Mapeo facial de emociones sintéticas

Raquel P. Conde López

Marzo 2005

A fuerza de perseverancia
alcanzó el caracol el arca de Noe.

Agradecimientos

A mi familia, por el apoyo. En especial a mis padres, Roberto y Mari Carmen por la paciencia. También a mis hermanos Eduardo y Sergio por estar ahí. Porque todos ellos han dado mucha caña, pero sé que están orgullosos de que "la niña" sea ingeniero. Porque no los cambiaría por nada, aunque a veces me los hubiera cargado.

A Ricardo por mil motivos. Por provocarme con tantas cosas nuevas, por hacer más grande mi mundo. Por creer en mí y darme la oportunidad de trabajar en su equipo. Por apoyarme en los malos momentos y cuidarme. Por ser grande como ingeniero y como persona. Por su amistad. Por tener siempre la puerta abierta y una silla en su despacho, a pesar de estar más que liado casi siempre.

A Pablo por su amistad todos estos años. Por tirar de mi en los malos momentos y celebrar conmigo los buenos. Por las clases magistrales. Por las cervezas y las risas. Por tantos buenos ratos. Por ser el más canalla. Por estar siempre de mi lado y apoyarme. Porque si he llegado a la meta es en gran parte por su culpa.

A Carlosm por todo el trabajo que ha hecho conmigo, por convertirme en una friki, por su paciencia y confianza en mí. Por reír conmigo y apoyarme en los malos momentos. Por sus consejos, por ser un tipo muy sabio. Por ayudarme en todo momento, pero sobre todo en la recta final y hacer que esto finalmente fuera realidad. Por los paseos por Madrid al salir del depar, por los mails y por portarse como un amigo.

A Julia por echarme más que un cable con la memoria y no asesinarme por comerme todos los acentos. Por ese dominio y control de los "papeles" que ninguno llegaremos a tener. Por la parte de culpa que tiene de que seamos un gran equipo.

A Anita, porque a pesar de los años y las nominaciones sigue estando ahí, porque lo bueno perdura en el tiempo. A Manuel por las eternas charlas arreglando el mundo delante de un café de comercio justo. A Rafa por los buenos ratos. Por la complicidad en tantos momentos. Porque somos demasiado distintos como para no chocar a veces, pero ambos seguimos enteros. A Silvia por cuando eramos una piña y podíamos con todo, porque a pesar de lo complicadas que se pusieron a veces las cosas, salimos de todas. Por las risas y por las lágrimas, porque nos hicieron fuertes. Por las cosas buenas. A Cris por tener siempre una sonrisa disponible. A Ceci por formar parte de la familia ASLab a pesar de ser del DIE. Gracias por estar ahí todo este tiempo. Por meternos en tantos líos y por apuntarse

a un bombardeo. A Paco por pertenecer a la secta. Por las vueltas y vueltas después de las clases de bailes de salón... A Ignacio por el psicoanálisis por el messenger, por las conversaciones absurdas y por las risas. A Carlos G. por pasarse siempre que puede por aqui y no olvidarse de nosotros. A la pandi porque nos hacemos mayores pero seguimos juntos después de tantos años y de tantas cosas... A los que dejaron que les "robara el alma" para este proyecto. A Belén por cuidar de su "hermana mayor" y por creer en mí.

A mis amigos de la escuela por compartir apuntes, clases y horas en la cafetería. Por las charlas en clase y en los pasillos, porque también ellos me enseñaron lo que es un ingeniero.

A todos aquellos, compañeros de pupitre y profesores, que se cruzaron en mi vida estos años. Porque de todos aprendí algo, porque algunos me enseñaron la valiosa lección de en lo que no quiero convertime.

Y a quien corresponda por ayudarme a que al fin, después de mucho esfuerzo, hoy pueda dar esto por terminado. ¡GRACIAS A TODOS!

Índice general

1. Introducción **1**

1.1. Motivación 1

1.2. Contexto 2

1.3. Objetivos del proyecto fin de carrera 5

1.4. Estrategia 5

1.5. Estructura de la memoria 5

2. Comunicación y emoción **7**

2.1. Interacción y diseño visual 8

2.2. Inteligencia emocional 9

2.3. Expresiones faciales 11

2.3.1. Descripción de las expresiones universales 12

2.3.2. Músculos de la cara 14

2.4. Animación facial 15

2.5. Modelo de Waters 18

2.5.1. Mecánica muscular 18

2.5.2. Vectores de músculos 18

2.5.3. El modelado de los músculos lineales 19

2.5.4. Propiedades elásticas 20

2.6. Aplicaciones 21

2.7. Conclusión: Ideas básicas 22

3. Una Cara para Higgs **25**

3.1. Introducción 25

3.2. La aplicación HiggsFace 25
3.3. Interacción con Higgs 26
3.4. Descripción de la aplicación HiggsFace 27
3.5. Diseño de la aplicacion HiggsFace 28

4. Plataformas software **31**
4.1. CORBA 32
4.1.1. El grupo OMG 32
4.1.2. La norma CORBA 33
4.1.3. La Tecnología CORBA 33
4.2. El Broker ICa 40
4.2.1. Arquitectura de Control Integrado 40
4.2.2. Instalación de ICa Broker 41
4.3. Omni 42
4.3.1. Instalación de Omni 42
4.4. Aria 42
4.4.1. Comunicación con el robot 43
4.4.2. Instalación de Aria 43
4.5. Cliente prototipo CORBA 43
4.5.1. Interfaz IDL 44
4.5.2. Estructura general del cliente C++ 46
4.5.3. Makefile 48
4.5.4. Ejecución 49
4.6. Servidor banco de pruebas 50
4.6.1. Qt 51
4.7. Aplicación final 53

5. Sistemas gráficos **55**
5.1. Introducción a los sistemas gráficos 56
5.2. Gráficos por ordenador 56
5.2.1. Creación de gráficos 3D 56

5.2.2. Animación . . . 57
5.2.3. Herramientas . . . 58
5.3. Blender . . . 58
5.4. Python . . . 60
5.5. Trabajo con Blender y Python . . . 61
5.5.1. Criterios de selección . . . 61
5.5.2. Prototipo Blender . . . 62
5.5.3. Animación . . . 66
5.5.4. Conclusiones . . . 68
5.6. OpenGL . . . 68
5.7. Trabajo con OpenGL . . . 72
5.7.1. Creación gráfica . . . 74
5.7.2. Animación . . . 80

6. Mapeo de emociones en Higgs 85
6.1. EmotionMapper . . . 86
6.2. Información sensorial . . . 87
6.3. Transformaciones emocionales . . . 87
6.3.1. Vector información sensorial . . . 88
6.3.2. Matriz de emociones . . . 89
6.3.3. Vector de información emocional . . . 90
6.4. Visualización . . . 92

7. Conclusiones y líneas futuras 93
7.1. Conclusiones . . . 93
7.2. Líneas futuras . . . 94
7.2.1. Nuevo servidor . . . 94
7.2.2. Teoría sobre las Emociones . . . 95
7.2.3. Aumentar el realismo de la imagen . . . 95

A. Ejemplos 97

B. Otras herramientas **99**

Índice de figuras

1.1. Parte de la interfaz del sistema HINT 2
1.2. Diagrama de caso de uso 4

2.1. Estudios de Duchenne 11
2.2. Expresiones universales de Eckman. 13
2.3. Músculos de la cara 16
2.4. Clasificación de los métodos de modelado 17
2.5. Modelo lineal de Waters 19
2.6. Secuencia de contracción muscular 20
2.7. Actividad muscular coseno 21

3.1. Robot móvil Pioneer 2-AT8 26
3.2. Esquema sencillo de la aplicación HiggsFace 27
3.3. Modelo UML elemental de la aplicación HiggsFace. 28

4.1. Servicio de objetos CORBA 34
4.2. Componentes de la arquitectura CORBA 35
4.3. Generación, integración y compilación 47
4.4. Simulador de Aria 50
4.5. Herramienta para Qt: Designer 52
4.6. Banco de pruebas 53

5.1. Captura de pantalla de Blender 2.36 59
5.2. Ejemplo de modelado con Blender 62
5.3. Imágenes auxiliares para el modelado 63

5.4. Modelo de una cara con Blender 64
5.5. Menú para la iluminación de Blender 64
5.6. Menú para el renderizado de Blender 65
5.7. Escena por defecto de Blender 65
5.8. Algunas opciones de Blender 66
5.9. Menú game engine de Blender 67
5.10. OpenGL Rendering Pipeline 71
5.11. Aplicación Facial Simulator para Windows 73
5.12. Ejemplo con OpenGL 74
5.13. Modelado de la cara con líneas 75
5.14. Modelado de la cara con polígonos 76
5.15. Distribución de los músculos en la cara 76
5.16. Esquema sencillo de la aplicación HiggsFace 84

6.1. Elementos de la aplicación HiggsFace 86
6.2. El EmotionMapper como aplicación lineal 91

Capítulo 1

Introducción

1.1. Motivación

La visualización de los estados funcionales de los sistemas técnicos tiene una gran importancia cuando de esta visualización se deriva la *correcta o incorrecta interpretación* de dicho estado por parte de un ser humano (consideremos, por ejemplo, la necesidad de comprender el estado de nuestro automóvil cuando lo vamos conduciendo).

Ésta ha sido una de las misiones tradicionales de las interfases de usuario donde no sólo la usabilidad del sistema técnico sino el desempeño global del sistema hombre-máquina dependen críticamente del modelo mental que el ser humano tiene de la máquina (ver Figura 1.1).

Al mismo tiempo, en la construcción de sistemas complejos de control intervienen muchos factores que determinan la necesidad de emplear arquitecturas software adecuadas. Esta adecuación se mide por el grado en que dichas arquitecturas permitan obtener tanto *características funcionales* determinadas (como son la satisfaccion de requisitos estrictamente necesarios en el ámbito del control) como *características no funcionales* (que son críticas a la hora de contruir un sistema software complejo).

Este proyecto fin de carrera se sitúa en la confluencia de ambas necesidades —visualización y arquitectura software— tratando de ofrecer una solución general al problema concreto de la percepción por parte de seres humanos no entrenados de estados técnicos globales de sistemas físicos.

Hay múltiples ejemplos que podemos ofrecer de esta necesidad de comprensión y que se han estudiado ampliamente en el ámbito de la psicología industrial, en el de la filosofía de la técnica y, más recientemente, en el ámbito de la robótica interactiva. Ejemplos paradigmáticos son el diseño de salpicaderos de automóviles, las interfases para operadores de procesos en caso de emergencias o los sistemas domóticos.

Obviamente en este proyecto no se trata de ofrecer una solución definitiva a todos los

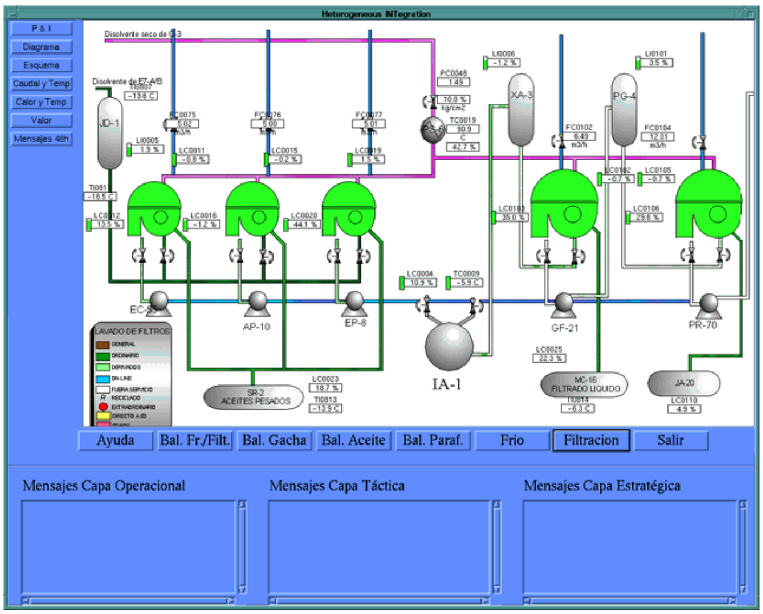


Figura 1.1: Una interfase de usuario diseñada para la operación de un sistema técnico debe permitir que el humano desarrolle un modelo mental adecuado para el desempeño del sistema hombre-máquina. En la figura se presenta parte de la interfase del sistema HINT [Alarcón et al., 1994].

problemas de visualización de estados técnicos —tarea posiblemente demasiado ambiciosa para un proyecto fin de carrera— sino de proponer una arquitectura de visualizacion concreta que se ofrece como una solución viable a uno de los problemas planteados por la necesidad de comprensión del sistema técnico que los interfases hombre-maquina plantean: la comunicación efectiva y sin entrenamiento previo a un humano de un estado técnico de un sistema físico. El objetivo es simple: *entender, de un vistazo, cómo está la máquina.*

1.2. Contexto

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 ditribuidos 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 ejecucion 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 multiples ámbitos de control:

- Control estratégico de procesos de fabricacion de cemento [Sanz et al., 2001].
- Gestión de emergencias en plantas químicas [Sanz et al., 2000].
- Sistemas de monitorización distribuida de tiempo real de produccion y distribución de energía eléctrica [Clavijo et al., 2000].

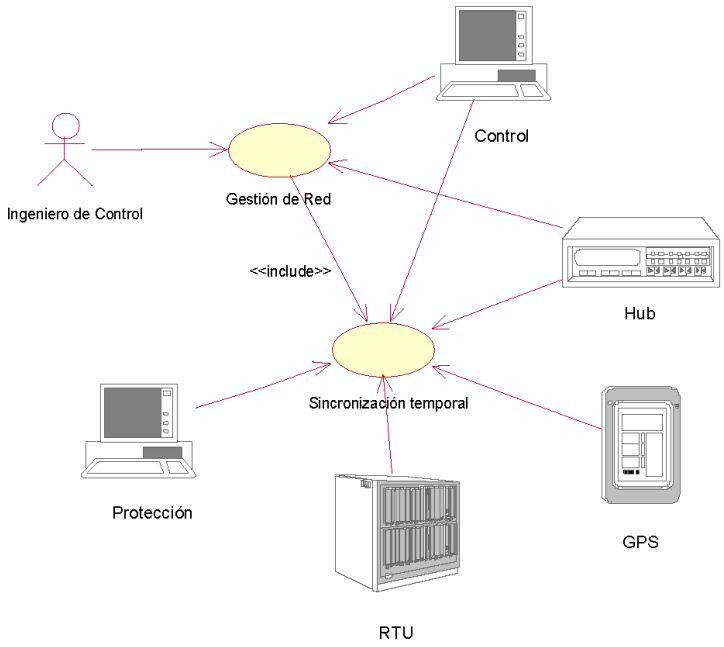


Figura 1.2: Diagrama de caso de uso para un despliegue de una aplicación de control y protección de redes.

- 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 contruyen 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 aplicacion, 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** y el nombre de este módulo es **SOUL::Face**.

1.3. Objetivos del proyecto fin de carrera

Los objetivos de este proyecto son muy simples —que no fáciles de alcanzar— y se describen sumariamente diciendo que 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.*

1.4. Estrategia

La estrategia a seguir para conseguir la correcta interpretación de los estado técnicos desde un punto de vista estratégico se basa en comunicacion emocional. Esta se detalla en el Capítulo 2 de esta memoria.

En pocas palabras podemos decir que la idea básica que guía este proyecto consiste en implementar un sistema metafórico-analógico de visualización de estados de sistemas técnicos mediante la representación tridimensional de caras antropomórficas correspondientes a estados emocionales humanos.

Decimos metafórico-analógico porque el sistema técnico no se diseña necesariamente para emular estado mentales humanos (como es el caso de la robótica humanoide). Así que cualquier interpretación del estado técnico en términos de emociones humanas es necesariamente metafórico. Sin embargo, las características del tipo de interacción hombre-máquina ulterior hace que veamos analogías de comportamiento derivables de la emoción que expresan estados intencionales de la máquina.

Así, una refinería podría "poner cara de susto" para indicar al operador humano la percepción, por parte del sistema técnico, de un riego inminente. Evidentemente, no podemos decir que la refinería está asustada —en el sentido humano— pero es obvio que el canal de comunicación facial-emocional proporciona, en este caso, un ancho de banda mucho mayor que los sistemas de alarma tradicionales.

1.5. Estructura de la memoria

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

Capítulo 1 / Introducción: Este capítulo es en el que se presenta el proyecto y se introducen sus ideas básicas.

Capítulo 2 / Comunicacion emocional: Donde se describen las ideas básicas del proyecto centradas en emplear visualizacion de estados técnicos mediante caras antropomórficas correspondientes a estados emocionales humanos.

Capítulo 3 / Una Cara para Higgs: En este capítulo se describe la aplicación específica para la que se implementa el sistema diseñado. **Higgs** es un robot móvil de Pioneer cuyo estado se pretende expresar mediante emociones.

Capítulo 4 / Plataformas software: En esta parte se describen las herramientas software que se utlizan para el desarrollo de la aplicación.

Capítulo 5 / Sistemas Gráficos: En este capítulo se describen las herramientas gráficas utilizadas para el desarrollo del proyecto así como el trabajo realizado con ellas.

Capítulo 6 / Mapeo de emociones en Higgs: Donde se describe el método con el que se obtiene la información sensorial de **Higgs** y cómo se traduce a estados emocionales que se visualizan en una cara antropomórfica.

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 / Ejemplos: Donde se recogen imágenes de las expresiones universales de Eckman para la cara diseñada para **Higgs**.

Apéndice B / Experimentos: Donde se recogen imágenes tomadas durante el desarrollo del proyecto a modo de experimento.

Apéndice C / Otras herramientas: Donde se recogen otras herramientas software utilizadas para el desarrollo de este proyecto.

Capítulo 2

Comunicación y emoción

En este capítulo se recogen unas ideas básicas acerca de la comunicación y las emociones. El objetivo de este proyecto es crear una interfaz de usuario de forma que, de un vistazo, se sepa cómo está el sistema físico en un determinado momento. Con las ideas que aqui se recogen se justifica la elección de la comunicación emocional mediante caras antropomórficas.

En la Sección 2.1 se describe la utilización de las imágenes como medio de comunicación de información.

La Sección 2.2 describe la importancia de las emociones en la toma de decisiones. De como razón y emoción son dos términos íntimamente relacionados cuando se habla de temas de inteligencia.

En la Sección 2.3 se trata el tema de la comunicación facial mediante expresiones. Se describe la existencia de una serie de expresiones universales (reconocidas por cualquier persona en cualquier parte del mundo) así como la descripción muscular de las mismas.

En la Sección 2.4 describe los modelos que existen para la generación y animación de caras de forma realista.

La Sección 2.5 describe el modelo muscular para la animación facial diseñado por Keith Waters.

La última parte, (Sección 2.7), recoge una serie de ideas acerca del tema tratado en este capítulo que se utilizan como base para el desarrollo del proyecto objeto de esta memoria.

2.1. Interacción y diseño visual

El diseño visual es una disciplina profesional que estudia los sistemas de información, con el objeto de convertir los datos en formas visuales, teniendo en cuenta los procesos perceptivos. Consiste en la creación de imágenes estéticas y funcionales con fines netamente comunicacionales.

El diseño visual estudia la generación y producción de la imagen fija, móvil, ambiental y digital, a partir de las estructuras de los lenguajes que operan, circulan y funcionan en la sociedad, con el objeto de entender la interactividad de los dispositivos y los datos con el observador. En nuestro caso este lenguaje es el lenguaje de la comunicación emocional entre humanos mediante la expresión facial. La importancia de este mecanismo de comunicacion queda manifiesta cuando, por problemas médicos, se pierde la capacidad de generar expresiones faciales (por ejemplo en el síndrome de Möebius [Cole, 1998]).

El diseño visual crea estructuras de comunicación e información visual, soportado por elementos estructurales del diseño y formales de los modelos de información y comunicación (emocional en nuestro caso). Así mismo, el diseño visual define los métodos para verificar, de forma experimental, la eficacia comunicativa de estos datos, con el propósito de reducir la entropía cognitiva.

Los sistemas de comunicación con sistemas técnicos emplean múltiples tecnologías, de las cuales la visualización no es sino una más; aunque dada la importancia de la percepción visual para los seres humanos —somos *animales visuales*— no es de extrañar que la visualización sea la forma de comunicación preferida[1]. Sin embargo, en la actualidad se entiende que muchos sistemas técnicos requieren de una nueva vizualización, que permitan entender, interiorizar e interpretar la información sobre el estado del sistema —el estado mental de la máquina, podríamos decir— de una manera mas dinámica y activa. El diseño visual, permite la creación de amplios sistemas comunicativos, basados en la ergonomía que permitan al usuario una relación más natural con dicha información. Es en relación con la ergonomía y con la eficacia donde la comunicación usando expresiones faciales encuentra su lugar.

Hay mucho trabajo realizado en el desarrollo de sistemas de visualizacion en ambientes técnicos (ver por ejemplo, todo el trabajo realizado en construccion de interfases de usuario para operadores de proceso) pero, en cierta medida, la visualizacion que aquí se investiga se aleja de estas líneas y entra más de lleno en la explotación de la conexión visual entre humanos.

[1]Hay otras, sin embargo, basadas en otras modalidades sensoriales o incluso basadas en comunicación directa —intrusiva y no intrusiva— con el cerebro

2.2. Inteligencia emocional

A pesar de las connotaciones negativas que en la actualidad tiene la palabra "sentimental" cuando forma parte del comportamiento, cualquier concepción de la naturaleza humana que deje de lado el poder de las emociones comete un error. Todos sabemos por experiencia propia que nuestras decisiones y nuestras acciones dependen tanto, y a veces más, de nuestros sentimientos como de nuestros pensamientos.

Todas las emociones son, en esencia, impulsos que nos llevan a actuar, programas de reacción automática con los que nos ha dotado la evolución. En cierto modo, las emociones son señales empleadas en los controladores complejos que nos mantienen en funcionamiento.

Desde nuestro punto de vista, las emociones son señales sintéticas de control estratégico. Sintéticas porque no responden a una magnitud concreta medida mediante cierto sensor sino que son producidas —calculadas— por el sistema de evaluación de nuestro control más global. Estratégicas porque estos bucles de control se sitúan en los niveles más altos de la jerarquía de controladores que nos mantiene vivos. Cada emoción, como señal de control, predispone al cuerpo a un tipo diferente de respuesta [Goleman, 1996]:

- el **enfado**: aumenta el flujo sanguíneo de las manos, haciendo más fácil empuñar un arma o golpear a un enemigo, también aumenta el ritmo cardíaco y la tasa de hormonas que, como la adrenalina, permitern generar la cantidad de energía necesaria para acometer acciones vigorosas
- el **miedo**: la sangre se retira del rostro (lo que explica la palidez y la sensación de "quedarse frío") y fluye a la musculatura esquelética larga (como las piernas, por ejemplo) favoreciendo así la huida. Al mismo tiempo, el cuerpo parece paralizarse, aunque sólo sea un instante, para calibrar, tal vez, si el hecho de ocultarse pudiera ser una respuesta más adecuada. Las conexiones nerviosas de los centros emocionales del cerebro desencadenan también una respuesta hormonal que pone al cuerpo en un estado de alerta general, sumiéndolo en la inquietud y predisponiéndolo para la acción, mientras la atención se fija en la amenaza inmediata con el fin de evaluar la respuesta más apropiada
- la **alegría**: aumento en la actividad de un centro cerebral que se encarga de inhibir los sentimientos negativos y de disminuir los estados que generan preocupación, al mismo tiempo que aumenta el caudal de energía disponible. En este caso no hay cambio fisiológico especial salvo, quizás, una sensación de tranquilidad que hace que el cuerpo se recupere más rápidamente de la excitación biológica provocada por las emociones perturbadoras
- la **sorpresa**: el arqueo de las cejas hace que aumente el campo visual y permite que penetre más luz en la retina, lo cual nos proporciona más información sobre el acontecimiento inesperado, facilitando así el descubrimiento de lo que realmente ocurre y permitiendo elaborar, en consecuencia, el plan de acción más adecuado

- el **desagrado**: tiene un gesto que transmite el mensaje de que algo resulta literal o metafóricamente repulsivo para el gusto o para el olfato. La expresión facial de disgusto (ladeando el labio superior y frunciendo ligeramente la nariz) sugiere un intento primordial de cerrar las fosas nasales para evitar un olor nauseabundo o para expulsar un alimento tóxico
- la **tristeza**: consiste en ayudarnos a superar una pérdida irreparable (como la muerte de un ser querido o un desengaño). La tristeza provoca una disminución de energía y del entusiasmo por las actividades vitales (especialmente las diversiones y placeres) y, cuanto más se profundiza y se acerca a la depresión, más se ralentiza el metabolismo corporal. Este encierro introspectivo nos brinda así la oportunidad de llorar una pérdida o una esperanza frustrada, sopesar sus consecuencias y planificar, cuando la energía retorna, un nuevo comienzo

Estas predisposiciones biológicas a la acción son moldeadas posteriormente por nuestras experiencias vitales y por el medio cultural en el que nos ha tocado vivir.

