UNIVERSITAT POLITÈCNICA DE VALÈNCIA

INGENIERÍA INFORMÁTICA

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

SISTEMA DE GESTIÓN Y CONTROL DE FICHEROS DE CONFIGURACIÓN PARA ROBOTS NAO APLICADO A COMPETICIONES SPL ROBOCUP

Proyecto Final de Carrera

Presentado por: Jorge Sanmartín Martínez

Dirigido por: Juan Francisco Blanes Noguera

Valencia, Septiembre 2011

AGRADECIMIENTOS

Quisiera dar las gracias a todas las personas que han formado parte de mi vida durante este paso por la Universidad, especialmente a mi familia y a mis amigos que me han apoyado durante estos años, y han hecho que esto sea posible.

Me gustaría agradecer también a todos los profesores que me han enseñado y formado durante sus clases, y a todos mis compañeros, muchos de ellos grandes amigos ahora, que me han acompañado durante este paso por la Universidad, y han hecho tan especial esta etapa de mi vida.

Sin todos ellos esto no hubiera sido posible.

RESUMEN

Este proyecto final de carrera tiene por objetivo dotar de flexibilidad al sistema empotrado del robot Humanoide Nao a la hora de configurar parámetros de control y ejecución de los procesos en ejecución. Para ello se diseñará y desarrollará un sistema informático, con el fin de gestionar un conjunto de ficheros de parametrización del control del sistema empotrado, incrementando de esta forma la flexibilidad del sistema y evitando errores sintácticos que desencadenan comportamientos no deseados por parte del robot.

Palabras Clave: ***Robótica humanoide, Sistema empotrado, Comunicaciones, Sintaxis XML***

Índice

1 – Introducción

- 1.1 Robots humanoides
- 1.2 Robot Nao
- 1.3 Entorno de desarrollo en Nao
- 1.4 Standard Platform League Robocup

2 – Objetivos

3 – Diseño de sistema

- 3.1 Secure Shell (SSH)
- 3.2 Secure Copy (SCP)
- 3.3 Archivo de configuración .INI
- 3.4 Archivo XML
- 3.5 Diccionario de datos

4 – Desarrollo de soporte de comunicaciones

- 4.1 Conexión SSH
- 4.2 Intercambio de ficheros
- 4.3 Traducción de .INI a XML
- 4.4 Edición y modificación de ficheros XML
- 4.5 Diccionario de datos

5 – Diseño y desarrollo de la interfaz gráfica de usuario

6 - Conclusiones

7– Anexo:

- 6.1 Manual de usuario
- 6.2 Manual de desarrollador

8 - Bibliografía

Capitulo 1

INTRODUCCIÓN

1.1 Robots humanoides

La robótica, según una de sus definiciones más sencillas, es la ciencia y la tecnología de los robots. Se ocupa del diseño, manufactura y aplicaciones de los robots. La robótica combina diversas disciplinas como son la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control.

Isaac Asimov fue el primero en introducir el término robótica tal y como lo entendemos en la actualidad, con una versión más humanizada y como una disciplina encargada de construir y programar robots. Este autor plantea una serie de reglas que debe cumplir todo robot, más conocidas como las Tres leyes de la robótica.

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

Estas leyes surgen como medida de protección para los seres humanos. Según el propio Asimov, la concepción de las leyes de la robótica quería contrarrestar un supuesto temor que el ser humano desarrollaría frente a unas máquinas que hipotéticamente pudieran rebelarse y alzarse contra sus creadores. Asimov crear un universo en el que los robots son parte fundamental durante los próximos años de la historia humana, incrementando su nivel de complejidad cada vez más.

Entender y aplicar lo anteriormente expuesto requiere verdadera inteligencia y consciencia del mundo real por parte del robot, algo que a pesar de los avances tecnológicos de la era moderna no se ha llegado.

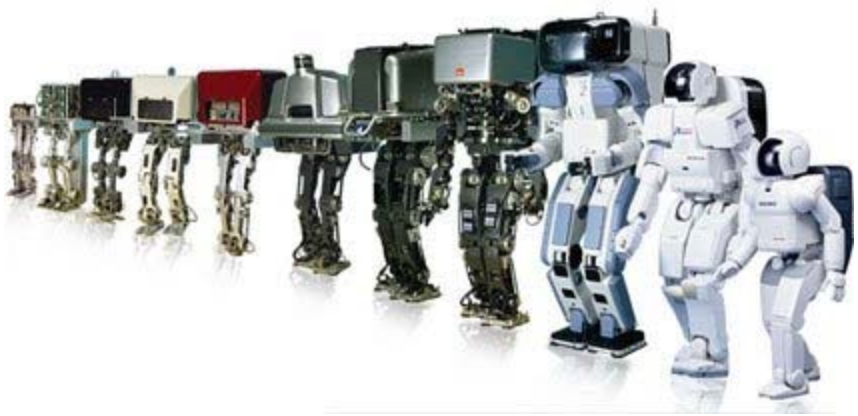

Figura 1.1

Podemos hacer una clasificación de los robots dependiendo de su arquitectura, definida por el tipo de configuración general del robot, y se puede subdividir en los siguientes grupos: poliarticulados, móviles, androides, zoomórficos e híbridos.

Los robots humanoides o androides son robots que intentan reproducir total o parcialmente la forma y el comportamiento cinemática del ser humano. Uno de los aspectos más complejos de estos robots y sobre el que se centra la mayoría de los trabajos, es el de la locomoción bípeda. En este caso, el principal problema es el control dinámica y coordinadamente en tiempo real del proceso y mantener simultáneamente el equilibrio del robot.

El androide siempre ha sido representado como una entidad que imita al ser humano tanto en apariencia como en habilidades mentales y de movimiento. Las principales ventajas que tienen los robots humanoides respecto a otro tipo de configuraciones son dos:

- Facilitan la interacción con otros humanos.
- Se adaptan rápidamente a entornos y herramientas diseñados para humanos.

Sin embargo, la construcción de un robot que imite convincentemente la libertad de gestos y movimientos humanos, es una tarea de una enorme complejidad técnica. De hecho, es un problema que está todavía abierto a la investigación y la mejora, aunque existen ciertos ejemplos bastante meritorios en este sentido, que imitan ciertas conductas y capacidades humanas.

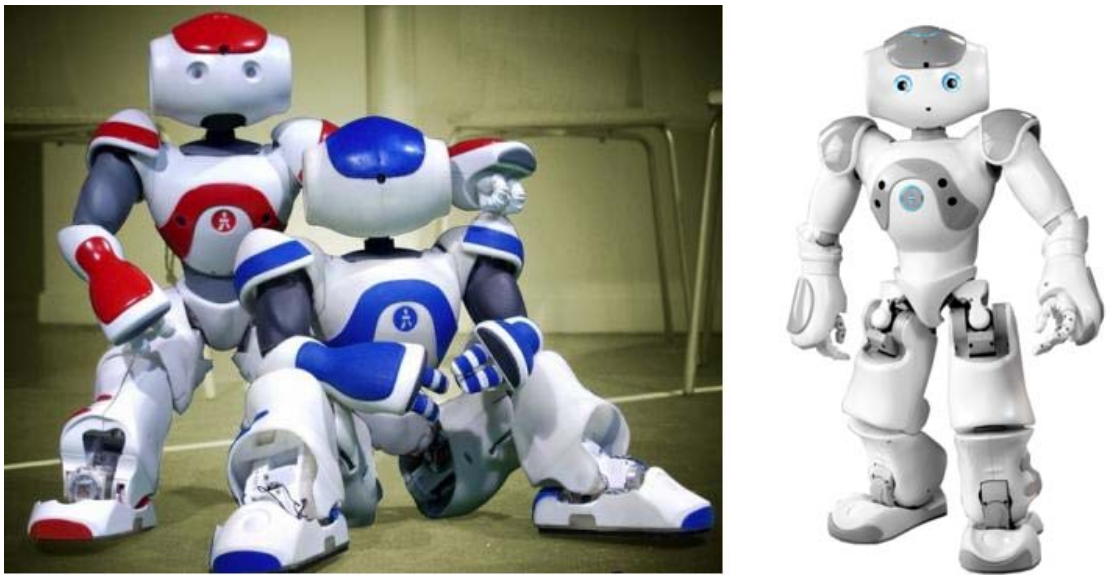

Figura 1.2

1.2 Robot Nao

Dentro de este grupo de robots humanoides se encuentra el modelo Nao de la empresa francesa Aldebaran Robotics usados en la SPL (Standard Platform League). Existen prototipos para utilizar en el área de la investigación en experimentos de locomoción, localización, reconocimiento de objetos y comunicación. En Agosto de 2007, Nao sustituye al perro robot Aibo de Sony como la plataforma estándar para la Robocup, un concurso internacional de robótica.

El modelo Nao de Aldebaran cuenta con un gran número de sensores y grados de libertad, de tecnología puntera, programable y controlable, permitiendo realizar movimientos precisos y coordinados. Nao ofrece una representación de la figura humana a tamaño reducido, mide 58 cm de altura, y pesa 4,3 kg y la carcasa está fabricada en plástico. Dispone de un cargador que funciona a 90-230 voltios, y una batería con una autonomía de unos 90 minutos aproximadamente. El robot dispone de módulos software que permiten la reproducción de archivos de texto a voz, localización basada en sonido, reconocimiento de formas y creación de efectos visuales mediante sus LEDs.

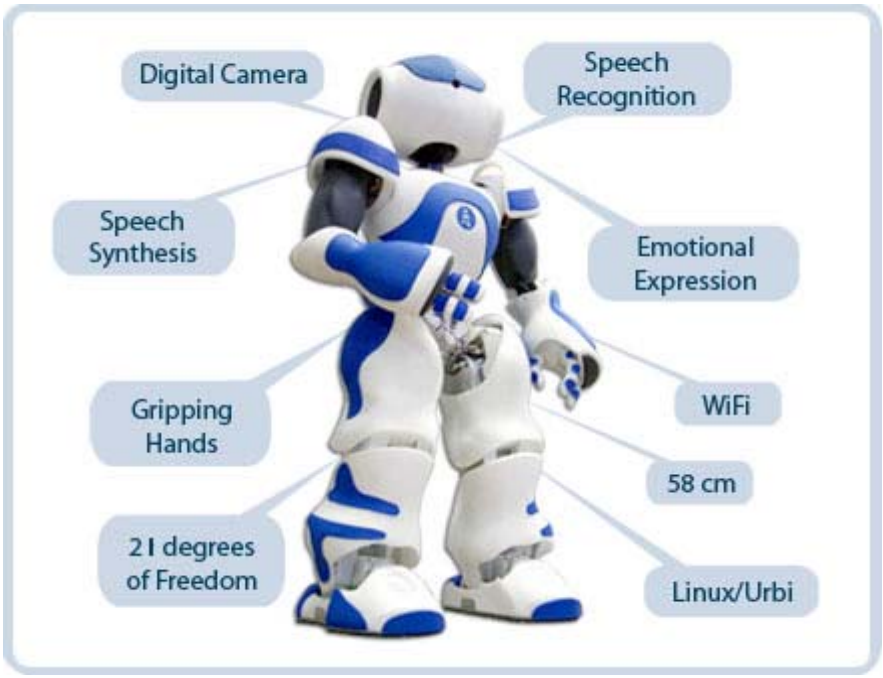

Figura 1.3

El robot Nao posee 21 grados de libertad, 2 en la cabeza, 4 en cada brazo, 1 en la pelvis y 5 en cada pierna. En la tabla 1.1 podemos ver la distribución de las articulaciones, sus movimientos, su rango de actuación y el tipo de actuador.

Part	Motion	Range (°)	Actuator type
Leg (left)	hip twist (45°)	-68 to 44	M1R11
	hip roll	-25 to 45	M1R11
	hip pitch	-100 to 25	M1R12
	knee pitch	0 to 130	M1R12
	ankle pitch	-75 to 45	M1R12
	ankle roll	-45 to 45	M1R11
Arm (left)	shoulder roll	0 to 95	M2R22
	shoulder pitch	-120 to 120	M2R21
	elbow roll	-120 to 120	M2R22
	elbow yaw	0 to 90	M2R21
Head	yaw	-90 to 120	M2R21
	pitch	-37 to 31	M2R22

Tabla 1.1

Además este modelo incorpora 2 altavoces, 4 micrófonos y 2 cámaras digitales integradas que facilitan la iteración con el medio. Respecto a los sensores, dispone de 32 sensores de efecto Hall, 1 giróscopo, 1 acelerómetro de 3 ejes y 1 sensor táctil en la cabeza del robot. Para hacerlo más vistoso, incorpora LEDs a lo largo de su cuerpo, que pueden servir para expresar desde un testigo de carga hasta un conjunto de instrucciones. La conexión al robot puede hacerse vía Wi-Fi IEE 802.11 g (inalámbrica) o Ethernet (con cable).

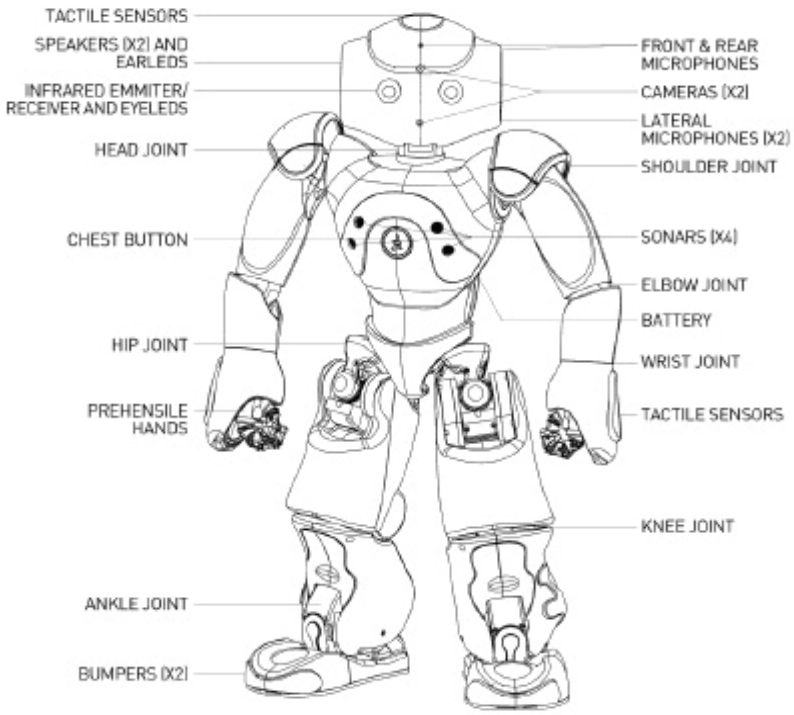

Figura 1.4

1.3 Entorno de desarrollo en Nao

El robot Nao se suministra con entorno de desarrollo llamado NaoQi. Este es un SDK que implementa un gran número de funciones de alto nivel y se encarga de comunicarse con los dispositivos de bajo nivel. También se encarga de manejar las comunicaciones. Para ejecutar órdenes sobre Nao es necesario hacerlo a través de

NaoQi. Se puede hacer también de forma remota (desde el PC) usando lenguajes de programación como C++, Python o URBI, y además podemos utilizar simuladores como Webots que son capaces de compilar programas ya que incorporan todas las librerías necesarias para crear programas propios. También se pueden compilar programas para ejecutarse en Nao, programados en C++, que se ejecutan en el sistema empotrado de Nao.

