* que los clientes pueden ser programados en diferentes lenguajes de programación.

- Invocación dinámica: se basa en el uso de la DII. Permite realizar invocaciones sin necesidad de que el cliente sepa cierta información en tiempo de compilación acerca del servidor, que sería necesaria en caso de utilizar el *stub*.

Una vez la petición llega al núcleo del ORB es transmitida hasta el lado del servidor, donde se sigue un proceso inverso de nuevo a través de dos mecanismos alternativos: el *skeleton* del servidor, análogo al *stub* del cliente, o la *DSI (Dynamic Skeleton Interface)*, contrapartida de la DII. Los servicios que proporciona un servidor CORBA residen en la implementación del objeto, escrita en un lenguaje de programación determinado. La comunicación entre el ORB y dicha implementación la realiza el adaptador de objetos (*Object Adapter*, OA), el cual proporciona servicios como:

- generación e interpretación de referencias a objetos
- invocación de métodos de las implementaciones
- mantenimiento de la seguridad en las interacciones
- activación o desactivación de objetos
- mapeo de referencias a objetos
- registro de implementaciones

Existen adaptadores de objetos más complejos, que proporcionan servicios adicionales, y que son utilizados para aplicaciones específicas (por ejemplo, bases de datos). Dos adaptadores de objetos básicos son el BOA (*Basic Object Adapter*) y el POA (*Portable Object Adapter*). El primero ha quedado

ya obsoleto, y suele utilizarse el segundo adaptador. El ORB proporciona además un POA preconfigurado que puede usarse si no se desea crear un POA específico para una aplicación, llamado *RootPOA*.

El adaptador de objetos necesita conocer cierta información acerca de la implementación del objeto y del sistema operativo en el que se ejecuta. Existe una base de datos que almacena esta información, llamada *Interface Repository* (IR), que es otro componente estándar de la arquitectura CORBA. El IR puede contener otra información relativa a la implementación como por ejemplo datos de depurado, versiones, información administrativa, etc.

Las interfaces de los objetos servidores pueden especificarse de dos maneras: o bien utilizando el lenguaje IDL, o bien almacenando la información necesaria en el IR. Para el caso de la invocación dinámica la interfaz DII accede al IR en busca de ésta información en concreto cuando un cliente la utiliza para realizar una petición. De este modo se hace posible que un cliente pueda invocar un servicio sin necesidad de conocer la descripción IDL de la interfaz del objeto servidor.

Por último, en el lado del servidor CORBA, los objetos que se encargan en última instancia de atender a las peticiones de servicio son los *servants*. Estos objetos contienen la implementación de las operaciones asociadas a cada servicio o método de la interfaz del objeto. No son visibles desde el lado del cliente; este sólo ve una entidad única a la que conoce como servidor. Dentro del servidor se crean y gestionan *servants* que atienden las peticiones que llegan al mismo. El adaptador de objetos es el encargado de hacer llegar dichas peticiones a los *servants*, crearlos o destruirlos, etc. Cada *servant* lleva asociado un identificador llamado *object ID*, valor que es utilizado por el adaptador de objetos para la gestión de los mismos.

4.3.3. Interoperabilidad entre ORBs

En la actualidad existen muchas implementaciones diferentes de ORBs, lo cual es una ventaja ya que cada desarrollador puede emplear aquella que mejor satisface sus necesidades. Pero esto crea también la necesidad de un mecanismo que permita a dos implementaciones diferentes comunicarse entre sí. También es necesario en determinadas situaciones proporcionar una infraestructura que permita a aplicaciones no basadas en CORBA comunicarse con aplicaciones basadas en esta arquitectura. Para satisfacer todos estos requisitos existe una especificación para lograr la interoperabilidad entre ORBs.

Las diferencias en la implementación del *broker* no son la única barrera que separa a objetos que funcionan en distintos ORBs, también existen otras dificultades como infraestructuras de seguridad o requisitos específicos

de sistemas en vías de desarrollo. Debido a esto, los objetos que funcionan en diferentes dominios (ORBs y entornos de los mismos) necesitan un mecanismo que haga de puente entre ellos. Este mecanismo debe ser suficientemente flexible para que no sea necesaria una cantidad de operaciones de "traducción" inmanejable, ya que la eficiencia es uno de los objetivos de la especificación de CORBA. Esta característica es crítica en sistemas de control, donde suele ser necesario alcanzar unos niveles determinados de seguridad y predecibilidad.

La interoperabilidad puede alcanzarse a través muchos procedimientos, los cuales pueden clasificarse en dos tipos principales: inmediatos e intermediados (*immediate/mediated bridging*). En los procedimientos intermediados los elementos de un dominio que interaccionan con los de otro dominio distinto, son transformados a un formato interno acordado por ambos dominios de antemano, y bajo este formato la información pasa de un dominio al otro. Este formato interno de la información puede basarse en una especificación estándar, como en el caso del IIOP del OMG, o bien puede ser un formato acordado de forma privada. Los procedimientos inmediatos se basan en traducir directamente la información desde el formato utilizado por un dominio, al formato utilizado por el otro, sin pasar por estructuras intermedias de datos. Esta solución es mucho más rápida, pero es menos general.

4.3.4. CORBA IDL

Como ya hemos mencionado, el lenguaje IDL (*Interface Definition Language*) es un lenguaje utilizado para especificar interfaces CORBA. A través de un compilador específico que procesa las definiciones escritas en IDL, se genera código fuente en el lenguaje de programación deseado en cada caso, código que es utilizado en las aplicaciones para crear servidores CORBA ya que proporciona la infraestructura de *skeletons* y *stubs* necesaria para que estos objetos puedan comunicarse con el ORB. IDL es un estándar ISO, y tiene las siguientes características principales:

- su sintaxis es muy similar a la de C++
- soporta herencia múltiple de interfaces
- independiente de cualquier lenguaje de programación y/o compilador, puede mapearse a muchos lenguajes de programación.
- permite independizar el diseño de la interfaz de la implementación del objeto CORBA en cuestión. La implementación puede cambiarse por otra distinta, manteniendo la misma interfaz, de forma que desde el "ex-

terior" el objeto sigue siendo el mismo y continúa ofreciendo idénticos servicios.

IDL no es un lenguaje de programación propiamente dicho, es únicamente un lenguaje declarativo. Tiene tres elementos principales: operaciones (métodos), interfaces(conjuntos de operaciones) y módulos (conjuntos de interfaces).

El IDL se mapea ("traduce") al compilar al lenguaje de programación deseado. Para el lenguaje C++, IDL sigue la siguiente tabla de conversión:

Tipo de datos IDL	***Tipo de datos C++***	***typedef***
short	CORBA::Short	short int
long	CORBA::Long	long int
unsigned short	CORBA::UShort	unsigned short
unsigned long	CORBA::ULong	unsigned long
float	CORBA::Float	float
double	CORBA::Double	double
char	CORBA::Char	char
boolean	CORBA::Boolean	unsigned char
octet	CORBA::Octet	unsigned char
enum	enum	enum

4.3.5. Bases de la construcción de aplicaciones CORBA

Para desarrollar un objeto CORBA se siguen, en general, los siguientes pasos:

1. **Diseño**: determinación de los servicios que debe proporcionar el objeto, e implementación de la/las interfaces IDL correspondientes.

2. **Compilación de IDL**: mediante el compilador de IDL se procesan las definiciones de interfaces y se generan los *skeletons* y *stubs* correspondientes, en un lenguaje de programación determinado.

3. **Implementación de servants**: utilizando las clases generadas por el compilador de IDL, se crean los *servants* de la aplicación, que contienen la implementación de las operaciones del objeto CORBA.

4. **Implementación del servidor**: el objeto CORBA debe proporcionar la infraestructura base para que la comunicación con el ORB sea posible; debe crear un adaptador de objetos para sus *servants*, etc.

5. **Compilación**: se procede a la compilación de los archivos de código fuente. Debe incluirse el código generado automáticamente por el compilador de IDL (la parte correspondiente a los *skeletons*).

El proceso se reduce básicamente a tres fases: generación e integración de código correspondiente a las interfaces, implementación de la funcionalidad del objeto CORBA y por último, compilación. Estos pasos son los correspondientes para el desarrollo de un objeto CORBA, es decir, un servidor. Para el desarrollo de un cliente, el proceso se reduce a la integración del código fuente generado correspondiente a los *stubs*.

4.4. Brokers

4.4.1. El Broker ICa

En esta sección se describe **ICa Broker** que es una implementación CORBA. Los proyectos que se realizan en el grupo ASLAb estan estrechamente relacionados, constituyendo, en cierta medida, piezas del mismo puzle **ICa**.

ICa Broker ha sido desarrollado en proyectos anteriores del grupo y posteriomente por un *spin-off* de alumnos de nuestro grupo (`www.scilabs.es`). Se sigue usando **ICa Broker** porque permite explorar aspectos particulares de la implementacion gracias al acuerdo UPM-SCILabs de acceso al código fuente sin tener que entrar en la complejidad de otros brokers *Open Source* como TAO.

Como ya se ha mencionado, este proyecto emplea recursos desarrollados en un proyecto anterior cuyo objetivo era el desarrollo del sistema de comunicación del robot móvil [Pareja, 2004]. En este proyecto se empleó **ICa Broker** como plataforma CORBA pero, como se ha indicado previamente en la Sección 4 esto no condiciona a tener que utilizarlo en este proyecto (recordemos la interoperabilidad entre ORBs descrita en la sección anterior).

ICa Broker es un broker CORBA –con un entorno de desarrollo asociado– diseñado para asistir la creación de sistemas software de medio y alto nivel en entornos industriales. Proporciona herramientas para el desarrollo de software distribuido, con prestaciones de tiempo real y tolerancia a fallos. ICa Broker 1.0 fue el resultado del trabajo desarrollado por el Departamento de Automática, Ingeniería Electrónica e Informática Industrial dentro del marco de un proyecto ESPRIT DIXIT en colaboración con varias empresas de toda Europa. ICa Broker RT 2.0 es la versión actual, propiedad de la empresa SCILabs con la que la UPM mantiene un acuerdo de colaboración. Esta versión cumple con la especificación Real-time CORBA 1.1 de la OMG.

Arquitectura de Control Integrado

ICa significa Arquitectura de Control Integrado. Este es el marco en el que se desarrolló este broker: para posibilitar el desarrollo de sistemas integrados de control.

Entendemos por **arquitectura** un diseño software a un determinado nivel de abstracción, centrado en los patrones de organización del sistema que describen cómo se particiona la funcionalidad y cómo esos elementos se interconectan e interrelacionan entre sí. Por un lado una arquitectura es un modelo de separación del sistema en componentes y por otro un modelo de coordinación de los mismos.

ICa implementa una metodología de orientación a componentes: módulos software que interaccionan unos con otros a través de una interfaz pública bien conocida. En el caso más general estos componentes se sitúan en un entorno distribuido y heterogéneo (tanto desde el punto de vista del hardware como de los sistemas operativos soportados) formado por una o varias plataformas hardware conectadas por una red que les permite interactuar a través de ella.