La mente racional y la mente emocional son consideradas como dos facultades relativamente independientes que reflejan el funcionamiento de dos circuitos cerebrales distintos aunque interrelacionados.

Las emociones se usan para enfatizar o ayudar a alcanzar una meta como actos intencionales de comunicación. En el mundo real las emociones de los demás influyen en nuestras decisiones y nuestro estado emocional. Pueden motivarnos y animarnos a conseguir ciertas cosas.

La idea de que las máquinas puedan tener un día emociones es un tema recurrente en ciencia-ficción. "2001: Una odisea del espacio", "Blade Runner" o "El hombre bicenterario" son algunos de los ejemplos que podríamos señalar. En la actualidad existe una gran diferencia entre ciencia-ficción y ciencia, pero muchos investigadores de inteligencia artificial consideran que es cuestión de tiempo superarla y hablan de la importancia de crear ordenadores emocionales.

Rosalind Picard [Picard, 1998] sugiere que los ordenadores influyen en las emociones humanas de forma tanto positiva como negativa, si comprendemos mejor estas influencias y podemos aplicar las positivas al diseño de los mecanismos de visualización e interacción, la gente se sentirá mejor al trabajar con ordenadores. Estar de buen humor tiene un impacto positivo en muchos aspectos de la creatividad y aumenta las posibilidades de éxito. Y como los estados de ánimo persisten y se contagian, los buenos sentimientos siguen cuando dejemos de trabajar con el ordenador.

Marvin Minsky en "The Society of Mind" [Minsky, 1978] reflexiona lo siguiente: *"la cuestión no es si las máquinas inteligentes pueden tener emociones sino si las máquinas pueden ser inteligentes sin emociones"*. Desde nuestra perspeciva, los subsistemas de control emocional, constituyen una parte importante de la generacion de comportamiento al más alto nivel y se imbrica, por tanto, en las estrategias más globales del agente inteligente.

2.3. Expresiones faciales

La cara es la parte más expresiva del cuerpo humano[2], el canal más importante de comunicación no verbal, es una buena interfaz para comunicar porque las personas tienen capacidad de reconocer expresiones e inferir de ellas estados emocionales.

El primero en demostrar la universalidad de las expresiones y su continuidad en hombres y animales fue Charles Darwin en 1872 en "The expression of emotions in man and animals"[Darwin, 1872] que fue publicado 10 años antes del más famoso "The origin of the species".

La investigación más importante sobre expresiones faciales fue desarrollada en 1862 por Duchenne[Duchenne, 1862], centrada en el estudio de los movimientos de los distintos músculos de la cara utilizando electrodos en puntos de la superficie del rostro (ver Figura 2.1). De este modo pudo clasificar los músculos en expresivos, inexpresivos o discordantemente expresivos. A pesar de que existen discrepancias entre su clasificación y la originada por los descubrimientos recientes, él fue el que definió esencialmente el campo.

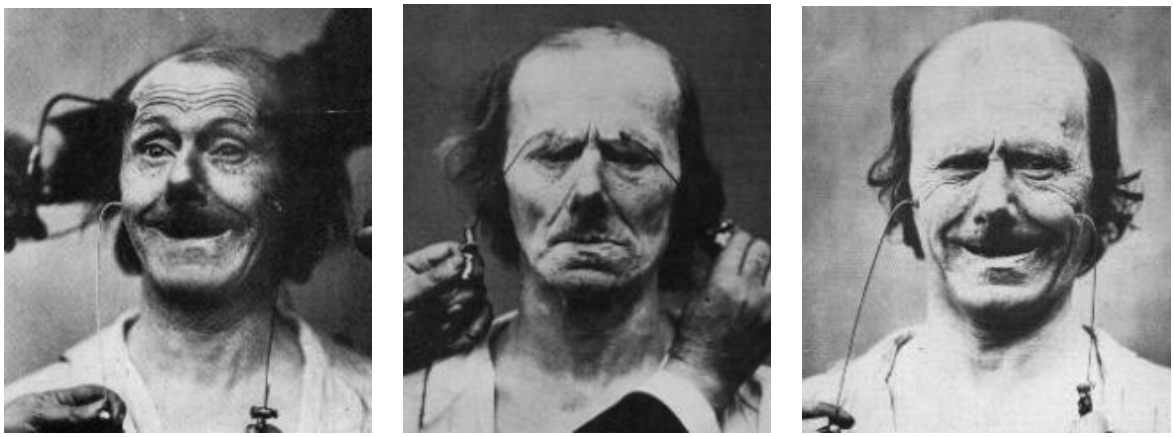


Figura 2.1: Estudios de Duchenne

El pionero del estudio analítico de la expresión de la cara humana fue Duchenne, pero no fue hasta los 70 que no se llevó a cabo un estudio más riguroso y preciso. Corrió a cargo de Paul Eckman y Wallace Friesen. Desarrollaron el Facial Action Coding System (FACS) [Ekman and Friesen, 1978] que permitía medir con rigor científico todos los movimientos musculares de la cara. Dividieron las expresiones en acciones de pequeños grupos de músculos que denominaron Action Units (AU).

Durante cinco años Eckman viajó alrededor del mundo para comprobar científicamente si los gestos y las expresiones difieren con la cultura, siguiendo las teorías de antropólogos

[2]La comunicación entre personas se produce mediante voz, las expresiones faciales y los gestos o posiciones del cuerpo, mas del 65 % de la información transmitida cara a cara forma parte de la banda no verbal.

destacados en aquella época. Según esta línea de pensamiento, los seres humanos aprendemos los gestos y las expresiones a través del contacto social, y éstos varían en función de la cultura. Pero el investigador recordó que Charles Darwin [Darwin, 1872] había dicho exactamente lo contrario: las expresiones humanas eran innatas y por tanto universales a todas las especies.

Utilizó la fotografía como soporte para sus investigaciones. Enseñó retratos a personas de cinco países diferentes para que identificasen la emoción de cada imagen y las interpretaciones coincidieron. También comprobó que cada cultura tiene sus propias normas que definen la forma socialmente aceptable de mostrar las emociones. Lo hizo mediante un experimento que consistía en mostrar películas con escenas quirúrgicas y accidentes a japoneses y estadounidenses. Si estaban solos tanto unos como otros movían los mismos músculos de la cara mientras que en presencia de un investigador los japoneses tendían a enmascarar más las emociones de desagrado con una sonrisa. Para confirmar los resultados obtenidos fué a cotejarlos a una cultura alejada de la civilización y convivió dos años con el pueblo *fore* en Papúa, Nueva Guinea.

Sus investigaciones concluyen que hay seis categorías que pueden considerarse como universales, esto es, reconocibles por cualquier persona de cualquier tipo y condición en cualquier parte del mundo. Estas categorías son: **alegría**, **tristeza**, **sorpresa**, **enfado**, **miedo** y **desagrado**. Cada una de estas ellas tienen un amplio rango de expresiones con distinta intensidad y variaciones en los detalles de las mismas.

Debemos a Gary Faigin [Faigin, 1990] la consideración de que hay expresiones no relacionadas con emociones que también están universalmente reconocidas: dolor, esfuerzo, somnolencia...

Las regiones más expresivas de la cara son los ojos, las cejas y la boca. Por eso son las zonas en las que se centra la descripción de las expresiones universales que se recoge en la siguiente sección.

2.3.1. Descripción de las expresiones universales

Dada la universalidad descubierta por Ekman [Ekman and Friesen, 1975], es posible hacer análisis y representaciones genéricas de estas emociones (ver Figura 2.2).

Alegría: en la expresión pura de alegría las cejas están relajadas. El párpado superior está ligeramente bajado y el párpado inferior está recto, siendo elevado por la mejilla. Los labios son finos y están fuertemente presionados contra el hueso. En la expresión de alegría se forman patas de gallo en los extremos de los ojos, un pliegue en forma de sonrisa bajo el párpado inferior, hoyuelos y un profundo pliegue nasolabial desde la nariz a la barbilla. Algunas variaciones de la alegría pueden ser risa desternillante, risa, sonrisa con la boca abierta, sonrisa melancólica, sonrisa ansiosa, sonrisa de agradecimiento, sonrisa tímida, sonrisa perversa, sonrisa con los ojos cerrados, falsa sonrisa y falsa risa. Las risas y sonrisas falsas se caracterizan por una disminución en las patas de gallo de los ojos, así como por

ausencia o sólo ligeros pliegues debajo del párpado inferior.

Tristeza: en la expresión pura de tristeza, la zona interior de las cejas se curva hacia arriba. La piel y los tejidos blandos debajo de la ceja se agolpan sobre el párpado superior. Los ojos se cierran ligeramente a consecuencia de la presión hacia abajo de los tejidos sobre el párpado y también por el movimiento hacia arriba del párpado inferior. La boca está relajada. Las arrugas asociadas a la tristeza incluyen pliegues horizontales en la frente; líneas verticales entre las cejas; pliegues oblicuos sobre el párpado superior y un pliegue en forma de sonrisa bajo el labio inferior. La tristeza puede tener muchas variaciones y diferentes intensidades: lloro con la boca abierta, lloro con la boca cerrada, tristeza reprimida, casi lloro e infelicidad. Estas variaciones pueden conllevar cejas completamente bajadas, ojos fuertemente cerrados, una protuberancia en la barbilla y un pliegue nasolabial muy pronunciado.

Figura 2.2: Expresiones universales de Eckman.

Enfado: en la expresión pura de enfado, los extremos de las cejas se empujan hacia abajo y se juntan. El extremo inferior de la ceja se pone al mismo nivel que el párpado elevado. El ojo está medio abierto, pero la presión de la frente impide que se vea el blanco del ojo por encima del iris. La boca está cerrada con el labio superior ligeramente comprimido. Las arrugas en el enfado incluyen pliegues horizontales sobre los párpados superiores y líneas verticales entre las cejas. Las variaciones posibles del enfado son: rabia y tensión. Dichas variaciones pueden implicar labios ligeramente presionados con protuberancia en la barbilla o un boca totalmente abierta con una mueca del labio superior con el labio inferior recto, enseñando los dientes superiores e inferiores.

Miedo: el miedo puede variar desde la preocupación al terror. Las cejas se elevan y se juntan. La parte inferior de las cejas se curva hacia arriba. Los ojos están alerta. La boca puede estar ligeramente abierta y caída, estando estirada en ambos lados. Las arrugas asociadas pueden incluir pliegues horizontales en la frente, líneas verticales entre los ojos, hoyuelos sobre las cejas y pliegues oblicuos sobre los párpados superiores. En la preocupación los labios se fruncen fuertemente y los bordes de los labios desaparecen. Se abulta la parte por debajo del labio inferior y sobre la barbilla. En el terror, tanto la boca como los ojos están totalmente abiertos. El labio superior se relaja mientras que el inferior se estira hasta mostrar los dientes inferiores. El pliegue nasolabial se vuelve recto y poco profundo. Pliegues en forma de paréntesis aparecen a los lados del labio inferior.

Sorpresa: en la sorpresa, las cejas se elevan rectas tanto como es posible. Los párpados superiores se abren lo máximo posible con los inferiores relajados. La boca se abre totalmente formando un óvalo. Se forman pliegues horizontales en la frente.

Desagrado: el desagrado puede variar desde el desdeño al rechazo físico. Las cejas se relajan y los párpados están relajados o tan sólo ligeramente cerrados. El labio superior se eleva en una mueca, a menudo asimétrica. El labio inferior se relaja. El pliegue nasolabial es más profundo a lo largo de la nariz. En el desdeño, los párpados pueden estar parcialmente cerrado con los ojos mirando hacia abajo. En el rechazo físico, las cejas se bajan, especialmente en los extremos interiores. Los ojos pueden estar casi cerrados, en un guiño. El labio superior se eleva en una mueca intensa que permite ver los dientes superiores. El labio inferior se eleva ligeramente. Hay líneas verticales entre los arcos superciliares, patas de gallo y por debajo del párpado inferior. También aparecen arrugas desde el lagrimal hacia el puente de la nariz así como una protuberancia en la barbilla.

2.3.2. Músculos de la cara

Los músculos en la cara se conocen habitualmente como los músculos de expresión facial. Algunos de estos músculos pueden también desempeñar otras funciones importantes, tales como el movimiento de mejillas y los labios durante el habla. Los músculos de expresión facial son superficiales (situados en el exterior del cráneo) y están unidos a una capa de grasa subcutánea y piel en los puntos de inserción. Cuando los músculos están relajados, el tejido graso rellena los hoyos existentes para dar al cráneo una forma suave.

Los músculos para la expresión facial trabajan sinérgicamente y no de forma independiente. El conjunto trabaja como un equipo coordinado y organizado, teniendo cada miembro un función específica. Los músculos interaccionan unos con otros. Por tanto, es difícil distinguir fronteras claras entre ellos. En el caso de la animación facial sólo los músculos principales de cada grupos se modelan.

Los músculos principales de expresión facial y sus efectos en la cara cuando se contraen son los siguientes[3]:

[3]Entre paréntesis los nombres correspondientes a la literatura inglesa.

- **Cigomático mayor (Zygomatic Major)**: desplaza la comisura de los labios superolateralmente. La parte inferior del surco nasolabial se profundiza cuando se estira hacia arriba y hacia fuera
- **Depresor del ángulo de la boca (Angular Depressor)**: tira de la comisura de la boca inferolateralmente. Profundiza la parte inferior del surco nasolabial y lo empuja hacia abajo
- **Elevador del ángulo de la boca (Labii Nasi)**: eleva la comisura y el labio superior, empujando la parte interior de la mejilla hacia el ojo
- **Elevador del labio superior y del ala de la nariz (Inner Labii Nasi)**: atrae en dirección superior el ala de la nariz y el labio superior. Se eleva y profundiza el surco nasolabial
- **Frontal interior (Frontalis inner)**: la parte media del músculo frontal eleva la parte interior de las cejas
- **Frontal exterior (Frontalis outer)**: la parte lateral del músculo frontal eleva la parte exterior de las cejas
- **Frontal (Frontalis major)**: eleva las cejas de forma que el arco formado es más prominente
- **Corrugador lateral (Lateral Corrugator)**: tira de la parte interna de las cejas
- **Corrugador de la ceja o superciliar (Corrugator Supercilli)**: junta las cejas

Su disposición puede verse en la Figura 2.3.

2.4. Animación facial

Desde el trabajo de Parke [Parke, 1972], pionero en los temas de animación facial, muchos investigadores han intentado generar un modelado y animación de rostros de forma realista. Los más ambiciosos han intentado que el modelado y renderizado se haga en tiempo real. Pero debido a la complejidad de la anatomía facial así como a la sensibilidad natural a la apariencia de los rostros, en la actualidad no existen sistemas ampliamente aceptados que generen las expresiones y emociones de forma realista y en tiempo real para un avatar (entendemos por avatar a toda representación digital de un personaje).

El objetivo de las caras tridimensionales es doble: que parezcan reales y que se muevan de forma real. Para lo cual es necesario fijarse en la anatomía y geometría del rostro. Éste está formado por músculos y huesos cubiertos de piel. Los huesos proporcionan el marco estático en el cual se anclan los músculos. Los músculos constituyen los organos

del movimiento y generan la expresion moviendo la piel. Los músculos también interaccionan unos con otros. Existen tres tipos de musculos en la cara: lineales/ paralelos, elípticos/circulares y planos. Un esquema de los músculos de la cara puede verse en la Figura 2.3.

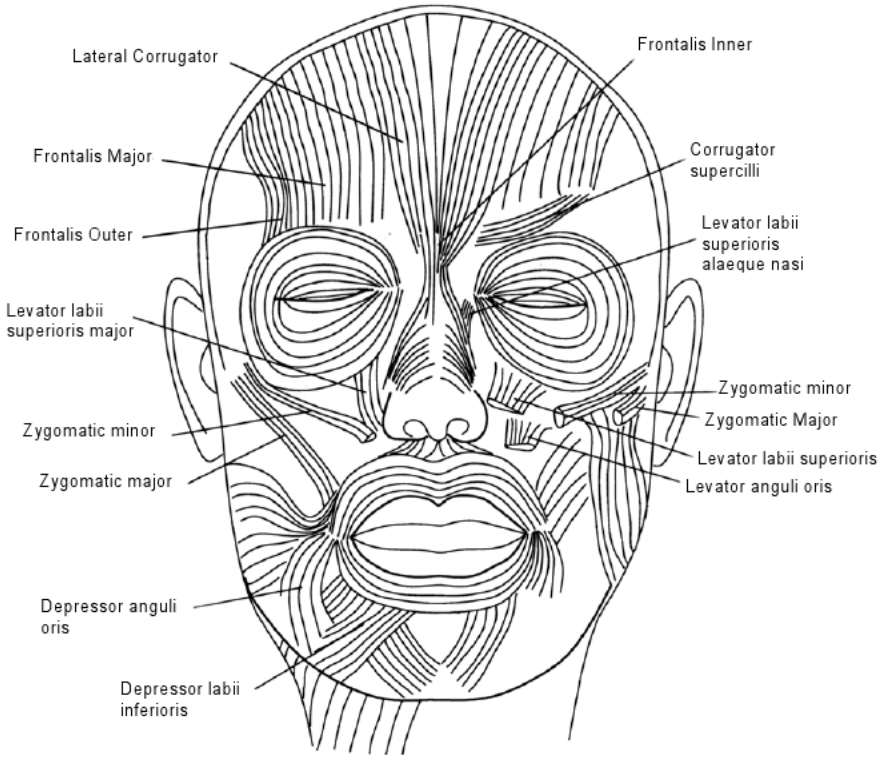


Figura 2.3: Músculos que intervienen en la generación de la expresión superficial de la cara [Waters and Parke, 1996].

Las caras en realidad son superficies muy complejas, no hay modelos que la representen y simulen en todos sus aspectos. Dependiendo de la aplicación que se esté desarrollando se necesitará un grado de detalle u otro. Para el caso de la animación de caracteres sintéticos podemos utilizar modelos que aproximen sólo algunos aspectos de la anatomía de la cara.

Al realizar el modelo facial podemos considerar que tenemos dos partes: la descripción geométrica y las posibilidades de animación. Ambas estan íntimamente relacionadas: la estructura del modelo determina su potencial animación. Puede que sea conveniente crear caracteres empáticos dependiendo del contexto (se odia al villano, se ama al héroe, se ríe con el cómico . . .)

Los detalles de la cara son muy importantes para obtener resultados reales (los ojos, el pelo de la cara (cejas, pestañas, barba . . .), los dientes, las encías y la lengua). Puede que también las orejas, aunque eso depende de la complejidad del modelo (son muy difíciles de modelar). La utilización de ojos detallados y dinámicos es muy importantes a la hora

de tener éxito en la expresividad de la animación (el movimiento de los ojos y los cambios de dilatación de la pupila añaden realismo a la animación, mientras que los detalles en la coloración del iris y las reflecciones en el globo ocular añaden vida a la cara). La cara ha de verse como parte de la cabeza, es necesario añadir cara, orejas, pelo, la parte de atrás de la cabeza y el cuello. Las orejas y el pelo le dan distinto aire a los personajes que se generen.

Para el modelado y la animación facial existen distintas técnicas que se dividen en dos categorías:

- manipulación de la geometría (*keyframes*, interpolación, parametrizaciones, métodos de elementos finitos, modelado basado en los movimientos de los músculos, etc.)
- manipulación de las imágenes (*morphing*, manipulación de texturas, etc.)

La figura 2.4 recoge una clasificación de los métodos de modelado y animación facial.

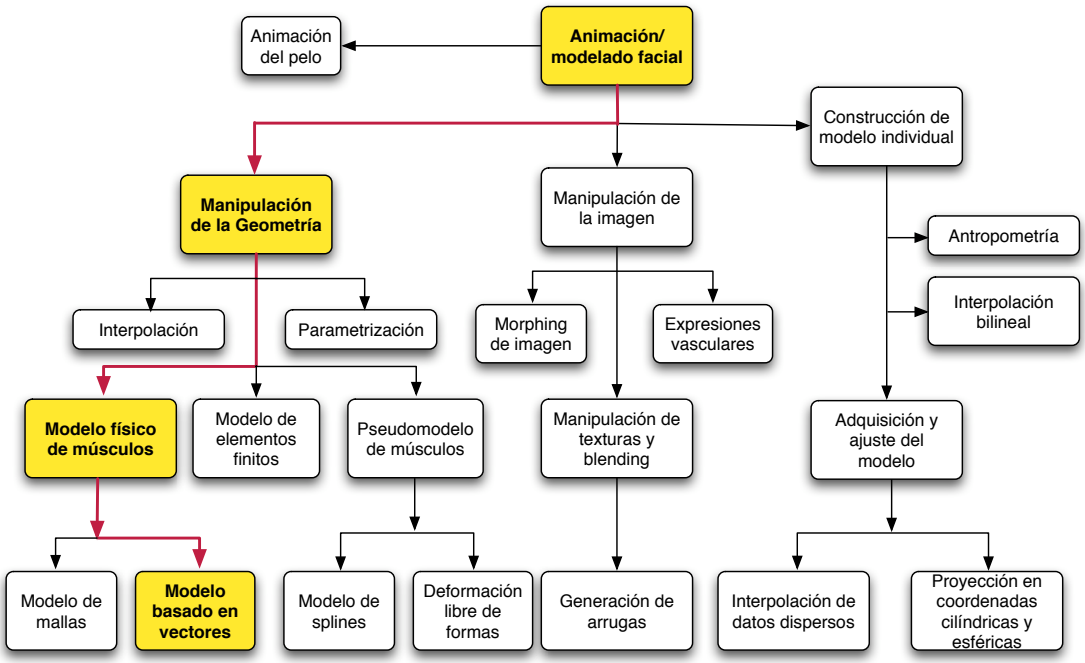


Figura 2.4: Clasificación de los métodos de modelado

2.5. Modelo de Waters

Esta sección contiene la descripción del modelo desarrollado por Waters [Waters, 1987] para la animación facial.

2.5.1. Mecánica muscular

Los músculos representan el motor principal de las expresiones faciales. Cuando un músculo se contrae, intenta mover los elementos anexos conjuntamente. Cuando se trata de músculos faciales, esta acción consiste en mover la piel hacia el punto de unión con el hueso.

Los músculos están formados por multitud de fibras larga llamadas miofibras en un diseño repetido. Cuando estas miofibras individuales se contraen, ejercen en conjunto una fuerza a lo largo de dos extremos forzando al músculo a encojerse. Los músculos se comportan como muelles ajustables de forma que las fuerzas que generan son función de la longitud y la estimulación neuronal.

Para modelar los músculos, una solución es la de ignorar la microestructura de las fibras musculares, modelando los músculos enteros como entidades básicas. Para mimetizar el efecto de la contracción muscular, se utiliza una función de deformación geométrica. Este procedimiento es con diferencia, el menos costoso computacionalmente y el que proporciona resultados aceptables.

Lo que se pierde con este modelo de deformación simplificado es la alteración de la geometría del músculo que se traduciría necesariamente en alteraciones de la posición de la piel que no serán adecuadamente modeladas.

2.5.2. Vectores de músculos

El considerar los músculos faciales como entidades individuales con puntos defnidos de unión y direcciones únicas, permite modelar un músculo como un vector en el espacio 3D. En este caso, el músculo tiene propiedades vectoriales de dirección y magnitud. La dirección es hacia el punto de unión con el hueso. La magnitud del desplazamiento depende de la constante elástica del músculo y la tensión creada por la contracción muscular. La magnitud es cero en el punto de unión y se incrementa hasta el máximo en el otro extremo de inserción en la piel.

Hay tres tipos básicos de músculos faciales que pueden modelarse: el músculo lineal, el músculo circular y el músculo plano.

- Los músculos lineales son fibras largas y paralelas, como por ejemplo el músculo cigomático mayor (o músculo de la risa). En estos músculos, la piel circundante se

contrae hacia el nodo estático de unión con el hueso hasta que, a cierta distancia, la fuerza disminuye a cero.

- Los músculos circulares están formados por fibras en forma de anillo, cuyo objetivo es cerrar orificios faciales tales como la boca o los ojos. En estos músculos, la piel se contrae hacia un centro imaginario de forma uniforme.

- Los músculos planos son áreas grandes y planas de fibras que no surgen de un punto origen concreto. Por tanto, se contraen hacia un nodo localizado en vez de a un grupo de nodos separados.

De hecho, el músculo plano es un conjunto de entidades musculares paralelas distribuidas sobre un área, siendo por tanto posible modelarlo como varios músculos lineales. El comportamiento y efecto de los músculos circulares también puede modelarse como varios muscúlos lineales orientados radialmente. Por tanto, el modelo del músculo lineal es el más importante de los tres.

2.5.3. El modelado de los músculos lineales

Waters describe el modelo de músculo lineal como sigue: para el músculo lineal es importante computar como el tejido adyacente, tal como el nodo p en la Figura 2.5 se ve afectado por la contracción del vector del músculo. Se supone que no existe desplazamiento en el punto de inserción en el hueso y que la máxima deformación tiene lugar en el punto de inserción en la piel. Como consecuencia, una disipación de la fuerza se pasa al tejido adyacente, a lo largo de ambos sectores en el diagrama.