NaoQi tiene diversas características y usos determinadas que cabe destacar:

- Multi-lenguaje: se puede programar en diversos lenguajes.
- Ejecución de procedimientos secuencialmente, en paralelo, o llamadas por eventos.
- Modularidad: se pueden ejecutar programas en local o remotamente mediante llamadas a los módulos de NaoQi.
- Multi-plataforma: al tener un sistema operativo incrustado con comunicaciones vía Internet (Wi-Fi o Ethernet), las llamadas pueden realizarse desde cualquier sistema operativo.
- Uso de la memoria compartida: se puede leer, escribir o subscribirse a datos.

Aunque existen otras opciones para poder programar en este tipo de robots, NaoQi es el que la empresa Aldebaran adjunto con el robot.

1.4 Standard Platform League Robocup

Este modelo Nao de la empresa Aldebaran son usados en la Standard Platform League Robocup. Robocup es una iniciativa internacional que se centra en la investigación en robots e inteligencia artificial, mediante diferentes competiciones como pueden ser la Robocup Robot Soccer, Robocup Rescue, Robocup Home y Robocup Junior. La Robocup Standard Platform League (SPL) es una liga de fútbol de robots en la cual todos los equipos compiten con idénticos robots, los cuales operan totalmente autónomos, no hay un control externo ni por humanos ni por ordenadores. Los equipos se centran principalmente en el desarrollo de software sobre el robot, más que en la parte mecánica. Dicha liga se desarrolla mediante partidos de futbol en un entorno estructurado de equipos de 4 robots (rojo vs azul). El campo de fútbol sobre el que se juega es una alfombra de 7,4x5,4 metros. Las dimensiones se pueden ver reflejadas en la figura 1.5.

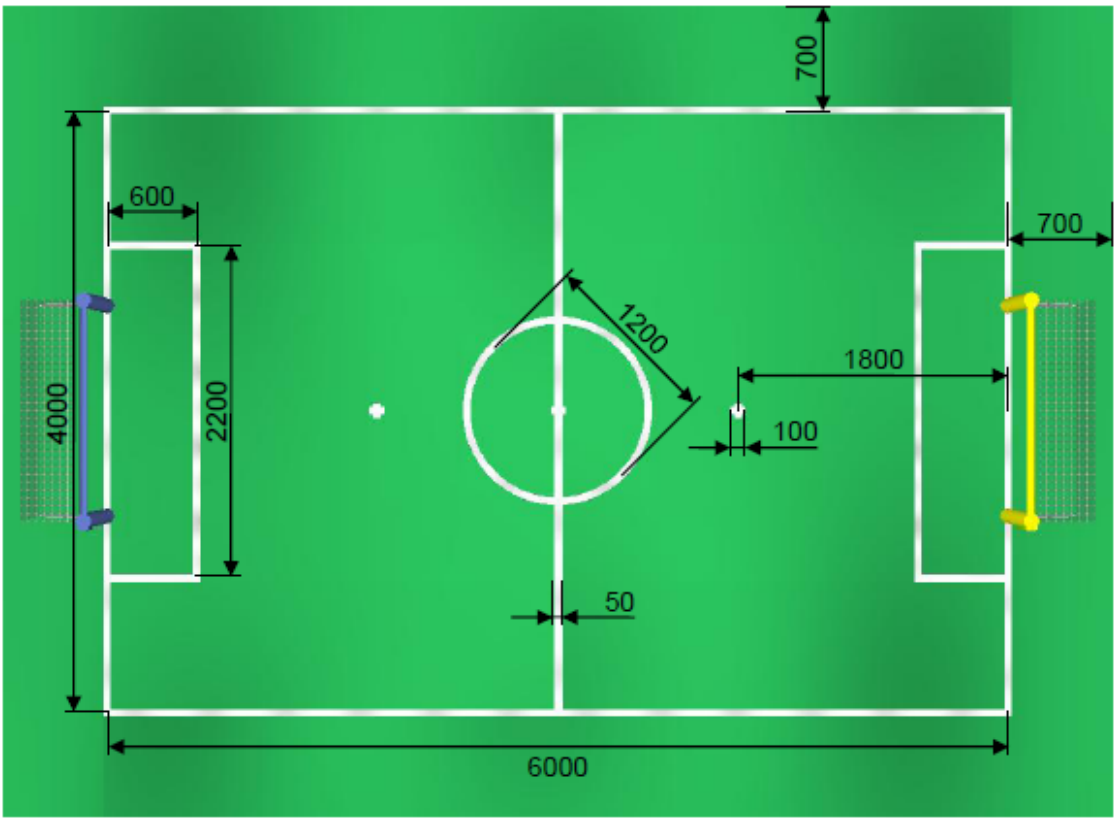

Figura 1.5

Otro de los elementos clave son las porterías, cuyas medidas se pueden ver en la figura 1.6.

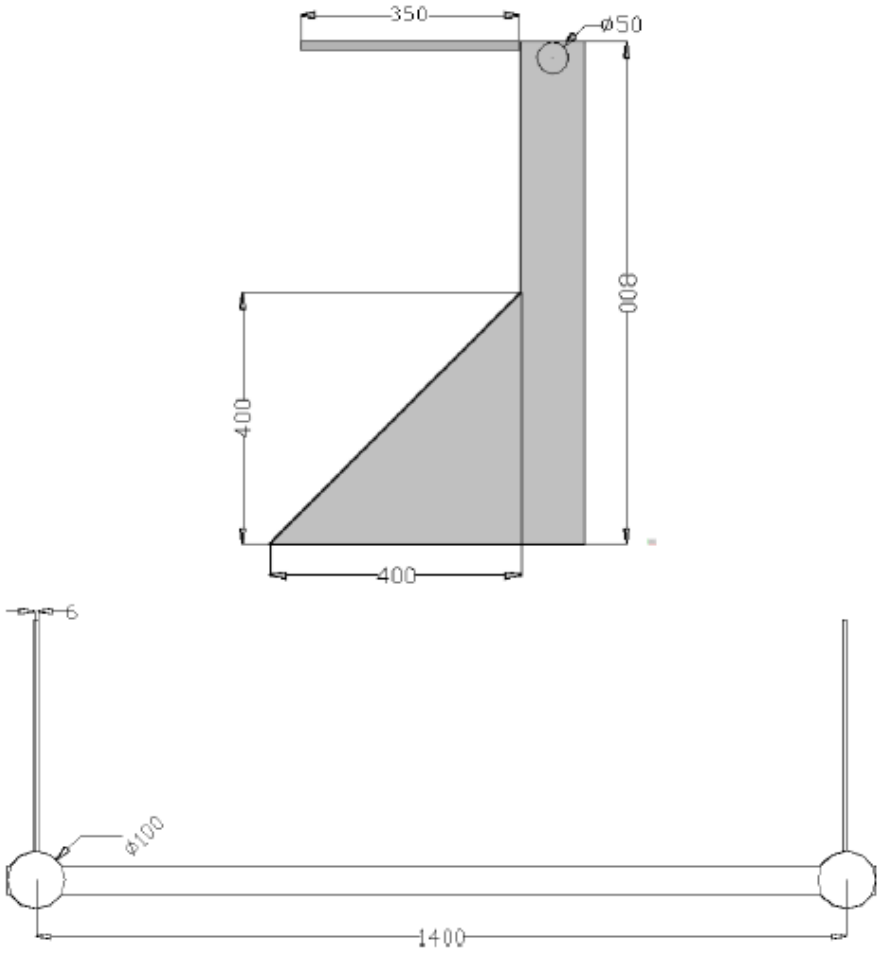

Figura 1.6

Los colores del campo son también muy importantes para que los sensores y cámara del robot funcionen correctamente. Todos los objetos del campo de la Robocup están ya codificados con un determinado color, que se pueden observar en la figura 1.7.

- El campo debe ser de color verde (no un color especifico, pero no debe ser muy oscuro).
- Las líneas del campo deben ser blancas.
- El equipo rojo defiende la portería amarilla.
- El equipo azul defiende la portería azul.
- Porterías. Los postes laterales y superior son ambos azul o amarillo, dependiendo del equipo. Los triángulos de soporte de los postes y la red son blancos.

La pelota oficial debe ser una pelota Mylec naranja. Sus características son 65 mm de diámetro y un peso de 55 gramos.

La comunicación de los equipos con los robots se basa exclusivamente en la utilización de las tarjetas de red wifi integradas en los Nao y los puntos de acceso que provee los organizadores del evento.

Figura 1.7

Las reglas del juego se basan en 2 partes de juego de 10 minutos cada una, con un descanso entre ellas de 10 minutos, durante los cuales los equipos pueden cambiar los robots, cambiar sus programas o cualquier actividad durante este tiempo de descanso. Los equipos deben de cambiar la portería que defienden y el color del equipo durante este descanso. Los goles son efectivos cuando la pelota entra completamente a través de la línea de gol. Después de un gol los robots vuelven a la posición inicial.

Los árbitros son las únicas personas permitidas dentro del área de juego. El árbitro principal es el que se encarga de la señalización del comienzo del juego, reiniciar el juego cuando se marque un gol, y señalar faltas o penaltis. El árbitro anuncia la interrupción del juego mediante un silbato, y después anuncia cual ha sido la causa de la interrupción. También anuncia el final de la primera parte con un pitido, y el final del partido con tres pitidos seguidos. Existen 2 árbitros asistentes que son los encargados de manejar los robots. Ellos son los encargados de inicializar los robots si el wireless no funciona o mover los robots manualmente si se llevan a cabo movimientos ilegales. Son también los encargados de recoger un robot y entregárselo al equipo, si ellos lo necesitan. Son los encargados de ejecutar las decisiones tomadas por el árbitro principal.

El principal cambio realizado del año 2010 a las actuales reglas en 2011, es que se ha cambiado el número de robots por equipo de 3 a 4, además de algunos cambios relacionados con los movimientos y las restricciones de los robots, adaptándolos a la nueva distribución de los robots en el campo.

Uno de los aspectos cruciales del sistema software empotrado en los robots son los ficheros de configuración que permiten fijar tanto aspectos de sistema (direcciones IP, número de jugador, etc.) como de control del mismo (tablas LUT, prioridades del sistema de threads, ficheros de comportamiento individual y de grupo, así como muchos otros). Debido a los cambios de normativa durante la competición y a los continuos cambios en los modos de ejecución de los robots, necesitamos una reconfiguración rápida y segura, que nos aporte dinamismo y minimizando el número de errores que puedan ocurrir.

Capitulo 2

OBJETIVOS

Este proyecto final de carrera tiene por objetivo dotar de flexibilidad al sistema empotrado del robot Humanoide Nao a la hora de configurar parámetros de control y ejecución de los procesos en ejecución.

Para cumplir con este objetivo general se ha trabajado en el diseño e implementación un sistema de gestión de ficheros de configuración bajo sintaxis XML, capaz de gestionar, modificar y actualizar los archivos de configuración de un robot, de manera rápida y eficaz, y crear un diccionario de datos en el sistema empotrado de control que el robot pueda interpretar para ejecutar dichos parámetros durante las competiciones.

Como etapas a cubrir en el desarrollo general del proyecto podemos enunciar:

- El diseño y desarrollo de una interfaz gráfica fácil de usar y amigable con el usuario, que sea capaz de mostrar toda la información necesaria, y de ejecutar las acciones necesarias.
- El desarrollo de las comunicaciones vía Internet, siguiendo unos protocolos seguros de forma que la información que enviemos llegue a su destino de forma fiable.
- El desarrollo de unos módulos de interpretación XML para sistemas de control, capaces de obtener la información contenida en este tipo de ficheros y de modificar estos ficheros para diferentes usos.
- La validación y verificación de la información contenida en estos ficheros de configuración, evitando de esta forma posibles errores que puedan llevar a comportamientos no deseados por parte del robot.

Por todo ello, esta aplicación está desarrollada para que dentro de la competición de la SPL Robocup, sea posible la modificación de estos archivos de configuración de manera rápida y segura, evitando de esta forma errores de sintaxis en los archivos que puedan a llevar a fallos en el comportamiento del robot durante la competición.

Capitulo 3

DISEÑO DE SISTEMA

3.1 Secure Shell (SSH)

El primer objetivo del proyecto es mantener una conexión abierta con un robot específico con la intención de poder hacer un intercambio de ficheros.

Para ello hacemos uso del protocolo SSH (Secure Shell) que nos sirve para acceder a maquinas remotas a través de la red. Nos permite manejar el equipo destino, mediante un intérprete de comandos, de manera remota desde nuestro propio equipo. Esto nos es de utilidad para el proyecto de forma que podremos analizar el sistema de archivos del robot y ejecutar órdenes en el robot desde nuestro equipo.

Existen diversos protocolos para la conexión y transferencia de archivos remotos, como puede ser Telnet, pero la principal ventaja de SSH con respecto a la seguridad, es que usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión. Para este tipo de conexiones, además de especificar la dirección del robot al que nos queremos conectar, deberemos introducir el usuario y contraseña, de formar que se asegura una conexión más segura.