En **ICa 1.0** se desarrollaron extensiones para adaptarse a las necesidades de los sistemas de control de procesos industriales, en los cuales aparecen restricciones de tiempo real, los sistemas deben tener tolerancia a fallos y existen limitaciones de velocidad importantes. Ejemplos de estas extensiones son los mecanismos de gestión de *timeouts* y de *checkpointing*. No renuncia a su aplicación en entornos sin estas necesidades, por lo que es aplicable en los sistemas que no requieran alguna de estas prestaciones simplemente renunciando a ellas. Existen otros frameworks que han sido diseñados para aplicaciones de carácter más ofimático y se aplican de forma forzada en entornos que se encuentran más allá de sus especificaciones de diseño.

Por último con el término **integrado** se hace referencia a una de las características principales de **ICa**: su orientación hacia la integración de tecnologías de control, plataformas y lenguajes diversos. A la hora de integrar componentes para formar un sistema de control existen técnicas diversas y los desarrolladores deben conocer y controlar todas ellas para utilizar la más adecuada en cada caso. Los sistemas de control se diseñan generalmente por capas, formando lo que se conoce con el nombre de pirámide de control. En las capas inferiores se intercambia información a altas velocidades y con un nivel de abstracción bajo, mientras que, a medida que se va ascendiendo por la misma, los flujos de datos disminuyen pero aumenta el nivel de abstracción de los mismos. En las capas superiores tenemos control inteligente, en el que la información que se intercambia es abstracta y desarrollada con metodologías diversas, proporcionando principalmente soporte a la decisión.

La integración de estas metodologías de control, junto con la integración de diversas plataformas y sistemas operativos son los objetivos de **ICa**.

4.4.2. Omni

OmniORB fue desarrollado por Olivetti Research Ltd (que en 1999 pasó a ser parte de AT&T Laboratories) para ser utilizado en pequeños equipos de entorno embebido que necesitaban de comunicación con ORBs comerciales en ordenadores sobremesa y servidores. Ha evolucionado con el tiempo a través de 11 versiones públicas para la comunidad CORBA, varias versiones beta y pequeños desarrollos. En la actualidad lo utilizan numerosos desarrolladores de programas en múltiples aplicaciones.

OmniORB es un broker que implementa la especificación 2.6 de CORBA. Es robusto y tiene grandes prestaciones, permitiendo la programación en C++ y Python. Es libre bajo los términos GNU General Public License. Es uno de los tres ORB al que se le ha concedo el Open Group's Brand de CORBA, lo que significa que ha sido probado y certificado para CORBA 2.1 (le faltan todavía características para ajustarse a la norma 2.6 como por ejemplo no se soportan las interfaces locales ni los objetos por valor).

Capítulo 5

Plataforma de pruebas: Higgs

Este capítulo está dedicado a la descripción de la plataforma de pruebas usada para este proyecto, que también sirve como plataforma para otros proyectos desarrollados dentro del grupo. Se describe la parte Hardware y Software de la misma para una mejor comprensión del robot con el que se trabaja.

5.1. Introducción

El laboratorio ASLab posee un robot móvil Pioneer modelo 2AT-8, "bautizado" con el nombre de *Higgs* (ver Figura 5.1), que se utiliza como plataforma de pruebas y que fue adquirido en febrero de 2003. El objetivo de esta adquisición fue incorporar al laboratorio una plataforma móvil e implementar sistemas de control inteligentes en la misma. Con la incorporación del robot al laboratorio ASLab se dispone de una nueva plataforma con la que comenzar a hacer pruebas de control inteligente y por tanto avanzar hacia la consecución de su objetivo a largo plazo: la creación de sistemas de control capaces de aprender a controlar cualquier sistema y de tener conciencia de sí mismos.

5.2. Características generales

El robot Pioneer 2-AT8 pertenece a la familia de robots móviles de *ActivMedia*. *ActivMedia Robotics* diseña y construye robots móviles inteligentes, así como sistemas de navegación, control y de soporte para la percepción sensorial de los mismos. Esta empresa ha vendido más de 1700 robots en todo el mundo y éstos han ganado varios y prestigiosos concursos.

Figura 5.1: Pioneer 2-AT8 (Fuente: *http://www.activrobots.com/*)

Figura 5.2: Robots ActivMedia Robotics (Fuente: *http://www.activrobots.com/*)

Pioneer 2-AT8 es una plataforma robusta que incorpora todos los elementos necesarios para la implementación de un sistema de navegación y control en el robot para un gran número de entornos del mundo real.

El tamaño del robot es pequeño en comparación con sus prestaciones. Pesa 14 kg con una batería incluída. Su estructura es de aluminio. Estas características le permiten transportar hasta 30 kg sobre el mismo. El microcontralador que gobierna los diversos dispositivos electrónicos conectados es un Hitachi H8S. El cuerpo del robot es de aluminio y aloja las baterías, los motores, los circuitos electrónicos y el resto de componentes. Además existe espacio para alojar diversos accesorios, como un PC de a bordo, un módem radio o ethernet radio o para incorporar sensores.

Los dispositivos electrónicos presentes en el robot son controlados por

el microcontrolador Hitachi H8S. Su cometido es manejar los actuadores, disparar y recoger la señal de los sónares, controlar la electrónica del robot y realizar el resto de funciones de bajo nivel. Es capaz de comunicarse con otras máquinas a través de una interfaz serie RS-232. El robot está alimentado con tres baterías de 12 voltios y 7 amperios-hora cada una. Son intercambiables entre sí y accesibles a través de una mini-puerta en la parte trasera del robot. Por lo tanto disponemos en total de 252 watios-hora, lo que asegura varias horas de autonomía para la plataforma.

5.2.1. Elementos añadidos

Con el objetivo de dotar al robot de los elementos hardware necesarios para constituir su cerebro y el sistema de comunicaciones es necesario añadir un PC de a bordo. La misión del PC de abordo es asegurar la comunicación con el microprocesador del robot, la ejecución de programas inteligentes de navegación y control del robot y la conexión del mismo con la red inalámbrica local del laboratorio ASLab [Pareja, 2004].

Una vez instalado el computador de a bordo, es necesario añadir los elementos hardware necesarios para habilitar la comunicación inalámbrica con la red local. Los elementos que se utilizan para la comunicación entre el robot y la red de ASLab son una tarjeta PCMCIA y un punto de acceso que permite comunicar la red inalámbrica con la red cableada del laboratorio.

Debido a que la tarjeta va insertada en la placa que va dentro del cuerpo del robot, al ser este de aluminio y estar cerrado en condiciones de trabajo, es necesario utilizar una antena amplificadora externa para permitir la comunicación inalámbrica. Esta antena fue adquirida junto con la tarjeta wireless y el Punto de Acceso. La tarjeta viene preparada para conectarla a la antena, una vez realizada la conexión colocamos la antena en la plataforma exterior del robot (ver Figura 5.3).

El dispositivo de almacenamiento elegido para albergar el sistema operativo de la placa es una CompactFlash (CF). Puede ser utilizada para el intercambio rápido de datos, de fácil utilización, como un pequeño disco duro de tamaño definido.

Se ha escogido Linux con un *kernel*[1] de la versión 2.6.x. Se elige instalar un kernel de esta serie ya que a partir de la versión 2.6 se incorporan características de tiempo real que nos resultan útiles a la hora de desarrollar software para la plataforma de pruebas (más información en [Fernández, 2003]).

[1] el *kernel* es el núcleo del sistema operativo

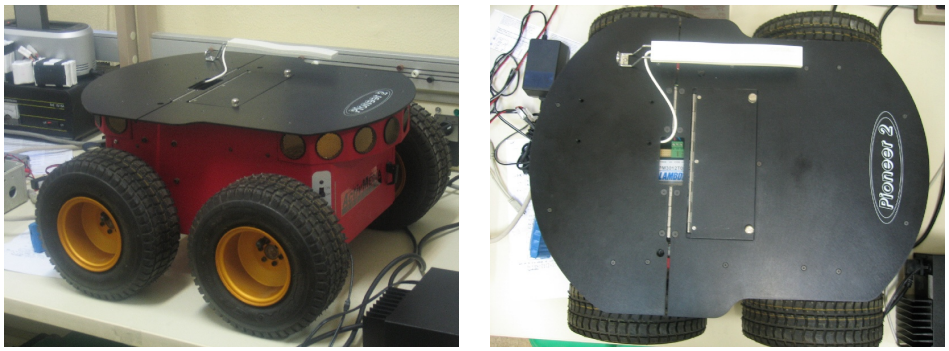

Figura 5.3: Ubicación de la Antena Amplificadora en el panel del robot para la comunicación inalámbrica [Conde, 2005a]

5.3. SW-Propio de la plataforma

5.3.1. Sistema Operativo AROS

El robot Pioneer 2-AT8 es controlado en primera instancia por un microprocesador Hitachi H8S. Para el desarrollo de la comunicación de éste con cualquier otra máquina es preciso utilizar una arquitectura cliente - servidor. Según este modelo, sobre el microcontrolador del robot corren los procesos servidores que se encargan del manejo y control de las tarjetas controladoras de los dispositivos electrónicos y de realizar las funciones de bajo nivel del sistema.

AROS (ActivMedia Robotics Operating System) es el conjunto de estos procesos servidores y constituye un sistema operativo que corre en el microprocesador. Es un software de bajo nivel cuyo cometido es manejar la regulación de velocidad de los motores, disparar y recoger la señal de los sonars, recoger las señales de los encoders y en general llevar a cabo todas las funciones de bajo nivel. Además es el responsable de transmitir por medio de comandos esta información a otra aplicación, la cuál será cliente de AROS. En nuestro caso, el cliente de AROS será un programa que correrá sobre la CPU de Higgs y que enviará comandos y recibirá información del microprocesador Hitachi H8S. Este proceso cliente de AROS será a su vez servidor de eventuales programas clientes de la red local.

Una de las principales características de AROS es que es actualizable y compatible con otros robots del mismo fabricante.

Protocolo de comunicación entre AROS y un cliente

AROS se comunicará con su cliente mediante una interfaz serie RS-232 usando un protocolo especial de comunicaciones para la transmisión de paquetes de datos, de un tipo desde el cliente al servidor y de otro distinto desde el servidor al cliente.

Estos paquetes son los siguientes:

- Los paquetes que envia el servidor, SIP (server information packets), informan al cliente sobre el estado del robot y ofrecen distintas lecturas sensoriales.
- Los comandos del cliente que estan determinados univocamente por un número ya que AROS posee un formato estructurado en comandos para poder recibir y responder a las ordenes del cliente.

Ambos protocolos son un flujo de bits de un máximo de 208 bytes cada uno que consisten en cinco elementos:

- Una cabecera de dos bytes.
- Un contador de un byte de los siguientes paquetes.
- El comando del cliente o el SIP del servidor.
- Argumentos o bits de datos.
- Checksum (chequeo) de dos bytes.

AROS ignora comandos o paquetes cuyo byte contador exceda de 252 o presente un checksum erróneo. Sin embargo el interfaz cliente - servidor está provisto de métodos para reconocer paquetes defectuosos, puesto que muchos de los comandos alteran variables de estado en el servidor. Examinando el valor de estas variables se pueden detectar y reutilizar comandos defectuosos.

Al diseñar aplicaciones cliente se han de tener en cuenta las limitaciones de la comunicación serie. En términos generales, se enviará un comando o un paquete de datos entre cada tres y cinco milisegundos.