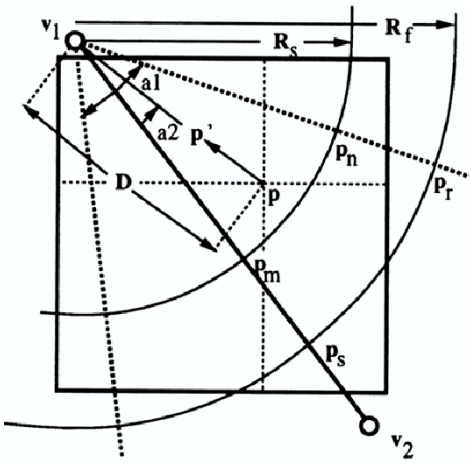


Figura 2.5: Modelo lineal de Waters y su zona de influencia

Para calcular el desplazamiento de un nodo arbitrario p situado en la malla un nuevo desplazamiento p' dentro del segmento $v_1 p_r p_s$ hacia v_1 a lo largo del vector p,v_1, se utiliza la siguiente expresión:

$$p' = p + akr\frac{pv_1}{||pv_1||}$$

Aquí, la nueva localización p' es una función de un parámetro de desplazamiento angular:

$$a = \cos(a2)$$

donde a2 es el ángulo entre los vectores (v_1,v_2) y (v_1,p), D es ||v_1-p||, r un parámetro de desplazamiento radial y k un constante fija que representa la elasticidad de la piel.

$$r = \begin{cases} \cos\left(\dfrac{1-D}{R_s}\right); & p \textit{ dentro de } v_1 p_n p_m p_1 \\ \cos\left(\dfrac{D-R_s}{R_1-R_s}\right); & p \textit{ dentro de } p_n p_r p_s p_m \end{cases}$$

Variando el factor de contracción del músculo, el desplazamiento de la piel parecerá moverse a lo largo del eje principal del músculo hacia el origen del vector como se muestra en la Figura 2.6.

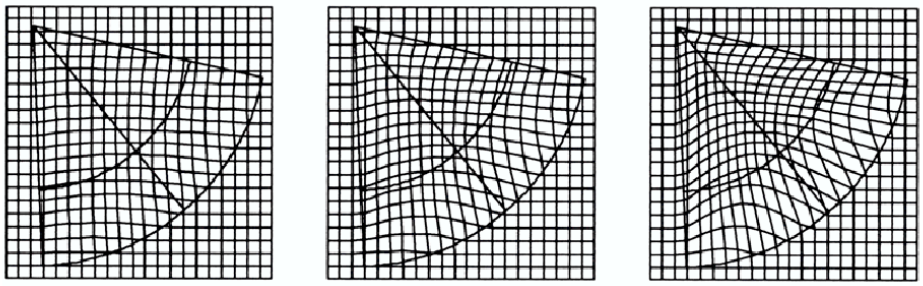


Figura 2.6: Secuencia progresiva de contracción muscular en una malla de 20x20. De izquierda a derecha con factores de 3, 5 y 10.

2.5.4. Propiedades elásticas

El modelo de los músculos lineales utiliza una función coseno como primera aproximación a las propiedades elásticas de la piel. Aún cuando esta aproximación produce resultados aceptables, es obvio que la elasticidad de la piel varía con la edad y de persona

a persona. Cambiando la función coseno por una interpolación no lineal o una función potencial, es posible variar la elasticidad de la malla y por tanto, simular la menor elasticidad de la piel según envejece.

Una función potencial incrementa las disminución a cero en los bordes de los vectores musculares Figura 2.7 muestra la función coseno elevada a la potencia x. Esta técnica permite una aproximación más flexible al modelado de los vectores de los músculos primarios.

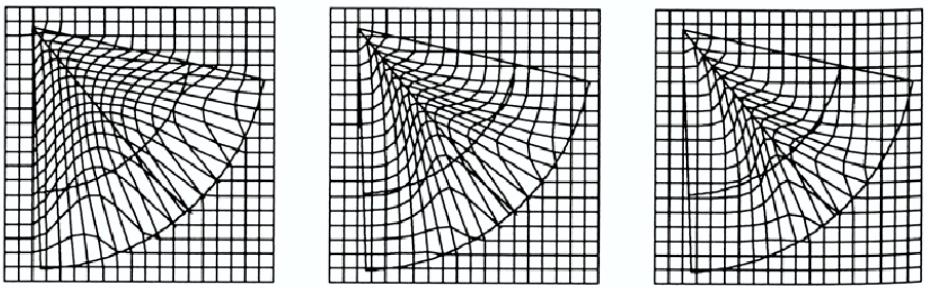


Figura 2.7: Actividad muscular coseno elevado a las potencias 0, 20 y 50 (de izquierda a derecha) con una contracción muscular de 2.0.

2.6. Aplicaciones

El crecimiento exponencial en la capacidad de computación en los últimos años ha llegado a tal nivel que la complejidad de las matemáticas empleadas en el tratamiento de gráficos tridimensionales puede resolverse en periodos suficientemente cortos como para producir animaciones 3D realistas. Asimismo, la disponibilidad de tarjetas gráficas que poseen procesadores 3D especializados ha convertido este tipo de animación en una realidad incluso para el público general. Esto trae consigo la aparición de nuevas industrias para satisfacer esta necesidad, así como la apertura de nuevas áreas para investigación.

Aunque el hardware existente para mostrar gráficos de calidad está disponible, no sucede los mismo con los métodos y técnicas para el modelado y la animación realista de personajes 3D, que están aún en desarrollo (aunque existen algunos productos internos explotados por empresas de animación y algunos productos comerciales en el ámbito de produccion de imagen sintética).

Algunas de las aplicaciones de esta tecnología, tanto desde el punto de vista industrial como de la investigación, son:

- Animación: la revolución de los gráficos 3D para la industria de animación ha producido un nuevo género de películas con títulos como “Toy Story” o “Final Fantasy”. No obstante, más allá de la representación fiable de personajes 3D, lo importante en

la creación de un personaje creíble es el realismo de sus expresiones y movimientos. Tradicionalmente, la animación se hace por parametrización, es decir, diferentes formas (morph) predefinidas se interpolan con diferentes pesos para representar una mezcla de expresiones faciales. Este método funciona con personajes de dibujos animados (p.ej. Toy Story) que poseen un número limitado de expresiones faciales. Sin embargo, la animación de caras humanas con anatomía perfecta (p.ej. Final Fantasy) utilizando el método anteriormente descrito requeriría un gran número de formas que pueden consumir demasiado tiempo.

- Juegos: la disponibilidad de aceleradores de tarjetas 3D ha cambiado el ámbito de los juegos en casa. La calidad de los gráficos procesados en tiempo real ha llegado a un nivel adecuado para que los rasgos faciales sean suficientemente detallados para ser animados. Por una parte, aparecen problemas similares a los existentes en animación. Por otra parte, en los juegos la expresión facial sólo es distinguible cuando ocupa la mayor parte de la imagen, lo que no es siempre habitual. Por tanto, la animación basada en músculo implica el consumo de recursos de computación que puede beneficiarse de la utilización de técnicas de reducción de complejidad basadas en la distancia.

- Teleconferencia: la habilidad para transmitir y recibir imágenes faciales es la base de la video o teleconferencia. A pesar del rápido crecimiento en la disponibilidad de ancho de banda para comunicaciones, se necesitan algoritmos de compresión. En lugar de transmitir datos de la imagen física, los algoritmos de visión artificial pueden utilizarse para extraer y transmitir estados de los músculos faciales de forma independiente. Unos pocos parámetros se transmiten al decodificador donde la cara humana se reconstruye, y allí, marco a marco se reflejan los cambios en los estados de los músculos en la representación 3D.

- Avatares: un área emergente es el uso de interfaces para el usuario, que utilizan personajes o agentes. La idea es que dichos agentes interaccionen con el usuario, de forma convincente para que el usuario crea que interactúa con un ser inteligente y/o emocional. Aquí, las expresiones faciales juegan un gran papel. La animación basada en músculo permitiría utilizar los mismos datos de expresiones para un gran número de agentes. La animación por parametrización requeriría un conjunto único de formas (morph) objetivos para cada personaje. Por el contrario, los estados musculares pueden utilizarse para distintos personajes.

2.7. Conclusión: Ideas básicas

La idea básica de este proyecto es que la comunicación máquina-hombre —en el sentido mencionado en el Capítulo 1 de comunicar estados técnicos globales— puede mejorar mucho con el uso de *caracteres sintéticos*. Según esta idea, dotaremos al sistema técnico (al robot,

al automóvil, a la refinería) de una cara antropomórfica para facilitar la relación. Esto se ha estudiado ampliamente en robótica humanoide, pero la propuesta de este trabajo es su aplicación a cualquier sistema técnico incluso no antropomórfico, ya que cualquier sistema tiene un estado global en términos de control —un estado emocional podríamos decir— susceptible de ser comunicado por este canal.

Para que la interpretacion de la cara sea efectiva no basta con la generación de cualquier representacion gráfica. En este proyecto se intenta investigar en técnicas para crear *agentes creíbles* para los humanos con los que interactúan. La habilidad de expresar emociones —importante para lograr el objetivo de la interpretación analógica— dependerá, necesariamente, de la calidad de la síntesis.

Las personas entienden a otras personas u objetos a través de la interacción y son expertos observadores del movimiento; detectando, de forma rápida, los movimientos no naturales o imposibles. Lo importante en este contexto es crear una *expresión inteligible* más que producir una cara humana exacta. Los gráficos de elevada calidad que son capaces de generar los ordenadores hacen que se puedan generar expresiones más reales, mejorando la comunicación hombre-máquina.

Las expresiones faciales están siendo cada ves más importantes en sistemas con interfaces humanoides (de hecho se investiga ampliamente en el ámbito de la robótica antropomórfica[4]). Los caracteres sintéticos son también cada vez mas populares[5] y, sin embargo, su comunicación facial es limitada. Las razones principales de que esto sea así son la falta de conocimiento sobre las expresiones faciales, especialmente su comportamiento dinámico, y la falta de métodos para crear y representar caras con el apropiado comportamiento emocional y cognitivo.

Existen diversas técnicas para la animación facial (ver Figura 2.4). El método seleccionado para este proyecto ha sido el realizar la animación facial basándonos en manipulaciones geométricas ayudándonos de las características físicas del modelo de músculos, utilizando vectores[6]. Es el denominado "Muscle Based Animation of Facial Expressions" desarrollado por Waters [Waters, 1987]. Consideramos que ésta es la mejor elección porque:

- Utiliza una simulación anatómica lo que le confiere realismo.
- Es rápida lo que permite su uso en tiempo real.
- Es un método relativamente simple lo que permite una implementacion en el alcance de este proyecto.

Por este método han demostrado interés tanto la industria del entretenimiento (juegos, películas de animación) como la comunidad de investigadores en general.

[4]Kismet: http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/kismet.html

[5]Un ejemplo de esto es que si hacemos una búsqueda del término "avatar" en el buscador Google encontramos más de veinticinco millones de enlaces.

[6]Camino señalado en la Figura 2.4

Diseñando de este modo nos aseguramos que podemos crear diferentes interfaces según la aplicación de la que formen parte con el mismo código, ya que entre unas caras y otras sólo varía la topología del rostro y la distribución de los músculos, sólo sería necesario que el programa cargara la adecuada en cada caso.

En el presente proyecto se ha diseñado una única interfaz sencilla, del estilo de una máscara que adopta las distintas expresiones faciales.

Capítulo 3

Una Cara para Higgs

3.1. Introducción

El proyecto **SOUL** del grupo ASLab pretende desarrollar una arquitectura de control reflexivo en el contexto de **ICa**. Esta arquitectura de control pretende conseguir un aumento de robustez en los sistemas de control basados en **ICa** por medio de la implementación de mecanismos de auto-consciencia.

Dentro de este contexto, este proyecto fin de carrera implementa un modulo reutilizable para la implementación de interfases basadas en caras. Este módulo se denomina **Faces**. Este proyecto también desarrolla una aplicación concreta de este módulo para proporcionar una cara a un sistema técnico: un vehículo de reducidas dimensiones.

3.2. La aplicación HiggsFace

En términos más precisos, la aplicación de demostración en la que se va a utilizar el módulo desarrollado en el presente proyecto es la comunicación de los estados globales del robot móvil Pioneer 2AT-8, denominado a partir de este momento en el resto de la memoria, como **Higgs**.

Higgs pertenece a la familia de robots móviles de ActivMedia Robotics[1]. Esta empresa diseña y construye robots móviles así como sistemas de navegación, control y de soporte a la percepción sensorial de los mismos. El Pioneer 2AT-8 es una plataforma robusta que incorpora los elementos necesarios para la implementación de un sistema de navegación y control del robot en entornos del mundo real.

El objetivo de la aplicación **HiggsFace** es visualizar, mediante una cara antropomórfica y por medio de expresiones emocionales, los estados internos del robot **Higgs**.

[1]Se puede conseguir más información en `www.activmedia.com`.

Figura 3.1: El robot Higgs. Un robot móvil Pioneer 2-AT8 de ActivMedia Robotics.

3.3. Interacción con Higgs

Para poder expresar los estados globales de **Higgs** mediante emociones hemos de comunicarnos con él. Este problema fue abordado en un proyecto desarrollado en ASLab el año pasado que tenía por título "Sistema de Comunicaciones para Pioneer 2AT-8" [Pareja, 2004]. El objetivo del mismo era la comunicación inalámbrica (para que la plataforma fuera autónoma) del robot móvil con la red local de computadores del departamento. Se desarrolló un software que permite la transmisión de toda la información sensorial proporcionada por el interfaz electrónico del robot a la red local, de modo que esta información está disponible en cualquier punto de la red y puede ser requerida en cualquier momento.

Esta implementación se hizo siguiendo las directrices de diseño de **ICa**, por lo que su reutilización en el contexto de este proyecto es prácticamente inmediata.

La aplicación se desarrolló mediante la implementación de un *servant* CORBA que encapsula la interfase proporcionada por el fabricante (denominada Aria). El objeto CORBA **Higgs**[2] es usado externamente por clientes CORBA siguiendo un esquema cliente/servidor.

El servidor **Higgs** se ejecuta en la CPU local del robot y se encarga de escuchar y atender las peticiones que le llegan por parte de los clientes que se ejecutan en cualquier máquina de la red local. En el diseño de **ICa** se prevé la posibilidad de que en algún momento los sistemas operativos y las plataformas de ejecución pueden ser diversos de modo que **Higgs** está preparado para la heterogeneidad del sistema. La tecnología CORBA nos proporciona esta capacidad de integración distribuida heterogénea.

Utilizando los resultados de dicho proyecto [Pareja, 2004], podemos obtener los estados mecánicos de **Higgs** mediante un cliente CORBA que le solicite al servidor que se ejecuta en

[2]Aunque el nombre del objeto CORBA es el mismo que el del robot, identificaremos de quien se trata en cada caso por el contexto (ver *nota aclaratoria* al final del capítulo).

Higgs los datos sensoriales para su posterior tratamiento y transformación de los mismos en visualización de estados emocionales.

La información que nos proporciona el servidor **Higgs** es la siguiente: velocidades, posición del robot, información de los sonares, y estado de la batería.

En el capítulo 6 veremos cómo se transforma esta información sensorial y cómo se mapean ésta a estados emocionales que serán expresados mediante expresiones faciales en un dispositivo de visualización.

3.4. Descripción de la aplicación HiggsFace

En la aplicación **HighsFace** intervienen tres elementos distribuidos:

- **Higgs**: un servidor CORBA que soporta la interfase Pioneer::Pioneer2AT.
- **HiggsFace**: un servidor CORBA que soporta la interfase SOUL::Face
- **EmotionMapper**: un programa intermedio (un agente activo **ICa**) que mapea estados de **Higgs** a estructura facial.

Un esquema sencillo de la aplicación puede verse en la Figura 3.2.

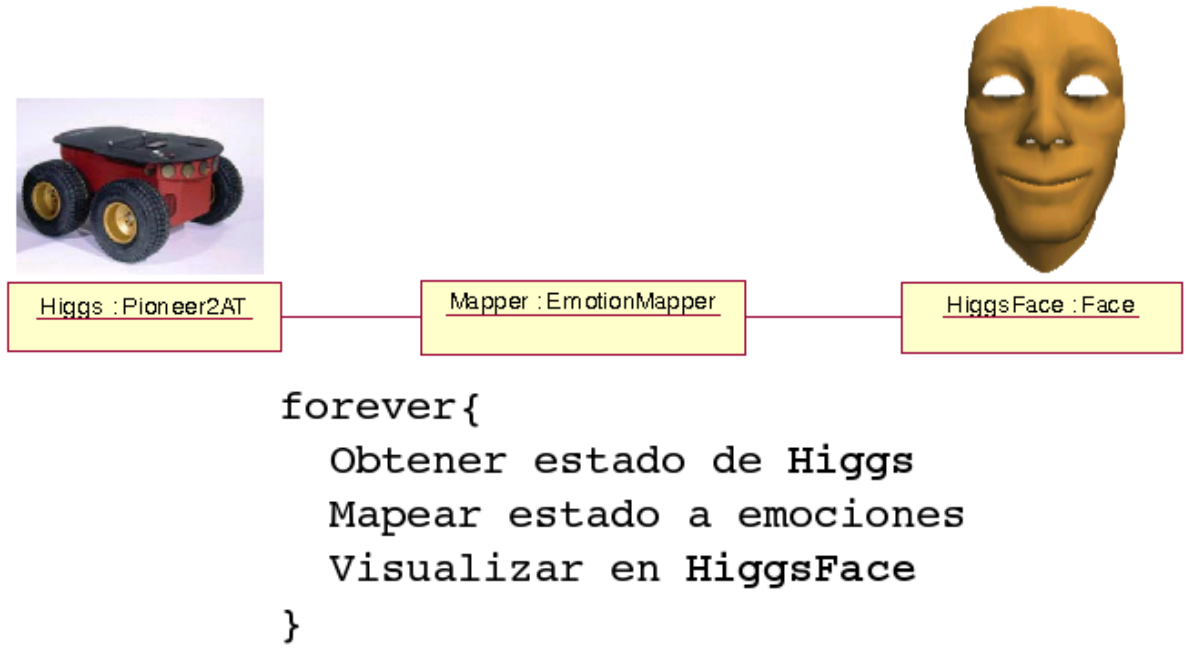


Figura 3.2: Esquema sencillo de la aplicación **HiggsFace**

El funcionamiento de la aplicación es simple: en Higgs se ejecuta el servidor CORBA Higgs, que nos proporciona la información sensorial del robot móvil. Esta información se mapea a estados faciales que se le comunican al servidor CORBA **HiggsFace** que se encarga de dibujar en la pantalla la cara correspondiente al estado mecánico del robot.

3.5. Diseño de la aplicacion HiggsFace

Uno de los criterios de diseño de **ICa** es la independencia de dominio de los componentes modulares.

En el caso que nos ocupa, el modulo **Higgs** implementa la interfase general **Pioneer::Pioneer2AT** que encapsula la funcionalidad del paquete Aria de ActiveMedia. Del mismo modo el módulo **HiggsFace** implementa la interfase **SOUL::Face**.

Ambas interfases han sido diseñadas para ser independientes de la aplicacion concreta (para proporcionar dicha independencia de dominio) y permiten la reutilizacion de los modulos en otras aplicaciones. Esto es posible ya que, por ejemplo **Higgs**, no implementa ninguna funcionalidad específica de la aplicacion construida en este proyecto.

Del mismo modo, el módulo empleado en la construccion de **HiggsFace** no implementa ninguna funcinalidad concreta de esta aplicación y podría ser empleado en cualquier otra (por ejemplo para generar caras de avatares o caras sintéticas en entornos de telepresencia humana).

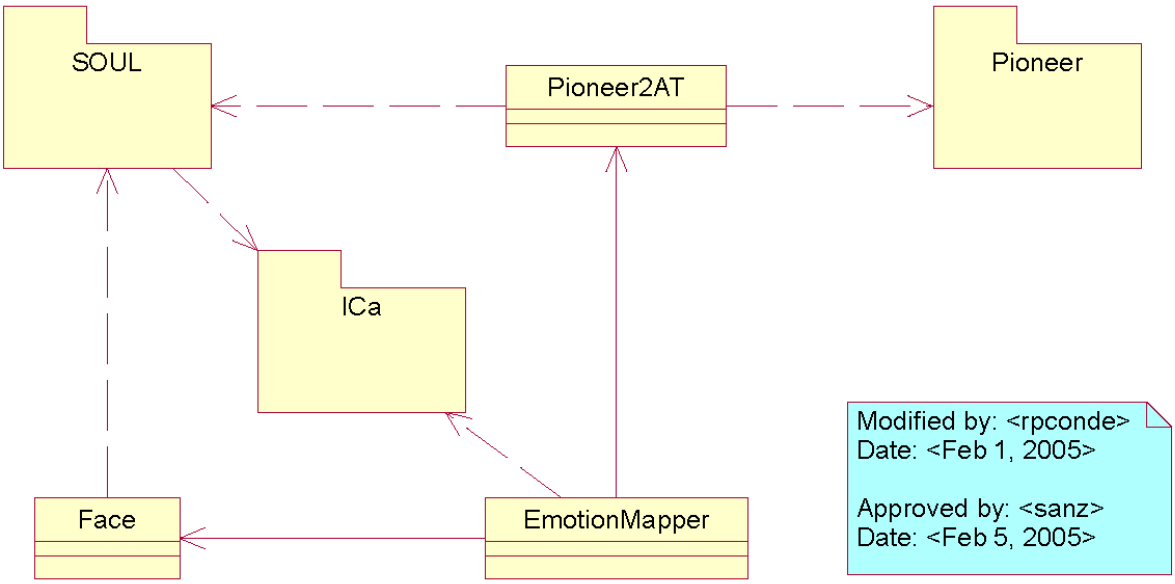


Figura 3.3: Modelo UML elemental de la aplicación HiggsFace.

Lo único que es específico de esta aplicación es el **EmotionMapper** ya que este módulo es el que vincula específicamente una interfase **Pioneer::Pioneer2AT** con una interfase **SOUL::Face**.

La aplicacion global recibe el nombre **HiggsFace** que es también el nombre empleado en el objeto CORBA que soporta la interfase **SOUL::Face** (ya que este es el centro del trabajo de este proyecto).

Nota aclaratoria: **Higgs** *tiene un cuerpo, una cara y un canal de conexión con el cuerpo. Todos ellos son* **Higgs**. *Podemos ser mas precisos y hablar del cuerpo de* **Higgs** *(el robot, un Pioneer 2AT8), la cara de* **Higgs** *(la cara, un* **SOUL::Face***) y el interfase del cuerpo de* **Higgs** *(el objeto software, un* **Pioneer::Pioneer2AT***). Desde el punto de vista del NameService CORBA,* **Higgs** *es el objeto que envuelve el cuerpo y* **HiggsFace** *es el objeto que gestiona la cara.*

Capítulo 4

Plataformas software

En este capítulo se describen las herramientas software que se utilizan para el desarrollo de la aplicación objeto del presente proyecto.

Como ya se dijo en la introducción, este trabajo se enmarca dentro de la filosofía **ICa** y eso, desde el punto de vista software, quiere decir estandarización: C, C++, POSIX y CORBA. Esta última es la tecnología de base para el desarrollo modular propugnado por **ICa** y por ello se le dedica una atención especial en este capítulo.

En la Sección 4.1 se describen aspectos generales de la especificación CORBA de la OMG (*e.g.* en qué consiste la tecnología y estructura general de la Object Management Architecture).

La Sección 4.2 se encarga de describir el Broker **ICa** como una implementación particular de CORBA que se va a utilizar en la aplicación (junto con otros brokers de terceros).

La Sección 4.3 describe otro broker utilizado para la implementación de los servidores CORBA de la aplicación desarrollada en este proyecto, el *broker* OmniORB.

En la Sección 4.4 se incluyen algunas notas sobre Aria, que es un paquete de clases suministrada por el fabricante del robot móvil para posibilitar el control del mismo. Este es el paquete encapsulado tras la interfase **Pioneer::Pioneer2AT**.

En la Sección 4.5 se describe el proceso de creación de un prototipo de la aplicación final **HiggsFace**.

En la Sección 4.6 se describen componentes auxiliares, y en particular, el servidor de pruebas. Este servidor se ha creado para poder generar los estados de **Higgs** cuando el robot no está operativo dado que, en concurrencia con este proyecto, los sistemas embarcados en él estaban siendo actualizados.

4.1. CORBA

La tecnología CORBA está orientada a los sistemas distribuidos heterogéneos. Un sistema distribuido heterogéneo es aquel compuesto por diferentes módulos software que interactuan 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.

4.1.1. 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

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

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.

4.1.2. 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 nombre 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.