Figura 3.1

Una de las formas de autenticación del usuario en SSH es mediante el cifrado de llave pública. Para este tipo de autenticación se dispone de una llave pública, que es la que se aloja en los equipos a los que se quiere acceder, y una llave privada, que solamente dispone el propietario y se mantiene en secreto. De esta manera cuando un usuario cifra algo con su llave privada, solamente es aceptado en el destino si se puede descifrar con la llave pública alojada en el equipo. Aun teniendo este par de llaves para el cifrado de mensajes, para mayor seguridad de la llave privada se puede pedir la contraseña al usuario. En sistemas Unix, la lista de llaves autorizadas para el acceso remoto está almacenada en la carpeta *authorized_keys* de ssh.

Como hemos dicho anteriormente el uso típico del protocolo SSH es el de conectarse a una maquina remota y poder ejecutar comandos, aunque también es posible la transferencia de archivos remotos mediante los protocolos SFTP y SCP, del que hablaremos más tarde. El puerto utilizado por defecto para este tipo de protocolos es el puerto 22 de TCP, aunque se puede modificar a un puerto no estándar como medida extra de seguridad.

Respecto a la arquitectura interna del protocolo SSH (definido en el RFC 4251) podemos ver las diferentes capas que la compone:

- Capa de transporte (RFC 4253). Esta capa se encarga de iniciar el intercambio de llaves, así como de la autenticación y de verificar la encriptación, comprensión e integración del mensaje. Provee a la capa superior de una interfaz capaz de mandar y recibir paquetes de 32.768 bytes cada uno (más de lo que está permitido en la implementación).

- Capa de autenticación de usuario (RFC 4252). Esta capa maneja la autenticación del cliente y le provee de distintos métodos de autenticación para poder conectarse. La autenticación debe ser parte del cliente, por lo que cuando se hace la petición de contraseña, es la parte cliente la que debe facilitarla y no la parte servidor. El servidor simplemente contesta a la petición de autenticación por parte del cliente. Los posibles métodos incluidos en la autenticación son los siguientes:

- contraseña: un método de autenticación mediante contraseña de texto plano por parte del cliente, con la facilidad de que la contraseña puede ser cambiada por parte del usuario.
- clave pública: un método de autenticación basado en la clave pública, que normalmente soporta al menos los pares de llaves DSA o RSA.
- interactiva por teclado: un modo versátil en el que el servidor es el que pide la contraseña y el cliente es el que debe de introducir la información por pantalla y enviársela de nuevo al servidor. Es usada sobre todo para la autenticación de contraseñas una sola vez como puede ser S/Key o SecurID.
- GSSAPI: un método de autenticación el cual provee un esquema extensible que provee SSH en el que se usan mecanismos externos como Kerberos 5 o NTLM. Este tipo de métodos son normalmente usados para implementaciones comerciales de SSH para su uso en organizaciones.

- Capa de conexión (RFC 4254). Esta capa define el concepto de canales, petición de canales y peticiones globales usando los servicios que provee SSH. Una conexión simple de SSH puede albergar diferentes canales simultáneamente, cada una de ellas transfiriendo datos en ambas direcciones. La petición de canales es usada para retransmitir datos específicos en canales fuera de la banda normal, como puede ser el cambio de tamaño de la ventana de un terminal.

- El almacenamiento SSHFP DNS (RFC 4255), provee un almacenamiento de claves públicas de host para añadir una verificación de la autenticación en las diferentes sesiones del host.

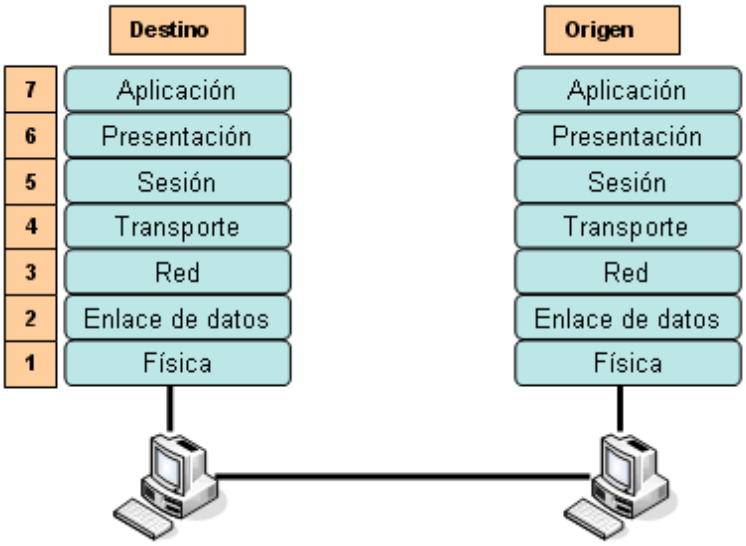

Figura 3.2

Este tipo de arquitectura abierta provee una considerable flexibilidad, permitiendo a SSH ser usado para una variedad de propósitos usando el Secure Shell. Las funcionalidades de la capa de transporte son comparables a la Capa de Seguridad de Transporte (TLS); la autenticación del usuario es extensible con múltiples métodos de autenticación; y la capa de conexión provee la habilidad de abrir múltiples sesiones secundarias en una sola conexión SSH, una característica comparable a BEEP y no disponible en TLS.

3.2 Secure Copy (SCP)

Tras analizar cuáles son las principales del protocolo SSH, con el que principalmente nos vamos a conectar remotamente a un equipo destino y ejecutar comandos sobre este, nos vamos a centrar ahora en el intercambio de ficheros.

Para el intercambio de ficheros entre equipos remotos hacemos uso de SCP o Secure Copy, el cual es un medio de transferencia seguro de archivos informáticos entre un host local y otro remoto o entre dos hosts remotos, usando el protocolo Secure Shell SSH. La característica principal de SCP es que los datos son cifrados durante su transferencia, para evitar que potenciales packet sniffers extraigan información útil de los paquetes de datos. Sin embargo, el protocolo mismo no provee autenticación y seguridad, sino que espera que el protocolo subyacente, SSH, lo asegure.

La descripción de un formato de comunicación en las cabeceras enviadas por la red es la que podemos ver en la figura 3.3:

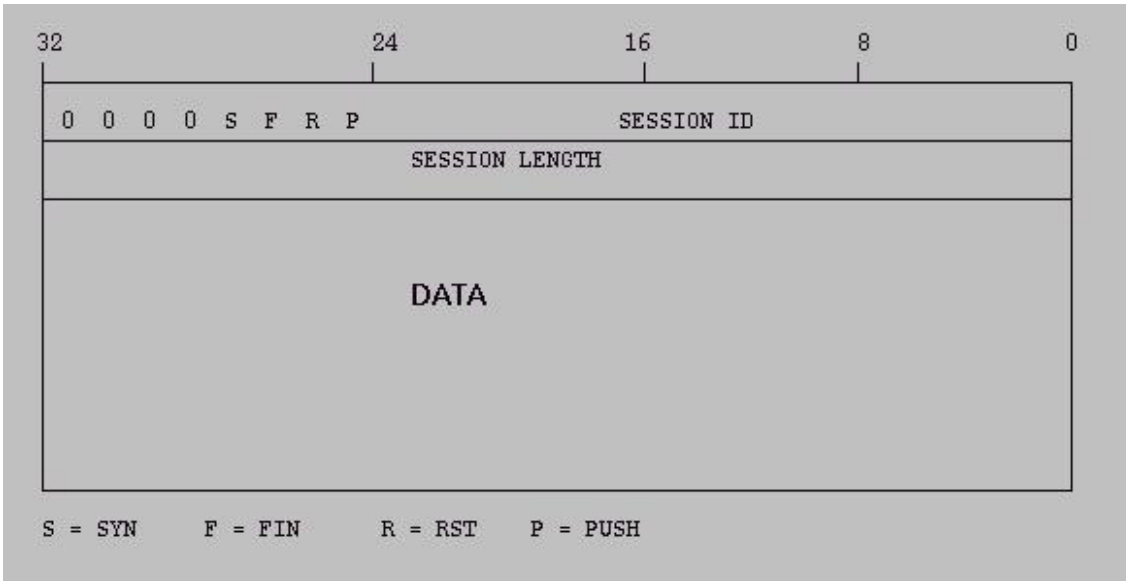

Figura 3.3

El modo SCP es un protocolo simple que deja al cliente y al servidor tener múltiples conversaciones sobre una TCP normal. Este protocolo está diseñado para ser simple de implementar. El servicio principal es el control del dialogo entre el servidor y el cliente, administrando sus conversaciones y agilizadas a un alto porcentaje. SCP puede solicitar de manera iterativa cualquier contraseña para establecer una conexión con un host remoto

El protocolo SCP implementa la transferencia de archivos únicamente. Para ello se conecta al host usando SSH y allí ejecuta un servidor SCP. Para realizar la subida, el cliente proporciona al servidor los archivos que desea subir y opcionalmente puede incluir otros atributos, como permisos, fechas, etc., lo que es una ventaja respecto a otros protocolos como FTP. Para descargar, el cliente enviar una solicitud por los archivos que desea descargar, el proceso de descarga está dirigido por el servidor y es el que se encarga de la seguridad del mismo.

El cliente SCP más utilizado es el programa scp del intérprete de comandos, que está incorporado en la mayoría de las implementaciones de SSH. La sintaxis del programa scp es similar a la sintaxis de la orden de copia cp:

scp usuario@host:directorio/ArchivoOrigen ArchivoDestino

scp ArchivoOrigen usuario@host:directorio/ArchivoDestino

Por tanto con los protocolos SSH y SCP, anteriormente explicados, tenemos las funcionalidades de conexión remota, ejecución de comandos remotamente y la transferencia de archivos entre equipos remotos.

Todo esto nos es muy útil en el proyecto para la conexión remota a un robot, y la transferencia de archivos entre el equipo local y el robot remoto, pudiendo hacer modificaciones sobre archivos contenidos en el robot y pudiendo enviarlos de nuevo tras la modificación.

3.3 Archivos de configuración .INI

Esta es la función principal de nuestro proyecto, y los archivos que queremos modificar son los archivos de inicialización del robot, los llamados archivos .INI.

Un archivo .INI consiste en un simple archivo de texto ASCII que contiene dos tipos de entradas:

- Secciones: permiten agrupar parámetros relacionados.
- Valores: definen parámetros y su valor. Primero se define el nombre del parámetro y después su valor separado por el signo de igualdad (=).
- Comentarios: permiten explicar el propósito de una sección o parámetro. Los comentarios comienzan con el carácter punto y coma (;) u otro caracteres como (# o //).

El significado de secciones y valores no está bien definido y cada aplicación puede reaccionar de manera diferente ante secciones duplicadas, parámetros duplicados o valores, que pueden consistir en texto, números o listas separadas por comas. Esto depende de la aplicación.

Un ejemplo de nuestro archivo INIT.INI tiene la forma que se muestra en la figura 3.4, donde se pueden observar los parámetros definidos con sus valores correspondientes, y los comentarios añadidos al fichero.

```
#--------------------------------------------------------------------
# INIT.INI
#--------------------------------------------------------------------
# Comments begin with # or //
#
# Each key/value pair on one line, in the format: KEY = VALUE
#
# Invalid lines are ignored, as is anything after the VALUE string
#
NO_IP=1
# Team Number - Game Controller ID
TEAM_NUMBER = 15

# Team Color -> 0: BLUE, 1: RED
TEAM_COLOR = 1

# Robot Number -> 1: GoalKeeper, 2-3: FieldPlayers
ROBOT_NUMBER = 3

# Periods
PERIOD_PERCEPTION = 0.18
PERIOD_CONTROL = 0.04

# Priorities
CMD_PRIORITY   = 90
CTRL_PRIORITY  = 91
PAM_PRIORITY   = 93
LOCO_PRIORITY  = 92
```

Figura 3.4

Tras ejecutar una aplicación, sus parámetros de configuración por defecto son los que están almacenados en el archivo .INI, y son los que se ejecutan al inicio para definir el estado inicial de la aplicación. Adicionalmente, cualquier usuario puede abrir el fichero .INI con un editor de texto y modificarlo, en caso de un mal funcionamiento de aplicación o que se quieran modificar los parámetros iniciales.

Pese a todo esto, queremos modificar estos archivos .INI por archivos XML. Con ellos conseguimos una organización de los ficheros en un formato más estructurado y una sintaxis más adecuada para que nuestro programa sea capaz de analizar y modificar estos archivos.

3.4 Archivo XML

El archivo XML, siglas en inglés de eXtensible Markup Language, es un metalenguaje extensible de etiquetas desarrollado por el Worl Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos; por tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de textos, hojas de cálculo, etc. XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XML y sus extensiones han recibido multitud de críticas por su nivel de detalle y complejidad. El mapeo del modelo de árbol básico de XML hacia los sistemas de tipos de lenguajes de programación o bases de datos puede ser difícil, especialmente cuando se utiliza XML para el intercambio de datos altamente estructurados entre aplicaciones. Pero este no era su primer objetivo al diseñarlo y en cambio dispone de otras muchas ventajas:

- Es extensible: después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML, por lo que posibilita el empleo de cualquiera de los analizadores disponibles.
- Si un tercero decide usar un documento creado en XML, es sencillo de entender su estructura y procesarla. Mejora así la compatibilidad entre aplicaciones, y podemos comunicar aplicaciones de distintas plataformas sin que importe el origen de los datos.
- Y una de las más importantes, es que transformamos fatos en información, pues se le añade un significado concreto y los asociamos a un contexto, con lo cual tenemos flexibilidad para estructurar documentos.

En cuanto a la estructura de un documento XML, la tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y

reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Las etiquetas tiene la forma <*nombre*>, donde nombre es el nombre del elemento que se está señalando.