Conexión cliente - servidor AROS

Para ejecutar cualquier tipo de programa de control y pilotaje sobre el robot (proceso cliente) es preciso, establecer una conexión con el servidor

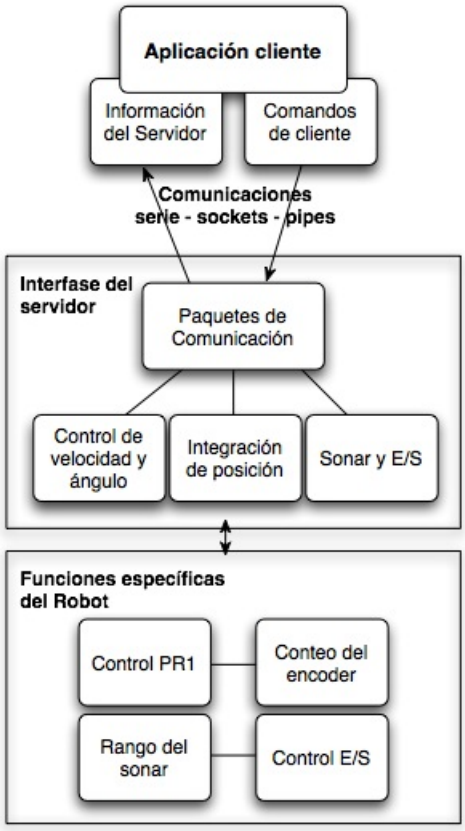

Figura 5.4: Arquitectura cliente - servidor

AROS a través de una interfaz serie. Después del establecimiento de la conexión el cliente enviará comandos y recibirá información del servidor.

Cuando se enciende el robot o se pulsa `RESET`, AROS opera en un modo especial llamado de espera. En este estado AROS escucha la posible llegada de paquetes de comunicación para establecer una conexión cliente - servidor. Para establecer la misma, la aplicación cliente debe enviar una serie de tres paquetes de sincronización (los comandos SYNC0, SYNC1 y SYNC2) y recuperar las respuestas del servidor.

Después de recibir el paquete SYNC2, AROS enviará información sobre la configuración del robot, con lo que se concluye el proceso de conexión. A continuación el cliente debe enviar el comando OPEN, que ordena al microprocesador Hitachi H8S la ejecución de funciones de mantenimiento y autochequeo y de inicio de diversos procesos servidores como el control de los sonar y los motores.

Además AROS incorpora un watchdog o programa de seguridad que espera que, una vez se ha conectado el cliente, se reciban al menos un paquete de comunicación cada cierto periodo de tiempo (este periodo es configurable).

5.3.2. SW Control del robot: ARIA

ARIA es un paquete software orientado a objetos programado en C++ que constituye una API (application-programming interface) para el control de diversos robots móviles de Activmedia.

La comunicación del microprocesador Hitachi H8S con cualquier máquina requiere de una arquitectura cliente - servidor. Los procesos servidores constituyen el sistema operativo llamado AROS. Éstos se limitan a la gestión de las tarjetas electrónicas y de las funciones de bajo nivel de la plataforma móvil.

Sin embargo en el lado del servidor AROS no se pueden llevar a cabo tareas inteligentes o de control. ARIA es el software del lado del cliente. Gracias a este paquete de clases nos podremos comunicar y controlar el robot desde aplicaciones cliente.

Mediante ARIA se podrán construir aplicaciones para el control de alto nivel del robot: desde la ejecución de simples comandos hasta la elaboración de comportamientos inteligentes (detectar y evitar obstáculos, reconocimiento de características de objetos, exploración, etc.).

ARIA está registrado bajo la GNU Public Licence, lo que significa que es un software de código abierto. Igualmente, todo el software desarrollado a partir de ARIA debe ser distribuido proporcionando el código fuente.

Comunicación con el robot

Una de las principales funciones de ARIA y el primer cometido de cualquier aplicación de control para el robot es establecer y gestionar la comunicación cliente - servidor AROS entre los dispositivos del robot y la aplicación cliente. ARIA suministra diversas clases para establecer esta conexión.

Después de establecer la conexión, las funciones principales que se realizan son la lectura y escritura de datos de los dispositivos.

ArRobot

ArRobot es la clase que constituye el corazón de ARIA. Actúa como pasarela de la comunicación cliente - servidor y gestiona la sincronización del sistema y la recolección y distribución de información referente al estado del mismo.

Figura 5.5: Esquema de la clase ArRobot de ARIA (Fuente: *http://www.activrobots.com/*)

ArRobot gestiona los detalles de bajo nivel al construir y enviar comandos al robot y al recibir y decodificar los paquetes de información del servidor (SIPs).

Los SIPs son enviados desde el robot cada 100 ms a través de la conexión realizada. ArRobot provee el llamado SIP handler, lo que permite gestionar la recepción de estos paquetes de forma síncrona.

Desde la parte del cliente se pueden enviar directamente comandos o usar los diversos métodos y acciones de ARIA. Estas acciones y métodos se convierten posteriormente, de forma invisible al programador, en comandos directos.

Por otra parte, cada vez que una aplicación cliente basada en ARIA recibe un SIP ArRobot lleva a cabo una serie de cinco tareas automáticamente:

- Gestión del SIP.
- Interpretación de la información sensorial.
- Gestión y resolución de las acciones del cliente
- Examen del estado del sistema.
- Realización de las tareas programadas

Además el programador puede añadir más tareas a las anteriormente citadas.

Comandos y acciones

La aplicación cliente puede controlar el robot a través de comandos directos, comandos de movimiento o a través de acciones.

Si se desea, es posible enviar comandos directos al robot a través de la clase ArRobot. Los comandos directos consisten en un byte, que constituye el número de comando a enviar. Puede estar seguido de uno o más argumentos.

El nivel inmediatamente superior a los comandos directos son los comandos de movimiento. La clase ArRobot incorpora métodos para enviar comandos explícitos de movimiento al robot. Mediante estos métodos se puede, por ejemplo, establecer la velocidad lineal y de rotación del robot, provocar su movimiento a una distancia determinada o detenerlo.

Por último se puede controlar el robot desde un nivel de abstracción superior. Es posible definir acciones o usar las ya construidas en Aria. Las acciones serían comportamientos o pautas que debe seguir el robot. Se deben establecer prioridades entre las diversas acciones que se han añadido al robot, pues durante la ejecución del programa pueden entrar en conflicto. Es decir, dependiendo del estado del robot y de la prioridad de las acciones, se deberá ejecutar una u otra acción en cada momento. Las clases que permiten la definición de acciones y la resolución de prioridades se llaman ArAction y ArResolver.

Las acciones (alto nivel) y los comandos (bajo nivel) pueden entrar en conflicto y provocar errores en el control. Entre los objetivos del proyecto no está el control inteligente del robot. Sin embargo, en futuros desarrollos el programador deberá ser cuidadoso y tener en cuenta este hecho.

Otras características

Aria es un potente y robusto paquete compuesto por más de cien clases programadas en C++. Por tanto, además de las características enunciadas anteriormente, posee otras muchas. Existen clases para el control de dispositivos como cámaras, láser o pinzas, clases con utilidades matemáticas o para el manejo de tiempos, etc. Se destacan las siguientes clases:

- ArKeyHandler y ArJoyHandler. Permiten el control del robot a través del teclado o de un joystick respectivamente.
- ArThread. Es una clase envolvente o wrapper de la libreria pthreads. Permite la creación de subprocesos o threads dentro de una aplicación.
- ArFunctor. Es una clase que permite la creación de punteros a funciones.

- ArSocket. Es una clase envolvente o wrapper de las librerías para el manejo de sockets.

5.3.3. Instalación de Aria

Los fabricantes del robot facilitan el paquete **ARIA-1.3-2.i386.rpm** para la instalación del software en las máquinas en las que se va a desarrollar la aplicación. Es necesario copiar el paquete en raíz y descomprimirlo. De este modo se crea el directorio `/usr/local/Aria`. Hay que compilar los archivos lo cual puede hacerse en un solo paso utilizando el comando "make everything". Una vez compilado con éxito queda instalado el paquete Aria y listo para su utilización.

5.3.4. Simulador del robot

El paquete Aria incluye un simulador llamado "SRIsim" cuya interfaz puede verse en la Figura 5.6 que permite simular la conexión al robot. De este modo se pueden realizar pruebas a la hora de ir desarrollando el software.

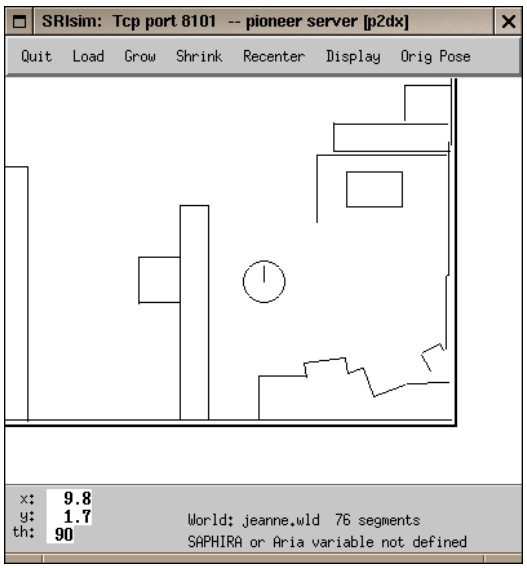

Figura 5.6: Simulador de Aria

Capítulo 6

Desarrollo

En este capítulo se recoge la evolución del presente proyecto fin de carrera para lograr el objetivo principal y dotar al sistema de un nuevo módulo de habla reutilizable. Desde el planteamiento del problema a solucionar hasta la obtención de resultados se establecieron distintas etapas progresivas que añadieran funcionalidades a lo ya obtenido o modificasen lo necesario para acercarnos a la consecución de nuestro objetivo.Todas ellas se recogen en las secciones siguientes.

6.1. Planteamiento del problema. Análisis de requisitos

Como ya se ha mencionado anteriormente,el objetivo final es desarrollar un nuevo módulo reutilizable que añada un nuevo componente al sistema ya existente.Este nuevo módulo constituirá una nueva interfaz entre el sistema y el operario, o posible interlocutor del sistema,posibilitando la comunicación hablada por parte del sistema.Para lograr nuestro propósito debemos conocer las limitaciones que nos afectan para poder así avanzar hacia nuestro fin.

El primer aspecto a tener en cuenta es la elección de una herramienta, siendo bastantes las alternativas disponibles como se ha visto en el Capítulo 3. El primer requisito que se considera es el de la utilización de una herramienta libre de generación de habla. La elección de esta característica no sólo se basa en la política interna del *ASLab* sobre utilización de software libre, sino en el ánimo de la proyectante de hacer este módulo accesible para toda persona dispuesta a incluirlo en su sistema. Este detalle supone la eliminación inmediata de gran parte de las posibilidades expuestas en el capítulo correspondiente a motores de generación de habla. Para establecer esta restricción se han considerado dos aspectos: las razones económicas y la posibilidad de

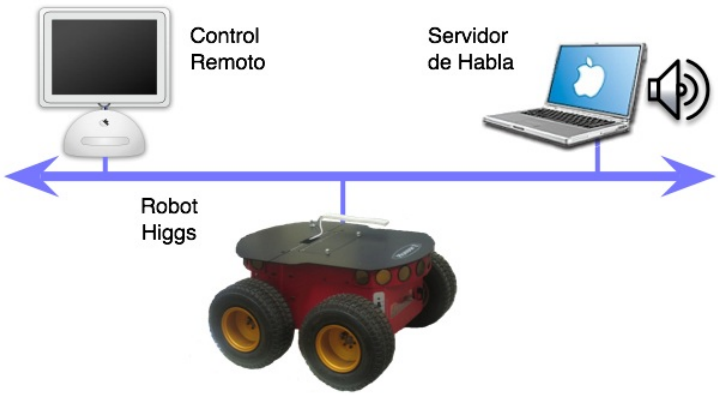

Figura 6.1: Objetivo del proyecto

poder utilizar este módulo en cualquier sistema con tecnología CORBA, ya sea de ámbito industrial como de uso personal, en cualquier red doméstica.