4.1.3. 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

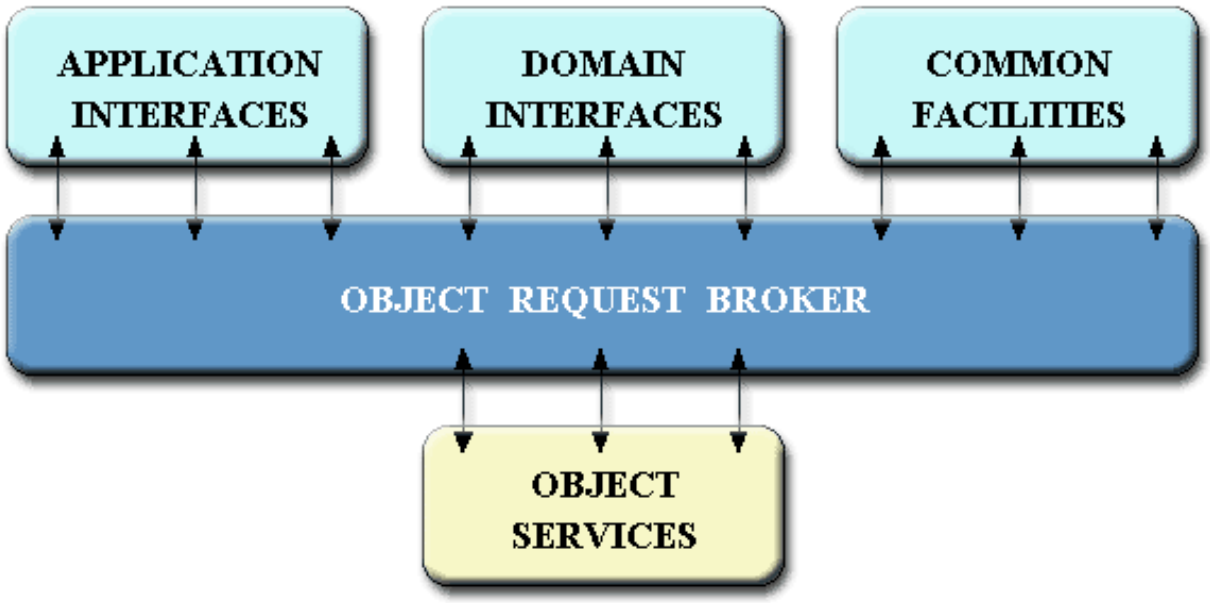


Figura 4.1: Servicio de objetos CORBA

de especificación del OMG[2].

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

[2]Más información en `www.omg.org`

- 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.

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

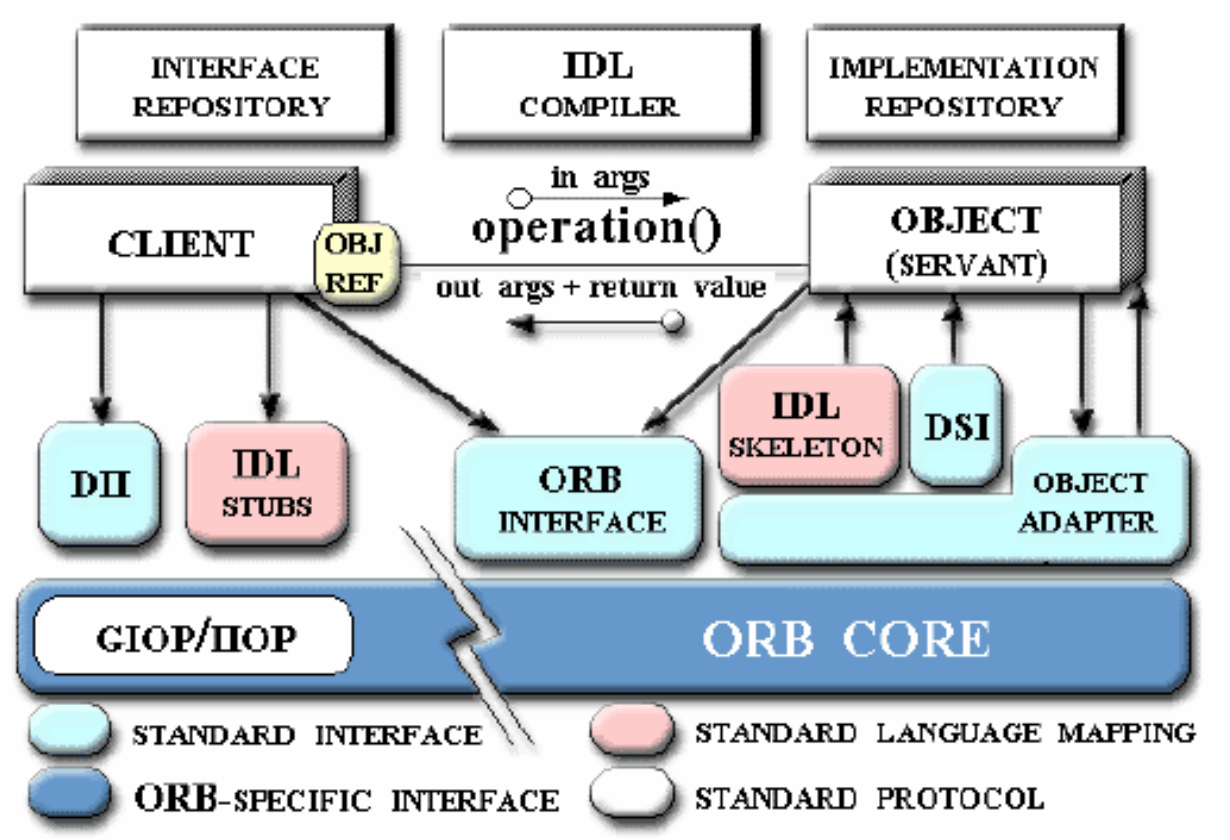


Figura 4.2: Componentes de la arquitectura CORBA

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 operación 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 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.

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, predecibilidad y seguridad.

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.

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 "exterior" 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

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*.

4.2. 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 codigo 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 4.1 esto no condiciona a tener que utilizarlo en este proyecto (recordemos la interoperabilidad entre ORBs descrita en la sección anterior), pero tener aplicaciones ya desarrolladas que lo utilizan es una buena razón para decantarnos por esa opción.

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 version 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.

4.2.1. 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 situan 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 extentiones 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**.

4.2.2. Instalación de ICa Broker

Para poder utilizar el broker de **ICa** es necesario instalar sus librerias en las máquinas en las que se vaya a desarrollar el proceso de creación del software[Chinchilla, 2003]. El sistema operativo elegido para el desarrollo de la aplicación que es objetivo de este proyecto es GNU/Linux, por lo que para la instalación de las mismas se utiliza el paquete correspondiente a este sistema operativo que fue proporcionado por SCILabs (creadores de **ICa Broker** RT 2.0).

El directorio `/usr/local/ICa` contiene las librerías, scripts y ficheros necesarios para la compilación de las aplicaciones basadas en el **ICa Broker**. Para completar la instalación se deben añadir algunas líneas a diversos ficheros del sistema:

- `/etc/hosts`, en la que se deben añadir las entradas para identificar la máquina en la que correrá el servicio de nombres (aplicación icans)
- `/etc/services`, en la que se deberá configurar a qué puertos y protocolos usa el broker para establecer la comunicación entre las distintas máquinas de la red local.

- `/etc/init.d/ICa`, hay que crearlo si se quiere que **ICa** arranque automáticamente como un servicio del sistema.
- `.bashrc`, añadir una serie de variables de entorno para facilitar la compilación de los programas

Con estos pasos se concluye la instalación de **ICa Broker**.

4.3. 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 terminos GNU General Public License. Es uno de los tres ORB al que se le ha concedio el Open Group's Brand de CORBA, lo que significa que ha sido probado y certificado para CORBA 2.1 (le faltan todavia características para ajustarse a la norma 2.6 como por ejemplo no se soportan las interfaces locales ni los objetos por valor).

Este es otro de los brokers usados en este proyecto para la implementación de los servidores de **HiggsFace**.

4.3.1. Instalación de Omni

Al ser Open Source se puede encontrar en la red[3] todo lo necesario para instalarlo, así como las instrucciones de como hacerlo dependiendo de la plataforma (Windows, Linux, Sun, etc.) sobre la que se esté trabajando.

4.4. Aria

Aria es un paquete software orientado a objetos programado en C++ que proporciona una API (Application Programming Interface) para el control de diversos robots móviles de ActivMedia[4]. Aria permite la construcción de aplicaciones para el control de los robots: desde la ejecución de simples comandos hasta la elaboración de comportamientos

[3] `http://omniorb.sourceforge.net/`
[4] Empresa fabricante del robot **Higgs**.

inteligentes como detectar y evitar obstaculos, reconocimiento de características de objetos, exploración...

Aria es un potente y robusto paquete compuesto por más de cien clases programadas en C++. Existen clases para el control de dispositivos como cámaras, láser o pinzas, clases con utilidades matemáticas o para el manejo de tiempos, *etc.*

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

4.4.1. Comunicación con el robot

Una de las principales funciones de Aria, y el primer cometido de toda aplicación de control para el robot, es establecer y gestionar la comunicación cliente-servidor AROS[5] entre los dispositivos del robot y la aplicación cliente. Suministra diversas clases para establecer esta conexión, una vez establecida, las funciones principales que se realizan son de lectura y escritura de datos en los dispositivos.

4.4.2. 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.

4.5. Cliente prototipo CORBA

Si bien las anteriores secciones se han dedicado a una descripción general de las herramientas, esta describe el uso de las mismas para la construcción de un prototipo de la aplicación objeto del presente proyecto.

Debido a las características CORBA descritas antes, existe total libertad a la hora de diseñar el cliente que se conecte a la aplicación servidor creada. Se utiliza invocación estática por lo que es necesario conocer la interfaz IDL con que se creó el servidor al que se conecta y ultilizar la misma para desarrollar el cliente, pero el lenguaje en el que se haga y el broker que se utilice puede elegirse libremente.

[5]AROS (ActivMedia Robotics Operating System), es el sistema operativo que se ejecuta en el microprocesador del robot móvil Pioneer-2AT8.

En "Sistema de Comunicaciones para Pioneer 2AT-8"?? se utilizaba **ICa** y C++ para desarrollar la aplicación cliente/servidor y este es el punto de partida en este caso, aunque nada obliga a ello ya que, como se explicó en la sección de CORBA, existe interoperabilidad entre ORBs y las aplicaciones cliente/servidor son independientes entre sí desde el punto de vista del lenguaje de implementación.

En las siguientes secciones se describen distintos aspectos referentes a la creación de un prototipo CORBA con el objetivo de familiarizarse con la programación para el desarrollo de la aplicación final. En la Sección 4.5.1 se describe la interfaz idl utilizada para la creación del mismo. La estructura general de un cliente CORBA con C++ se describe en la Sección 4.5.2. Las últimas secciones estan dedicadas a la compilación del programa (Sección 4.5.3) y la Sección 4.5.4 al modo en que se ejecuta el binario generado.

4.5.1. Interfaz IDL

Para la realización del programa cliente debe ser conocida la interfaz que proporciona el servidor 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{

const long NUMSONARS = 16;
typedef float SonarArray[16];

struct Pioneer2AT_State {
  float Velocity;
  float LeftVelocity;
  float RightVelocity;
  float RotVelocity;
  float XCoordinate;
  float YCoordinate;
  float ThCoordinate;
  SonarArray Sonar_Range;
  float ClosestSonar;
  float ClosestRange;
  float Battery;
};

interface Pioneer2AT{
```

```
      // Get all-in-one robot state
     Pioneer2AT_State getFullState();

     // Connection management
     long connect();
     void disconnect();

     // Velocities information
     float getVelocity();
     float getLeftVelocity();
     float getRightVelocity();
     float getRotVelocity();

     // Velocities manipulation and movement
     void  setVelocity(in float vel);
     void  setLeftRightVelocity(in float leftvel,in float rightvel);
     void  setRotVelocity(in float vel);
     void  moveDistance(in float vel);

     // Robot coordinates information
     float getXCoordinate();
     float getYCoordinate();
     float getThCoordinate();

     // Sonar information
     long  getSonar_Range(in long num_sonar);
     short getClosestSonar();
     long  getClosestRange();

     // Battery information
     float getBattery();

   // CORBA object remote control
     void init();
     void finish();

};

};
```

El nombre del módulo creado es **Pioneer**. En la primera parte del IDL se declaran los tipos de datos y se define la estructura **Pioneer2AT_State** que guarda toda la información sensorial del robot móvil. También se define un tipo de dato (SonarArray) para guardar

en un vector de números con coma flotante toda la información de los 16 sensores de los que dispone. En la segunda parte del IDL se declara la interfaz **Pioneer2AT** en la que se definen los métodos disponibles para el módulo **Pioneer**. En este caso el módulo sólo posee una interfaz pero podrían definirse tantas como fueran necesarias y los métodos disponibles son:

- obtener el estado del robot, esto es, toda su información sensorial (en una estructura)
- conexión/ desconexión al robot
- velocidad lineal y de rotación con la que se mueve
- manipulación de la velocidad y posición del robot
- posición del robot
- estado de la batería
- control del servidor CORBA

El siguiente paso es generar los *stubs* y los *skeleton* con el compilador idl correspondiente a **ICa** (que se llama ADL). Se utiliza el *stub* generado para implementar el cliente; los *skeleton* se usan para implementar la parte del servidor[6]. Un esquema general del proceso puede verse en la figura 4.3

4.5.2. Estructura general del cliente C++

Al principio de la función "main" del programa principal de la aplicación cliente hay que realizar una serie de pasos relacionados con CORBA, la forma en la que se hacen estos pasos es siempre la misma. En los siguientes fragmentos de código se recogen estos pasos así como los comandos CORBA correspondientes:

1. Instanciar las clases.

```
RTCORBA::RTORB_var rtorb;
CORBA::Object_ptr nameService;
CosNaming::NamingContext_var namingContext;
CosNaming::Name name;
RTPortableServer::POA_var rtpoa;
PortableServer::POAManager_var mgr;
```

2. Obtener la referencia del ORB, si es nula el programa aborta su ejecución.

[6]Existe un servidor diseñado con la misma interfaz IDL para ejecutarse en la CPU de **Higgs** y atender peticiones

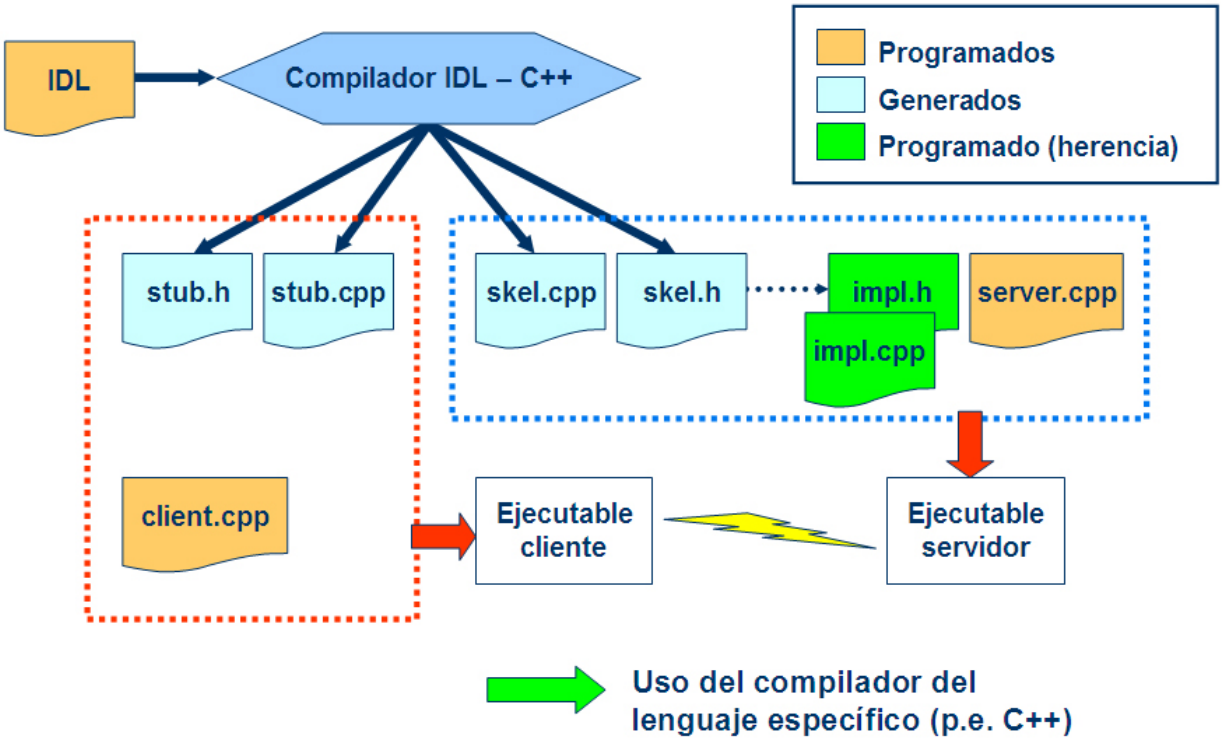


Figura 4.3: Generación, integración y compilación

```
cout << "Obtaining a RTORB reference...";
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
CORBA::Object_var obj = orb->resolve_initial_references("RTORB");
rtorb = RTCORBA::RTORB::_narrow(obj);
cout << "done." << endl;

if (CALL_IS_NIL(rtorb.in()))
    cerr << "ERROR: Couldn't get a reference to RTORB" << endl;
```

3. Obtener la referencia del POA.

```
cout << "Obtaining a Root POA reference...";
CORBA::Object_var obj2 = rtorb->resolve_initial_references("RootPOA");
rtpoa = RTPortableServer::POA::_narrow(obj2);
cout << "done." << endl;

if (CALL_IS_NIL(rtpoa.in()))
    cerr << "ERROR: Couldn't get a reference to RTPOA" << endl;
```

4. Activar el POA Manager.

```
cout << "Activating POA Manager...";
mgr = rtpoa->the_POAManager();
mgr->activate();
cout << "done." << endl;
```

5. Obtener el NameService.

```
cout << "Resolving name service...";
nameService = rtorb->resolve_initial_references("NameService");
namingContext = CosNaming::NamingContext::_narrow(nameService);
cout << "done." << endl;

CosNaming::Name robot_name;

robot_name.length(1);
robot_name[0].id = CORBA::string_dup("SERVIDOR_ROBOT_SERVANT");
robot_name[0].kind = CORBA::string_dup("");
```

6. Resolver el nombre a una refrencia al servant.

```
cout << "Resolving servant name...";
CORBA::Object_var object_robot = namingContext->resolve(robot_name);
cout << "done." << endl;
```

7. Obtener el manejador del servant.

```
cout << "Obtaining handle for servant...";

miRobot = Pioneer::Pioneer2AT::_narrow(object_robot);
cout << "done." << endl;
```

4.5.3. Makefile

Una vez escrito el código de la aplicación es necesario compilarlo. Con este objetivo se escribe el archivo Makefile.

Make es una herramienta de generación de código, muy usada en los sistemas operativos tipo GNU/Linux. Lee las instrucciones escritas en el archivo Makefile que reciben el nombre de dependencias. La herramienta *Make* se usa para las labores de creación de fichero ejecutable, la limpieza de los archivos temporales en la creación del fichero...

El Makefile correspondiente al prototipo CORBA es el siguiente:

4.5.4. Ejecución

Una vez generado el binario, para ejecutarlo se debe seguir el proceso siguiente:

1. Lanzar el servidor de nombres (*icans*) para obtener el NamingService.IOR
2. Lanzar la aplicación servidor pasándole como parámetro el IOR del *NameService*
3. Lanzar la aplicación cliente pasándole como parámetro el IOR del *NameService*

El servidor de nombres genera la referencia al objeto (IOR) que es un parámetro que le tenemos que pasar al ejecutable. Depende de varios factores tales como la máquina en la que corra el servidor de nombres y de cuando se haya lanzado el mismo, de modo que es distinto en cada ocasión. Para facilitar la tarea en c3p0 (uno de los ordenadores del departamento) se está ejecutando constantemente la aplicación *icans* (ICa Naming Service) por lo que el IOR del NameService es siempre el mismo y podemos crear *scripts* en los que se indica que se ejecute el binario con el IOR permanente del *NameService* como parámetro.

El NamingService.IOR correspondiente a c3p0 es el siguiente:

```
IOR:010000002800000049444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e674
36f6e746578743a312e3000010000000000000008a000000010101000f0000003133382e313030
2e37362e3234300000d1070000430000000004000000d4e616d655365727669636526f6f74504
f41000049444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f6e74657874
3a312e3000003100010000000200000001a0000000100000001000000280000000a00000001000
0000100000000040000000
```

De modo que el lanzador del cliente queda de la siguiente forma:

```
./ClienteRobotB2 -ORBInitRef IOR=‘cat NamingService.IOR‘
```

Por cuestiones de mantenimiento del robot móvil necesita un nuevo computador abordo para satisfacer las necesidades de otro proyecto que se está desarrollando ahora en el grupo, Higgs no se encuentra operativo en todo momento. Afortunadamente el paquete Aria incluye un simulador llamado “SRIsim” cuya interfaz puede verse en la Figura 4.4 que permite simular la conexión al robot.

El problema que se plantea al no poder conectar directamente el cliente al robot es que los movimientos del mismo en el simulador no se pueden controlar ya que el cliente diseñado solo le pide la información, no lo teleopera en modo alguno, por lo que el simulador no sirve para probar si la cara se anima según las distintas situaciones en las que se pueda encontrar Higgs. Se haría necesaria la creación de otro cliente que nos permitiera provocar en el robot los distintos estados para comprobar que la interfaz diseñada recorre todo el espectro de

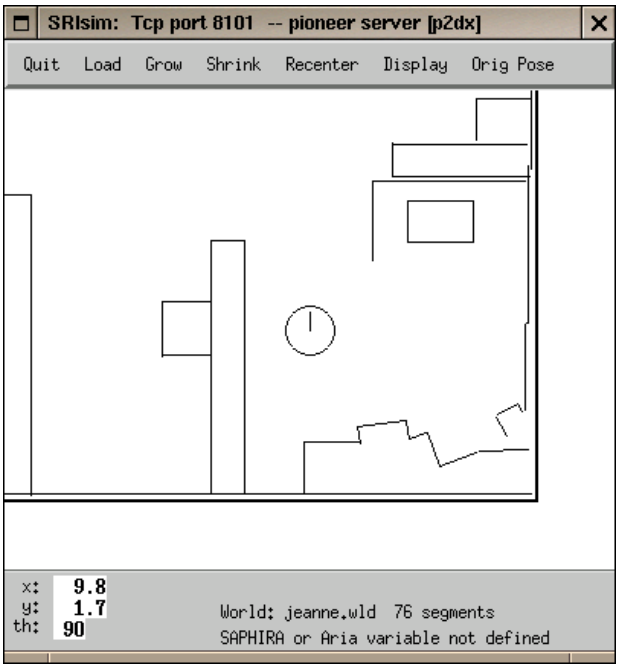


Figura 4.4: Simulador de Aria

situaciones mecánicas posibles y las mapea a los estados emocionales correspondientes. Otra alternativa es crear una aplicación en la que se pudiera simular los estados de Higgs que interesaran en cada caso. Esta segunda es la opción elegida: se crea una aplicación que actúa como banco de pruebas para el módulo **SOUL::Face**.

4.6. Servidor banco de pruebas

Como alternativa se crea un banco de pruebas que actúa como "simulador del simulador", esto es, una interfaz gráfica formada por barras y displays con la que se interactúa por medio del ratón para generar los distintos estados mecánicos con objeto de comprobar que se generan las expresiones adecuadas para cada caso. Esta interfaz se diseña con Qt y se conecta mediante CORBA utilizando las librerias de OmniORB al programa de dibujo gráfico propiamente dicho. Sustituye al robot o al simulador, actuando como servidor y permitiendo la generación de la información sensorial que se utiliza posteriormente en el proceso de mapeo a emociones.

4.6.1. 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.

Un poco de historia. Inicialmente Qt apareció como biblioteca desarrollada por Trolltech[7] 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.

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"[8] (Figura 4.5).

"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 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++).

[7] www.trolltech.com

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

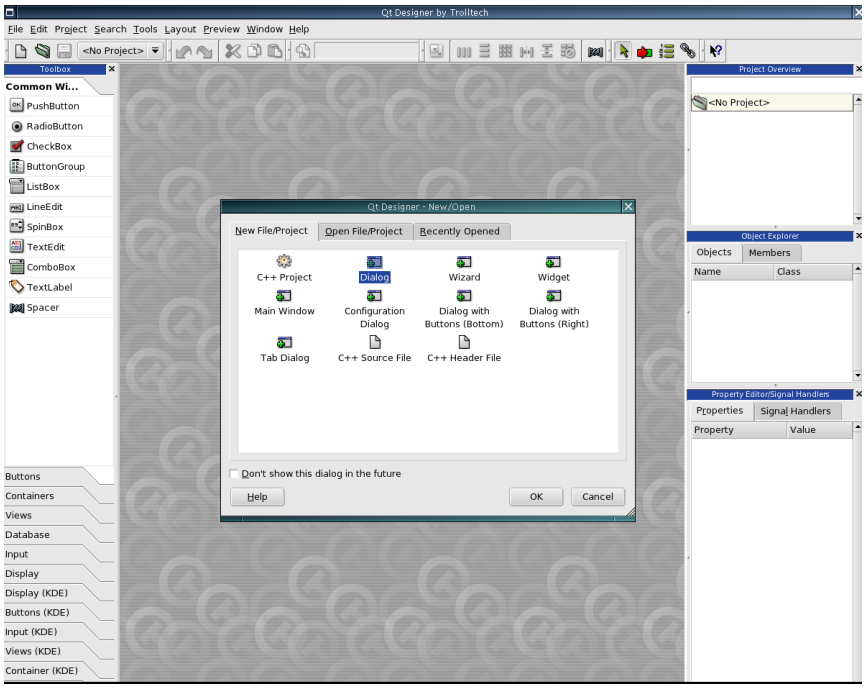


Figura 4.5: Herramienta para Qt: Designer

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 4.6.

Figura 4.6: Banco de pruebas: simulador del simulador

Makefile

El Makefile utilizado para la compilación de la aplicación diseñada con Qt es el siguiente:

Ejecución

Una vez generado el binario se ejecuta y proporciona su IOR (el del servidor, para que los clientes puedan referirse a él directamente sin en *NameService*) que es necesario pasar como parámetro a cualquier aplicación cliente que quiera conectarse al servidor diseñado.

4.7. Aplicación final

La aplicación final se diseña según lo descrito en el capítulo 3 utilizando las librerías de Omni_ORB. En los casos en los que no se conecta al robot se utiliza el banco de pruebas diseñado con Qt que implementa la funcionalidad del servidor que se ejecuta en la CPU de **Higgs** y nos permite generar estados globales del sistema. La equivalencia de estos estados mecánicos a estados emocionales se puede visualizar en un display mediante una cara.

Capítulo 5

Sistemas gráficos

El diseño gráfico es una forma de comunicación visual. Puede aplicarse a muchos medios, ya sean impresos, digitales, audiovisuales...

El mundo del diseño gráfico es un mundo fascinante. Encontramos múltiples ejemplos en el cine, la publicidad, los vídeojuegos... los resultados son sorprendentes.

El presente capítulo se organiza del siguiente modo:

- En la Sección 5.1 se da una visión general de los sistemas gráficos y las distintas aplicaciones que éstos tienen.
- En la Sección 5.2 se describen conceptos generales acerca de los gráficos por ordenador, desde qué son hasta las herramientas disponibles en el mercado para trabajar con ellos, pasando por la descripción de las fases básicas presentes en el proceso de creación de gráficos 3D.
- Las Secciones 5.3, 5.4 describen dos herramientas que se utilizan en el desarrollo del proyecto: Blender y Python. Se recogen en ellas conceptos generales como qué son, algo de su historia y las características principales de cada una de ellas.
- La Sección 5.5 describe el trabajo realizado con las herramientas antes presentadas en el desarrollo del proyecto. Se describe el trabajo realizado con Blender y Python y cómo se ha de abandonar esa vía al no satisfacer las especificaciones del proyecto los resultados que con ellas se pueden obtener.
- La Sección 5.6 describe la forma en la que se ha diseñado finalmente la parte gráfica del proyecto. Empieza con cuestiones generales sobre OpenGL para terminar describiendo la aplicación realizada.

5.1. Introducción a los sistemas gráficos

Los gráficos por ordenador son el campo de computación visual donde se utilizan ordenadores tanto para generar imágenes de forma artificial como para integrar o modificar información espacial y visual recogida del mundo real. Este campo se puede dividir en varias áreas:

- Renderizado 3D en tiempo real (utilizado en juegos)
- Renderizado para creación de vídeos y películas
- Edición de efectos especiales (para películas y televisión)
- Edición de imágenes y modelado (utilizado en ingeniería)

Inicialmente el desarrollo de los sistemas gráficos por ordenador tuvo lugar en entornos académicos. Con el tiempo, su uso en películas y televisión demostró su utilidad para la creación de efectos especiales y de animación con sorprendentes resultados. Desde "2001: Una odisea del espacio" donde los efectos de gráficos por ordenador se creaban mediante animación manual, pasando por "Final Fantasy" que fue un hito en cuanto a animación se refiere por los resultados de elevada calidad obtenidos con sus personajes, hasta llegar a las últimas películas de Pixar, "Buscando a Nemo" o "Los Increíbles".

5.2. Gráficos por ordenador

Los gráficos por ordenador se crean mediante la utilización de software especializado para el diseño. En general, el término "gráficos por ordenador" se utiliza para referirse por una parte al proceso de creación y por otra al estudio de técnicas gráficas para el diseño.

5.2.1. Creación de gráficos 3D

El proceso de creación de gráficos 3D por ordenador se divide en tres fases básicas:

1. Modelado
2. Composición de la escena
3. Renderizado (creación de la imagen final)

Modelado. La etapa de modelado consiste en dar forma a objetos individuales que luego serán usados en la escena. Existen diversas técnicas: modelado poligonal, modelado con NURBS . . . Los procesos de esta etapa pueden incluir la edición de la superficie del objeto o las propiedades del material (color, luminosidad, características de reflexión, transparencia u opacidad), agregar texturas, mapas de relieve (*bump-maps*) y otras características a los objetos. También pueden incluir algunas actividades relacionadas con la preparación del modelo 3D para su posterior animación[1].

El modelado puede ser realizado por programas dedicados (como Lightwave 3D, Rhinoceros 3D o Moray), un componente de una aplicación (Shaper, Lofter en 3D Studio) o por un lenguaje de descripción de escenas (como en POV-Ray).

Composición de la escena. Esta etapa involucra la distribución de objetos, luces, cámaras y otras entidades en una escena que será utilizada para producir una imagen estática o una animación.

La iluminación es un aspecto especialmente importante de la composición de la escena: contribuye al resultado estético y a la calidad visual del trabajo.

Renderizado. Se llama *render* al proceso final de generar la imagen o animación a partir de la escena creada. El proceso de render necesita una gran capacidad de cálculo, pues requiere simular muchos procesos físicos complejos. La capacidad de cálculo que se puede realizar en un PC se ha incrementado rápidamente a través de los años, permitiendo un grado superior de realismo en los gráficos 3D.

5.2.2. Animación

Por animación se entiende la creación de imágenes en movimiento. Para crear la "ilusión de movimiento" se muestra una imagen en la pantalla del ordenador que es rápidamente sustituida por una nueva imagen, similar a la anterior pero ligeramente modificada. Para engañar a la vista y al cerebro, haciéndoles creer que se está viendo un objeto en movimiento, las imágenes deben dibujarse a 24 frames por segundo o más rápido (un "frame" es un imagen completa). Por encima de 70 frames/segundo no mejora el realismo por el modo en el que la vista y el cerebro procesan las imágenes. Por debajo de 24 frames/segundo la mayoría de la gente percibe extraños fenómenos disminuyendo el efecto de realismo. Por ejemplo, en los dibujos animados convencionales utilizan 12 frames/segundo para reducir el número de dibujos. Sin embargo, los gráficos por ordenador necesitan de tasas mayores para mejorar el realismo de la imagen.

En animación es muy frecuente utilizar la técnica de *doble buffering* para evitar el *flick-*

[1] A los objetos se les puede asignar un esqueleto de modo que el movimiento del mismo afecte automáticamente al modelo.

ering (parpadeo) originado porque la pantalla se borra y se redibuja de forma constante. El proceso que se sigue es el siguiente: cuando la imagen se renderiza se hace en el llamado buffer secundario, en el que el ordenador dibuja la imagen haciendo los cambios necesarios antes de mostrarla. Mientras tienen lugar los cambios, lo que se muestra en pantalla es el contenido del buffer primario. Cuando la imagen se completa, se dibuja en pantalla el contenido del buffer secundario. Este proceso puede hacerse de dos formas:

- el contenido del buffer secundario se copia en el primario
- se intercambian los buffers primario y secundario (de forma que el primario pasa a secundario y viceversa)

Este cambio debe hacerse de forma imperceptible para el usuario o de otro modo se produce la llamada ruptura de la imagen, que disminuye el efecto perseguido por la animación, es decir, la "ilusión de movimiento".

5.2.3. Herramientas

Para el diseño gráfico en sus distintas fases(modelado, animación...) existen en el mercado multitud de herramientas:

- Herramientas especializadas en ciertos procesos como por ejemplo:
  - modelado: Rhino3D, Wings3d ...
  - texturizado: UVMapper, DeepPaint ...
  - render: YaFray, 3DLight ...
  - animación: Pose, AnimationMesher ...
- Entornos integrados para todos los procesos, muy caros en general. Este es el caso de los conocidos 3DStudio Max, SoftImage, Maya o LightWave. Existe también alguna alternativa libre a los mismos como es el caso de Blender.

5.3. Blender

Blender es un programa de modelado y animación libre (gratuito), con características como soporte para programación bajo Python. Es mucho menos conocido que otros programas para animación 3D, como 3DStudio, Lightwave o Maya, pero es una herramienta de gran potencia para el diseño gráfico.

Un poco de historia

Originalmente el programa fue desarrollado como una aplicación propia por el estudio de animación holandés NeoGeo; el principal autor, Tom Roosendaal, fundó la empresa "Not a Number Technologies" (NaN) en junio de 1998 para desarrollar y distribuir el programa.

La compañía llegó a la bancarrota en 2002 y los acreedores acordaron ofrecer Blender como un producto de código abierto y gratuito bajo los términos de la GNU GPL a cambio de 100.000€. El 18 de julio de 2003, Roosendaal creó la "Fundación Blender" (sin ánimo de lucro) para recoger donaciones; el 7 de septiembre se anunció que se habia reunido el dinero necesario para liberar Blender y el código fuente se hizo público el 13 de octubre.

Por tanto, Blender es Open Source, es decir, que el código fuente del programa está disponible para examinarlo y generar mejoras a este para su inclusión en las nuevas versiones[2]. La última versión, Blender 2.36 (Figura 5.1), está disponible en la web[3] para instalar en los distintos sistemas operativos (tiene versiones para Windows, Linux, Irix, Sun Solaris, FreeBSD y Mac OS X bajo la GNU Public License).

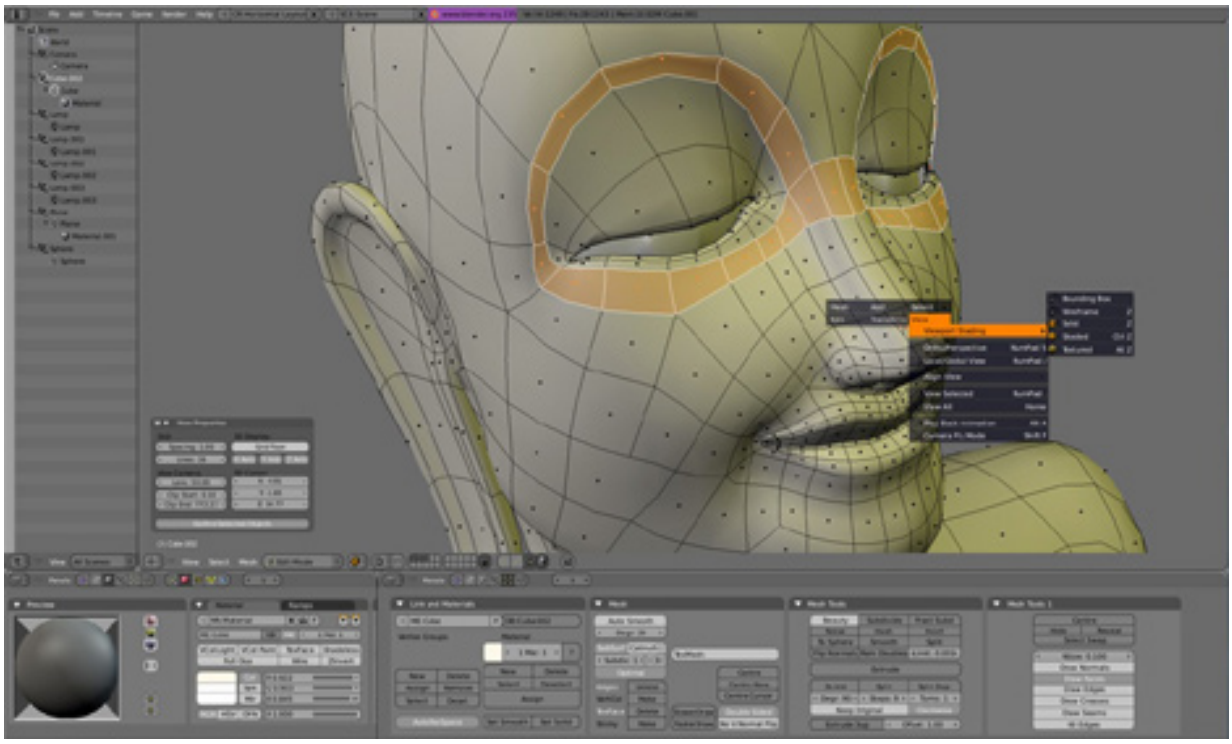


Figura 5.1: Captura de pantalla de Blender 2.36

Como ejemplo de su buen estado se puede señalar que se ha utilizado como una de las herramientas de animación para la película "Spiderman 2".

[2]Más información sobre la evolución de blender en `www.blender.org`

[3]`www.blender3d.org`

Características

Las características principales de esta herramienta de diseño gráfico son:

- Es multiplataforma, libre, gratuita y con un tamaño de origen realmente pequeño (alrededor de 5 MB dependiendo del sistema operativo)
- Permite utilizar una gran variedad de primitivas geométricas, incluyendo curvas, mallas poligonales, NURBS, metaballs (objetos de tres dimensiones con características físicas del mercurio)
- Junto a las herramientas de animación se incluyen cinemática inversa, deformaciones por armadura o cuadrícula, vértices de carga y partículas estáticas y dinámicas.
- Permite la edición de audio y sincronización de vídeo.
- Posee características interactivas para juegos (detección de colisiones, recreaciones dinámicas y lógica)
- Posibilita un renderizado interno versátil y la integración externa con el potente trazador de rayos libre de YafRay.
- Mediante el lenguaje Python se pueden automatizar o controlar distintas tareas
- Acepta formatos gráficos como TGA, JPG, Iris, SGI, o IFF. También puede leer ficheros Inventor.

5.4. Python

Python es un lenguaje de programación de *scripts* (*script* es el nombre que se da a un archivo, o conjunto de archivos, que contiene instrucciones, y que necesita de un programa intérprete para ejecutarse). Permite dividir el programa en módulos reutilizables desde otros programas. Dispone de una gran colección de módulos estándar que se pueden utilizar como base de los programas (o como ejemplos para empezar a aprender Python). También posee otros que proporcionan E/S de ficheros, llamadas al sistema, *sockets* y hasta interfaces GUI (Interfaz Gráfica con el Usuario).

Un poco de historia

Python fue creado como sucesor del lenguaje ABC por Guido van Rossum en 1990 cuando trabajaba en el Stichting Mathematisch Centrum (CWI)[4]. En 1995 continuó su

[4] `www.cwi.nl`

trabajo en Python en el CNRI[5] donde creó muchas versiones del lenguaje. En mayo del 2000 Guido y el equipo de desarrollo de Python se trasladan a BeOpen.com y se forma el BeOpen PythonLabs. En octubre de este mismo año, PythonLabs pasa a Digital Creations (actualmente Zope Corporation). En 2001, se crea la "Python Software Foundation" (PSF)[6], una organización sin ánimo de lucro que surge específicamente para proteger la libertad de Python como lenguaje de código abierto.

El nombre del lenguaje proviene de la afición de su creador original por los humoristas británicos Monty Python. El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura como de diseño.

Características

Las características principales de Python son:

- Es un lenguaje de alto nivel sencillo, potente y rápido que está en constante crecimiento[7].
- Es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa pues no es necesario compilar ni enlazar. Se dice que un lenguaje es interpretado si sus instrucciones se ejecutan secuencialmente a partir del código fuente (el intérprete va recibiendo líneas de código que traduce a código máquina para que se ejecuten).
- El intérprete se puede utilizar de modo interactivo. Esto facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa. También es una calculadora muy útil.

5.5. Trabajo con Blender y Python

5.5.1. Criterios de selección

Se eligió Blender como herramienta para el diseño y la animación del modelo facial que es objeto de este proyecto por tres motivos fundamentales:

- Es libre (se dsitribuye bajo la GNU Public License)
- Es multiplataforma (existen versiones para Windows, Linux, Mac OS X...)

[5] `www.cnri.reston.va.us`
[6] Más información en la página web `www.python.org/psf/`
[7] Más información en `www.python.org`

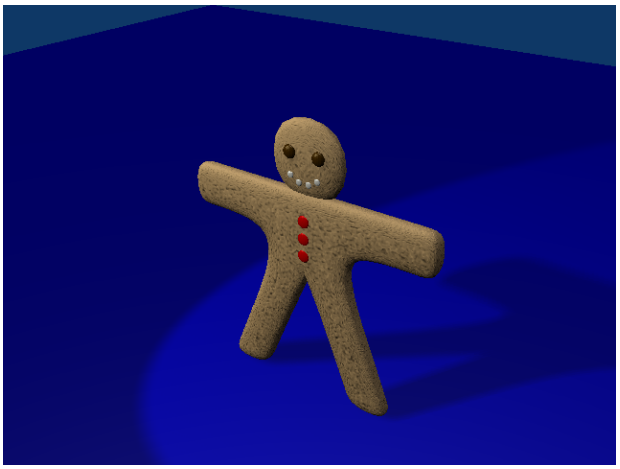


Figura 5.2: Ejemplo de modelado con Blender: “Ginger”

- Es un entorno integrado para la animación gráfica (con la misma herramienta podemos trabajar en las distintas etapas del diseño grafico: modelado, texturizado y animación)

5.5.2. Prototipo Blender

* **Aprendizaje de la herramienta**. Durante el desarrollo de este proyecto la versión de Blender ha cambiado varias veces, añadiendo prestaciones a la herramienta. La más interesante de ellas desde el punto de vista de la animación es que el motor de juegos (*game engine*), congelado desde la versión 2.28, volvió a formar parte de Blender a partir de la 2.34 (agosto de 2004). El *game engine* permite la interacción con el programa en tiempo real y esto es una característica necesaria para la implementación del software que es objeto de esta memoria.

  El objeto a crear es una cara antropomórfica que posteriormente habrá de animarse, pero el primer paso en este proceso es familiarizarse con la herramienta. Blender es una herramienta muy potente con un entorno de ventana muy distinto del usual, muy distinto incluso de los entornos de ventana de otras aplicaciones similares dedicadas al diseño gráfico. Por este motivo antes de empezar con la construcción del modelo de la cara se construyeron otros elementos más simples. Como apoyo en este proceso de aprendizaje se utilizaron los recursos disponibles en la red así como la “Guía oficial de Blender” [Roosendaal and Selleri, 2004]. Una de las creaciones de esta etapa de aprendizaje puede verse en la Figura 5.2.

  Una vez alcanzado el dominio suficiente de la herramienta se comenzó con el diseño de la cara antropomórfica para el proyecto.

* **Modelado**. Se construyó una cara mediante puntos utilizando como soporte dos imagenes de una cara, una de frente y otra de perfil (Figura 5.3). El motivo por el

cual se utilizaron como referencia fue por cuestión de proporciones ya que el objetivo es crear una cara antropomórfica de apariencia normal. Aprovechando la simetría de la cara se modela la mitad y mediante proyección especular generamos la otra mitad. Los puntos que definen la cara se unen formando triangulos o cuadrados (la forma óptima para facilitar el posterior proceso de renderizado es la triangular). Cuantos más puntos, más polígonos y mayor definición. El resultado final del proceso de modelado esta formado por unos 500 puntos y puede verse en la Figura 5.4.

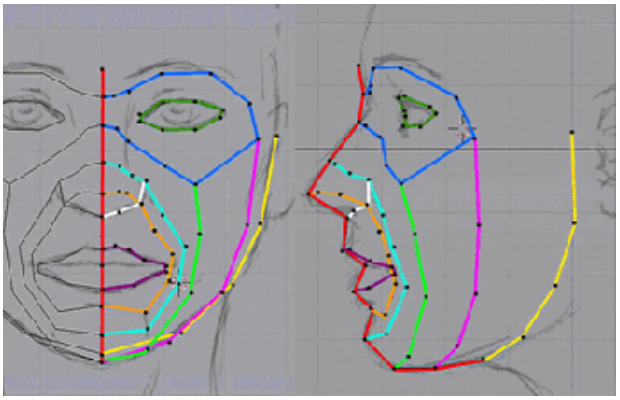


Figura 5.3: Imágenes auxiliares para el diseño de la cara

* **Composición**. Es momento de determinar la distribución de los objetos, las luces y las cámaras en la escena. El único objeto de la escena es la cara creada en el proceso de modelado. El tema de iluminación es muy importante para el realismo de la imagen generada. Blender dispone de distintos tipos de luces y de un menú en el que definir los parámetros de las mismas (ver Figura 5.5). Es necesario poner en la escena al menos una luz y una cámara ya que de otro modo no se vería nada al renderizar en el paso siguiente. La escena por defecto de Blender trae una luz y una cámara (ver Figura 5.7), que nos sirven para la escena sencilla que se está creando.

* **Renderizado**. Después del proceso de modelado y composición se puede generar la imagen final, es decir, se puede renderizar la escena. El menú de Blender correspondiente al renderizado permite definir algunas opciones tales como el tamaño de la ventana en la que veremos el resultado (Figura 5.6). Una vez seleccionado como se quiere que se realice el *render*, se ejecuta y se abre una ventana con la escena renderizada.

* **Dificultades en la creación gráfica con Blender**.

  Blender tiene una peculiar interfaz gráfica de usuario como puede verse en la Figura 5.7. Es poco intuitiva pero al mismo tiempo muy eficiente y productiva una vez se llega a conocer. El usuario se comunica con el programa vía teclado y ratón (como Blender hace un uso intensivo de de los mismos, existe una "regla de oro" común entre los usuarios de Blender: *¡mantén una mano sobre el teclado y la otra sobre el*

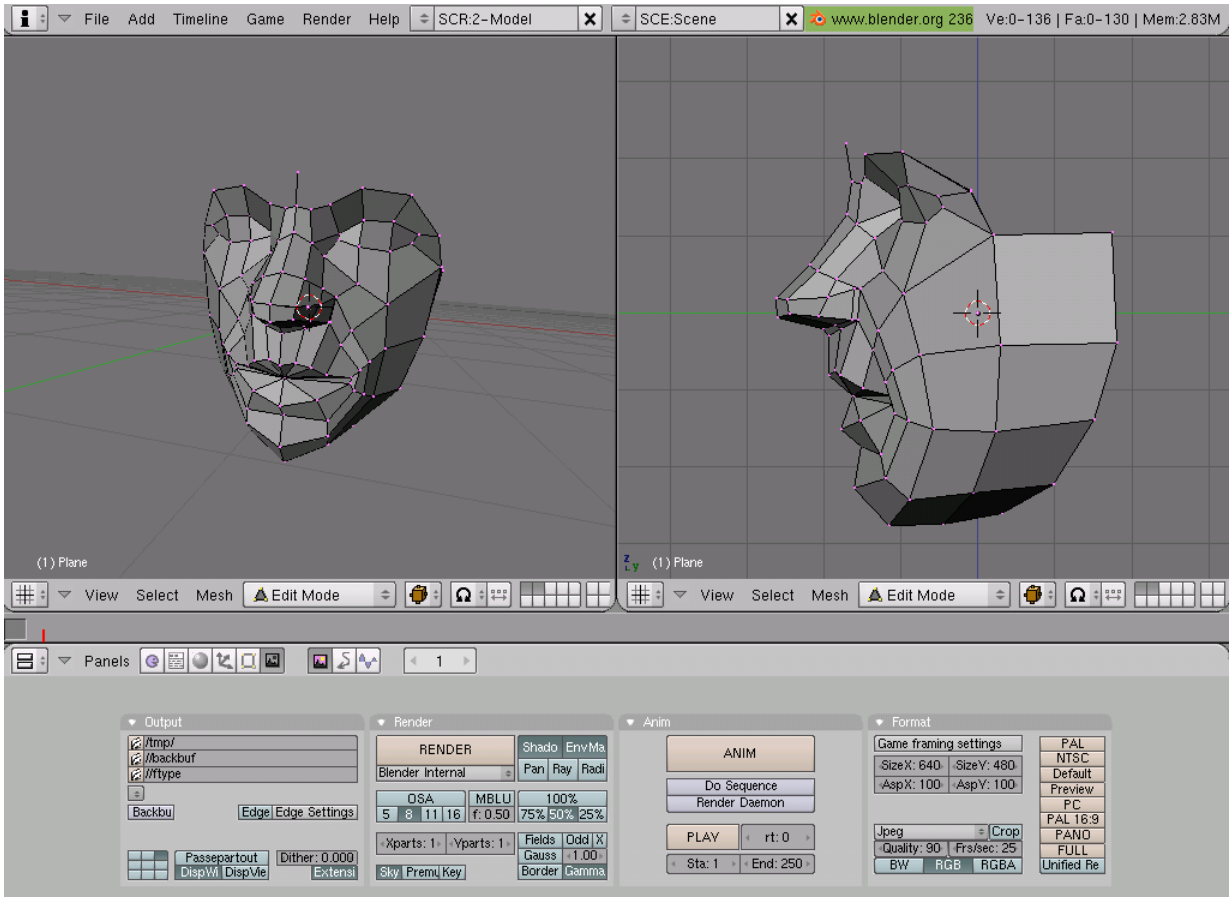


Figura 5.4: Diseño con Blender de una cara antropomórfica

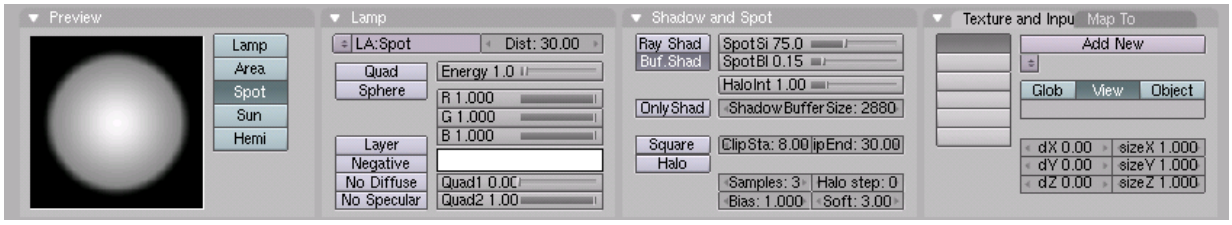


Figura 5.5: Menú para la iluminación de Blender

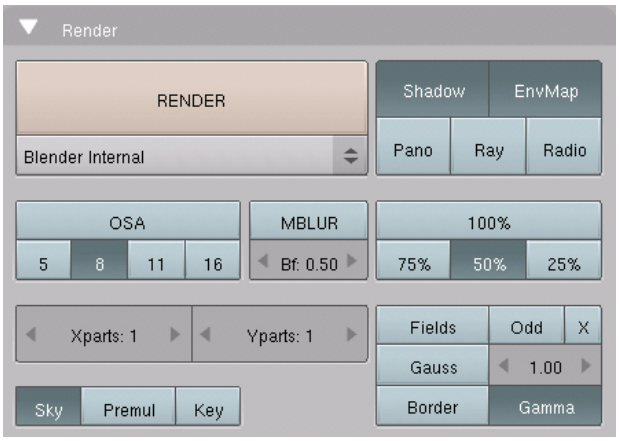


Figura 5.6: Menú para el renderizado de Blender

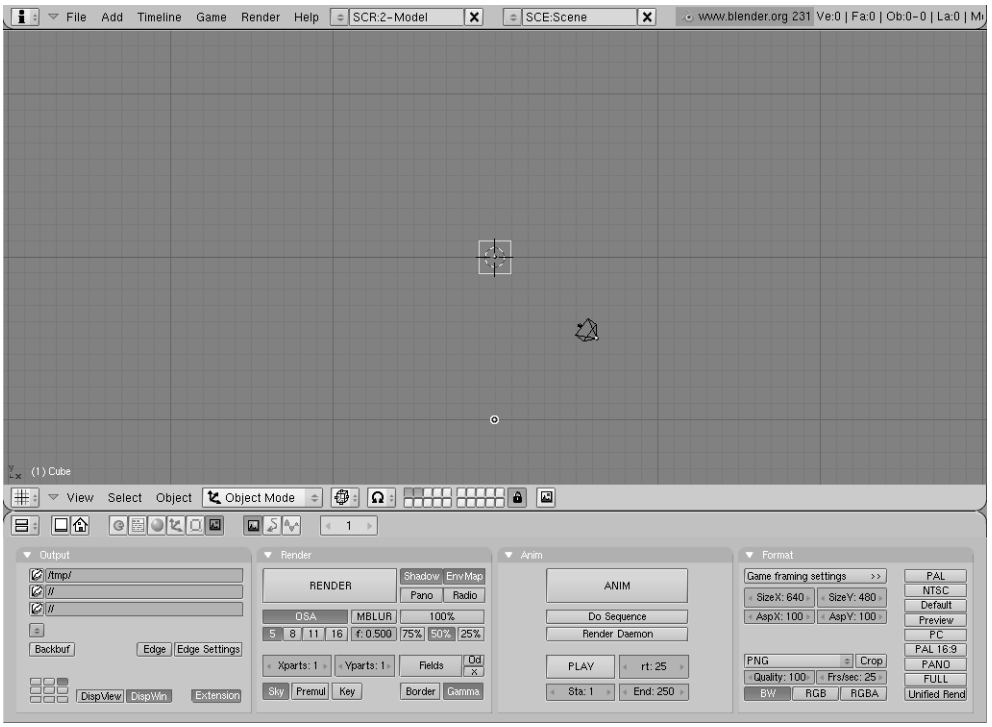


Figura 5.7: Escena por defecto de Blender

ratón!). La interfaz de Blender hace uso de los tres botones del ratón y de un amplio rango de teclas rápidas.

Los resultados de elevada calidad[8] hacen que el esfuerzo en dominar la herramienta esté recompensado ampliamente por los resultados que se pueden obtener.

Si se quiere "deshacer" una acción, Blender ofrece esa posibilidad (Mesh Undo), pero

[8]En `www.blender3d.org` se encuentra una galería de imágenes creadas con Blender

Figura 5.8: Algunas opciones de Blender

sólo funciona para mallas en modo edición y para una única malla, siempre y cuando se esté trabajando con ella. Esto dificulta el proceso de modelado ya que hay errores que una vez cometidos no pueden subsanarse y la única opción en estos casos es volver a una versión guardada del trabajo o volver a empezar.