En la figura 3.5 podemos observar el archivo de inicialización INIT.INI anterior en formato XML.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <root>
3      <NO_IP>1</NO_IP>
4      <TEAM_NUMBER > 15</TEAM_NUMBER >
5      <TEAM_COLOR > 1</TEAM_COLOR >
6      <ROBOT_NUMBER > 3</ROBOT_NUMBER >
7      <PERIOD_PERCEPTION > 0.18</PERIOD_PERCEPTION >
8      <PERIOD_CONTROL > 0.04</PERIOD_CONTROL >
9      <CMD_PRIORITY   > 90</CMD_PRIORITY   >
10     <CTRL_PRIORITY  > 91</CTRL_PRIORITY  >
11     <PAM_PRIORITY   > 93</PAM_PRIORITY   >
12     <LOCO_PRIORITY  > 92</LOCO_PRIORITY  >
13 </root>
14
```

Figura 3.5

En este archivo INIT.xml se puede observar la misma información que en el archivo anterior, pero de una manera estructurada, donde los parámetros pasan a ser las etiquetas y el valor correspondiente está definido entre las etiquetas. Esta sería una manera simple de representar nuestra información en forma de árbol, donde la etiqueta *root* es nuestro nodo raíz, y partir de él están representados el resto de parámetros con sus valores como nodos hojas.

Ahora bien, para que un documento XML funcione correctamente y no contenga errores, debe de seguir algunas definiciones básicas. Los documentos denominados “bien formados” (del inglés well formed) son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente con cualquier analizador sintáctico (parser) que cumpla con la norma.

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas y los elementos con contenido deben estar correctamente cerrados.
- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial.
- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.
- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos “entendibles” por las personas.

Por último vamos a hablar de las partes que compone un documento XML. Un documento XML está formado por el prólogo y por el cuerpo del documento, así como texto de etiquetas que contiene una gran variedad de efectos positivos o negativos en la referencia opcional a la que se refiere el documento, hay que tener mucho cuidado de esa parte de la gramática léxica para que se componga de manera uniforme.

- Prologo. Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento, etc.
- Cuerpo. A diferencia del prólogo, el cuerpo no es opcional en un documento XML. El cuerpo debe contener solo un elemento raíz, característica indispensable también para que el documento esté bien formado, sin embargo es necesaria la adquisición de datos para su buen funcionamiento.
- Elementos. Dentro del cuerpo están los elementos XML que pueden contener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.
- Atributos. Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Estos deben de ir entre comillas.

- Comentarios. A modo informativo para el programador, los comentarios han de ser ignorados por el procesador. Los comentarios en XML tienen el siguiente formato: *<!- - Esto es un comentario - ->*

Que un documento esté bien formado solamente se refiere a su estructura sintáctica básica, es decir, que se componga de elementos, atributos y comentarios como XML especifica que se escriban. Ahora bien, cada aplicación de XML, es decir, cada lenguaje definido con esta tecnología, necesitará especificar cuál es exactamente la relación que debe verificarse entre los distintos elementos presentes en el documento.

Cualquier procesador de texto, que sea capaz de producir archivos .txt es capaz de generar XML, aunque en los entornos de desarrollo, se facilita, ya que se reconoce los formatos, y ayuda a generar un XML bien formado.

Una de las herramientas que más tarde va a ser utilizada y vale la pena explicarla son los parsers. Un parser o analizador sintáctico lee el documento XML y verifica que es XML bien formado, algunos también comprueban que el código XML sea válido. El parser es la herramienta principal de cualquier aplicación XML, no solamente podemos comprobar que nuestros documentos son bien formados y validos, sino que también podemos incorporarlos a nuestras aplicaciones, de manera que estas pueden manipular y trabajar con documentos XML.

La parte importante de los parsers, radica en comprender su organización, para saber cómo trabajan y de esta forma entender como extraemos la información. Para hacernos una idea del funcionamiento, deberemos imaginar el documento XML como un árbol cuya raíz es la clase principal, y el resto como hijos u hojas del mismo. De esta forma, entendiendo el documento XML como árbol, podemos recorrer todos los hijos de la raíz e ir extrayendo la información que nos sea útil. Nosotros utilizamos este tipo de organización de forma que, definimos una etiqueta root que va a ser nuestra raíz, y a partir de ella nacen todas sus hojas, que son nuestros atributos con los valores definidos.

La forma que podría tener un árbol que representará la información de uno de los documentos XML que vamos a utilizar, como el del ejemplo anterior, sería el de la figura 3.6.

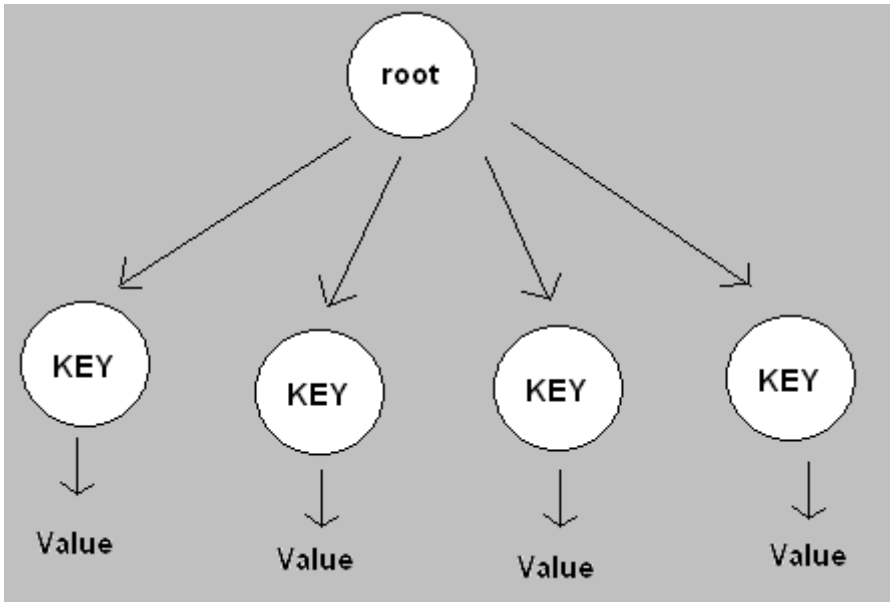

Figura 3.6

3.5 Diccionario de datos

El diccionario de datos es una estructura de datos donde vamos a guardar la información que vamos a ir leyendo del archivo XML. En nuestro caso esta estructura de datos va a estar formada por arrays que se van a encargar de almacenar tanto las claves como los valores de los atributos que vayamos a utilizar.

El diccionario de datos se encarga de verificar que no haya atributos repetidos dentro del mismo documento, y además de esto nos encargamos de revisar los valores de los atributos, en caso de que alguno de ellos no estuviera definido o tuviera un valor no deseado. Con esto nos aseguramos el correcto funcionamiento de nuestro robot, al pasarle valores de configuración correctos y asegurándonos que no hay atributos repetidos, lo que podría ocasionar fallos de coherencia a la hora de cargar dichos valores.

Con todo esto el diccionario de datos nos proporciona una utilidad flexible para almacenar nuestros datos y poder utilizarlos posteriormente en el robot, además de una gestión de variables vacías, evitando de este modo comportamientos no deseables por el robot durante su ejecución.

Capitulo 4

DESARROLLO DE SOPORTE DE COMUNICACIONES

En este capítulo vamos a pasar a describir la implementación de las diferentes partes del programa. Básicamente tenemos cuatro apartados que hemos implementado en la aplicación: la conexión SSH, el intercambio de ficheros mediante SCP, la traducción de ficheros de inicialización .INI a ficheros XML y la edición y modificación de los ficheros XML.

4.1 Conexión SSH

Como hemos dicho en el capitulo anterior, la conexión SSH nos permite establecer una conexión remota con otro equipo, en nuestro caso un robot, y ejecutar comandos remotamente a través de la red. Para la implementación de la conexión SSH nos hemos apoyado en la librería JSch. JSCH (Java Secure Channel) es una librería de utilidades de comunicación para JAVA que nos ofrece JCraft con licencia BSD. Principalmente esta librería permite conectarse a un servidor vía SSH.

En nuestro código tenemos dos funciones básicas como son conectar y exeComando. La función conectar, como podemos ver en la figura 4.1, se encarga de iniciar una sesión en el puerto 22 de TCP mediante el usuario y el host al que nos queremos conectar, y solicitando la contraseña para poder autentificarse en esta sesión. Luego simplemente se inicia la conexión, y queda abierta para poder ejecutar comandos remotamente. Tenemos otra función que se encarga de desconectar la sesión cuando hemos acabado, o si queremos conectarnos a otro equipo remoto.

```
15  public class ConectionSSH {
16
17
18      private final static String newline = "\n";
19      Channel channel;
20      Session session;
21
22
23      public void conectar(String user, String host, String pasw) throws JSchException{
24
25          JSch jsch = new JSch();
26          session = jsch.getSession(user, host, 22);
27          session.setPassword(pasw);
28          session.setConfig("StrictHostKeyChecking", "no");
29          session.connect(30000);
30
31      }
```

Figura 4.1

La otra función básica es la de exeComando, que se encarga de ejecutar comandos, como si fuera una línea de intérprete de comandos. En esta función simplemente cabe destacar que abrimos un canal sobre la sesión ya abierta anteriormente, donde podremos ejecutar los comandos. Si el comando que deseamos ejecutar es correcto, tendremos una salida por pantalla con el resultado, mientras que si el comando es erróneo no nos mostrará nada por pantalla, y en todo caso podremos seguir introduciendo comandos por la línea de interprete.

```
32      public void exeComando(String comando, javax.swing.JTextArea resultado) {
33          try {
34              channel = session.openChannel("exec");
35              ((ChannelExec) channel).setCommand(comando);
36              channel.setInputStream(null);
37
38              ((ChannelExec) channel).setErrStream(System.err);
39
40              InputStream in = channel.getInputStream();
41
42              channel.connect();
```

Figura 4.2

Con esto conseguimos la conexión SSH a un equipo remoto y poder ejecutar comandos sobre él.

4.2 Intercambio de ficheros

A continuación nos vamos a centrar en el intercambio de ficheros mediante el protocolo SCP. Para ello nos hemos apoyado de nuevo en la librería JSch, que se encarga de conectarnos remotamente con el equipo que nosotros solicitemos.

La mentalidad es la misma que en la conexión SSH, abriendo una sesión con un user y un host determinado y solicitando la contraseña correspondiente. Una vez conectados a la maquina remotamente, abrimos un canal sobre la sesión y ejecutamos los comandos. En este caso el comando a ejecutar es único, la orden de copia remota scp, cuya sintaxis es la siguiente:

scp usuario@host:directorio/ArchivoOrigen ArchivoDestino

scp ArchivoOrigen usuario@host:directorio/ArchivoDestino

Como podemos ver, tenemos dos posibilidades con esta instrucción, enviar un ArchivoOrigen a un destino, o traer un ArchivoDestino al origen. La instrucción a ejecutar es la misma pero cambia su sintaxis, y es por ello que tenemos las dos posibilidades en la aplicación. Tras ejecutar la instrucción y ver que los archivos son correctos, se procede a la copia remota del fichero de un equipo a otro. En nuestro caso los archivos que nos interesa traer son archivos de inicialización .INI para poder traducirlos a XML, o principalmente archivos XML que están alojados en el robot y que queremos modificar y volver a enviarlos al robot de nuevo.

4.3 Traducción de .INI a XML

Otra de las opciones que se han implementado en la aplicación es la traducción de ficheros .INI en ficheros XML que son los que vamos a utilizar más tarde. Actualmente los ficheros de configuración son del formato .INI, que hemos detallado anteriormente, y el nuevo objetivo es poder trabajar con archivos XML.

Para implementar esta funcionalidad, lo primero que hacemos es leer línea por línea el fichero .INI, y descartamos las líneas que contengan comentarios o líneas vacías que no nos aportan información. Con ello conseguimos quedarnos solamente con las líneas que contienen información que nos interesa, y que vamos a procesar a continuación. Los archivos .INI, tienen el siguiente formato *“key=value”*, lo que nos va a facilitar la lectura de los parámetros y los valores de configuración. Una vez leídos, nos toca darle formato al archivo XML. Para ello, introducimos una primera línea con información, e introducimos la raíz de nuestro archivo que se va a llamar *“root”*. El resto de información, que van a ser los parámetros y sus valores, van a ser todos hojas de la raíz, siendo cada parámetro una hoja del árbol. Cuando ya hayamos terminado de leer completamente el fichero y escrito los parámetros con sus respectivos valores en el archivo XML, cerraremos la raíz *“root”*.

El código de la figura 4.3 es la forma que tiene la función que se encarga de esta acción.

```
279     private void jButtonconvertActionPerformed(java.awt.event.ActionEvent evt) {
280         // TODO add your handling code here:
281
282         if(ficheroini.contains(".ini") && ficheroxml.contains(".xml")){
283
284             //Creamos fichero XML a partir del .INI
285             StringBuilder StrXML = new StringBuilder();
286
287             File archivo = null;
288             FileReader fr = null;
289
290             String key=null;
291             String value=null;
292
293             PrintStream xml = null;
294             try {
295                 xml = new PrintStream(ficheroxml);
296             } catch (FileNotFoundException ex) {
297                 Logger.getLogger(ControlPanel.class.getName()).log(Level.SEVERE, null, ex);
298             }
299             java.util.Date fecha = new Date();
300             StrXML.append("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n");
301
302             StrXML.append("<root>\n");
303
304             try{
305                 archivo = new File (ficheroini);
306                 String linea;
307                 fr = new FileReader (archivo);
308                 BufferedReader br = new BufferedReader(fr);
309                 while((linea=br.readLine())!=null){
310                     if(!linea.startsWith("#") && !linea.isEmpty()){
311                         System.out.println(linea);
312                         StringTokenizer tokens = new StringTokenizer(linea,"=");
313                         key=tokens.nextToken();
314                         value=tokens.nextToken();
315                         StrXML.append("\t<").append(key).append(">");
316                         StrXML.append(value);
317                         StrXML.append("</").append(key).append(">\n");
318                     }
319
320                 }
321
322                 StrXML.append("</root>");
```

Figura 4.3

Tras conseguir esta traducción, ya disponemos del fichero XML con el que vamos a trabajar a partir de ahora. Este archivo va a contener la misma información que el antiguo fichero de configuración, pero con el formato XML. Ahora nuestro objetivo es extraer la información contenida en el archivo XML. Para ello hacemos un “parseo” del archivo, recorriendo todo el árbol; primero extraemos la raíz del árbol con la instrucción “getRootElement” y a partir de ahí conseguimos la lista de hijos, que a continuación recorremos para conseguir la información.