Dentro de las características de la herramienta a elegir se considera valorable la posibilidad de:

- configurarla de manera personalizada, poder modificar sus parámetros para conseguir unas prestaciones distintas en función de las necesidades,
- poder interactuar con otros programas,
- poder añadir voces de creación propia,
- una herramienta multilingüe, aunque como lenguaje por defecto se utiliza el inglés por tratarse del idioma más utilizado, y con mayor proyección, en el ámbito general de la tecnología y la industria,
- una herramienta capaz de adaptarse a entornos de vocabulario ilimitado, de forma que su utilización sea posible no sólo en ámbitos industriales sino también en los domésticos.

En el sistema actualmente se cuenta con los módulos descritos en el Capítulo 1, habiendo sido desarrollados todos ellos con la tecnología CORBA, descrita ya en el Capítulo 4, pero con distintos distribuidores: **ICa** es empleado en el módulo de comunicaciones y **mico** en el módulo *SOAR* , por ejemplo. Para que nuestro módulo pueda ser integrado dentro de este sistema debemos plantear como requisito la utilización, desde el principio del desarrollo, de esta tecnología para su realización, ya que, de esta forma,

la integración correcta está prácticamente garantizada: gracias a sus características de interoperabilidad, si nos ajustamos al estándar[1], nuestro módulo será totalmente portable independientemente del distribuidor elegido.

Otras decisiones que se deben tomar son las relativas al lenguaje de programación y el sistema operativo a utilizar. En cuanto al lenguaje de programación se utilizan C y C++ por ser los más familiares al proyectista y estar soportados, y ampliamente documentados, para su utilización con CORBA[M. Henning, 1999]. Se elige Linux como sistema operativo por disponer de una red operativa en el departamento de Automática y más concretamente en el ASLab, donde, además de disponer de un ordenador donde instalar todo el software necesario para el desarrollo, ya se encuentran instalados y disponibles varios servicios CORBA (entre ellos dos servidores de nombres, uno de ICa y otro de TAO). A pesar de estas elecciones cabe destacar el hecho de que otros lenguajes y sistemas operativos pueden ser utilizados gracias a las características de la tecnología empleada.

6.2. Elección de herramienta

Tras una primera selección buscando software libre y con el resto de las restricciones anteriores, de entre todas las herramientas disponibles la que más prestaciones aporta y cumple con todas las condiciones impuestas es *festival*, desarrollada por el CSTR (The Centre for Speech Technology Research) de la universidad de Edimburgo.

Festival es multilingüe: actualmente soporta inglés (británico, escocés y americano) y español, aunque el idioma más desarrollado es el inglés. Cuenta además con la ayuda de otros grupos para el desarrollo de nuevas voces en otros idiomas y su incorporación al sistema. Existe también la posibilidad de crear nuevas voces siguiendo las indicaciones y las herramientas que se encuentran disponibles a través del proyecto *FestVox* desarrollado por el grupo de habla de la universidad Carnegie Mellon. Está escrito en C++ y usa la librería de la universidad de Edimburgo llamada *Speech Tools* para la arquitectura de bajo nivel, y para el control un intérprete de comandos basado en Scheme(SIOD). Dispone de módulos independientes que pueden ser configurados externamente para seleccionar:

- conjunto de fonemas
- léxicos

[1] visitar la página web de la OMG para obtener las últimas modificaciones: http://www.omg.org

- reglas *letter-to-sound*
- tipo de normalizado de texto, pre-procesado o *tokenización*, para poder seccionar las distintas frases en pequeñas porciones o segmentos, también llamados *tokens*
- marcado de texto
- entonación y duración

Dispone además de varios sintetizadores de ondas:

- basados en *difonemas*: LPC (Linear Predictive Coding) de excitación residual
- soporte de base de datos MBROLA
- distribuidos bajo licencia gratuita del tipo X11, que permite su uso tanto comercial como no comercial sin ningún tipo de restricción.
- generación de módulos de estadísticas, *ngramas* (que permiten describir dependencias sintácticas locales como,por ejemplo, un modelo de Markov), CART (Classification And Regression Tree),
- soporte para JSAPI
- carga XML para relaciones

6.3. Instalación y pruebas con *festival*: parámetros e introducción de datos

6.3.1. Instalación

Una vez seleccionada *festival* como la herramienta a utilizar se realiza la instalación del software necesario a partir de los archivos comprimidos disponibles en la página web del CSTR: *http://www.cstr.ed.ac.uk/projects/festival/*.La versión a fecha de realización del proyecto corresponde al archivo *festival-1.95-beta.tar.gz*. Son también necesarios los archivos del paquete de librerías *speech-tools* que se encuentran disponibles en la misma página, siendo descargado el archivo *speech_tools-1.2.95-beta.tar.gz*. Además debemos descargar los archivos correspondientes a las voces que queramos para el sistema y los distintos léxicos que necesitemos en función de las voces. Ya que vamos a construir un sistema multilingüe elegimos los siguientes ficheros:

- *festvox_kedlpc8k.tar.gz*,voz masculina americana que usa bases de datos de excitación residual LPC,
- *festvox_ellpc11k.tar.gz*, voz masculina española que todavía no se encuentra desarrollada al completo pero que puede resultar adecuada por el momento,
- *festvox_cmu_us_slt_arctic_hts.tar.gz*, voz femenina americana basada en la voz llamada "slt" de CMU_ARTIC que usa el sintetizador basado en HTS (HHM-Based Speech Synthesis System) del instituto de tecnología de Nagoya,
- *festvox_rablpc8k.tar.gz*, voz británica masculina con pronunciación RP(Received Pronunciation [2])
- *festlex_CMU.tar.gz*, donde se encuentra el léxico necesario para las voces americanas,
- *festlex_OALD.tar.gz*, necesario para las voces británicas, se trata de una versión para ordenador del Oxford Advanced Learners'Dictictionary,
- *festlex_POSLEX.tar.gz*, incluye el léxico necesario para las voces inglesas tanto británicas como americanas.

Los únicos requisitos del sistema para la instalación son:

- una máquina Linux
- un compilador de C++
- GNU make
- hardware de audio

Al descomprimirlos se generará una nueva carpeta raíz con el nombre de *festival* y otra para *speech_tools*. Ya que *festival* se basa en ella, primeramente configuramos el sistema para *speech_tools* mediante consola siguiendo el procedimiento normal en Linux:

[2]pronunciación del inglés británico, originalmente basada en el habla de la clase alta del sudeste de Inglaterra, y característica del inglés hablado en los colegios y en las Universidades de Oxford y Cambridge. Hasta hace poco era el inglés estándar utilizado para las retransmisiones británicas.

```
cd speech_tools
./configure
make
make install
make test
```

Si existe un archivo de configuración para la distribución otra opción a `./configure` es:

```
cp config/config-dist config/config
chmod +w config/config
```

El archivo *config/config* es del que hereda la mayor parte de la configuración de *festival* por lo que puede modificarse aquí lo necesario para cada sistema además de en el correspondiente archivo del mismo *festival*(al que se refieren todos los "Makefiles"). El último comando nos informa de la posible falta de algún archivo o programa necesario. El mismo procedimiento se lleva a cabo para *festival*, siendo muy recomendable realizar el último test, ya que podemos detectar la falta de voces o si interfieren unas con otras durante el mismo.

Por último, para que ambos programas estén disponibles, es necesario añadir su localización en la variable de entorno `PATH` (puede hacerse de manera permanente modificando el archivo oculto *./bashrc* del directorio raiz del usuario)

```
export PATH=$PATH:/Proyecto/festival/bin
export PATH=$PATH:/Proyecto/speech_tools/bin/
```

6.3.2. Usando *Festival*

Para familiarizarse con el manejo y probar las distintas posibilidades que ofrece se llevan a cabo pequeños programas sencillos que leen texto introducido por pantalla, lo almacenan en un archivo y posteriormente hacen que *festival* lo lea; también se desarrollan scripts en Scheme para modificar las voces usadas y la velocidad utilizada. Estas primeras pruebas se realizan tanto de forma local como remota ya que entre las opciones disponibles destaca la de operar como servidor, escuchando peticiones de clientes a través del puerto por defecto (1314). Además con la distribución se incluyen un cliente y un servidor tipo que pueden utilizarse directamente o como base para desarrollar unos propios.

Existen distintas formas de utilizar esta herramienta: con el intérprete de comandos, mediante la línea de comandos sin entrar en el intérprete, en modo servidor, aprovechando el cliente y servidor suministrados con la distribución o creando unos propios.

```
cris@cris:~$ festival
[1] 23850
cris@cris:~$ Festival Speech Synthesis System 1.95:
 beta July 2004
Copyright (C) University of Edinburgh, 1996-2004.
 All rights reserved.
For details type `(festival_warranty)'
festival>3
3
festival> (+ 1 2)
3
festival> (SayText "hello" nil)
#<Utterance 0xb6f77e18>
festival>
```

Al lanzar la aplicación nos encontramos con este intérprete de comandos que espera que le introduzcamos órdenes escritas en una variedad de Scheme: SIOD (Scheme In One Defun). Se trata de un lenguaje de programación derivado de LISP, diseñado para ser consistente y que fue elegido como lenguaje básico para realizar los scripts de *festival* porque:

- es un lenguaje fácil de analizar e interpretar por las máquinas,
- ofrece la posibilidad de recoger "basura"por lo que el manejo de objetos es fácil y seguro,
- ofrece una estructura de datos consistente para representar parámetros,reglas etc.,
- es un lenguaje con el que los autores se encuentran familiarizados,
- es adecuado para su uso en sistemas embebidos.

La estructura fundamental en este lenguaje es la *s-expression*. Se trata de un átomo, o una lista de *s-expressions* representados entre paréntesis y que se evalúan recursivamente[3]

[3]Para mas información sobre este tema se recomienda visitar http://www.swiss.ai.mit.edu/projects/scheme/ , o consultar [E. Charniak, 1985]

```
3
(1 2 3)
(a (b c) d)
((a b) (c d))
(+ 1 2)
```

Sin embargo este tipo de interfaz no resulta adecuada a nuestras necesidades, pues requiere un aprendizaje previo por parte del usuario final de las distintas órdenes y comandos para lograr sintetizar frases.