Otra característica de esta herramienta es que no solicita confirmación para cerrar la ventana de dibujo, de modo que si no se ha guardado previamente el trabajo realizado se perderían todas las modificaciones posteriores realizadas tras "salvar" por última vez. Esto tiene sus ventajas en las ocasiones en las que se comete un error que no se puede deshacer, pero en la mayoria de los casos, esta peculiaridad del programa es un inconveniente.

5.5.3. Animación

Dejando a un lado los procesos de texturizado (le dan realismo a la imágen pero es secundario para los objetivos que persigue este proyecto), se pasa al proceso de animación. Debido a que el objetivo del proyecto es que la cara se mueva según lo requiera un programa externo a Blender es necesario que utilicemos Python como interfaz entre ellos. Ha de usarse de forma conjunta al *game engine* ya que, como se explicó antes, es el único modo que existe para que podamos interactuar con Blender en tiempo real. Python no sólo se utiliza en el *game engine*, también sirve para modelar y animar.

* **Game engine** . Es el motor de juegos de Blender. Permite la interacción con el programa en tiempo real. En el *game engine* existen tres partes: los sensores, los controladores y los actuadores. Haciendo un símil se puede decir que los sensores actúan como los sentidos, los controladores como el cerebro y los actuadores como los músculos. De modo que los sensores reaccionan ante un evento, los controladores recogen los eventos de los sensores y calculan el resultado, y los actuadores realizan la acción sobre los objetos. Existen distintos tipos de sensores ("always", teclado, ratón...), de controladores (AND, OR, expresiones, scripts de Python[9]) y actuadores (movimiento, propiedades, cámara, sonido ...).

  El modo en que actua el *game engine* es el siguiente: se le asocia un sensor a un objeto. Dependiendo del tipo de sensor elegido se disparará en un momento o en otro (el "always" cada cierto tiempo, el "keyboard" cuando se produzca un evento por teclado ...). Relacionado con ese sensor existe un controlador que se une al actuador (Figura 5.9).

Figura 5.9: Menú *game engine* de Blender

* **Python**. El *script* de Python se encarga de dos cosas: por un lado de interactuar con Blender y por el otro de recoger los datos que indican cómo debe cambiar la imagen. Es necesario comunicar Python con otro programa (el cliente CORBA descrito en el Capítulo 3) que se ejecuta dentro de la misma máquina. Se eligen los *sockets* para la comunicación de ambos. Se crea una estructura cliente/servidor. El cliente es el programa en Python y el servidor es el programa CORBA (que está escrito en C++).

* **Proceso de animación**. Para el proceso de animación es necesario unir tres elementos: el servidor CORBA que indica qué expresión poner, el cliente Python que lo recoge y se lo comunica al tercer elemento, Blender, mediante un *script* actuando como controlador en el *game engine*. Para conectar cliente y servidor se utilizan *sockets*.

  Existen muchos tipos de *sockets*, los de uso mas extendido son los *sockets unix* y los *inet*. Como los dos programas se ejecutan en la misma máquina en principio pueden usarse los *sockets unix* que son más rápidos que los *inet*, pero debido a que Python no posee implementación para esta clase de *sockets* se trabaja con los *inet*. Se crean prototipos de cliente/servidor en C++ y en Python, para en el último momento desarrollar la aplicación mixta en la que el cliente es Python y el sevidor es C++.

[9]Permiten generar un comportamiento más completo

* **Dificultades para la animación con Blender y Python**. De modo similar al proceso de modelado, lo primero es explorar las características del *game engine* para poder utilizarlo para los objetivos del proyecto. Se realizaron pruebas con figuras sencillas. Comenzaron a surgir problemas ya que las operaciones que se pueden realizar a los objetos con el *game engine* son limitadas: no todas las operaciones que se pueden hacer con los objetos tienen su equivalente para el *game engine* (como por ejemplo, girar un objeto según los ángulos de Euler). Esta funcionalidad de Blender está pensada para el diseño de juegos, por lo que viene provista de funciones físicas tales como gravedad o colisiones entre objetos. Las pruebas con objetos sencillos no daban los resultados esperados, las operaciones se complicaban, por lo que en este punto se abandonó el camino basado en Blender para el desarrollo de la aplicación. Blender es una herramienta muy potente pero no es lo suficientemente flexible, para el desarrollo completo de la aplicación.

5.5.4. Conclusiones

Como quiera que Blender es una herramienta de diseño gráfico integrada cuya misión es facilitar la creación de gráficos que no se adapta a las necesidades de este proyecto y que no es más que un programa diseñado sobre la librería de diseño grafico por excelencia (OpenGL), el camino que toma el proyecto a partir de este punto es utilizar para el diseño y la animación OpenGL directamente.

5.6. OpenGL

Debido a los problemas encontrados al realizar el modelado y animación de la aplicación objetivo del presente proyecto descritas en la sección anterior, se decide recurrir directamente a OpenGL para resolver el problema planteado. Se descarta recurrir a otra herramienta gráfica ya que, tras la experiencia con Blender, es claro que esta aplicación tiene unas características muy especificas que son poco comunes en el mundo del diseño grafico (interactuar en tiempo real utilizando otra aplicación como fuente de eventos ante los que reaccionar).

OpenGL es una librería de funciones que permite visualizar gráficos 3D en nuestra aplicación. La gran ventaja de esta librería es que aisla la aplicación del hardware disponible; por tanto, si se dispone de una tarjeta aceleradora, no hará falta cambiar el programa para aprovechar la potencia de la tarjeta.

Librerías adicionales de OpenGL

La librería principal de OpenGL suministra todas las funciones necesarias para mostrar un entorno 3D aunque hay algunas operaciones que son algo tediosas de realizar utilizando

solo esta librería. Para esto existen también unas librería auxiliares:

- La librería *GLU (OpenGL Utility Library)* es una librería de utilidades que proporciona acceso rápido a algunas de las funciones más comunes de OpenGL, a través de la ejecución de comandos de más bajo nivel pertenecientes a la librería OpenGL propiamente dicha.
- La librería *GLUT (OpenGL Utility Toolkit)* es un paquete auxiliar que proporciona una interfaz independiente de la plataforma para crear aplicaciones de ventanas totalmente portables. GLUT nos permite crear ventanas y controlar la entrada independientemente de la plataforma utilizada
- La librería *GLAUX*, muy parecida a *GLUT*, es la que Microsoft ha desarrollado para Windows. Mantiene practicamente la misma estructura que la librería *GLUT* con el defecto de que solo es aplicable para Windows, mientras que GLUT sirve para cualquier plataforma.
- La librería *GLX* es la utilizada para trabajar en un sistema de X-Windows (GNU/Linux), permite no sólo renderizar en la máquina local, sino tambien a traves de red.

También hay otras librerías más específicas para el control de entrada, sonido, red.... (OpenGL es una librería gráfica, no fue diseñada para eso).

OpenGL está formado por aproximadamente 250 comandos de los cuales unos 200 corresponden al núcleo de OpenGL y los restantes a *GLU*. *GLUT* proporciona una API que facilita enormemente el aprendizaje de la programación con OpenGL. Se recomienda su uso para aquellas aplicaciones que no requieran interfaces muy sofisticadas ya que para esos casos es mejor utilizar las herramientas del sistema de ventanas que estemos utilizando (para Windows se usa la libreria *GLAUX* y para X-Windows la libreria *GLX*). *GLUT* no es OpenSource, su creador Mark Kilgard sigue manteniendo el copyright. Es necesario descargar e instalar el paquete para poder utilizarlo. Al descomprimirlo aparece un archivo "readme" en el que se detallan los pasos a seguir para su instalación dependiendo de la plataforma en la que se esté trabajando. La versión de *GLUT* utilizada en el presente proyecto es la 3.6.

Un poco de historia

OpenGL es un estándar sobre gráficos por ordenador, hoy día es uno de los más conocido del mundo. En 1982 nació en la Universidad de Standford el concepto de *graphics machine* y este fue utilizado por Silicon Graphics Corporation en su propia estación Silicon IRIS para crear un renderizador. Así nacio la librería IRIS GL. A raíz de esto, en 1992, muchas empresas del hardware y software se pusieron deacuerdo para desarrollar conjuntamente una librería gráfica libre: OpenGL. Entre estas empresas destacaban Silicon Graphics Inc.,

Microsoft, IBM Corporation, Sun Microsystems, Digital Equipment Corporation (DEC), Hewlett-Packard Corporation, Intel e Intergraph Corporation. Así nació OpenGL (Open Graphics Library).

Características

Entre las características de OpenGL podemos destacar que:

- Es multiplataforma.
- La gestión de la generación de gráficos 2D y 3D se realiza por hardware ofreciendo al programador una API sencilla, estable y compacta.
- Su escalabilidad ha permitido que no se haya estancado su desarrollo, permitiendo la creación de extensiones, una serie de añadidos sobre las funcionalidades básicas, para aprovechar las crecientes evoluciones tecnológicas[10].
- Las operaciones se puden agrupar en jerarquias, lo que proporciona gran potencia y flexibilidad a la hora de programar.
- OpenGL funciona como una máquina de estados, por lo que el orden en el que se realizan las operaciones es importante. El orden de las acciones resulta crítico en la mayoria de las ocasiones. Sus variables de estado tienen unos valores por defecto que podemos modificar para adaptarlos a la aplicación que estemos desarrollando. Inicialmente la mayoria de estas variables de estado están desactivadas. Activarlas supone que el renderizado sea más lento pero más realista. Es necesario un compromiso entre ambos para obtener los resultados deseados.

Un programa en OpenGL puede ser complicado, la estructura básica puede sintetizarse en dos ideas:

- Inicialización de ciertos estados que controlan como renderiza[11] OpenGL
- especificar los objetos que han de renderizarse

OpenGL Rendering Pipeline

La mayor parte de las implementaciones de OpenGL siguen un mismo orden en las operaciones, que en su conjunto crean lo que se suele llamar "OpenGL Rendering Pipeline" (Figura 5.10).

[10] Más información acerca de OpenGL en `www.opengl.org`

[11] Renderizado es el proceso en el cual el ordenador crea imágenes de modelos u objetos. Las primitivas geométricas con las que se construye cualquier objeto en OpenGL son puntos, líneas y polígonos y vienen definidos por sus vértices

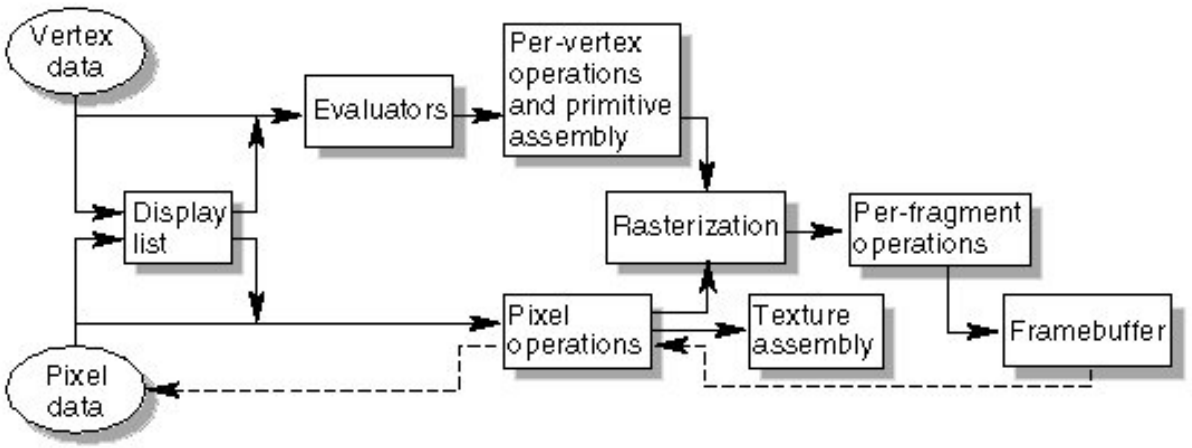


Figura 5.10: OpenGL Rendering Pipeline

Sobre el *vertex data* se pueden aplicar *evaluators* para describir curvas o superficies parametrizadas mediante puntos de control. Luego se aplican las *pre-vertex* operations que convierten a los vertices en primitivas. Aquí es donde se aplican las transformaciones geométricas como rotaciones, traslaciones, etc. por cada vértice. En la sección *primitive assembly* se elimina lo que queda fuera del plano de proyección.

Por la parte de *pixel data* están las *pixel operations*. Aquí los pixels son desempaquetados desde algún array del sistema (como el *framebuffer*) y tratados (escalados, etc.). Luego, si estamos tratando con texturas, se preparan eln la seción *texture assembly*.

Ambos caminos convergen en la *rasterization* donde son convertidos en fragmentos. Cada fragmento será un pixel del *framebuffer*. Aquí es donde se tiene en cuenta el modelo de sombreado, la anchura de las líneas o el antialiasing.

En la última etapa, las *perfragment operations*, es donde se preparan los texeles (elementos de textura) para ser aplicados a cada pixel, la "fog" (niebla), el z-buffering, el blending, etc. Todas estas operaciones desembocan en el *framebuffer*, donde obtenemos el render final.

Iluminación

La iluminación en OpenGL permite crear escenas más realistas, se basa en luces y materiales. Una luz es una fuente de iluminación sobre la escena, emite un haz de un color determinado, dividido en las tres componentes de color RGB. Un material determina la cantidad de cada color que refleja un objeto determinado.

Una fuente de luz es la entidad que aporta luz a la escena. Una fuente puede emitir distintos tipos de luz, que son complementarios y pueden emitirse a la vez por la misma fuente. Los tipos de luz son:

- emitida: luz emitida por un objeto. No se ve afectada por ningún otro tipo de luz.
- difusa: es la luz que incide sobre un objeto y proviene de un determinado punto. La intensidad con la que se refleja en la superficie del objeto puede depender del ángulo de incidencia, dirección, etc. Una vez incida sobre un objeto se refleja en todas las direcciones.
- especular: es la luz que al incidir sobre un objeto, se ve reflejada con un ángulo similar al de incidencia. Es la luz que produce los brillos.
- ambiental: a esta categoría corresponde el resto de luz que aparece debido a la reflexión residual de la luz que ya se ha reflejado sobre muchos objetos, y es imposible determinar su procedencia. Es algo así como la iluminación global de la escena.

La iluminación con OpenGL se calcula a nivel de vértice, es decir, por cada vértice se calcula su color a partir del material activo, las luces activas y cómo estas luces inciden sobre el vértice. Se emplean las normales de los vertices[12] (vector perpendicular a la cara que se esta dibujando). Las normales son salientes por la parte delantera de la cara. Por defecto la parte delantera de la cara es aquella en que sus vertices se dibujan en sentido antihorario (aunque esto puede cambiarse modificando el valor de la variable de estado correspondiente).

Z-Buffer

El z-buffer o buffer de profundidad es un algoritmo que sirve para determinar los puntos que se dibujan en la pantalla. Cada vez que se va a renderizar un pixel, comprueba que no haya dibujado antes en ese pixel una posición que este más cerca repecto a la cámara. Este algoritmo funciona bien para cualquier tipo de objetos: cóncavos, convexos, abiertos y cerrados.

5.7. Trabajo con OpenGL

Para el modelado de la cara, teniendo en cuenta que debe posiblitarse la animación posterior, se utiliza como base el denominado "código de Waters". El modelo de Waters se basa en la animación utilizando músculos como describe la Sección 2.5.

La aplicación se construye basandose en el trabajo desarrollado por Varian Qua [Qua, 2002]. Utiliza el código de Waters para desarrollar una aplicación llamada *"Facial Simulator"* sobre Windows utilizando las librerías MFC de Microsoft. Esta aplicación consiste en una interfaz gráfica formada por dos partes: una cara y los controles para cambiarla por medio

[12]Excepto para renderizar NURBS y otras funciones gráficas de alto nivel, OpenGL no genera automáticamente las normales, de modo que hay que especificarselas

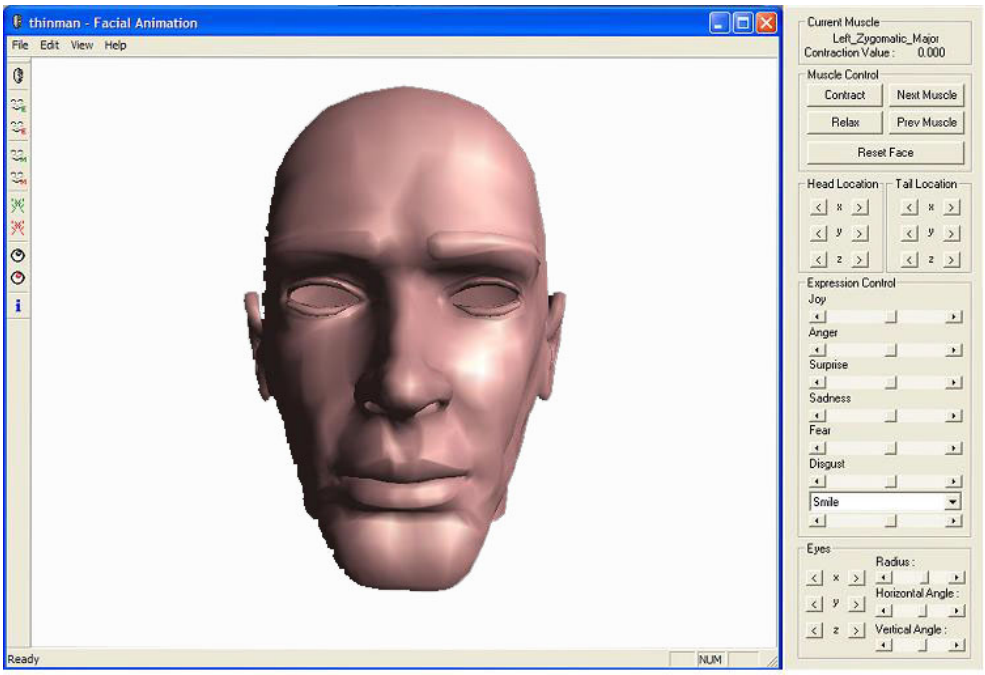


Figura 5.11: Aplicación *"Facial Simulator"* para Windows

del ratón para generar distintas expresiones en la misma. Esta interfaz puede verse en la Figura 5.11.

La reutilización que este proyecto hace del código consiste en uso de la estructura en que se organiza el programa y las funciones generales para dibujar vertices, calcular normales... rediseñando la parte de OpenGL para que no dependa del sistema de ventanas en el que se ejecute la aplicación utilizando exlusivamente las librerias *GL*, *GLU* y *GLUT*. También es necesario diseñar de nuevo toda la parte de animación para adecuarla a los objetivos de este proyecto.

La aplicación está construida de tal modo que la cara y los músculos son datos que están en un archivo. El programa abre los archivos, lee los puntos y los dibuja en la pantalla utilizando los comándos de OpenGL. De este modo se podrá utilizar este programa para generar caras distintas que vienen determinadas por dos archivos: uno en el que se especifica la posición de los puntos de la cara y otro en el que se especifica la posición y zona de influencia de los músculos sobre la misma. De modo que podemos disponer de múltiples caracteres definidos cada uno de ellos por dos archivos.

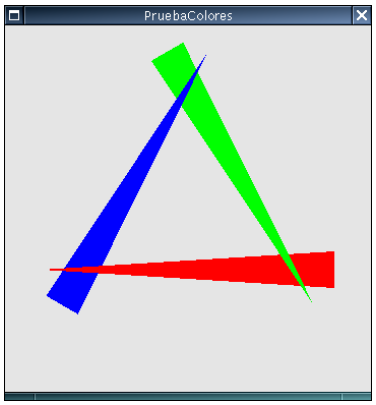


Figura 5.12: Ejemplo con OpenGL

5.7.1. Creación gráfica

* **Aprendizaje de la herramienta**. El primer paso para poder desarrollar la aplicación es familiarizarse con la filosofía y sintaxis que utiliza OpenGL. Los recursos utilizados en esta etapa fueron el "Redbook" [Woo et al., 1999] y los tutoriales de NeHe[13].

  Se comenzó dibujando cosas sencillas para luego ir complicando la escena. Lo primero que se hizo fue aprender a dibujar una ventana. Después se dibujaron formas y se les aplicó color (un ejemplo puede verse en la Figura 5.12). También se dibujaron figuras y se rotaron. Por último se hicieron pruebas para aplicar texturas.

  OpenGL es una máquina de estados y, como se ha indicado antes, el orden en el que se realizan las operaciones es crítico en muchos casos. Un esquema sencillo que deben seguir los programas en OpenGL es:

  1. Activar las opciones que van a ser persistentes en la escena (poner la cámara, activar la iluminación global . . .)
  2. Activar las opciones que establecen el estado de un objeto específico como la posición en el espacio o la textura.
  3. Dibujar el objeto.
  4. Desactivar las acciones propias de ese objeto (volver a la posición inical, desactivar su textura . . .) para que no afecten a objetos que se dibujen posteriormente.
  5. volver al punto 2 hasta haber dibujado todos los objetos.

  Concluido este proceso de aprendizaje se pasa al modelado de la cara, composición de la escena y su posterior animación.

[13] http://nehe.gamedev.net, página de referencia obligada para OpenGL

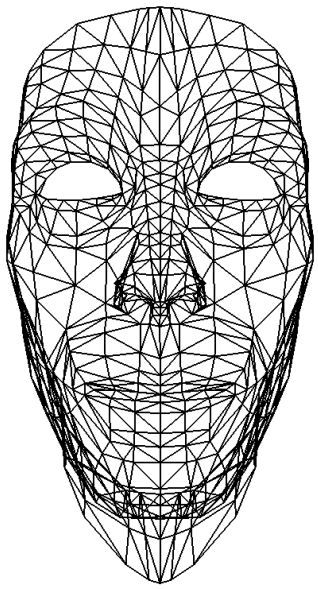


Figura 5.13: Modelado de la cara con líneas

* **Modelado**.

  El objeto que se está modelando para esta aplicación es una cara antropomórfica. Deben posibilitarse también las posteriores tareas de animación de la misma (como ya se indicó en la Sección 2.4 del presente documento). Por eso en esta parte se habla de dos funciones, la que dibuja los polígonos de la cara y la que se encarga de dibujar la estructura muscular (que es la que se utiliza para animar la cara).

  Los prototipos de estas funciones son: `void paint_polygons (HEAD *face, int type, int normals);` y `void paint_muscles (HEAD *);`. Se encargan de leer los archivos correspondientes y dibujar mediante las primitivas de OpenGL la información que proporcionan en la pantalla.

  La cara se puede generar mediante líneas (Figura 5.13) o mediante polígonos (Figura 5.14).La representación que se va a utilizar es la generada mediante polígonos. La realizada mediante líneas es interesante para ver la disposición de los músculos (Figura 5.15). Para aumentar el realismo de la imágen diseñana se les aplica color a los polígonos de la cara. El esquema de color seleccionado para la aplicación es el RGBA. La visualización del color depende del tipo de luces que iluminen la escena como se ha indicado anteriormente en el presente capítulo.

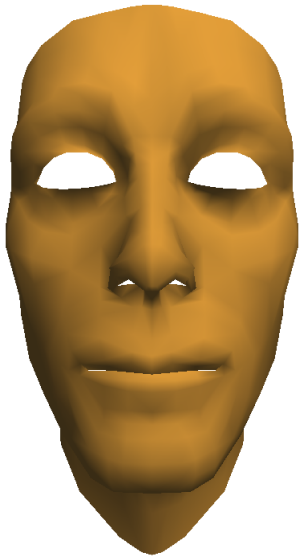


Figura 5.14: Modelado de la cara con polígonos

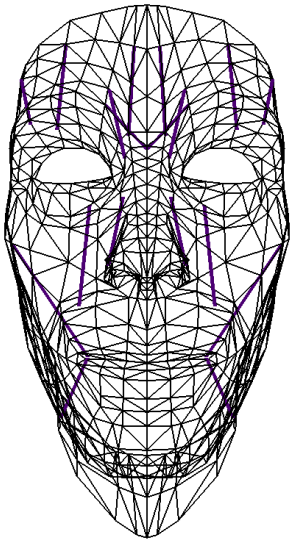


Figura 5.15: Distribución de los músculos en la cara

* **Composición**.

  En esta etapa se construye la escena: se colocan los objetos, se situan las luces... Este proceso se realiza en el programa principal. Su estructura es la siguiente[14]:

  - Declaración de las cabeceras de las librerias necesarias para la parte gráfica (Son necesarias gl.h y glu.h pero si el programa utiliza glut solo es necesario incluir glut.h ya que esta asegura que las otras dos son incluidas, y poner las tres sería dar información redundante) y demás librerías de carácter general que se necesiten así como las específicas del programa que se esta diseñando.

    ```
    #include <GL/glut.h>

    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>

    #include"Head.h"
    ```

  - Declaración de las variables globales que se van a utilizar en el programa, como por ejemplo las relativas a las luces (tipo de luz o posición). Para configuraciones de luces y materiales más complejas que las que se usan en esta aplicación es conveniente crear un fichero llamado "iluminacion.h" en el que se definan las luces que se van a utilizar en la escena así como los distintos tipos de materiales e incluir en el programa principal la cabecera correspondiente.

    ```
    float ambient_light[] = { 0.3, 0.3, 0.45, 1.0 };
    float source_light[] = { 0.9, 0.8, 0.8, 1.0 };
    float light_pos[] = { 30.0, 70.0, 100.0, 1.0 };
    ```

  - Declaración de las funciones que se van a utilizar en el programa (y que están definidas en otro archivo del código fuente), como por ejemplo dibujar los polígonos de la cara o pintar los músculos:

    ```
    void paint_muscles(HEAD *);
    void paint_polygons(HEAD *, int, int);
    ```

  - Función principal, de estructura claramente definida:
    - Inicialización de los modos con los que se creará la ventana como por ejemplo los buffers que utiliza, el más importante por temas de animación es el GLUT_DOUBLE que indica que se usa doble buffer para evitar el *flickering*.

      ```
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_ALPHA|GLUT_DEPTH);
      ```

    - Establecer el tamaño y posición de la pantalla

[14]Se ponen trozos de código a modo de ejemplo