```
330                 //Parseo del XML
331
332
333                 // Creamos el builder basado en SAX
334                  SAXBuilder builder = new SAXBuilder();
335                  // Construimos el arbol DOM a partir del fichero xml
336                  Document doc = null;
337                 try {
338                     doc = builder.build(new FileInputStream(ficheroxml));
339                 } catch (JDOMException ex) {
340                     Logger.getLogger(XMLconvert.class.getName()).log(Level.SEVERE, null, ex);
341                 }
342
343                  // Obtenemos la etiqueta raíz
344                  Element raiz = doc.getRootElement();
345                  // Recorremos los hijos de la etiqueta raíz
346                  List<Element> hijosRaiz = raiz.getChildren();
347                  int i=0;
348                  for(Element hijo: hijosRaiz){
349                     // Obtenemos el nombre y su contenido de tipo texto
350                     String nombre = hijo.getName();
351                     editor.nombrar_label(nombre,i);
352                     String texto = hijo.getValue();
353                     i++;
354
355                  }
```

Figura 4.4

Este tipo de parseo, en forma de árbol, lo vamos a utilizar continuamente durante la aplicación puesto que vamos necesitar extraer la información de los ficheros, para actualizarlos y modificar o añadir determinados parámetros.

4.4 Edición y modificación XML

Y por último, nos vamos a centrar en la edición y modificación de los ficheros XML. Como hemos dicho, nuestro objetivo principal en este proyecto es el de poder modificar ficheros de configuración del robot y volver a enviarlos. La forma de proceder va a ser la de leer un fichero XML ya creado y con información, o la de crear un fichero desde el inicio, con la ayuda de una plantilla que nos va a ayudar en este trabajo.

Si queremos construir un fichero de configuración desde cero, vamos a tener la ayuda de una plantilla, que vamos a cargar inicialmente, en el que se nos van a presentar todos los valores posibles de los parámetros que podemos dar valor en cada tipo de fichero. Para ello hemos definido unas plantillas con los siguientes campos:

- Edit: Indicaremos si el campos es editable “true” (podemos escribir datos sobre él), o si no lo es “false” (deberemos elegir los datos de una lista). Este campo

tiene un valor especial “extra” para ciertos campos que van a aparecer dependiendo del número de extras que indiquemos.

- Tipo: Indicaremos el tipo de variable que vamos a utilizar en este campo, normalmente utilizaremos los tipos “int” para enteros o “string” para cadenas de texto.
- Min_Range: En el caso de que el campo sea editable, indicaremos cual es el valor mínimo que se le puede al atributo.
- Max_Range: Igual que en el caso anterior, si el campo es editable, indicaremos el valor máximo que puede tomar el atributo.
- Default_Value: Por defecto, le asignaremos un valor al campo.
- Enum_Value: En el caso de que el campo no sea editable, necesitaremos ofrecer al usuario una lista con los valores que puede elegir. En este campo añadiremos los valores que va a contener la lista, para que el usuario pueda elegir uno entre ellos.

En nuestra aplicación esto se verá reflejado en una pantalla en la que iremos rellenando los valores de los parámetros, o dejando en blanco (valor “null”) los que no queramos configurar.

El documento de la figura 4.5 es la forma que tendría un ejemplo de plantilla para un tipo de fichero determinado, en el que se pueden observar sus campos y los valores permitidos.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<root>
    <INITIAL_ROLE>
            <edit> false </edit>
            <tipo> string </tipo>
            <min_range> null </min_range>
            <max_range> null </max_range>
            <default_value> follow </default_value>
            <enum_value> och follow jorge roberto null </enum_value>
    </INITIAL_ROLE>
    <INITIAL_POSX>
            <edit> true </edit>
            <tipo> int </tipo>
            <min_range> 1000 </min_range>
            <max_range> 2000 </max_range>
            <default_value> 1200 </default_value>
            <enum_value> null </enum_value>
    </INITIAL_POSX>
    <INITIAL_POSY>
            <edit> true </edit>
            <tipo> int </tipo>
            <min_range> 1000 </min_range>
            <max_range> 2000 </max_range>
            <default_value> 1500 </default_value>
            <enum_value> null </enum_value>
    </INITIAL_POSY>
```

Figura 4.5

La otra opción es la de cargar un fichero XML con valores ya configurados. En este caso nosotros nos vamos a limitar a mostrar estos campos por pantalla y permitir su modificación mediante los campos correspondientes. Ambas opciones se basan en la plantilla del fichero que vamos a editar, el cual nos proporcionará todos los datos acerca del tipo de los parámetros y los valores que estos pueden tomar.

La implementación de esta parte, se basa como hemos dicho en cargar una plantilla definida en un archivo XML, el cual vamos a tener que parsear, para extraer la información de cada uno de los parámetros. El parseo de las plantillas es muy parecido al de los ficheros, recorriendo todos los hijos de la raíz, y en este caso, dentro de cada hijo dispondremos de toda la información. Toda esta información se va a almacenar en arrays, que son los que luego deberemos de consultar para ver que se cumplen todas las restricciones de los valores introducidos. Cada parámetro va a disponer de 5 arrays donde va a almacenarse toda la información.

- Listatext: va a contener el nombre del parámetro.

- Listacombo: va a contener la lista de valores permitidos, en el caso de que el parámetro no sea editable y debamos elegir entre unos valores dados. Este campo solo va a tener información si el parámetro no es editable.
- Listaedittext: en el caso de que el parámetro sea editable, este es el campo donde vamos a almacenar el valor que se va a introducir por pantalla.
- Listarangomin: en el caso de que el parámetro sea editable, este parámetro indicará el rango mínimo que puede tomar el valor introducido. Nos servirá para comprobar esta restricción.
- Listarangomax: es el mismo caso que el anterior, pero indicará el rango máximo.

Esto nos va a permitir, no poder introducir una cadena de texto en un parámetro que solo admite enteros, o de introducir enteros que sobrepasan el rango permitido por el parámetro.

A la hora de modificar un determinado parámetro, deberemos introducir en su campo correspondiente el valor que le queramos asignar, o elegir de la lista de sus posibles valores. Con esto lo que haremos será modificar en el array, el valor correspondiente al parámetro que queramos modificar, siempre consultando que cumple todas las restricciones exigidas, y que más tarde escribiremos en el nuevo archivo XML.

Hay algunos casos especiales en el que determinados ficheros tienen campos extras, con información adicional, no especificados desde un principio y que pueden cambiar. Esto lo hemos abordado mediante el campo editable de la plantilla, donde le hemos asignado el valor extra. Con esto lo que conseguimos es identificar este tipo de parámetros especiales, que luego tendremos que manejar de diferente manera, para poder introducirlos según la cantidad de extras que el usuario quiera añadir al fichero.

A la hora de aplicar los cambios y guardar el fichero, lo que hacemos es primero de todo verificar que los parámetros cumplen las restricciones exigidas. Para ello consultamos los arrays que contienen la información de los rangos y los tipos permitidos por el valor. Si todo esto se cumple, pasamos a guardar el fichero en formato XML, creando una raíz *“root”*, y creando tantas hojas en el árbol como parámetros vayamos a añadir al fichero.