Utilizando la línea de comandos es posible indicar distintos comportamientos, por ejemplo, con llamadas de la forma:

1) `festival --language spanish --tts fichero.txt`, con ella se produce la lectura del contenido del archivo de texto que se introduce como parámetro en el lenguaje indicado. El inconveniente es que para configurar la herramienta, el usuario final necesita localizar y modificar en el directorio de librerías el archivo *init.scm* que se carga al iniciar el programa y que debe estar escrito en Scheme,

2) `festival fichero.scm`, con esta segunda se inicia el intérprete de comandos de *festival* cargando la configuración inicial escrita en Scheme contenida en el fichero indicado,

3) `festival -b fichero.scm`, con esta llamada se realiza la misma operación que en la anterior pero en modo no interactivo o por lotes,

4) `festival_client --server 138.100.76.208 --prolog conf.scm --asyn --ttw --aucommand 'na_play $FILE'fichero.txt` con la cuarta llamada utilizamos un programa cliente que viene en la distribución indicando en la línea de comandos la IP del servidor que debe atender la llamada de manera asíncrona , con la configuración inicial contenida en el archivo *conf.scm* e indicando por último el archivo de texto que debe leerse.Se recurre en este caso a la utilización de una función de *speech_tools* llamada *na_play*.

Son muchas, por tanto, las formas de conseguir sintetizar voz, pero para configurar la herramienta es necesario tener conocimientos previos. Por esta razón se piensa en desarrollar un servidor que realice esta función por el usuario final, realizando desde dentro una llamada a festival. Éste, a su vez, también estará actuando como servidor a la espera de recibir peticiones por parte de clientes con lo que se disminuye el tiempo necesario para responder

pues no es necesario inicializar las bases de datos nada más que al lanzar festival como servidor. Para garantizar la seguridad en el acceso se recurre a establecer una lista restringida de clientes autorizados no estableciéndose en principio la necesidad de contraseña.

6.4. Creación de objetos CORBA

Para la implementación del servidor y el cliente introducimos la tecnología CORBA basándonos en las indicaciones recogidas en [M. Henning, 1999] y eligiendo ***omni*** como la distribución a utilizar. Las razones para esta elección se basan en la disponibilidad de una buena documentación que permite un desarrollo sencillo y, por otro lado, también nos permite demostrar la interoperabilidad entre orbs, ya que el resto de módulos utilizan ***Ica***, reforzando así la apuesta por esta metodología estándar.

Tras la descarga[4] e instalación de *omniORB-4.0.6*, nos dedicamos al diseño del idl para nuestro servidor de habla.

```
#ifndef _TALKINGMACHINE_IDL_
#define _TALKINGMACHINE_IDL_

module TalkingMachine {
// tipos para los perfiles
enum pnum { p1, p2, p3, p4 };
enum idioma { spanish, english};
enum sex { male, female} ;
enum enation { uk, us};

// estructura para los perfiles de usario
struct Profile {
pnum profnum; // profile_number
idioma language; // idioma
sex gender; // sexo
enation nationality; // uk-us english
float volume;
float speed;
};

interface Talk {
```

[4]disponible a través de *http://sourceforge.net/*

```
void read_it (in string textin);
void set_profile (in Profile perfil);

boolean disable_alerts();
boolean enable_alerts();

void set_volume (in float vol_level);
void set_speed (in float speed_level);
};
};

#endif //_TALKINGMACHINE_IDL_
```

Con la intención de hacer transparente al usuario todo aquello relacionado con comandos de *festival* se definen dentro de esta interfaz las siguientes funciones:

- `read_it` esta función recoge como argumento el texto que debe pronunciarse,
- `set_profile` se usa para cambiar las distintas caracteristicas de las voces disponibles, los datos se introducen en forma de una nueva estructura, definida por nosotros, en la que existen los siguientes campos: número de perfil, lenguaje (puede ser inglés o español por el momento), género de la voz (masculino o femenino), nacionalidad (para distinguir en el caso de voces inglesas), volumen y velocidad.
- `disable_alerts` permite inhabilitar la emisión de posibles alarmas acústicas,
- `enable_alerts` habilita de nuevo las alarmas,
- `set_volume` establece el volumen a utilizar,
- `set_speed` establece la velocidad con la que debe reproducirse la voz.

6.4.1. Creación del servidor de habla

Como hemos mencionado anteriormente, el desarrollo ha sido incremental, primeramente se ha desarrollado un servidor de habla básico que no utiliza el *servidor de nombres*, y una vez que se ha probado su funcionamiento con un cliente, también básico, se ha añadido esa funcionalidad que permite un uso

mucho más accesible de cualquier módulo CORBA: sin necesidad de conocer el IOR del módulo, simplemente con su idl, podemos pedir al *Naming Service* que lo localice por nosotros (siempre y cuando se haya registrado previamente en él claro).

Los distintos pasos a seguir por tanto son:

1. inicializar el ORB:

```
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
```

2. obtener el adaptador de objetos RootPOA:

```
CORBA::Object_var obj=
  orb->resolve_initial_references("RootPOA");
PortableServer::POA_var root_poa=
  PortableServer::POA::_narrow(obj);
```

3. crear un servant del servidor:

```
Talk_impl* mi_robot= new Talk_impl();
```

4. activar el objeto en el RootPOA:

```
PortableServer::ObjectId_var oid =
  root_poa->activate_object(mi_robot);
```

5. activamos el POA Manager:

```
root_poa->the_POAManager()->activate();
```

6. en el caso de utilizar el *Naming Service* obtenemos la referencia del objeto y lo registramos en el servidor de nombres utilizando la función*bindObjectToName*, y se comprueba el éxito de la operación:

```
obj=mi_robot->_this();
if( !bindObjectToName(orb, obj) )
       return 1;
```

7. quedamos a la escucha de peticiones:

```
orb->run();
orb->destroy();
```

En la función `bindObjectToName` se reciben como parámetros las referencias del *ORB* y del objeto, se resuelve la referencia al *Naming Service* y se registra en él el objeto con el nombre que más adelante tendrá que utilizar todo cliente que quiera ser atendido por el servidor, en este caso `speaker/server` , siendo `speaker` el contexto en el que se incluye el objeto:

```
CosNaming::NamingContext_var inc;
CORBA::Object_var obj2;

obj2=orb->resolve_initial_references("NameService");
inc=CosNaming::NamingContext::_narrow(obj2);

   CosNaming::Name name;
   name.length(1);
   name[0].id=CORBA::string_dup("speaker");
   name[0].kind=CORBA::string_dup("");
   CosNaming::NamingContext_var speakerContext;

   speakerContext=inc->bind_new_context(name);

CosNaming::Name objectName;
     objectName.length(1);
     objectName[0].id=CORBA::string_dup("server");
     objectName[0].kind=CORBA::string_dup("");

      speakerContext->bind(objectName, obj);
      speakerContext->rebind(objectName, obj);
```

Uso de omniNames

El servidor de nombres de **omni**:*omniNames* necesita que se le indique el puerto donde debe esperar peticiones en el momento en que se inicia por primera vez a través de la opción `-start` (como veremos después, para simplificar su uso con corbaname URIs , se inicia por defecto en el puerto 2809 y con la clave `NameService` en el contexto raíz). A partir de ese momento el puerto a escuchar se determina en función del contenido de los archivos de registro que se realizan. Cuando se inicia de manera correcta devuelve, a través de la salida estándard de errores, el IOR de su contexto raíz.

OmniORB 4.0 soporta el INS (Interoperable Naming Service) que permite, entre otras características, configurar de una manera portable

`ORB::resolve_initial_references()` a través de dos argumentos estándard en la línea de comandos. Uno de ellos es `ORBInitRef` (debe escribirse con un guión al introducirlo como parámetro en la línea de comandos del terminal), que nos permite resolver la referencia al *Naming Service* en la llamada a nuestras aplicaciones de cualquiera de estas formas:

```
./aplicacion -ORBInitRef NameService=IOR:0083717...
./aplicacion -ORBInitRef NameService=corbaname::host
```

Además del formato correspondiente a las cadenas de caracteres que forman el IOR, omniORB 4.0 soporta recibir URI (Uniform Resource Identifier) en dos formatos distintos. Estos formatos tienen multiples opciones pero la que se considera más sencilla para su uso en el proyecto para indicar la situación del servidor de nombres es la recogida en el ejemplo anterior, y por tanto será la utilizada de ahora en adelante.

6.4.2. Primer prototipo de cliente

Para realizar las primeras pruebas se diseña un cliente que utiliza un diálogo gráfico en el que se introduce el texto a sintetizar. Para la creación del diálogo se ha utilizado **Qt**.

Qt

La biblioteca Qt (o simplemente Qt) es una herramienta de programación para desarrollar interfaces gráficas de usuario. Una interfaz gráfica de usuario (GUI) es un método para facilitar la interacción del usuario con el ordenador a través de la utilización de un conjunto de imágenes y objetos (como iconos o ventanas) además de texto. Surge como evolución de la línea de comandos de los primeros sistemas operativos y es pieza fundamental en un entorno gráfico.

Inicialmente Qt apareció como biblioteca desarrollada por Trolltech[5] en 1992 siguiendo un desarrollo basado en el código abierto, pero no libre. Se usó activamente en el desarrollo del escritorio KDE (entre 1996 y 1998), con un notable éxito y rápida expansión. Esto fomentó el uso de Qt en programas comerciales para el escritorio, situación vista por el proyecto GNU como amenaza para el software libre.

Para contrarrestar ésta se plantearon dos ambiciosas iniciativas: por un lado el equipo de GNU en 1997 inició el desarrollo del entorno de escritorio GNOME con [GTK+] para GNU/Linux. Por otro lado se intenta construir una biblioteca compatible con Qt pero totalmente libre, llamada Harmony.

[5] www.trolltech.com

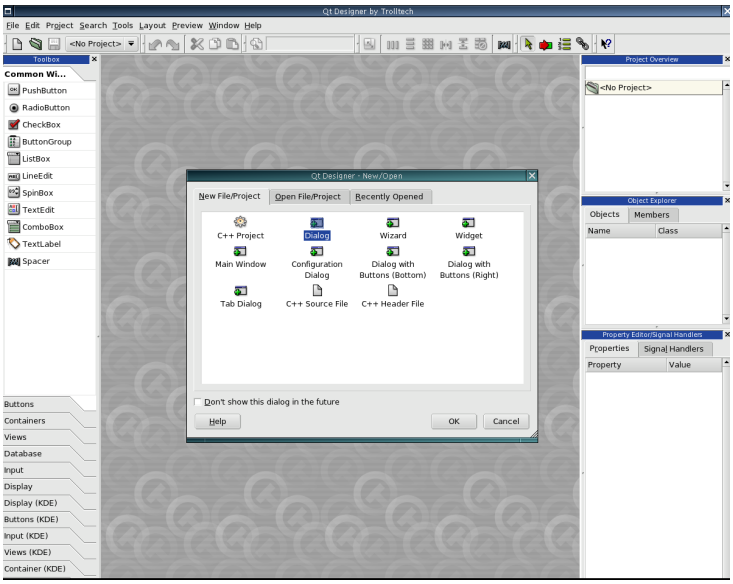

Figura 6.2: Herramienta para Qt: Designer

Qt cuenta actualmente con un sistema de doble licencia: por un lado dispone de una licencia GPL para el desarrollo de software de código abierto (open source) y software libre gratuito y por otro una licencia de pago para el desarrollo de aplicaciones comerciales.

Designer

Como ayuda para el diseño de la interfaz gráfica con Qt se utiliza la herramienta "Designer"[6] (Figura 6.2).

"Designer" nos permite la creación de diálogos interactivos con distintos objetos (barras, displays ...). Para estos elementos se definen sus propiedades tales como el nombre, tamaño, tipo de letra, etc. Estos son clases dentro de la aplicación que estamos diseñando. Posteriormente será necesario implementar la funcionalidad mediante el código que se debe ejecutar cuando se recibe el evento correspondiente.