```
glutInitWindowSize(640, 480);
glutInitWindowPosition(0, 0);
```

- Creación de la ventana gráfica con el nombre que quiera que aparezca en el título de la ventana.

  ```
  glutCreateWindow("myFacE");
  ```

  Este nombre saldrá siempre y cuando no hagamos una llamada a la función `glutFullScreen()` que maximiza la ventana a todo el área disponible ocultando el nombre de la ventana.

- Llamada a la función que dibuja cada frame (utiliza el mecanismo de CALLBACK).

  ```
  glutDisplayFunc(&DrawGLScene);
  ```

  La función a la que llama utilizando CALLBACK debe intercambiar los buffers para evitar el parpadeo. Esto se hace mediante la instrucción de GLUT `glutSwapBuffers()`. Justo antes de hacer la llamada a esta función es necesario llamar a la función `glFlush()` de GL que se encarga de la actualización de los buffers.

- Llamada a la función que captura eventos del teclado y/o del ratón. En nuestro caso se utiliza la captura de eventos por teclado para hacer pruebas previas a la aplicación final (utiliza el mecanismo de CALLBACK).

  ```
  glutKeyboardFunc(&keyPressed);
  ```

- Llamada a la función que hace modificaciones entre frames necesaria para las animaciones. Recoge las operaciones que se realizan entre una frame y la siguiente que hacen que la escena vaya cambiando (utiliza el mecanismo CALLBACK).

  ```
  glutIdleFunc(&idle_function);
  ```

  La función a la que llama utilizando CALLBACK debe hacer que los cambios que se realicen en las variables de dibujo se guarden para que la próxima vez que se dibuje estos cambios se reflejen. Esto se hace mediante la función de GLUT `glutPostRedisplay()`.

- Lanzar el capturador de eventos que lleva al programa a que entre en un bucle infinito y se empiece a dibujar (mientras no se llame a esta función no se dibuja nada en la pantalla)

  ```
  glutMainLoop();
  ```

- Definición de las funciones que dibujan cada frame, entre frames, captura de teclado... por ejemplo la función que se ejecuta cuando se produce un determinado evento por teclado (mediante CALLBACK) es la siguiente:

```
void keyPressed(unsigned char key, int x, int y)
{
   usleep(10000);
   if(key==ESCAPE){
```

```
                face_reset(face);
                make_expression(face, face->current_exp);
        }
    }
```

Las funciones de *GLUT* a las que se pasa como parámetro la dirección de otra función con la sintaxis `glut*(&funcion)` utilizan el mecanismo de CALLBACK, de modo que la función será llamada para hacer una operación específica cada vez que se produzca un evento. La función CALLBACK más importante es `glutDisplayFunc()` que se ejecuta cada vez que se refresca la pantalla.

Se suele definir una función que realiza la inicialización de OpenGL y recoge las operaciones que solo han de hacerse una vez en el programa como son:

- Seleccionar el color de fondo de pantalla.
- Activar la iluminación.
- Indicar la profundidad de la escena y habilitar la comparación de profundidad.
- Suavizar las aristas.
- Seleccionar las características de los materiales.
- Seleccionar la matriz de proyección.
- Seleccionar el tipo de proyección con que se van a ver las imágenes.
- Seleccionar la matriz de visualización/modelado.

El código de la función de inicialización de OpenGL descrita es el siguiente:

```
void InitGL(int Width, int Height)
{
    //fondo de pantalla: blanco
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

    //iluminación
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    //profundidad
    glClearDepth(1.0);
    //dibujo lo q queda mas cerca
    glDepthFunc(GL_LESS);
    //habilito la comparacion de profundidad
    glEnable(GL_DEPTH_TEST);

    //pinto poligonos suavizando las aristas
```

```
    glShadeModel(GL_SMOOTH);

    //características de los materiales
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

    //matriz de proyección (le cargo la matriz identidad)
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    //perspectiva de visualización de la imagen
    gluPerspective(45.0f, (GLfloat) Width / (GLfloat) Height, 0.1f, 100.0f);

    //matriz de visualización/modelado
    glMatrixMode(GL_MODELVIEW);
}
```

* **Renderizado**. Una vez montada la escena es momento de crear la imagen final, es decir, de renderizarla[15]. Para ello es necesario generar el binario de la aplicación diseñada. Se hace mediante el siguiente Makefile:

  La parte más importante del Makefile es la siguiente:

  ```
  LIBS = -lX11 -lXi -lXmu -lglut -lGL -lGLU -lm
  ```

  Se refiere a las librerias de OpenGL con las que se debe enlazar el binario. Son las librerias *GL*, *GLU* y *GLUT*.

  Al ejecutar el binario, se abre una ventana y se ve la cara diseñada (Figura 5.14).

5.7.2. Animación

Una vez concluye el proceso de modelado, composición y renderizado el siguiente paso es la animación de la cara. Durante el proceso de modelado se ha tenido en cuenta que posteriormente viene un proceso de animación. Este es el motivo por el cual se dibujan los músculos sobre la cara. Son las herramientas en las que se basa el proceso de animación.

Los pasos generales para animar una escena son los siguientes:

1. Actualizar los datos de la figura (se realizan en la función `idle_function`).
2. Borrar la pantalla.

[15] cómo renderiza internamente OpenGL se explica en la Sección 5.6

3. Dibujar la figura.

4. Volver al punto 1.

Antes de construir la aplicación final que anima la cara diseñada según los estados globales de **Higgs** se construyen una serie de prototipos. Se parte del dibujo estático de la cara en estado neutro (Figura 5.14).

Prototipo: expresiones básicas. Este prototipo pretende que se dibujen las seis expresiones básicas de Ekman [Ekman and Friesen, 1978](alegría, tristeza, miedo, sorpresa, desagrado y enfado). Como se ha indicado antes, los músculos son los elementos que han sido implementados en el modelado de la aplicación para facilitar la tarea posterior de animación.

Los músculos se caracterizan por tener una determinada tensión. Cada una de las expresiones universales se caracteriza por la tensión de los músculos que componen la estructura de la cara. Es de nuevo en un archivo donde se encuentran las tensiones de cada músculo para cada uno de los estados emocionales. A modo de ejemplo, la parte del archivo correspondiente a la alegría es el siguiente:

```
Alegria
Left_Zygomatic_Major 1.10       Right_Zygomatic_Major 1.00
Left_Angular_Depressor 0.00     Right_Angular_Depressor 0.00
Left_Frontalis_Inner 0.80       Right_Frontalis_Inner 0.90
Left_Frontalis_Major 0.20       Right_Frontalis_Major 0.20
Left_Frontalis_Outer 0.10       Right_Frontalis_Outer 0.30
Left_Labi_Nasi 0.00             Right_Labi_Nasi 0.00
Left_Inner_Labi_Nasi 0.00       Right_Inner_Labi_Nasi 0.00
Left_Lateral_Corigator 0.00     Right_Lateral_Corigator 0.00
Left_Secondary_Frontalis 0.00   Right_Secondary_Frontalis 0.00
```

Se utiliza como ayuda el teclado para implementar la función prototipo. Esta función es a la que llama `glutKeyboard` cuando se presiona la tecla `escape` del teclado (se ha definido que reaccione ante esa tecla). Lo que hace es que, cada vez q se pulsa esa tecla, va recorriendo el archivo en el que se encuentran las distintas expresiones. De modo que se van mostrando una tras otra en la pantalla. El orden con el que se visualizan es el siguiente: neutra, alegría, tristeza, miedo, sorpresa, desagrado y enfado. Los resultados obtenidos pueden verse en el Apéndice A de la presente memoria.

El código de dicha función es el siguiente:

```
void keyPressed(unsigned char key, int x, int y)
{
```

```
        usleep(10000);
        if(key==ESCAPE){
                face_reset(face);
                face->current_exp++;
                if (face->current_exp>= face->nexpressions)
                        face->current_exp=0;
                make_expression(face, face->current_exp);
        }
}
```

Prototipo: expresión viva. Este prototipo pretende que la imagen estática que se dibuja cuando no se produce ningún evento tenga mayor sensación de realismo. Lo que se hace es añadirle a la tensión de los músculos una oscilación de un seno de poca amplitud entre un frame y el siguiente. De este modo que se produce la sensación de que esta respirando.

Esta función se implementa en `idle_function` que es la función que se ejecuta entre frames. Su código es el siguiente:

```
void idle_function()
{
        usleep(20000);

        // codigo para generar la expresion "viva"
        static float t;
        float ts=0.07;
        t=t+ts;
        float amplitud=0.0006;
        float inc=amplitud/tts*sin(tt); // a lo senoidal

        int e=0;
        int n;
        for (n = 0; n < face->nmuscles; n++)
        {
                float muscle_contraction = face->expression[e]->m[n];

                activate_muscle(face,
                        face->muscle[n]->head,
                        face->muscle[n]->tail,
                        face->muscle[n]->fs,
                        face->muscle[n]->fe,
                        face->muscle[n]->zone, +inc);
```

```
        glutPostRedisplay();
}
```

Prototipo: cambio de expresión. El objetivo es pasar de una expresión a otra. La función lee los datos de dos expresiones del archivo de expresiones e interpola entre sus valores mediante una senoidal.

La función se implementa en el `idle_function` y se ejecuta entre frames, el código de la misma es el siguiente:

```
void idle_function()
{
        usleep(20000);

        // codigo para cambiar de expresion
        static float t;
        float ts=0.050;
        t=t+ts;
        int m;

        // generacion de la interpolacion entre los niveles musculares de
        // las expresiones i y j
        face_reset(face);
        int i=0;
        int j=1;

        for (m = 0; m < face->nmuscles; m++)
        {
                float muscle_contraction0 = face->expression[i]->m[m];
                float muscle_contraction1 = face->expression[j]->m[m];

                // genero una senoidal entre los valores
                float ampl=abs(muscle_contraction0-muscle_contraction1)/2.0;
                float punto_medio=(muscle_contraction0+muscle_contraction1)/2.0;
                float inc=ampl*sin(t);
                float valor=inc+punto_medio;

                activate_muscle(face,
                        face->muscle[m]->head,
                        face->muscle[m]->tail,
                        face->muscle[m]->fs,
                        face->muscle[m]->fe,
                        face->muscle[m]->zone, valor);
```

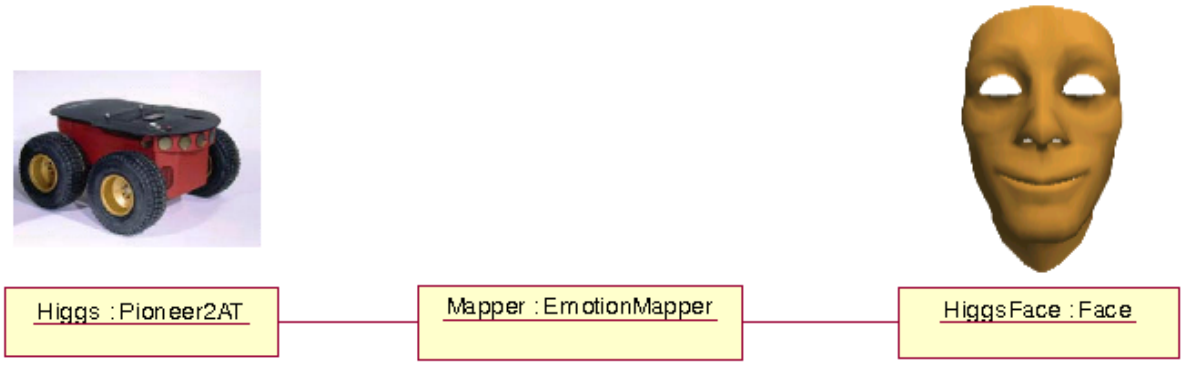


Figura 5.16: Esquema sencillo de la aplicación **HiggsFace**