La figura 4.6 sería un ejemplo del fichero final, con los parámetros y sus valores modificados:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--Fichero creado el: Thu Jul 28 12:56:04 CEST 2011-->
<root>
    <INITIAL_ROLE>roberto</INITIAL_ROLE>
    <INITIAL_POSX> 1200</INITIAL_POSX>
    <INITIAL_POSY> 1500</INITIAL_POSY>
    <KICKOFF_POSX> 1200</KICKOFF_POSX>
    <KICKOFF_POSY> 1500</KICKOFF_POSY>
    <HFSM_PLAY_ALONE>jorge</HFSM_PLAY_ALONE>
    <HFSM_ROLE_CAPTAIN>roberto</HFSM_ROLE_CAPTAIN>
    <HFSM_ROLE_ATTACKER>follow</HFSM_ROLE_ATTACKER>
    <HFSM_ROLE_DEFENDER>follow</HFSM_ROLE_DEFENDER>
    <HFSM_ROLE_SUPPORTER>jorge</HFSM_ROLE_SUPPORTER>
    <CTRL_RUN_TYPE>RUN_AS_ROBOCUP_MATCH</CTRL_RUN_TYPE>
    <ROLE_BLOCKED_AS_PLAY_ALONE>FALSE</ROLE_BLOCKED_AS_PLAY_ALONE>
    <ROBOT_BLOCKED_AS_CAPTAIN>null</ROBOT_BLOCKED_AS_CAPTAIN>
    <HFSM_EXTRA_NUMBER> 4</HFSM_EXTRA_NUMBER>
    <HFSM_EXTRA_NUMBER1>0</HFSM_EXTRA_NUMBER1>
    <HFSM_EXTRA_NUMBER2>0</HFSM_EXTRA_NUMBER2>
    <HFSM_EXTRA_NUMBER3>0</HFSM_EXTRA_NUMBER3>
    <HFSM_EXTRA_NUMBER4>4</HFSM_EXTRA_NUMBER4>
</root>
```

Figura 4.6

4.5 Diccionario de datos

El caso del diccionario de datos lo abordamos de forma diferente. Como hemos dicho anteriormente, el robot Nao permite su programación en lenguaje C++ entre otros. Nosotros hemos utilizado este lenguaje de programación para definirnos una clase “dictionary”, donde vamos a gestionar el diccionario de datos.

El procedimiento va a ser similar al anterior, ya que tenemos que leer un archivo XML e introducir sus valores en el diccionario de datos. Para ellos vamos a hacer uso de un parser para C++. El parser que hemos utilizado es TinyXML, ya que es un pequeño parser, muy simple programado en C++, además de ser libre y open source. Aunque es un parser simple, tiene todas las funcionalidades que vamos a necesitar en nuestro proyecto. Es capaz leer y escribir archivos XML y de parsear un archivo XML en un árbol DOM, de manera que la lectura va a ser simple

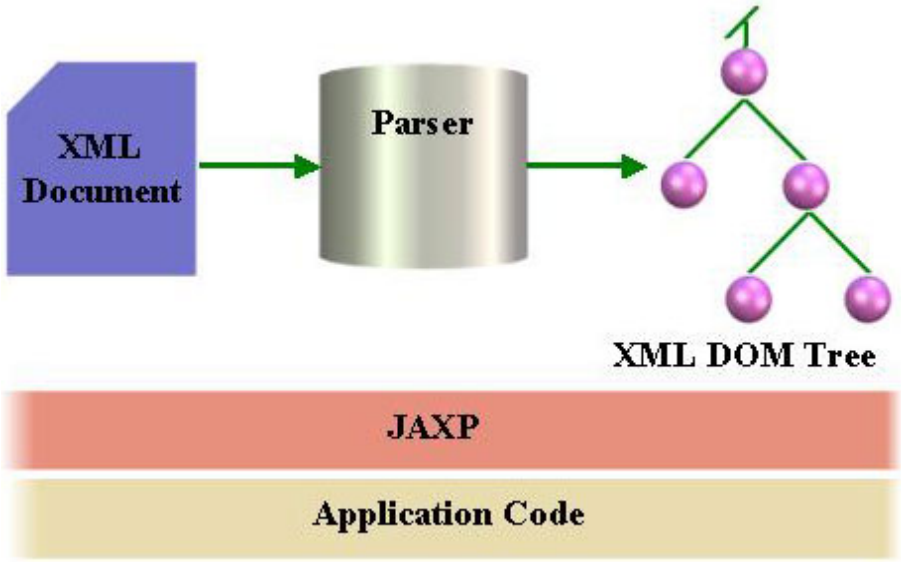

Figura 4.7

Capitulo 5

DISEÑO Y DESARROLLO DE LA INTERFAZ GRÁFICA DE USUARIO

En este capítulo vamos a tratar los distintos elementos que hemos utilizado para elaborar nuestra interfaz gráfica de la aplicación y como interactúa con el usuario.

Primero de todo cabe destacar que la aplicación está implementada en lenguaje de programación JAVA, con la herramienta de desarrollo Netbeans IDE 7.0.

JAVA es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsytems a principios de los años 90. Sus principales características son que usa la metodología de la programación orientada a objetos, permite la ejecución de un mismo programa en múltiples sistemas operativos, incluye por defecto soporte para trabajo en red, está diseñado para ejecutar código en sistemas remotos de forma segura, y es fácil de usar y toma lo mejor de otros lenguajes orientados a objetos como puede ser C++. El entorno de desarrollo que hemos elegido para trabajar con JAVA es Netbeans IDE 7.0, ya que está hecho principalmente para JAVA. Existe un número importante de módulos para extender Netbeans y además es un producto libre y gratuito sin restricciones de uso.

La interfaz gráfica de usuario es la forma mediante la cual se le presenta la información al usuario. Nuestra aplicación se va a basar en una interfaz gráfica de ventanas, que van a contener toda la información y van a permitir la entrada de datos para los procesos que queramos ejecutar. Vamos a disponer de una ventana principal, donde vamos a poder encontrar los diferentes procesos que podemos ejecutar, y a partir de la ventana principal vamos a poder ejecutar procesos en diferentes subventanas. Dentro de una ventana podemos disponer de distintos objetos que nos van a proporcionar información o la capacidad de ejecutar determinadas acciones.

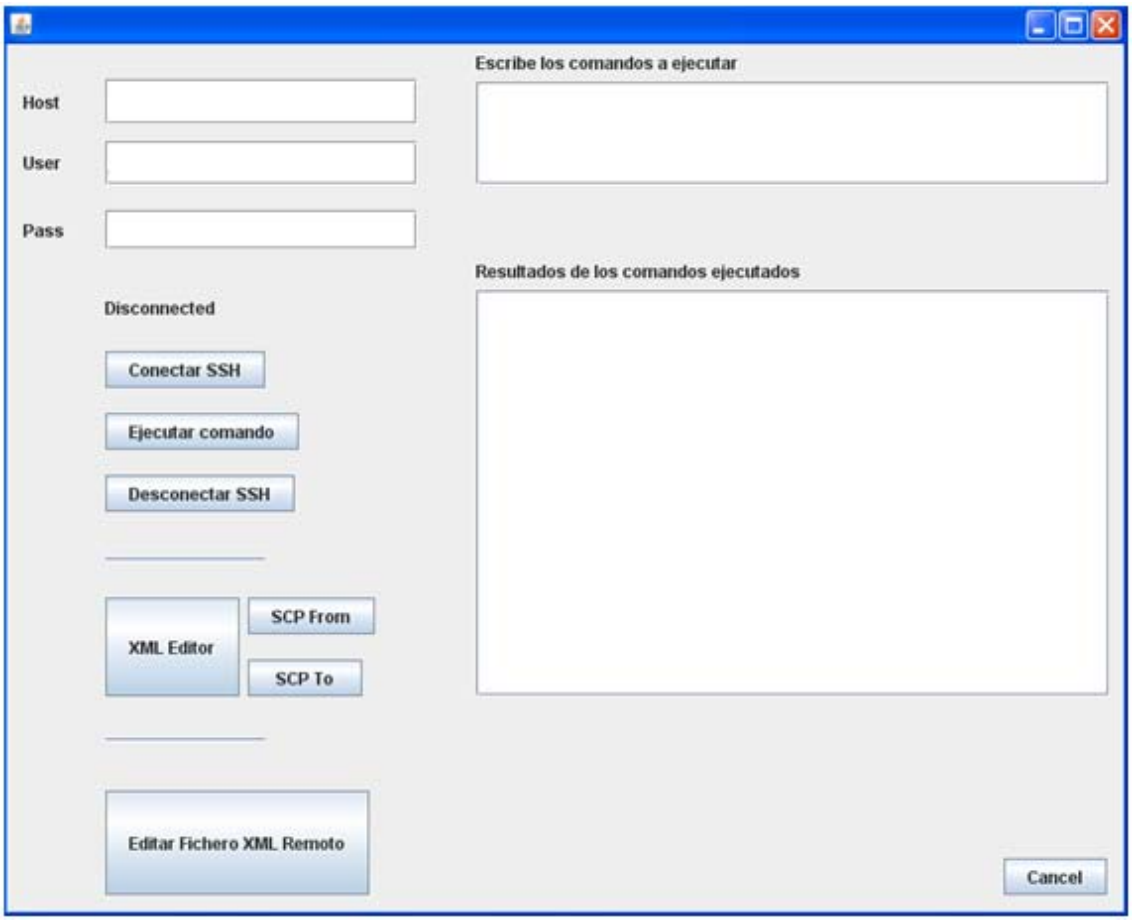

Figura 5.1

Podemos disponer de objetos informadores, como pueden ser cuadros de texto o mensajes de información, que informan al usuario de cuál es el estado de la aplicación en un momento determinado o de si la información está siendo tratada adecuadamente. También disponemos de mensajes de errores, que nos informan de que debido a algún error no se puede continuar con la acción que queríamos realizar y que deberemos volver a intentarlo.

También disponemos de objetos accionadores, como pueden ser los botones, que se encargan de iniciar ciertas acciones que el usuario quiere realizar. Este es el caso de determinadas acciones que queremos llevar a cabo, como puede ser el de conectar por SSH a una maquina remota, en el que deberemos presionar un accionador determinador y esperar una respuesta.

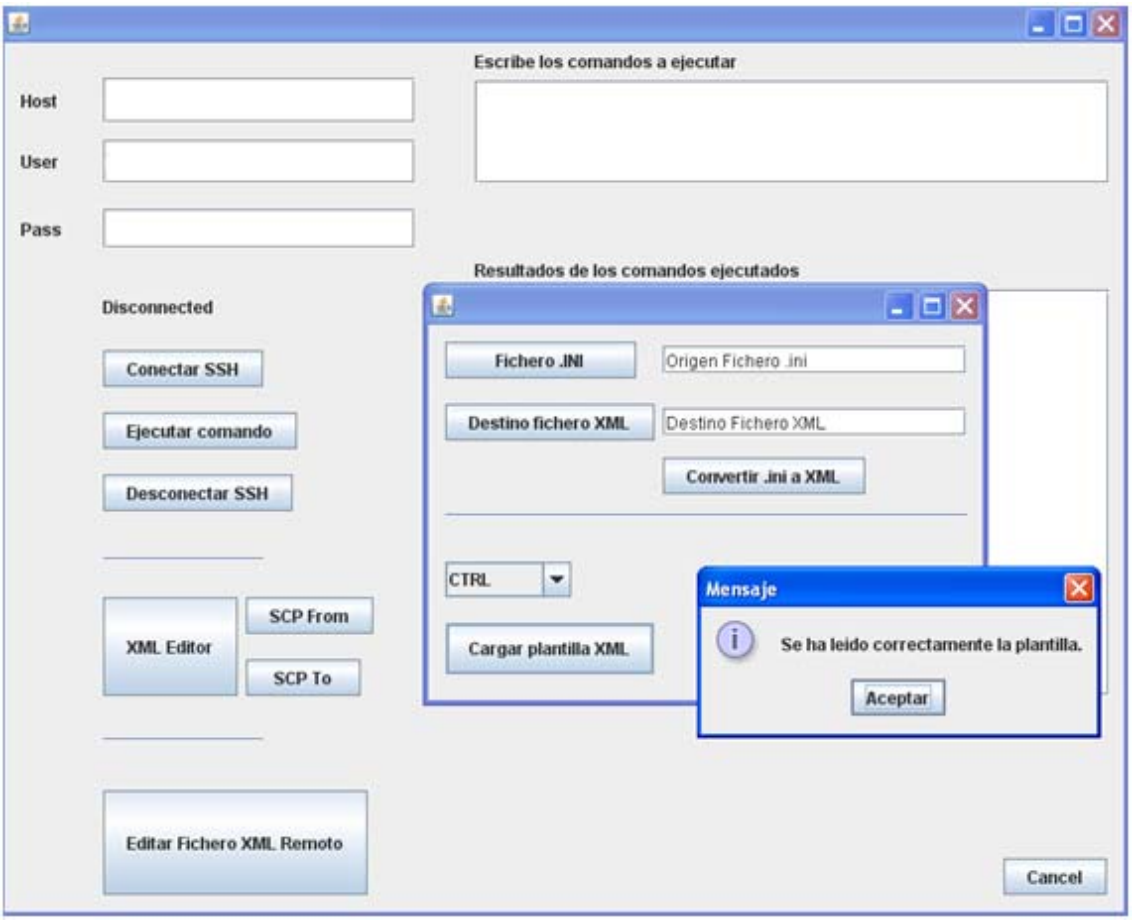

Figura 5.2

Como hemos dicho vamos a disponer de una ventana principal que se va a ejecutar al iniciar nuestra aplicación. En ella vamos a disponer de tres regiones, que se van a encargar de distintas acciones.

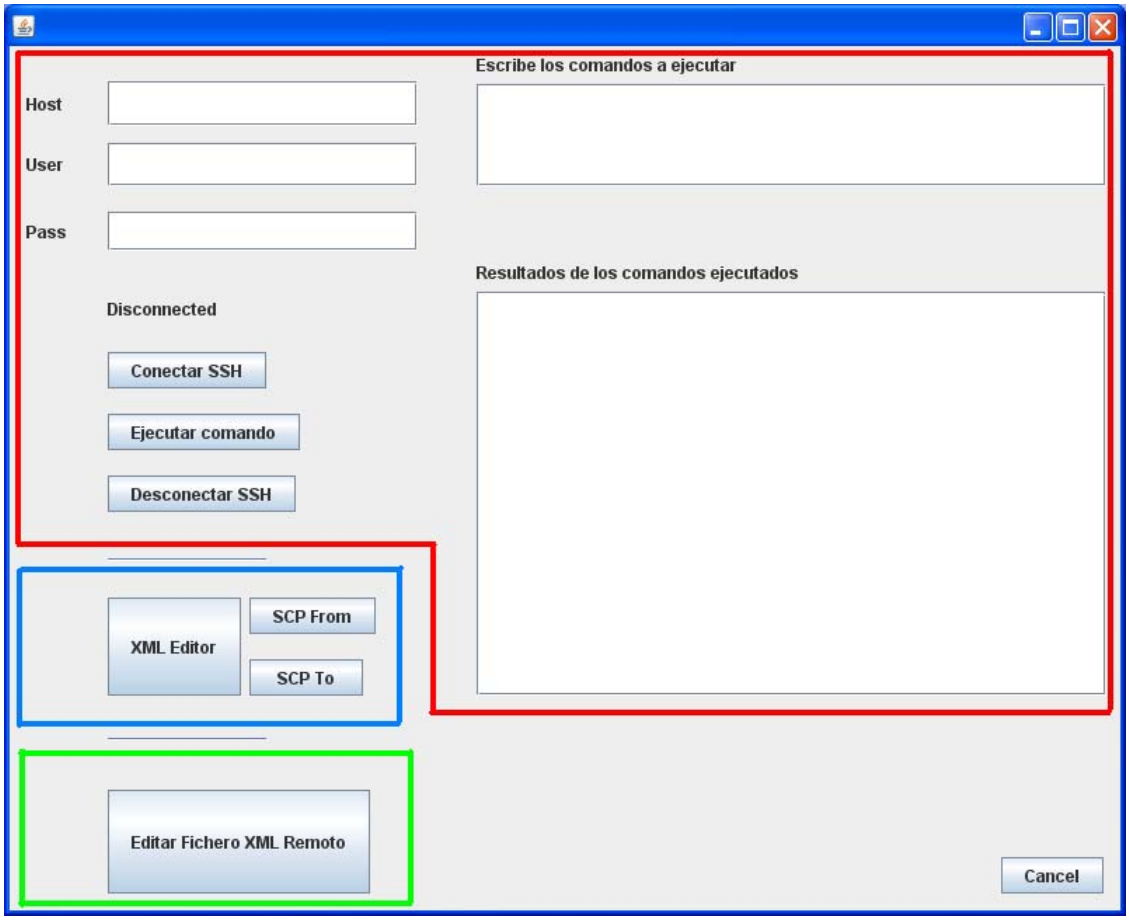

Figura 5.3

Como se puede ver en la figura 5.3, disponemos de la región superior, que se va a encargar de realizar la conexión SSH con el equipo remoto y ejecutar comandos remotos sobre él. Para ello disponemos de los campos de texto necesarios para el host al que queremos conectarnos, y el usuario y contraseña con el que queremos autenticarnos. En la parte derecha disponemos de un recuadro donde escribiremos los comandos que queramos ejecutar, como si fuera una línea de comandos de una consola, y conseguiremos los resultados correspondientes en el recuadro de la parte inferior.

La región intermedia se encarga de la transferencia de archivos entre los equipos remotos, mediante los botones “SCP From” y “SCP To” dependiendo de la dirección en que queramos enviar los archivos. El botón “XML Editor”, nos va a llevar a una subventana donde podremos tanto convertir archivos de configuración .INI en ficheros XML, como editar estos archivos XML mediante la ayuda de una plantilla.