Resultado del trabajo con "Designer" se obtiene un archivo de extensión ".ui". Utilizando el compilador de ui (ui compiler) para generar la descripción de la clase ".h" y su implementación ".cpp"(*) (implementación de la clase). Como la parte de Qt emplea las palabras clave "SLOT" y "SIGNAL" que no significan nada para C++, es necesario pasar el ".h" por el compilador

[6]http://www.trolltech.com/products/qt/designer.html

de objetos de Qt MOC (Meta Object Compiler) que se encarga de crear "glue code"(*) de modo que se genera código C++ correspondiente a los mecanismos de comunicación "SIGNAL-SLOT" (una especie de mapeo o traducción a C++).

Es necesario implementar el comportamiento de los distintos elementos del diálogo, es decir, indicar qué tipo de acción realizarán cuando se produzca un evento. Los archivos ".cpp" descritos anteriormente son creados directamente a partir del ".ui" generado por el "Designer" por lo que no conviene tocarlos, ya que cualquier cambio que se haga en ellos se perderá al modificar la interfaz.

El proceso seguido para la implementación de la funcionalidad característica de cada elemento de la interfaz diseñada es crear una clase que herede de la clase generada en la implementación del ui, de modo que en esta nueva clase se definan los métodos virtuales de la clase base.

Una vez esté implementada la funcionalidad es necesario pasar el ".h"(*) por el MOC porque aparecen de nuevo mecanismos de "SIGNAL-SLOT" y es necesario generar el "glue code"(*) para que el compilador de C++ pueda entenderlo.

Para generar el binario es necesario escribir la aplicación principal(*), en la que se instancia la clase derivada en un entorno de aplicación de Qt.

Es momento de reunir los archivos marcados con (*), compilarlos para generar los ficheros de código objeto, enlazar estos con las librerias de Qt y obtener de este modo el binario de la aplicación.

Interfaz

La interfaz diseñada para la aplicación es la que se muestra en la Figura 6.3.En ella se recoge el texto a sintetizar a través de una ventana, mediante una lista desplegable se puede seleccionar la voz deseada y moviendo los cursores de las barra inferiores regular la velocidad y el volumen.

6.4.3. Cliente definitivo

Una vez conseguida esta primera interfaz, se avanza un poco más intentando crear un cliente para la plataforma Pioneer2AT-8, también llamada Higgs. Este nuevo cliente debe ser capaz de comunicarse con él y solicitar datos relativos a su estado y posición. Inicialmente sólo se considera la petición del nivel de batería, para que, si se encuentra por debajo de un cierto nivel dé una "voz de alarma". Nos encontramos por tanto con un sistema distribuido propiamente dicho: por un lado tenemos el **robot móvil** que lleva en su interior un servidor que llamaremos **higgsOpServer** y también el

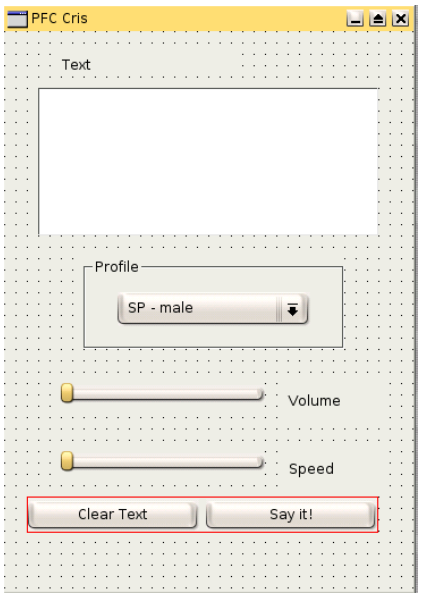

Figura 6.3: Diálogo cliente.

cliente (de ambos servidores) , y por otro un **ordenador personal** en el que se encuentra instalado de manera local la herramienta *festival* y donde también tenemos el **servidor de habla**.

Para la realización del programa cliente debe ser conocida la interfaz que proporciona el servidor de la plataforma para poder hacer peticiones, debido a que utilizamos invocación estática. El proyecto "Sistema de Comunicaciones para Pioneer 2AT-8"[Pareja, 2004] diseñó una aplicación cliente/servidor basada en una interfaz que ha sido modificada posteriormente para adecuarse a las necesidades de las aplicaciones que se desarrollan en el grupo. La siguiente interfaz idl se utiliza para la aplicación:

```
module Pioneer {

//////////////////////////// DATA TYPES

typedef  sequence<long>   SonarRangeSeq;
typedef  sequence<boolean>  SonarNewReadingSeq;

//////////////////////////// DATA STRUCTURES
```

```
struct TimeStamp{
    long   secs;
    long   usecs;
};

struct Pioneer2AT_State {
    boolean     motorsEnabled;
boolean     sonarsEnabled;
    boolean     emergencyStop;
    boolean     connected;
    boolean     running;
    boolean     moving;
    boolean     leftMotorStalled;
    boolean     rightMotorStalled;
    boolean     leftBreakTriggered;
    boolean     rightBreakTriggered;
    float      battery;
    boolean     newReadings;
    TimeStamp time_Stamp;
};

struct Pioneer2AT_Movement {
    float      speed;
    float      leftSpeed;
    float      rightSpeed;
    float      rotSpeed;
    float      maxSpeed;
    float      maxRotSpeed;
    float      absMaxSpeed;
    float      absMaxRotSpeed;
    float      accel;
    float      rotAccel;
    boolean     headingDone;
    boolean     movementDone;
    TimeStamp   time_Stamp;
};

struct Pioneer2AT_Position {
    float     x;
    float     y;
    float     t;
```

```
    float     heading;
    float     distanceDiff;
    float     headingDiff;
    TimeStamp   time_Stamp;
};

struct Pioneer2AT_Sensing {
    boolean             hasSonars;
    boolean             hasBumpers;
    unsigned short      numSonars;
    unsigned short      numFBumpers;
    unsigned short      numRBumpers;
    unsigned short      closestSonar;
    unsigned long       closestRange;
    SonarRangeSeq       SonarRanges;
    SonarNewReadingSeq  SonarNews;
    TimeStamp   time_Stamp;
};


////////////// INTERFACE

interface Pioneer2AT{

    // Data exchanging methods -----------------
    Pioneer2AT_State    getRobotState    ();
    Pioneer2AT_Movement getRobotMovement ();
    Pioneer2AT_Position getRobotPosition ();

    void getRobotSensing (out Pioneer2AT_Sensing sensing );

    // Robot fast state checking ----------------------
    boolean isReady     ();
    boolean isConnected ();
    boolean isRunning   ();
    boolean isStalled   ();
    boolean isEmergency ();
    boolean isMoving    ();
    boolean isEnabled   ();
    boolean isBreaked   ();
    float  getBattery  ();
```

```
TimeStamp getTime();
void setTimeToNow();

    // CORBA object remote control -----------------
    void            init            ();
    long            connect         ();
    void            start           ();
    void            stop            ();
    void            disconnect      ();
    void            finish          ();
    void            setCycleTime    (in unsigned long ms);
    unsigned long   getCycleTime    ();
    unsigned long   getRefreshTime  ();
    void            setRefreshTime  (in unsigned long ms);

    // Movement information ------------------------------
    float  getVelocity         ();
    float  getMaxVelocity      ();
    float  getAbsMaxVelocity   ();
    float  getLeftVelocity     ();
    float  getRightVelocity    ();
    float  getAcceleration     ();
    float  getRotVelocity      ();
    float  getMaxRotVelocity   ();
    float  getAbsMaxRotVelocity();
    float  getRotAcceleration  ();
    boolean isHeadingDone       ();
    boolean isMovementDone      ();
    float  getHeading          ();
    float  getHeadingDiff      ();
    float  getDistanceDiff     ();

    // Movement control ----------------------------------
    void    enableMotors        ();
    void    disableMotors       ();
    boolean areMotorsEnabled    ();
    void    setVelocity         ( in float vel);
    boolean setMaxVelocity      ( in float maxVel);
    boolean setAbsMaxVelocity   ( in float maxVel);
    void    setLRVelocity       ( in float leftVel, in float rightVel);
    void    setAcceleration     ( in float accel);
```

```
    void    setRotVelocity      ( in float rotVel);
    boolean setMaxRotVelocity   ( in float maxRotVel);
    boolean setAbsMaxRotVelocity( in float maxRotVel);
    void    setRotAcceleration  ( in float rotAccel);
    void    moveDistance        ( in float distance);
    void    setHeading          ( in float heading);
    void    setDeltaHeading     ( in float delta);

    // Position information ----------------------------------
    float getX             ();
    float getY             ();
    float getTh            ();
    float getCompass       ();

    // Sensing -----------------------------------------------
    boolean         hasSonars           ();
    boolean         areSonarsEnabled    ();
    void            enableSonars        ();
    void            disableSonars       ();
    boolean         hasFrontBumpers     ();
    boolean         hasRearBumpers      ();
    unsigned short  getNumSonars        ();
    unsigned short  getNumFrontBumpers  ();
    unsigned short  getNumRearBumpers   ();
    unsigned short  getClosestSonar     ();
    unsigned long   getClosestSonarRange();
    unsigned long   getSonarRange  (in unsigned short numSonar);
    boolean         hasSonarNewReadings (in unsigned short numSonar);

};//INTERFACE

};//MODULE
```

Para cumplir con nuestro objetivo se recurre al procedimiento `getBattery()` de la `interface Pioneer2AT`, que devuelve el voltaje actual de la batería del robot(campo `battery` de la estructura `Pioneer2AT_State`) . Se ha diseñado una función, `getObjectReference`, que, de manera análoga a la función `bindObjectToName` del servidor, llama al *Naming Service* en busca de la referencia al objeto `PIONEER` y al objeto `speaker/server`.

6.5. Validación

Para comprobar el funcionamiento de nuestros programas existen dos posibilidades, conectarse físicamente al robot móvil o utilizar un simulador, llamado SRIsim, proporcionado por los fabricantes de la plataforma y que se encuentra incluido en la distribución de *Aria*.En el caso del simulador el servidor de nombres utilizado es el de **omni**, mientras que en el caso de la conexión con el robot se puede realizar la comprobación usando dos servidores de nombres distintos, el de **omni** instalado localmente en el ordenador personal de nombre **simak**, y el que se encuentra ya instalado en el AsLab y que corresponde a **ICa**. Realizar la validación con los dos sería lo ideal para comprobar la interoperabilidad entre ambos.

En cualquiera de los dos casos existe una secuencia de actuación para las llamadas a los distintos programas implicados en las pruebas:

1. se lanza el *servidor de nombres* si no se encontraba ya disponible,
2. iniciamos bien el simulador SRIsim, o bien se conecta el robot, en cuyo caso automáticamente iniciará el servidor higgs-op-server para el suministro de datos,
3. en el caso del simulador, se inicia el servidor higgs-op-server,
4. se inicia el servidor de habla[7]
5. se inicia el cliente.

6.5.1. Comprobación en el simulador SRIsim

El problema que se plantea al utilizar el simulador es que los movimientos del robot simulado no se pueden controlar ya que el cliente diseñado sólo le pide la información, no lo teleopera en modo alguno, por lo que el simulador no sirve para probar las distintas situaciones en las que se pueda encontrar, entre otras cosas porque su nivel de batería siempre es el máximo. Esta situación hace necesaria la creación de otro cliente que mueva a Higgs. Este nuevo cliente utiliza el servidor de voz para anunciar las acciones y movimientos del robot, desde el momento de su activación hasta su desconexión tras realizar una pequeña traslación y un giro sobre su eje.