```
        }
        glutPostRedisplay();
}
```

Aplicación final: animación de la cara de Higgs. Un esquema sencillo de la aplicación final puede verse en la figura 5.16. Según este esquema el método seguido para el proceso de animación en la aplicación final es el que se describe a continuación: el servidor CORBA **Higgs** le proporciona al **EmotionMapper** un vector con información sensorial del robot. Éste se encarga de traducirla a información emocional que viene dada por un vector que tiene por componentes las tensiones correspondientes a cada músculo de la cara. Por último el **EmotionMapper** le pasa esta información al servidor CORBA **HiggsFace** que la utiliza para dibujar la cara.

La función `idle_function` que se utiliza para realizar las operaciones correspondientes entre frames, se encarga de utilizar los valores almacenados en el vector de estado emocional que recibió el servidor**HiggsFace** en la última llamada sobre su interfaz. De este modo, cuando se redibuje la pantalla se hará con las nuevas tensiones de los músculos de la cara correspondientes al estado emocional de **Higgs**.

Capítulo 6

Mapeo de emociones en Higgs

Este capítulo describe cómo se realiza el mapeo a emociones del estado de **Higgs**. Por mapeo se entiende la traducción del estado sensorial[1] a un estado emocional[2] sino proporcionar un mecanismo para implantar una teoría concreta en forma de *plug-in* (en particular como un agente activo **ICa**. El objetivo no es desarrollar una teoría sobre las emociones. Se pretende habilitar un mecanismo para transformar los estados del robot en estados emocionales y posibilitar la visualización de éstos mediante una cara antropomórfica. Cualquier teoría sobre las emociones que quiera implementarse después podrá hacerse cambiando el modo de mapear la información.

La aplicación no se diseña para emular estados humanos, sino que convierte los estados globales del sistema en estados emocionales que expresa mediante caras de modo que estas sean fácilmente entendibles por el usuario. De esta forma, de un vistazo, se puede ver cómo se encuentra el sistema en un determinado instante.

Este capítulo describe la parte de la aplicación que ha sido denominada **EmotionMapper** que se encarga de solicitar al robot su estado, mapearlo a un estado emocional y enviar la información al programa gráfico que se encarga de dibujar la cara. Se organiza del siguiente modo:

- En la Sección 6.1 se describe, a modo de introduccion, la misión del **EmotionMapper** y la estructura del mismo.
- En la Sección 6.2 se describe la información sensorial que nos proporciona el servidor CORBA **Higgs**. Describe el tipo de información que proporciona al **EmotionMapper** cuando este le solicita en un determinado momento el estado del robot móvil.
- En la Sección 6.3 se describe cómo el **EmotionMapper** realiza la traducción de los estados globales del sistema en estados emocionales, esto es, el modo en el que la

[1] Estado de los sensores del robot (tanto exteroceptivo como propioceptivo)
[2] Estado sintético de alto nivel empleando en el control estratégico de una máquina

información sensorial se transforma en información emocional que es visualizada posteriormente en una cara antropomórfica.

- En la Sección 6.4 se indica cómo el **EmotionMapper** transmite la información emocional al servidor CORBA **HiggsFace** para que actualice la posición de la cara al estado emocional determinado por el estado del sistema en ese momento.

6.1. EmotionMapper

El **EmotionMapper** es la parte de la aplicación que se utiliza para transformar la información sensorial en información emocional que pueda ser utilizada por la parte gráfica para representar mediante expresiones faciales en una cara antropomórfica el estado del sistema (Figura 6.1).

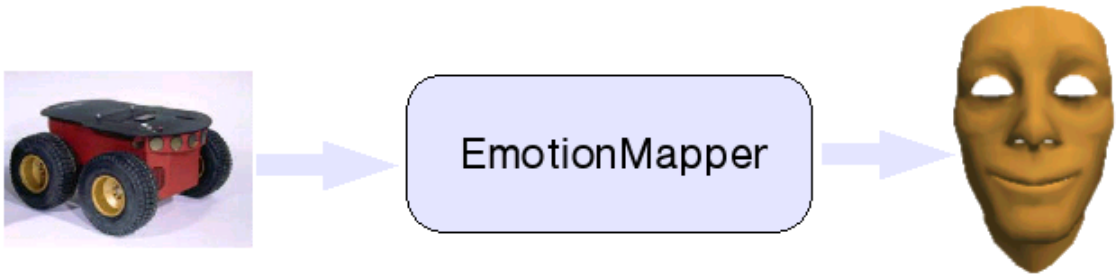


Figura 6.1: Elementos de la aplicación **HiggsFace**

Higgs no tiene emociones humanas sino estados mecánicos que abstraemos a estados emocionales que se expresan mediante expresiones faciales para que, de un vistazo, el operador sepa cómo se encuentra el sistema. Hay que "jugar" con la información sensorial que proporciona el robot (se describe en la Sección 6.2) para convertirla en información emocional que guarde relación con el estado del robot.

6.2. Información sensorial

El robot móvil tiene un programa servidor ejecutandose en su CPU que atiende peticiones a través de la red inalámbrica. Este servidor fue desarrollado en un proyecto anterior [Pareja, 2004] y proporciona cierta información del robot que se va a utilizar para transformar a información emocional. El **EmotionMapper** es un agente **ICa** que actúa como cliente de ese servidor solicitándole la información sensorial de **Higgs** en un determinado instante. El servidor le proporciona esa información mediante un vector con los datos pedidos en ese momento. La información que le proporciona el servidor **Higgs** es la siguiente[3]:

- velocidad de traslación del robot (**Velocity**)
- velocidad de rotación del robot (**RotVelocity**)
- velocidad de las ruedas derecha e izquierda (**RightVelocity**, **LeftVelocity**)
- coordenadas X, Y y θ del robot respecto al sistema de referencia del robot (**XCoordinate**, **YCoordinate**, **ThCoordinate**)
- lecturas de los sonares[4] (**Sonar_Range**)
- batería (**Battery**)

Este vector le proporciona al **EmotionMapper** el estado global del sistema. Una vez tiene esta información ha de transformarla en estados emocionales. Este proceso (el "mapeo" propiamente dicho) se describe en la siguiente sección.

6.3. Transformaciones emocionales

Entre los objetivos del proyecto no está el desarrollo de una teoría sobre las emociones. El objetivo perseguido es dotar al módulo de la funcionalidad necesaria para comunicar su información sensorial mediante estados emocionales que se visualizan en una cara antropomórfica por lo que se limita a transformar la información sensorial que recibe de algún modo en información emocional que pueda utilizarse para dibujar la cara.

En el mapeo de emociones intervienen tres elementos: el vector de información sensorial (V_R), la matriz de emociones (M_e) y el vector de información emocional (V_C). A continuación se describe cada uno de estos elementos.

[3]Entre paréntesis el nombre correspondiente en la estructura proporcionada por el servidor.
[4]Dispone de 16 sonares, aunque no todos tienen porqué estar activos.

6.3.1. Vector información sensorial

Está formado por la información sensorial proporcionada por el robot que se utiliza para el mapeo de emociones. No todos los datos que son proporcionados por el servidor al hacerle la consulta sobre el estado de **Higgs** se utilizan para realizar la transformación a estados emocionales. De entre los datos que proporciona el servidor se eligen aquellos que son susceptibles de poder significar algo para el estado global del robot teniendo en cuenta que vamos a hacer una analogía para transformarlos en emociones.

Para la aplicación que se está desarrollando, la información proporcionada por el servidor CORBA **Higgs** que se selecciona para su conversión a emociones es la siguiente:

- estado de la batería
- coordenada X
- coordenada Y
- velocidad rueda derecha
- velocidad rueda izquierda
- velocidad de rotación

Es necesario escalar al rango

$$[0, 1]$$

los valores de este vector ya que los valores que en él se encuentran corresponden a variables muy distintas que no tienen relación entre sí. De este modo cada variable influye en la emoción final de manera parecida, independientemente de la magnitud de la misma (que nos conduciría a valores erróneos porque los rangos de variación de las mismas son muy distintos).

En la siguiente tabla se reflejan las distintas variables consideradas así como los rangos de variación de cada una de ellas:

Variable	**Rango**
batería	0–13 (V)
coordenada X	$(*)$ (m)
coordenada Y	$(*)$ (m)
velocidad rueda derecha	0–250 (mm/s)
velocidad rueda izquierda	0–250 (mm/s)
velocidad de rotación	0–50 (mm/s)

Los rangos de variación de las variables coordenada X y coordenada Y se representa por $(*)$ porque realmente no están limitados, incluso pueden ser negativos. El sistema

de coordenadas de referencia se establece en el momento en el que el robot se conecta y empieza a moverse pero para poder utilizar estos valores en el mapeo de información sensorial a información emocional que se está realizando deben limitarse. Consideramos que el rango de variación es de -5 a 5 metros.

Una vez la información sensorial de **Higgs** en un determinado instante le llega al **EmotionMapper**, se cogen de esa vector las componentes que interesan para mapear el estado global a estado emocional y se pasan a valores por unidad. El vector obtenido tras este proceso es V_R (vector de información sensorial).

El orden de estas variables en el vector V_R es el siguiente[5]:

$$V_R = \begin{bmatrix} \text{Batería} \\ \text{V rueda dcha} \\ \text{V rotación} \\ \text{V rueda izqda} \\ \text{Coordenada } X \\ \text{Coordenada } Y \end{bmatrix}$$

6.3.2. Matriz de emociones

La matriz de emociones es una matriz de dimensiones $m \times n$ siendo m el número de músculos que se utilizan para la animación y n el número de expresiones existentes. De modo que el elemento a_{ij} de la matriz de emociones representa la tensión del músculo i correspondiente a la expresión j.

En el caso de la aplicación que se desarrolla en este proyecto, la matriz de emociones es de 18×6. Esta matriz es,

[5]La razón por la que se ordenan de este modo se indica en la parte del vector de información emocional.

$$M_e = \begin{cases} \text{filas: músculos (2 de cada)} \quad m = 1 \ldots 18 \begin{cases} \text{Cigomático mayor} \\ \text{Depresor del ángulo de la boca} \\ \text{Frontal interior} \\ \text{Frontal} \\ \text{Frontal exterior} \\ \text{Elevador del ángulo de la boca} \\ \text{Elevador del labio superior} \\ \text{Corrugador lateral} \\ \text{Corrugador superciliar} \end{cases} \\ \text{columnas: expresiones} \qquad n = 1 \ldots 6 \begin{cases} \text{Alegría} \\ \text{Enfado} \\ \text{Sorpresa} \\ \text{Tristeza} \\ \text{Miedo} \\ \text{Desagrado} \end{cases} \end{cases}$$

La forma en la que se construye la matriz es la siguiente: se dispone de un archivo en el que se indica la tensión de cada músculo para cada una de las seis expresiones universales de Ekman[6]. La aplicación desarrollada abre dicho archivo y lee sus valores rellenando la matriz M_e (matriz de emociones) que se utiliza para el mapeo.

6.3.3. Vector de información emocional

Este vector representa las tensiones de cada uno de los músculos de la cara para el estado global del sistema. Se obtiene como producto de la matriz de emociones por el vector de información sensorial:

$$V_C = M_e \cdot V_R$$

siendo:

$$\begin{aligned} M_e &= [a_{ij}]_{m \times n} \\ V_R &= [r_j]_{n \times 1} \\ V_C &= [c_j]_{m \times 1} \end{aligned}$$

[6]Los valores de la tensión de cada músculo en cada una de las expresiones se obtienen probando valores hasta generar expresiones naturales.

de modo que:

$$c_i = \sum_{j=1}^{m} (a_{ij} \cdot r_j)$$

lo que significa que cada variable pondera una columna de la matriz de emociones. De modo que el orden de las variables de información sensorial del vector V_R es importante ya que cada una de ellas pondera una emoción, es decir, le confiere un peso a esa emoción en el estado final.

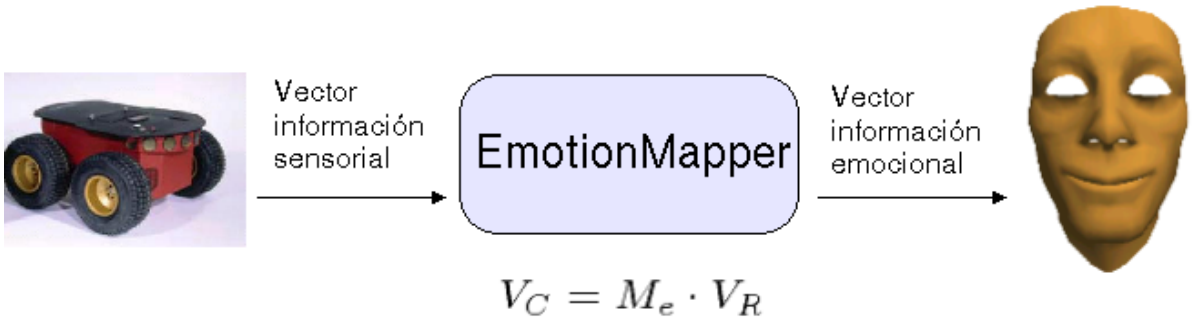


Figura 6.2: El **EmotionMapper** como aplicación lineal

El vector obtenido no siempre puede utilizarse directamente. Es necesario modificarlo antes si la tensión de alguno de los músculos de la estructura supera los valores tensionales máximos para evitar obtener expresiones imposibles (que los puntos se salgan de la cara). En la siguiente tabla se recogen las tensiones máximas de los distintos músculos que constituyen la estructura de la cara (su disposición se puede ver en la Figura 2.3).

Si alguno de los valores sobrepasa el umbral se satura ese valor, es decir, toma el valor máximo de tensión posible para ese músculo. Si todos los valores están dentro de los rangos de variación correspondientes a cada músculo, no será necesario hacer nada y su utilización es inmediata.

Una vez obtenido el vector de información sensorial (V_C) y modificado en caso de ser necesario, el **EmotionMapper** lo transmite al servidor CORBA **HiggsFace** que utiliza su información para dibujar la expresión correspondiente.

Músculos (lado izquierdo)	**Tmáx.**	**Músculos (lado derecho)**	**Tmáx.**
Cigomático mayor	1.10	Cigomático mayor	1.00
Depresor del ángulo de la boca	0.70	Depresor del ángulo de la boca	0.70
Frontal interior	3.10	Frontal interior	3.90
Frontal	0.60	Frontal	0.40
Frontal exterior	2.00	Frontal exterior	2.00
Elevador del ángulo de la boca	1.80	Elevador del ángulo de la boca	2.00
Elevador del labio superior	1.20	Elevador del labio superior	1.10
Corrugador lateral	2.30	Corrugador lateral	2.40
Corrugador superciliar	0.50	Corrugador superciliar	0.60

6.4. Visualización

Una vez que el **EmotionMapper** ha transformado la información sensorial en información emocional ha de pasársela al sevidor **HiggsFace** para que este la trate de la forma conveniente y en la cara se visualicen las nuevas tensiones de los músculos que dan lugar a la expresión emocional del estado global del sistema.

Capítulo 7

Conclusiones y líneas futuras

Para finalizar con la memoria de este proyecto fin de carrera titulado **"Mapeo facial de emociones sintéticas"** en este capítulo se describen las conclusiones que se sacan del trabajo realizado así como posibles líneas de investigación que podrían seguirse en el futuro y revertirían en mejoras para el mismo.

En la Sección 7.1 se recuerdan los objetivos, el trabajo realizado así como las conclusiones que se sacan del proyecto que se describe en esta memoria.

En la Sección 7.2 se describen las líneas de trabajo que quedan abiertas tras la conclusión de este proyecto que supondrían mejoras de la aplicación desarrollada.

7.1. Conclusiones

El objetivo de este proyecto es la implementación de un sistema para la visualización del estado global de un sistema técnico mediante una cara antropomórfica utilizando la expresión facial como médio de comunicación. Así mismo se desarrolla un patrón de diseño para poder implantar cualquier teoría emocional en forma de agente activo. También se implementa una aplicación concreta: **HiggsFace** (plataforma móvil Pioneer 2AT-8).

La aplicación consta de tres partes que se han ido describiendo por separado en la memoria.

Pioneer 2AT: Es el servidor CORBA que se ejecuta en la CPU de **Higgs** y se encarga de proporcionar la información sensorial. Esta parte fue desarrollada anteriormente para otro proyecto [Pareja, 2004], en un principio sólo era necesario hacer uso de él, pero la necesidad de hacer pruebas y la falta de disponibilidad del robot en algunos momentos hicieron necesaria la creación de un simulador que actuara como banco de pruebas de la aplicación. Fue necesaria la creación de un servidor CORBA con la misma interfaz que el servidor que se ejecuta en el robot para que cuando la aplicación

se conecte a **Higgs** se obtengan los mismos resultados que en las pruebas.

EmotionMapper: Este es un agente intermedio que se encargaba por un lado de recoger la información sensorial y por otro de generar información emocional a partir de ella mediante un proceso de mapeo. En este proyecto facilita un mecanismo para pasar de las unas a las otras mediante transformaciones lineales de la información.

Face: Este bloque que se encarga del dibujo y gestión de la representación gráfica del estado emocional de **Higgs**. Une dos cosas: por un lado es un servidor CORBA que atiende peticiones que le llegan de clientes y por otro es un programa de dibujo gráfico basado en OpenGL. Ambos interactuan de modo que lo que la parte gráfica dibuja los valores que le pasa el **EmotionMapper** dependiendo del estado que se infiere de la información sensorial proporcionada por **Higgs** (o en su defecto por el banco de pruebas construido para suplirle).

El resultado concreto obtenido en el proyecto es un sistema que dibuja distintas expresiones faciales según el valor de las variables del sistema robótico Pioneer 2AT-8. El valor principal del desarrollo es, sin embargo, su reutilizabilidad en otras aplicaciones.

7.2. Líneas futuras

Este proyecto deja abiertas varias líneas de trabajo. Algunas se relacionan directamente con la aplicación específica realizada para este proyecto: **HiggsFace**, aunque también las hay de carácter más general como por ejemplo la relacionada con el mapeo de emociones.

7.2.1. Nuevo servidor

La aplicación específica para la que se ha implementado el software diseñado es **HiggsFace**. El objetivo perseguido era hacer que **Higgs** se comunicara mediante emociones que se visualizan en una cara antropomórfica. Para ello el proceso que se se sigue es transformar la información sensorial que proporciona el servidor CORBA de **Higgs** en información emocional utilizando el **EmotionMapper**.

El servidor que se ha utilizado en esta aplicación fue diseñado en otro proyecto [Pareja, 2004] para satisfacer las necesidades específicas de aquel y no de éste, por lo que una línea de trabajo futura para mejorar esta aplicación es el diseño de un nuevo servidor. Aria (API para el control del robot) proporciona más funciones de las que se implementaron en su momento para el servidor. Algunas de estas funciones proporcionan información más adecuada a la hora de inferir un estado global del sistema, por ejemplo es el caso de ver si el robot se está moviendo o no, si está atascado o si tiene los motores conectados. Toda esta información es susceptible de significar algo para el estado global del sistema, mucho

más que variables como la posición que se ha utilizado para el mapeo en la aplicación desarrollada (se utilizó la información disponible).

Por lo que una de las líneas futuras que queda abierta tras la finalización de este proyecto es desarrollar un servidor que proporcione más información acerca del robot para aumentar la funcionalidad del mismo y poder así expresar mejor su estado global utilizando la expresión emocional.

7.2.2. Teoría sobre las Emociones

La otra línea de trabajo posible tiene que ver con el **EmotionMapper**. Como se indicó en el Capítulo 6 el objetivo de este proyecto no era el diseño de una "Teoría sobre las Emociones" sino dotar al sistema de un mecanismo que transformara la información sensorial disponible en información emocional.

En este proyecto no se ha utilizado ninguna teoría concreta. Se disponía de una serie de datos a partir de los cuales se implementó un mecanismo de demostración para transformar la información proporcionada por esos datos en emociones que se visualizan en una cara antropomórfica.

Una interesante puerta que queda abierta tras este proyecto es la investigación de una Teoría sobre las Emociones para el mapeo a información emocional del estado de los distintos sistemas físicos con los que nos pudieramos encontrar, esto es, habilitar métodos que expliquen cómo realizar la traducción de un estado global del sistema a un estado emocional que sea entendible por el usuario y que guarde analogía con la expresión emocional humana.

7.2.3. Aumentar el realismo de la imagen

La interfaz diseñada en este proyecto para la comunicación del estado global de un determinado sistema (en este caso el robot móvil **Higgs**) es una cara de apariencia antropomórfica. De modo que evitamos que sea necesario un "entrenamiento" especial para comprender lo que se quiere significar ya que de la expresión facial se infiere el estado del sistema (de igual modo que en la comunicación entre personas se infiere el estado del interlocutor). La cara de **Higgs** es una máscara dotada de gran expresividad de modo que es capaz de generar expresiones fácilmente reconocibles por las personas (ver las imágenes del Apéndice A). Con esto se cumple el objetivo del presente proyecto. Pero el diseño gráfico de la misma es susceptible de mejora.

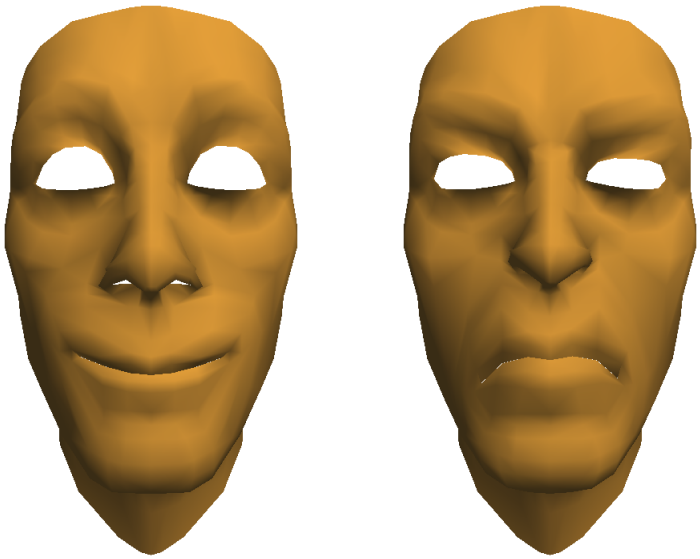
Aumentar el realismo de la imagen utilizando el diseño gráfico es una línea futura de trabajo. Se puede diseñar que tenga ojos (y que se muevan), pelo en la cabeza y en la cara (cejas, barba...) así como textura de piel en la cara en lugar de color. Todas estas características, como se explicó en el Capítulo 2, contribuyen al realismo de la imagen y consiguen que la interfaz diseñada sea más creíble. Lo único que aporta expresividad a

la imagen son los ojos y su movimiento (no es una tarea sencilla conseguir una mirada natural), el resto de las características son sólo estéticas, pero en cualquier caso este es el modo de proporcionar interfaces más naturales para el usuario.

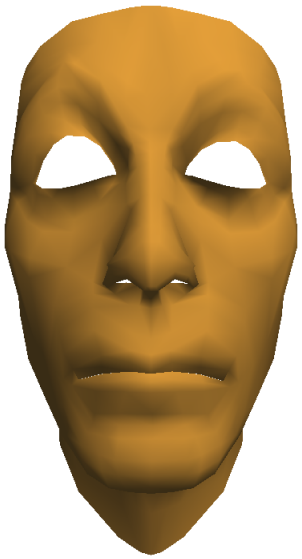
Apéndice A

Ejemplos

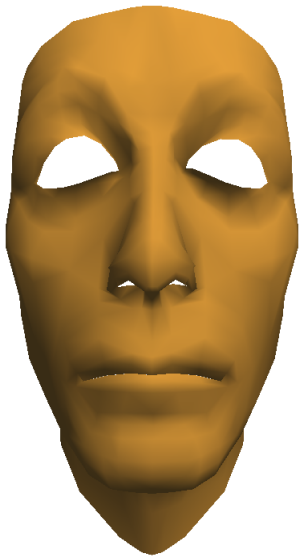
En este apéndice se recogen imágenes de la interfaz diseñada en distintas posiciones. Se recogen las expresiones universales de Ekman [Ekman and Friesen, 1975].



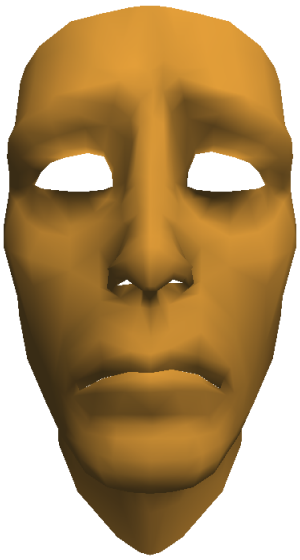
Alegría Enfado



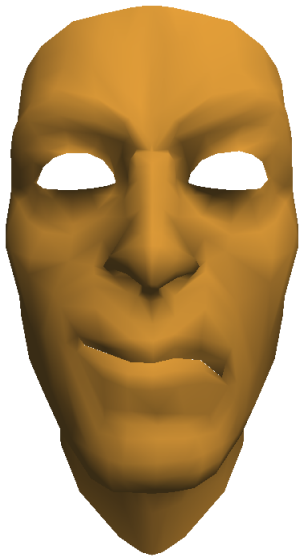
Sorpresa



Tristeza



Miedo



Desagrado

Apéndice B

Otras herramientas

Además de las herramientas descritas el los Capítulos 4 y 5, 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 el programa que se encarga de dibujar y gestionar la cara que expresa el estado del robot (programa C++).

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. 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 imágenes de la cara diseñada, 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.

Bibliografía

[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*.

[Bartneck, 2002] Bartneck, C. (2002). eMuu-An Embodied Emotional Character For The Ambient Intelligent Home.

[Bartneck, 2003] Bartneck, C. (2003). Interacting with an embodied emotional character. In *DPPI '03: Proceedings of the 2003 international conference on Designing pleasurable products and interfaces*, pages 55–60. ACM Press.

[Blanz and Vetter, 1999] Blanz, V. and Vetter, T. (1999). A morphable model for the synthesis of 3d faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194. ACM Press/Addison-Wesley Publishing Co.

[Boden, 1977] Boden, M. A. (1977). *Inteligencia artificial y hombre natural*. Editorial Tecnos.

[Braccini et al., 2002] Braccini, C., Lavagetto, F., Costa, M., and Pockaj, R. (2002). A solution for model-independent animation of mpeg-4 faces.

[Brilliant, 2003] Brilliant, K. (2003). *Building a digital human*. Charles River Media, INC.

[Bui et al., 2003] Bui, T. D., Heylen, D., and Nijholt, A. (2003). Improvements on a simple muscle-based 3d face for realistic facial expressions. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*, page 33. IEEE Computer Society.

[Chinchilla, 2003] Chinchilla, R. (2003). *Installing ICa*. Hard Real Time CORBA IST Project.

[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, 1998] Cole, J. (1998). *About Face*. A Bradford Book.

[Darwin, 1872] Darwin, C. R. (1872). *The expression of emotions in man and animals*. John Murray.

[Duchenne, 1862] Duchenne, C. B. (1862). *The Mechanism of Human Facial Expression*. R. Andrew Cuthbertson.

[Ekman and Friesen, 1975] Ekman, P. and Friesen, W. (1975). *Unmasking the face. A guide to recognizing emotions from facial clues*. Englewood Cliffs, New Jersey: Prentice-Hall.

[Ekman and Friesen, 1978] Ekman, P. and Friesen, W. (1978). Facial action coding system (FACS): a technique for the measurement of facial action. Technical report, Cosulting Psychologist Press, Palo Alto, CA.

[El-Nasr et al., 1999] El-Nasr, M. S., Ioerger, T. R., Yen, J., House, D. H., and Parke, F. I. (1999). Emotionally expressive agents. In *CA '99: Proceedings of the Computer Animation*, page 48. IEEE Computer Society.

[Evans, 2001] Evans, D. (2001). *Emotion, a very short introducction*. Oxford University Press.

[Fabri et al., 1999] Fabri, M., Moore, D. J., and Hobbs, D. J. (1999). The emotional avatar: Non-verbal communication between inhabitants of collaborative virtual environments. *Lecture Notes in Computer Science*, 1739:245–248.

[Faigin, 1990] Faigin, G. (1990). *The Artist's Complete Guide to Facial Expressions*. Watson-Guptill Publicatiobs.

[Fàbregas, 1993] Fàbregas, J. (1993). *El arte de leer el rostro*. Martinez Roca.

[García, 2003] García, J. (2003). Curso de introducción a opengl (v1.0).

[Goleman, 1996] Goleman, D. (1996). *Inteligencia emocional*. Ed. Kairós.

[Kopk and Daly, 2003] Kopk, H. and Daly, P. W. (2003). *Guide to LaTeX*. Addison Wesley, 4th edition.

[Marschner et al.,] Marschner, S. R., Guenter, B., and Raghupathy, S. Modeling and rendering for realistic facial animation. pages 231–242.

[Michi Henning, 1999] Michi Henning, S. V. (1999). *Avance CORBA programming with C++*. Addison Westley.

[Minsky, 1978] Minsky, M. (1978). *Society of Mind*. Simon.

[Mittelbach and Goossens, 2004] Mittelbach, F. and Goossens, M. (2004). *The LaTeX Companion*. Addison Wesley, 2nd edition.

[Ostermann, 1998] Ostermann, J. (1998). Animation of synthetic faces in mpeg-4. In *CA '98: Proceedings of the Computer Animation*, page 49. IEEE Computer Society.

[Pareja, 2004] Pareja, I. (2004). Sistema de comunicaciones para pioneer 2at-8. Master's thesis, ETSII-UPM.

[Parke, 1972] Parke, F. I. (1972). Computer generated animation of faces. In *ACM'72: Proceedings of the ACM annual conference*, pages 451–457. ACM Press.

[Paull, 2002] Paull, D. B. (2002). *Programming dynamic character animation*. Charles River Media, INC.

[Picard, 1998] Picard, R. W. (1998). *Los ordenadores emocionales*. Editorial Ariel.

[Qua, 2002] Qua, V. (2002). *Muscle Based Facial Animation*. PhD thesis, Imperial College.

[Roosendaal and Selleri, 2004] Roosendaal, T. and Selleri, S. (2004). *The official Blender 2.3 guide*. Blender Foundation.

[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.

[Terzopoulos and Waters, 1993] Terzopoulos, D. and Waters, K. (1993). Analysis and synthesis of facial image sequences using physical and anatomical models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(6):569–579.

[Trappl et al., 2002] Trappl, R., Petta, P., and Payr, S. (2002). *Emotions in humans and artifacts*. Massachusetts Institute of Technology.

[Waters, 1987] Waters, K. (1987). A muscle model for animation three-dimensional facial expression. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 17–24. ACM Press.

[Waters and Parke, 1996] Waters, K. and Parke, F. I. (1996). *Computer Facial Animation*. AK Peters, Ltd.

[Woo et al., 1999] Woo, M., Neider, J., Davis, T., and Shreiner, D. (1999). *OpenGL Programming Guide: the official guide to learning OpenGL, version 1.2, 3rd edition*. Addison-Wesley.