Y por último, disponemos de la región inferior que contiene un único botón que se va a encargar de la edición de ficheros XML remotos. Esta región dispone de un solo botón

grande ya que va a ser la más utilizada por el usuario que nos va a llevar una subventana donde podremos elegir el archivo remoto que queremos editar y poder enviarlo de nuevo.

Esta ventana principal contiene las funciones principales de nuestra aplicación, y utilizando sus botones podremos acceder a ejecutar las acciones correspondientes. De esta forma tenemos en una sola ventana todas las funciones permitidas y según la que queramos ejecutar pues accederemos a la subventana correspondiente.

A la hora de poder editar un fichero XML o traducir un determinado fichero .INI, deberemos buscar entre nuestros archivos cuál es el seleccionado. Para ello necesitamos proporcionar la ruta completa donde se aloja el archivo. Insertar esta información en cuadro de texto puede ser algo complicado, ya que cualquier error puede conllevar un fallo en el que no se encuentre el archivo en la ruta definido. Para evitar este tipo de errores, a la hora de elegir un archivo proporcionamos una ventana con la que vamos a poder explorar nuestro sistema de archivo y elegir el adecuado. Con ello conseguimos la ruta completa del archivo seleccionado, evitando errores y pudiendo consultar cuales son los archivos disponibles.

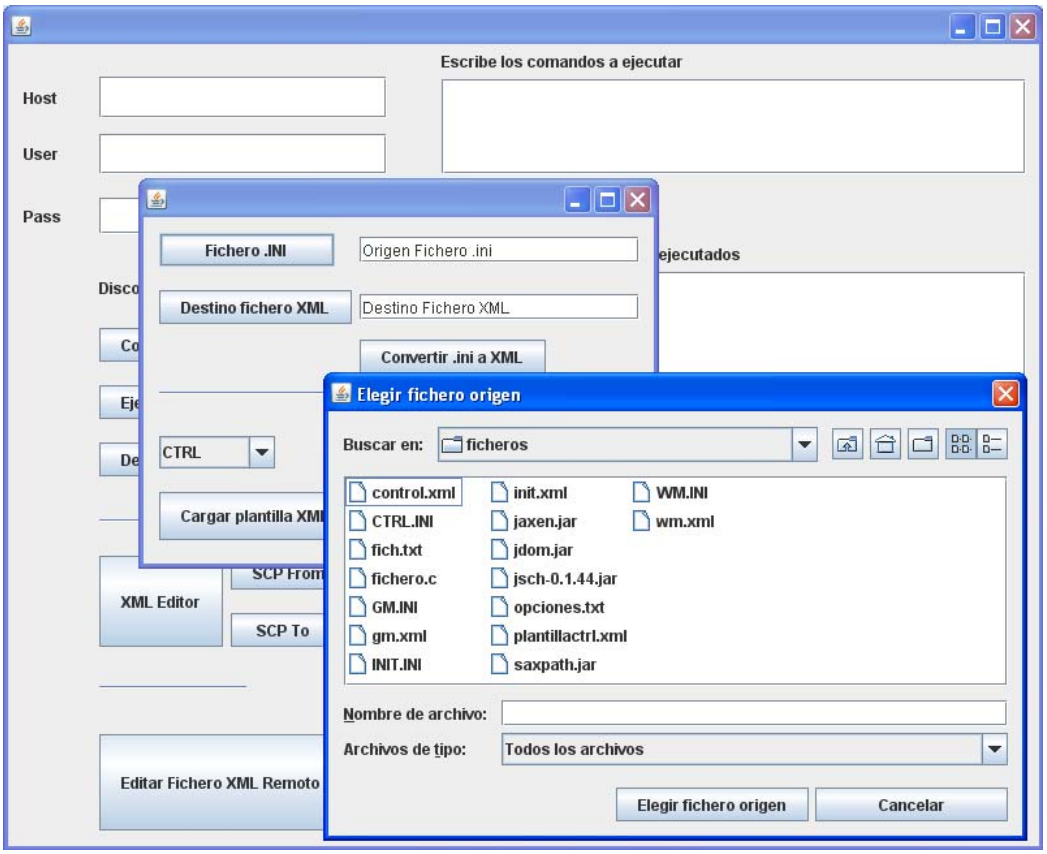

Figura 5.4

Capitulo 6

CONCLUSIONES

El objetivo planteado de este proyecto final de carrera era dotar de flexibilidad al sistema empotrado del robot Humanoide Nao a la hora de configurar parámetros de control y ejecución de los procesos en ejecución.

Para cumplirlo se ha trabajo en el desarrollo del sistema de gestión de ficheros de configuración para uso durante la competición.

Podemos decir que la aplicación final ha cumplido el objetivo marcado. Basándonos en los objetivos particulares:

- Se ha diseñado y desarrollado una interfaz gráfica multiplataforma capaz de permitir al usuario navegar entre sus diferentes ventanas y de manera clara y sencilla ejecutar las acciones necesarias.
- Se ha desarrollado un sistema de comunicaciones seguro, gracias al protocolo SSH que se ha utilizado, permitiendo la conexión segura a un equipo remoto.
- Se han desarrollado unos módulos de interpretación XML, capaces de obtener la información contenida en estos ficheros, pudiendo modificarlos y guardarlos para su uso como ficheros de configuración para los robots.
- Se ha conseguido validar y verificar la información de estos ficheros de configuración, permitiendo el uso fiable de estos ficheros y evitando comportamientos no deseados por parte del robot durante la competición.

La aplicación ha sido testeada y aplicada a los robots Nao de competición, obteniendo unos resultados satisfactorios, y que cumplen los objetivos marcados al comienzo de este proyecto.

Como futuras mejoras posibles a realizar sobre este proyecto en incluir dentro del sistema desarrollado de gestión de ficheros aquellos relacionados con los procesos de visión artificial, en concreto las tablas de calibración de colores de los objetos de interes (Look Up Tables). Estos ficheros son generados mediante otra herramienta en un proceso de calibración y guardados en archivos de texto de gran tamaño.

Capitulo 7

ANEXO

6.1 Manual de usuario

A continuación vamos a describir el funcionamiento de la aplicación, para que un usuario sin formación sobre la aplicación, pueda hacer uso de ella y de sus funcionalidades.

La aplicación será distribuida en una carpeta donde se incluirán tanto el ejecutable .JAR de la aplicación JAVA, como una carpeta lib que deberá acompañar en todo momento al ejecutable, con las librerías necesarias para poder ejecutar la aplicación. Todos los archivos que guardemos, sin especificar una ruta determinada, quedaran almacenados en esta carpeta junto al ejecutable. Además las plantillas de los ficheros XML deben estar también incluidas en esta carpeta junto al ejecutable para que puedan ser leídas. Las plantillas deberán ser de formato XML y contener el nombre plantillaXXX.xml, donde las XXX son el nombre del archivo al que hace referencia la plantilla.

Cualquier usuario puede modificar las plantillas de los ficheros abriendo el fichero mediante un editor de textos, siempre respetando el cuerpo y los campos que contiene el fichero de plantilla. Se pueden añadir nuevas variables, y añadir más opciones en el campo “enum_value” simplemente añadiendo a la lista, los valores que queramos que luego se muestren en el desplegable para elegir.

En la Figura 6.1.1, podemos ver un ejemplo donde tenemos la aplicación .JAR, la “plantillactrl.xml” y el archivo “CTRL.xml”.

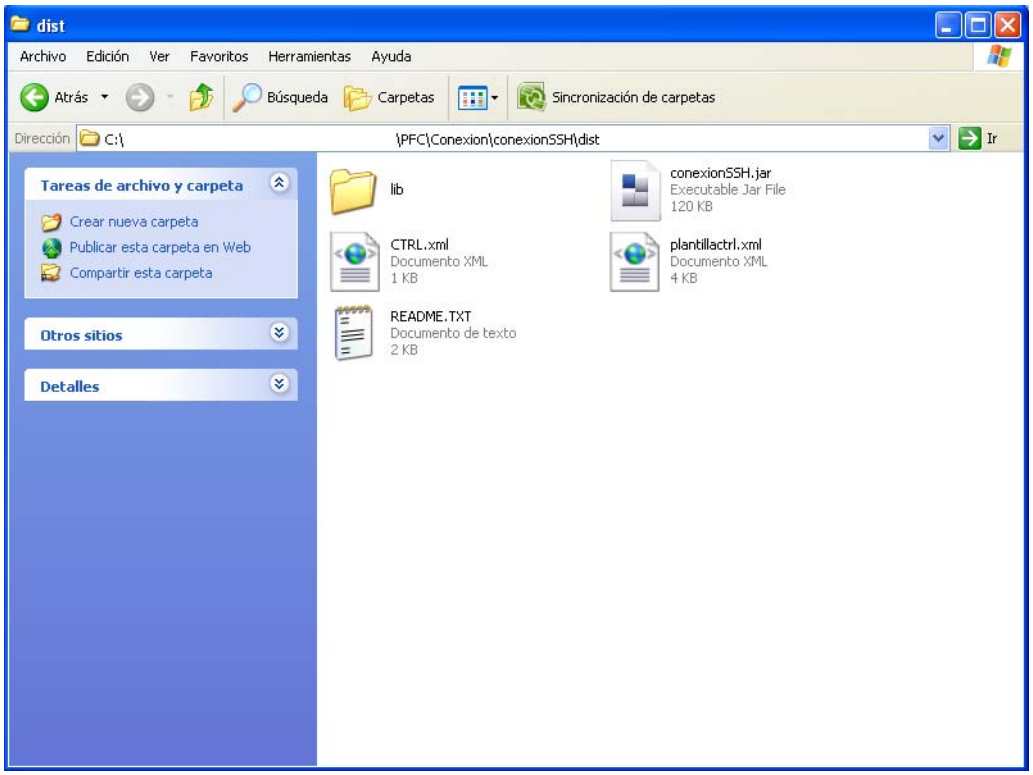

Figura 6.1.1

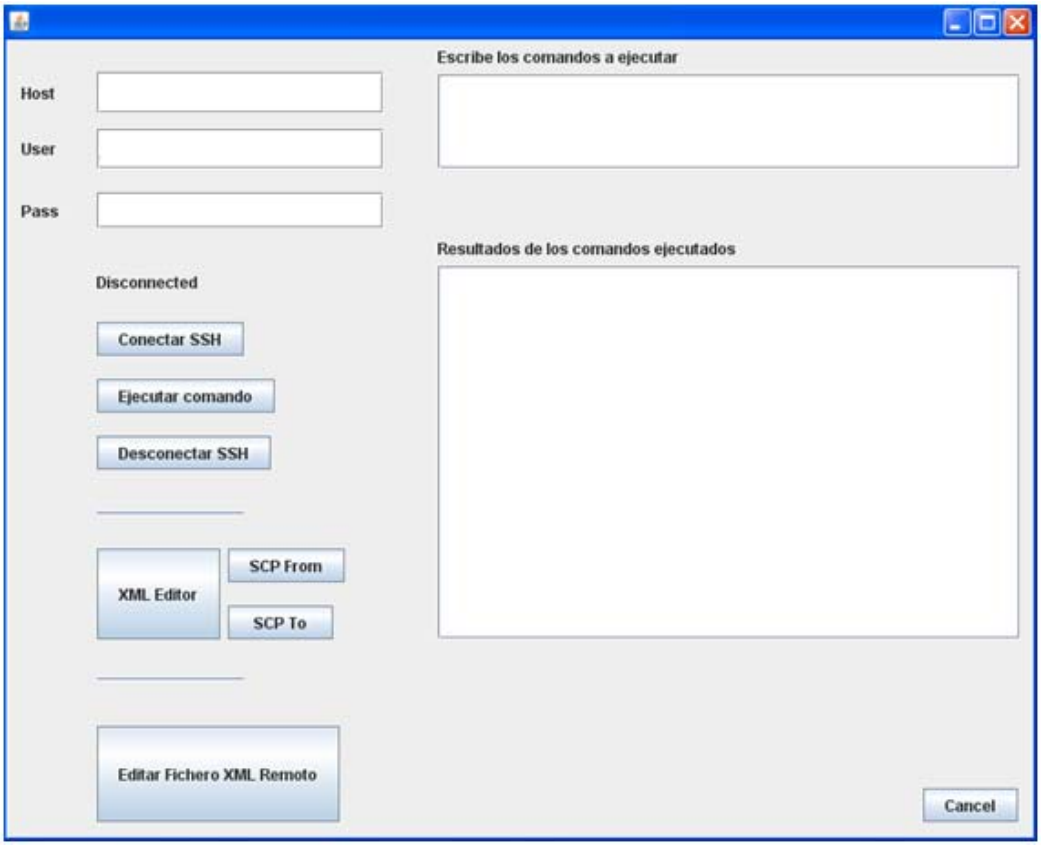

Figura 6.1.2

La pantalla principal de nuestra aplicación es la que podemos ver en la Figura 6.1.2. Como podemos observar está separado en tres regiones, que se van a encargar de las diferentes funcionalidades del sistema.

Primero de todo podemos observar los cuadros de texto de la parte superior. Su utilidad va a ser la de simular una terminal de consola sobre la que vamos a poder ejecutar comandos. Para ello lo primero que debemos hacer es elegir el host al que queremos conectarnos, y el usuario y contraseña con el que vamos a autenticarnos. Una vez hecho esto, pulsaremos sobre el botón “Conectar SSH” para conseguir establecer la conexión. Si la conexión ha tenido éxito, el texto “Disconnected” pasará a “Connected” para saber que estamos conectados y podemos comenzar a ejecutar comandos. A partir de ahora podremos comenzar a escribir comandos sobre el cuadro de texto de la parte superior derecha, y apretando el botón “Ejecutar comando”, recibiremos la respuesta por el cuadro de resultados de comandos. Podremos ejecutar todos los comandos que necesitemos sobre el cuadro de texto, siempre pulsando sobre el botón de ejecutar para que este tenga efecto. Disponemos del botón “Desconectar SSH” para terminar la conexión con el host, y poder volver a conectarse a otro host diferente o autenticarse con otro usuario distinto. Podremos ver cuando nos hemos desconectado puesto que volveremos a ver el texto “Disconnected” en la pantalla.