[7]el orden de este último respecto al otro servidor puede variar, pero siempre debe realizarse después de iniciar el servidor de nombres y antes de lanzar el cliente.

Durante la validación con el simulador se usa el servidor de nombres de **omni** que se encuentra instalado localmente.Como ya se ha visto anteriormente para iniciar el servidor de nombres por primera vez basta con añadir en la línea de comandos `omniNames -start`. Aparecerá por pantalla el IOR asociado al *Naming Service* y una frase indicando que se ha realizado un archivo de registro. Durante el inicio de la aplicación se lleva a cabo la creación de este archivo, en el directorio indicado en la variable de entorno OMNINAMES_LOGDIR, en el que se recoge en primer lugar el número de puerto por el que "escucha" el servidor de nombres (siendo por defecto el 2809) y a continuación los distintos accesos de clientes: se añaden los IOR de los distintos nombres registrados en el servidor de nombres, indicando si se tratan de objetos, o bien contextos, y también se registra la destrucción de los vínculos establecidos a los distintos nombres.Se recoge, de esta forma, todo cambio sufrido por el gráfico de nombres. Esta forma de almacenar los datos de los clientes nos sirve de ayuda en el caso de que, por cualquier causa, el servidor de nombres deje de funcionar y sea necesario reiniciarlo. Al volver a funcionar, `omniNames` busca este archivo y lo lee de forma que puede volver a registrar los mismos nombres con los mismos IORs que tenían antes del fallo, con lo que las llamadas de clientes no se pierden. Este archivo además es revisado, con una periodicidad configurable, permitiendo la eliminación de los vínculos inútiles.

Para las llamadas a las aplicaciones es necesario tener en cuenta la distribución con la que se han desarrollado, existiendo diferencias a la hora de introducir la referencia del *Naming Service* en el caso de ser uno u otro. Esto se debe a que cada distribuidor intenta introducir algo específico suyo que lo diferencie del resto, siendo éste uno de los posibles problemas que pueden surgir a la hora de interconectar ORBs distintos.

Para iniciar el servidor `higgs-op-server` hay que introducir en la línea de comandos un argumento, con el IOR del *Naming Service*, para configurar `ORB::resolve_initial_references()` adecuadamente, mediante la opción `-ORBInitRef` :

```
./higgs-op-server -ORBInitRef IOR=IOR:010000002b000000...
```

o bien sacándolo del archivo donde se encuentre recogido mediante un lanzador de la forma:

```
./higgs-op-server -ORBInitRef IOR=‘cat NSior_file‘ &
```

Para el servidor de habla, llamado genéricamente *robot*, se puede recurrir a cualquiera de las formas indicadas anteriormente en la sección 6.4.1 pero, ya que nos encontramos trabajando de forma local, se elige:

```
./robot -ORBInitRef NameService=corbaname::simak &
```

Por último, en el caso del cliente hay que hacer una pequeña modificación en la línea de comandos y añadir una opción para cambiar la forma en que omniORB usa GIOP. Este cambio habilita el modo *menor denominador común*, (Lowest Common Denominator) que inhabilita diversas características de IIOP y GIOP, que se sabe dan problemas a la hora de tratar con otros ORBs y que, desgraciadamente, era nuestro caso con **ICa**.

```
./cliente -ORBlcdMode 1 -ORBInitRef NameService=corbaname::simak &
```

Recurriendo a las herramientas, distribuidas conjuntamente con `omniORB-4.0.6`, denominadas `nameclt` y `catior` se comprueba la correcta inscripción de los distintos servidores en el *Name Service*. `Nameclt` recibe como argumento el IOR del *Naming Service* y mediante la operación `list context-name` realiza un listado de los contextos y objetos con el nombre especificado. `Catior` permite ver los componentes del IOR que se le pasa como argumento y que podemos conseguir, para cada servidor, de los archivos de registro de `omniNames`. Realizada esta comprobación se puede decir que: el proceso se realizó con éxito, tanto en su conexión a través del servidor de nombres, como en la obtención final de frases audibles indicando las acciones realizadas por el robot simulado.

6.5.2. Higgs habla

En la última etapa de nuestro desarrollo se lleva a cabo la última prueba, esta vez con el Pioneer2AT-8. Recientemente se ha incorporado un **Servidor de tiempo real** al sistema gracias a otro proyecto fin de carrera desarrollado por Adolfo Hernando [Hernando, 2005]. En él se ha mejorado el servidor disponible añadiéndole características de tiempo real. Para ello ha sido necesaria la incorporación de un nuevo sistema operativo que implemente características de tiempo real blando con las restricciones impuestas por las comunicaciones inalámbricas del hardware existente, de forma que pueda controlar el robot móvil Pioneer2AT-8 que constituye la plataforma de pruebas común a todos los proyectos. Este nuevo servidor ha permitido el poder suministrar servicios a varios clientes a la vez, lo que ha hecho posible el teleoperar el robot a la vez que se le pedían datos a través de nuestro programa cliente.

El planteamiento es el mismo que en la etapa anterior de comprobación con el simulador. La única diferencia es que en este caso el sistema es realmente distribuido y varía la utilización del servidor de nombres: Higgs tiene por defecto asignada la utilización de un servidor de nombres en el arranque del servidor, cuyo IOR se encuentra en el robot en el archivo

`root/servidor/servidor.sh` y que corresponde al *Naming Service* de **ICa** que se encuentra disponible en el ASLab. Se pretende realizar la comprobación del software desarrollado utilizando cualquiera de los dos servidores de nombres distintos, demostrando de esta forma la interoperabilidad característica de CORBA.

Se utiliza el servidor de nombres de **omni** por lo que es necesaria la modificación del mencionado archivo `servidor.sh` introduciendo el IOR de `omniNames` que actualmente está corriendo en el ordenador personal llamado *simak* . Las llamadas a las aplicaciones serán por tanto las mismas que en la sección anterior, salvo en el caso del servidor `higgs-op-server` que se inicia de manera remota, una vez modificado el archivo con el IOR, mediante las llamadas:

```
ssh root@138.100.76.247
Password:
/etc/init.d/higgs-srv stop
/etc/init.d/higgs-srv start
```

Como resultado se escuchan las frases programadas para indicar el comienzo de actividad del robot, su movimiento y su desactivación posterior, consiguiendo así el objetivo propuesto.

Después de comprobar el correcto funcionamiento se procede a modificar de nuevo el archivo `root/servidor/servidor.sh` para incluir el IOR original de **ICa**. En caso de querer validar nuestro sistema con él, sería necesario repetir la secuencia anterior de órdenes para habilitar de nuevo el servidor de datos del robot. En este caso también es necesario cambiar las llamadas del resto de aplicaciones en cuanto a la forma de introducir el IOR del *Naming service* a la forma `IOR=IOR:00003641741....`

Capítulo 7

Conclusiones y líneas futuras

En este último capítulo se recogen las conclusiones obtenidas del proyecto fin de carrera que se describe en esta memoria. En una primera parte se recuerdan los objetivos y el trabajo realizado, y en la segunda se describen las líneas de trabajo que quedan abiertas tras la conclusión de este proyecto, que supondrían mejoras de la aplicación desarrollada, y posibles líneas de investigación que podrían seguirse en el futuro.

7.1. Conclusiones

El objetivo de este proyecto es el de dotar a un sistema distribuido de una nueva interfaz que permita al usuario recibir información de una manera más adecuada a sus necesidades, centrándose en la comunicación verbal como herramienta y consiguiendo de esta forma mejorar la relación hombre-máquina. Para llevarlo a cabo se desarrolla un módulo reutilizable utilizando la tecnología CORBA que, por sus características intrínsecas, permite una integración satisfactoria dentro del sistema en el que se engloba el proyecto.

Se divide la evolución del proyecto en diversas etapas que, progresivamente, van añadiendo funcionalidad hasta la creación de una aplicación CORBA que actúe como servidor de habla para el sistema. Para comprobar su potencial se implementan también unas aplicaciones cliente. La primera establece una interfaz gráfica para la síntesis de texto, mientras que la segunda actúa como cliente solicitando datos a otro servidor CORBA, ya disponible con anterioridad en el sistema, y en función de ellos realiza peticiones al servidor de habla obteniendo frases audibles que indican el éxito obtenido.

Nos encontramos, por tanto, con un nuevo módulo cuya principal misión es la de actuar como "mensajero" entre dos entidades bien distintas y que sin embargo pueden encontrar un punto de unión a través de la comunicación. La finalidad es una implantación no sólo en el ámbito industrial sino también en el ámbito doméstico ya que el software utilizado se encuentra disponible de manera gratuita en Internet.

7.2. Líneas futuras

Como posibles mejoras a desarrollar para nuestro servidor se barajan varias opciones:

- Gracias a la elección de *festival* como herramienta para la síntesis de voz, se pueden crear **nuevas voces** para añadir al sistema sin grandes problemas. Para ello es necesario seguir las indicaciones recogidas en [Alan W. Black, 2003], o a través de la red en:

  http://festvox.org/festvox/festvox_toc.html,

  en donde se explica paso a paso el proceso a seguir, desde los requisitos necesarios hasta la integración final, pasando, entre otros temas, por la construcción de reglas *letter-to-sound* o de modelos prosódicos, y dando las bases para el lenguaje de programación de *festival*: Scheme.

- Otra posibilidad es la utilización de lenguajes de marcado, como por ejemplo **SABLE**, para poder establecer, entre otras características, el volumen, el género de la voz a utilizar, las pausas prosódicas deseadas (con el fin de enfatizar algún detalle), la enfatización, la velocidad, etc

- Otra posible línea de investigación con los módulos disponibles es la integración del servidor de habla con el módulo que mapea el estado del sistema mediante caras. Una posible sincronización entre ambos, moviendo los labios de forma adecuada mientras se sintetizan sonidos supondría un gran avance en el objetivo de este proyecto de hacer la comunicación hombre-máquina más amigable, ya que por factores psicológicos una **cabeza parlante** predispondría al usuario a una mayor confianza respecto a su trato con el sistema.

- Lograr una comunicación hablada bidireccional con los posibles múltiples usuarios del sistema distribuido de control de Higgs.

Dentro del ASLab se encuentran actualmente en desarrollo distintos proyectos fin de carrera que pueden entrar en relación directa con nuestro módulo de habla. Entre ellos podemos mencionar un **sistema de reconocimiento de voz** cuyo objetivo es la creación de un módulo capaz de identificar a su interlocutor dentro de un grupo conocido de personas, y otro de **reconomiento visual** capaz de identificar personas cuyas fotos se encuentren en una base de datos. Ambos módulos podrán recurrir al módulo de habla para responder ante las situaciones de reconocimiento de la persona en cuestión llamándola por su nombre propio, por ejemplo.

Por último cabe destacar que la posibilidad de poder incorporar este módulo de habla a cualquier red doméstica (o industrial) de ordenadores abre un gran abanico de posibilidades:

- para servicios de telecomunicaciones,
- como intermediario para eventos (como el anuncio de la llegada de nuevo correo electrónico),
- sistemas de ayuda para invidentes,
- ayuda en el aprendizaje de nuevos idiomas,
- aplicaciones en la industria del juguete para la obtención de juguetes "parlanchines",
- como ayuda para la corrección de disfunciones en el habla,
- a nivel educativo,
- etc.

Apéndice A

Otras herramientas utilizadas

Además de las herramientas descritas el los capítulos anteriores, en la realización de este proyecto y su correspondiente memoria se han utilizado otras. A continuación se muestra una lista de las mismas así como en qué se empleó cada una de ellas.

Eclipse: la plataforma Eclipse está diseñada para la construcción de entornos de desarrollo que puedan ser utilizados para la construcción de aplicaciones web, aplicaciones Java de todo tipo, programas C++, y Enterprise JavaBeans (EJBs). En este proyecto se ha utilizado Eclipse para realizar los distintos programas en C++ desarrollados.

LaTeX: es un conjunto de macros de TeX. La idea principal de LaTeX es ayudar a quien escribe un documento a centrarse en el contenido más que en la forma. La calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. LaTeX es Software Libre bajo licencia LPPL (LaTeXProject Public License). Se ha utilizado para escribir esta memoria que recoge el trabajo realizado en el proyecto.

Kile: es un editor sencillo para TeX y LaTeX. Se ha utilizado para generar esta memoria. Se utilizan las funciones integradas de las que dispone, que hacen que la creación del documento sea más sencilla (autocompleta comandos TeX/ LaTeX posibilita generar y ver el documento con un solo click, tiene un asistente para introducir la bibliografía ...).

GIMP(GNU Image Manipulation Program): es un programa de manipulación de imágenes del proyecto GNU. Es la alternativa más firme del software libre al popular programa de retoque fotográfico Photoshop. En este proyecto se ha utilizado para hacer capturas de las imáge-

nes del diálogo diseñado así como para el retoque de otras imágenes que aparecen a lo largo de la memoria.

Open Office.org: es un proyecto basado en el código abierto para crear una *suite* ofimática. Es bastante compatible con los formatos de fichero de Microsoft Office, ya que puede leer directamente los archivos creados con dicha *suite* ofimática, aunque tiene su propio formato de archivos basado en el estándar XML. Se ha utilizado los módulos "Draw" (módulo de dibujo vectorial) e "Impress" (presentaciones) para crear las figuras que se ven a lo largo de la memoria.

Kate: es un editor de texto avanzado para el proyecto KDE. Entre otras características soporta el resaltado automático de sintáxis (para lenguajes de programación), manejador de archivos y emulador de terminal (basado en Konsole) integrados y el uso de plugins. Se ha utilizado, junto con Eclipse, para el desarrollo de programas.

Apéndice B

Manuales de referencia

En este apéndice se indican algunos de los distintos recursos que han sido utilizados para la elaboración de este proyecto y que pueden resultar de ayuda para su entendimiento y posterior uso. Se recogen desde los más básicos referentes a Linux hasta las más avanzadas referencias a CORBA.

B.1. Linux

En el momento de elegir el sistema operativo a utilizar la proyectante no tenía ninguna experiencia anterior con Linux por lo que fue necesario el aprendizaje de su manejo, tanto de los programas propios de cada distribución como de la utilización de la consola: los comandos más básicos y los no tan básicos. En las direcciones siguientes es posible encontrar documentación útil sobre estos temas:

http://alts.homelinux.net/
: aquí se encuentra información sobre software libre

http://linux.about.com/library/glossary/blglos_index_n.htm
: aquí sobre linux en general

http://www.zonasiete.org/manual/index.html
: este manual es bastante útil para el aprendizaje, resultando muy claro y bastante completo.

B.2. C++

Para las dudas y consultas a la hora de programar en C++
http://cplus.about.com/od/beginnerctutorial/l/aa122202a.htm

B.3. Síntesis de voz

En este tema son infinidad de páginas las que pueden visitarse y que resultan de interés por lo que sólo se incluyen unas cuantas:

http://freetts.sourceforge.net/docs/index.php
: recoge información del sistema FreeTTS.

http://festvox.org/
: es la página principal desde la que es posible la descarga de festival (manuales y documentación) y la de festvox en sí para la construcción de nuevas voces.

http://tcts.fpms.ac.be/synthesis/mbrola.html
: página principal del proyecto MBROLA desde el que puede desargarse y donde se indican los sistemas compatibles con él.

http://www.speech.kth.se/wavesurfer/index.html
: para obtener código libre que permite la visualización y modificación de sonidos.

http://cslu.cse.ogi.edu/

B.3.1. Festival

http://www.cstr.ed.ac.uk/projects/festival/
: página del proyecto

http://festvox.org/docs/
: servidor desde el que es posible descargar documentación de festival y speech_tools, manuales y tutoriales.

http://festvox.org/festvox/book1.html
: documento incompleto para la construcción de nuevas voces.

B.4. Scheme

Además de [E. Charniak, 1985] se puede recurrir a:

http://www.swiss.ai.mit.edu/projects/scheme/
: todo sobre este lenguaje basado en Lisp.

http://www-2.cs.cmu.edu/Groups/AI/html/cltl/clm/index.html
: recoge un índice de las funciones más usadas en Lisp.

B.5. SABLE

Se puede encontrar la documentación necesaria para escribir un archivo con marcado SABLE en:
http://www.bell-labs.com/project/tts/sable.html

B.6. Qt

Para obtener documentación on-line de todas las clases que maneja *designer* para la creación de diálogos:
http://doc.trolltech.com/

B.7. CORBA

http://linas.org/linux/corba.html
: una introducción muy general a la tecnología, con enlaces a distintos brokers.

http://adams.patriot.net/ tvalesky/freecorba.html
: recoge los enlaces de CORBA de código libre

http://www.omg.org/gettingstarted/corbafaq.htm
: las preguntas más frecuentes sobre CORBA contestadas por la organización que establece el estándar.

Bibliografía

[Alan W. Black, 2003] Alan W. Black, K. A. L. (2003). Building synthetic voices.

[Alan W. Black, 1994] Alan W. Black, P. T. (1994). CHATR: a generic speech synthesis system.

[Alarcón et al., 1994] Alarcón, M., Rodríguez, P., Almeida, L., Sanz, R., Fontaine, L., Gómez, P., Alamán, X., Nordin, P., Bejder, H., and de Pablo, E. (1994). Heterogeneous integration architecture for intelligent control. *Intelligent Systems Engineering*.

[Clavijo et al., 2000] Clavijo, J. A., Segarra, M. J., Sanz, R., Jiménez, A., Baeza, C., Moreno, C., Vázquez, R., Díaz, F. J., and Díez, A. (2000). Real-time video for distributed control systems. In *Proceedings of IFAC Workshop on Algorithms and Architectures for Real-time Control, AARTC'2000*, Palma de Mallorca, Spain.

[Cole et al., 1995] Cole, R., Hirschman, L., and Atlas, L. (1995). The challenge of spoken language systems: research directions for the nineties.

[Conde, 2005a] Conde, R. (2005a). *Manual de Higgs, plataforma Pioneer 2AT-8*, 1.0 edition.

[Conde, 2005b] Conde, R. (2005b). Mapeo facial de emociones sintéticas.

[Dutoit, 1997] Dutoit, T. (1997). High-quality text-to-speech synthesis: an overview, Australia: Special issue on speech recognition and synthesis. *Journal of Electrical & Electronics Engineering*, 17(1):25–37.

[E. Charniak, 1985] E. Charniak, D. M. (1985). *Introduction to Artificial Intelligence*. Addison-Wesley Publishing Company.

[Fernández, 2003] Fernández, A. (2003). El maravilloso mundo de Linux 2.6. www.escomposlinux.org/wwol26.html.

[Hernando, 2005] Hernando, A. (2005). Versión RT-CORBA del servidor Pioneer2AT-8.

[Klatt, 1987] Klatt, D. (1987). Review of text-to-speech conversion for english. *Journal of the Acoustical Society of America*, (82):737–793.

[Lemmetty, 1999] Lemmetty, S. (1999). Review of speech synthesis technology. Master's thesis, Helsinki University of Technology.

[M. Henning, 1999] M. Henning, S. V. (1999). *Advanced CORBA Programming with C++*. A. Wesley, third edition.

[Pareja, 2004] Pareja, I. (2004). Sistema de comunicaciones para PIONEER 2-AT8.

[R. Potter, 1947] R. Potter, G. Kopp, H. G. (1947). Visible speech.

[R. Sproat, 1999] R. Sproat, M. Ostendorf, A. H. (1999). The need for increased speech synthesis research: Report of the 1998 NSF workshop for discussing research priorities and evaluation strategies in speech synthesis.

[Robotics, 2002] Robotics, A. (2002). *ActivMedia Robotics'Pioneer 2-H8S Operations Manual*, 1.0 edition.

[Sanz, 2002] Sanz, R. (2002). Embedding interoperable objects in automation systems. In *Proceedings of 28th IECON, Annual Conference of the IEEE Industrial Electronics Society*, pages 2261–2265, Sevilla, Spain. IEEE Catalog number: 02CH37363.

[Sanz, 2003] Sanz, R. (2003). The IST HRTC project. In *OMG Real-Time and Embedded Distributed Object Computing Workshop*, Washington, USA. OMG.

[Sanz et al., 1999a] Sanz, R., Alarcón, I., Segarra, M. J., de Antonio, A., and Clavijo, J. A. (1999a). Progressive domain focalization in intelligent control systems. *Control Engineering Practice*, 7(5):665–671.

[Sanz et al., 1999b] Sanz, R., Matía, F., and Puente, E. A. (1999b). The ICa approach to intelligent autonomous systems. In Tzafestas, S., editor, *Advances in Autonomous Intelligent Systems*, Microprocessor-Based and Intelligent Systems Engineering, chapter 4, pages 71–92. Kluwer Academic Publishers, Dordretch, NL.

[Sanz et al., 2001] Sanz, R., Segarra, M., de Antonio, A., and Alarcón, I. (2001). A CORBA-based architecture for strategic process control. In *Proceedings of IFAC Conference on New Technologies for Computer Control*, Hong Kong, P.R. of China.

[Sanz et al., 2000] Sanz, R., Segarra, M., de Antonio, A., Alarcón, I., Matía, F., and Jiménez, A. (2000). Plant-wide risk management using distributed objects. In *IFAC SAFEPROCESS'2000*, Budapest, Hungary.

[Sanz et al., 1999c] Sanz, R., Segarra, M. J., de Antonio, A., and Clavijo, J. A. (1999c). ICa: Middleware for intelligent process control. In *IEEE International Symposium on Intelligent Control, ISIC'1999*, Cambridge, USA.

[Sanz and Zalewski, 2003] Sanz, R. and Zalewski, J. (2003). Pattern-based control systems engineering. *IEEE Control Systems Magazine*, 23(3):43–60.

[Sutton et al.,] Sutton, S., Cole, R., de Villiers, J., Vermeulen, P., M.Macona, Yan, Y., Kaiser, E., Rundle, B., K.Shobaki, P.Hosom, Kain, A., Wouters, J., Massaro, D., and Cohen, M. Universal speech tools: the CSLU Toolkit.

[Sánchez, 2005] Sánchez, S. (2005). Integración de SOAR en ICa.

[W. Walker, 2002] W. Walker, P. Lamere, P. K. (2002). FreeTTS. a performance case study.