Ahora nos vamos a centrar sobre la región central, donde se ubican los tres botones. En ellos podemos ver el “XML Editor”, “SCP From” y “SCP To”. Pasemos a describir uno por uno su apariencia y funcionalidades.

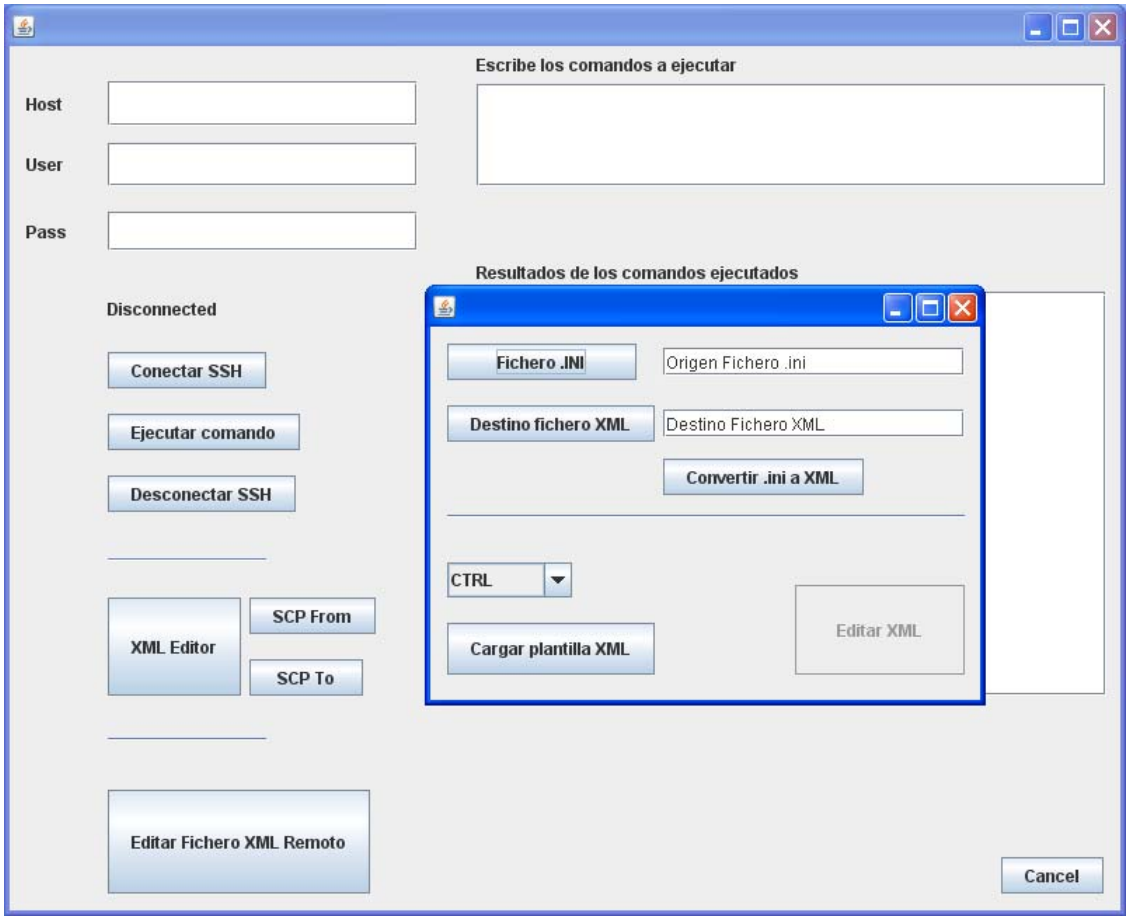

Figura 6.1.3

Si pulsamos sobre el botón “XML Editor” nos aparecerá la siguiente ventana de la Figura 6.1.3. En esta ventana tenemos dos funcionalidades, la primera es la de traducir un archivo .INI en archivo XML. Para ello elegiremos en nuestro PC local, el archivo .INI que queremos traducir, para ello tendremos la ayuda de una ventana que nos guiará por nuestro gestor de archivos para elegir el adecuado. Una vez elegido el origen, debemos elegir donde queremos ubicar nuestro archivo XML final. Para ello volvemos a disponer de una ventana que nos ayudará a elegir la carpeta donde queremos guardar nuestro archivo XML. Es importante guardar nuestro archivo con extensión .xml, para no dejar el archivo sin formato.

Una vez elegido el origen y destino de nuestros ficheros, simplemente deberemos pulsar el botón “Convertir .INI a XML”. Si alguno de los ficheros no tiene el formato correspondiente, nos saldrá una advertencia avisándonos sobre ello, y si todo sale correctamente no saldrá una ventana confirmando la acción y la carpeta donde se aloja el fichero destino.

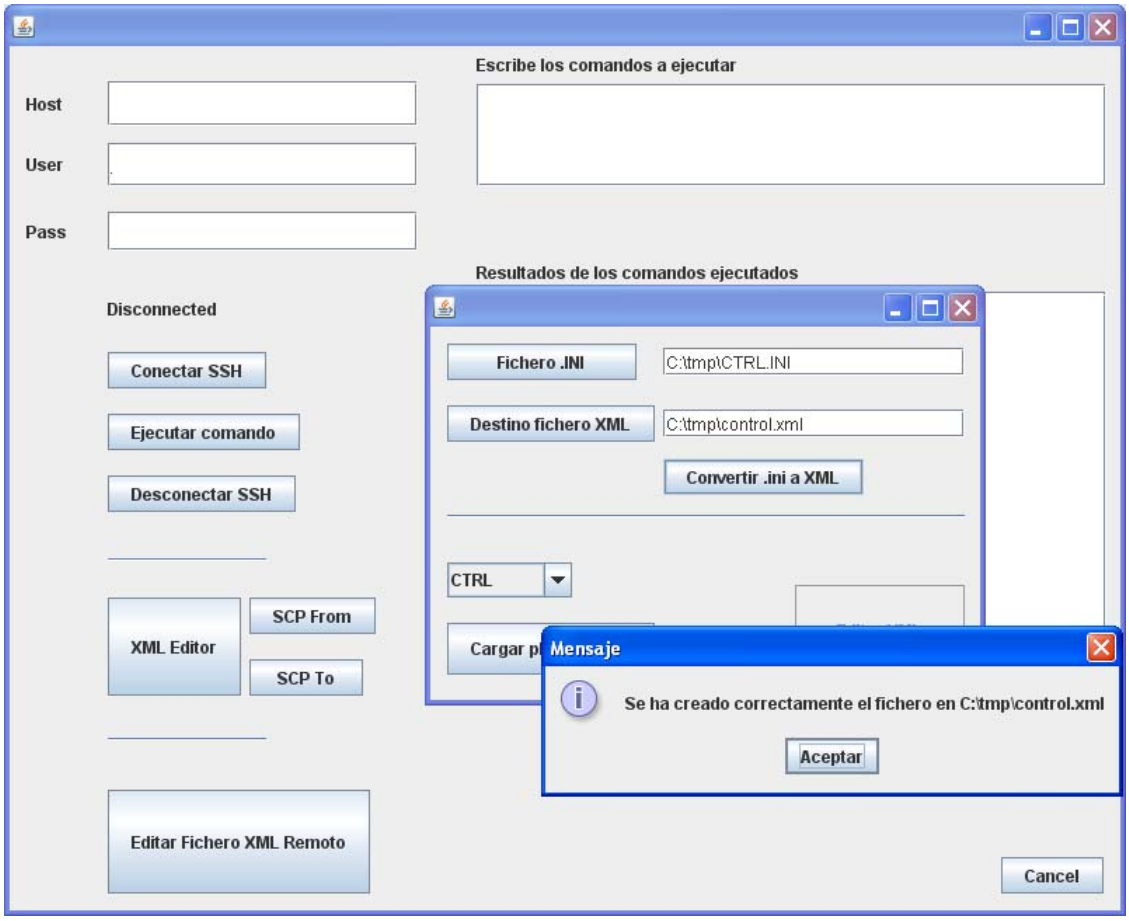

Figura 6.1.4

En la Figura 6.1.4 podemos ver como cargamos un el fichero “CTRL.INI” de la carpeta “C:\tmp\” y alojamos el fichero XML en la misma carpeta con el nombre “control.xml”. Como todo ha ido correctamente nos sale el mensaje de confirmación.

Pasamos a la segunda funcionalidad que disponemos en la Figura 6.1.2, que es la de editar un fichero XML para poder darle los valores que nosotros deseemos a las variables del fichero. En este caso, como podemos ver en la Figura 6.1.2 disponemos de un desplegable donde podremos elegir el tipo de fichero que queremos editar. En este caso vamos a editar un fichero “CTRL.xml”. Elegimos “CTRL” en el despegable y pulsamos el botón “Cargar plantilla XML”. Nos aparecerá un mensaje de confirmación diciendo que hemos encontrado la plantilla y se ha podido cargar correctamente. Con esto conseguimos que se nos muestre una nueva pantalla en la que ya le podremos dar valores a las variables que contiene el fichero “CTRL.xml”, como la que podemos en la Figura 6.1.5.

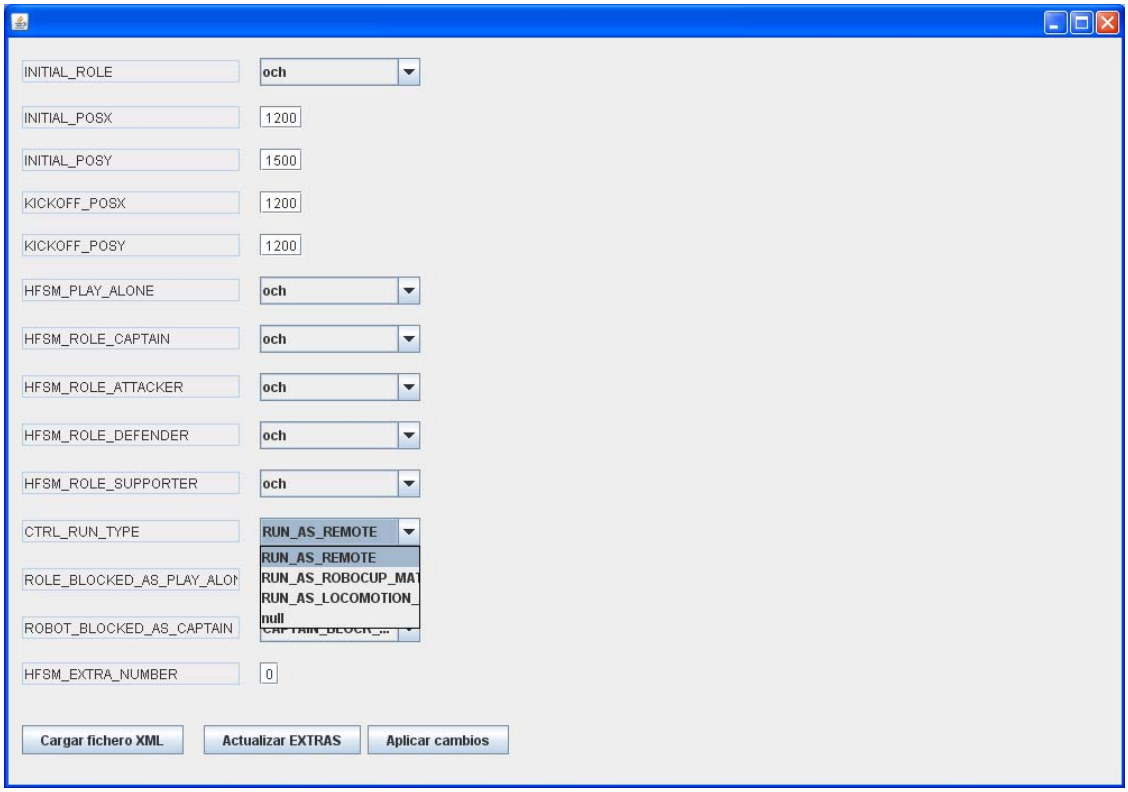

Figura 6.1.5

En ella podemos observar las variables editables en el fichero “CTRL.xml”, y en estos momentos ya podremos modificar todas las variables que nosotros necesitemos. Como podemos ver existen desplegables en algunas variables, ya que sus posibles valores están limitados y por tanto debemos elegir de la lista que se nos ofrece. Para los valores que están más abiertos, disponemos de unos cuadros de texto donde deberemos introducir el valor deseado, este valor luego se comprobará que esté dentro de los rangos permitidos para según cada variable.

Cabe destacar que si una variable queremos no darle un valor determinado, deberemos elegir el valor “null”, para que luego al leer el robot el fichero final, sepa que esta variable no va a contener ningún valor, y por tanto evitar comportamientos no deseados.

Disponemos del botón “Cargar fichero XML” situado en la parte inferior izquierda, que nos va a servir para cargar los valores de las variables desde un fichero XML que ya tengamos en nuestro PC local. Si pulsamos este botón dispondremos de una ventana donde podremos elegir un fichero XML dentro de nuestro gestor de archivos y que al cargar dicho fichero, todas las variables se van a cargar con los valores que contenga el fichero XML. Esto nos puede ser útil si queremos modificar un fichero que ya

tenemos creado, y que simplemente queremos cambiar un valor determinado, sin tener que modificar el resto.

Como podemos ver existe un botón de “Actualizar EXTRAS”, esto ocurre porque en determinados ficheros como puede ser el “CTRL” existe una variable que puede contener variables EXTRAS, según el valor que tome la variable “HFSM_EXTRA_NUMBER”. Es por ello que hemos añadido esta funcionalidad. Si queremos añadir EXTRAS a nuestro fichero deberemos indicar el número en el cuadro de texto de la variable anterior, y pulsar en “Actualizar EXTRAS”, de esta forma nos aparecerán tantos EXTRAS como hayamos indicado, y podremos de esta forma darle valor a estas nuevas variables.

Por último disponemos del botón “Aplicar cambios”, este botón nos sirve para guardar los cambios que hemos hecho en las variables mediante el editor, y guardarlo en un fichero XML nuevo. Para ello, si pulsamos en el botón de aplicar cambios nos aparecerá una ventana para examinar nuestro gestor de archivos y guardar en nuestro PC local el nuevo fichero XML en la carpeta que nosotros elijamos. Cabe destacar que deberemos darle extensión .xml al archivo para poder guardarlo correctamente, sino hacemos esto nos aparecerá un mensaje avisándonos que el fichero que queremos guardar no es XML.

En la Figura 6.1.6 podemos ver un ejemplo de un fichero que hemos modificado, le hemos añadido 7 variables EXTRAS y hemos modificado los valores de algunas variables, y que nos disponemos a guardar los resultados en la carpeta ficheros con el nombre “ejemplo.xml”.

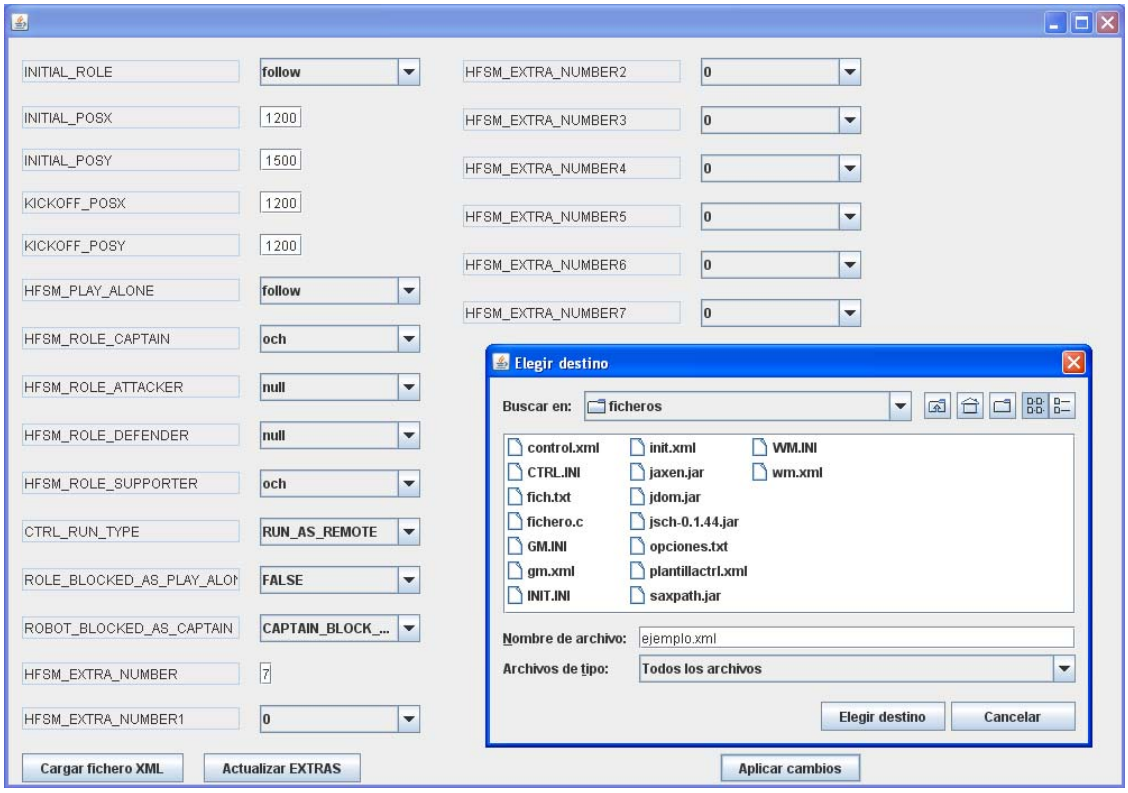

Figura 6.1.6

Como hemos dicho en la parte central de nuestra aplicación existen 2 botones, “SCP From” y “SCP To”. Estos son los encargados de transferir archivos entre el PC local y el PC remoto al que queremos conectarnos, en nuestro caso un robot. Si pulsamos sobre el botón “SCP From” se nos abrirá una ventana como la de la Figura 6.1.7.

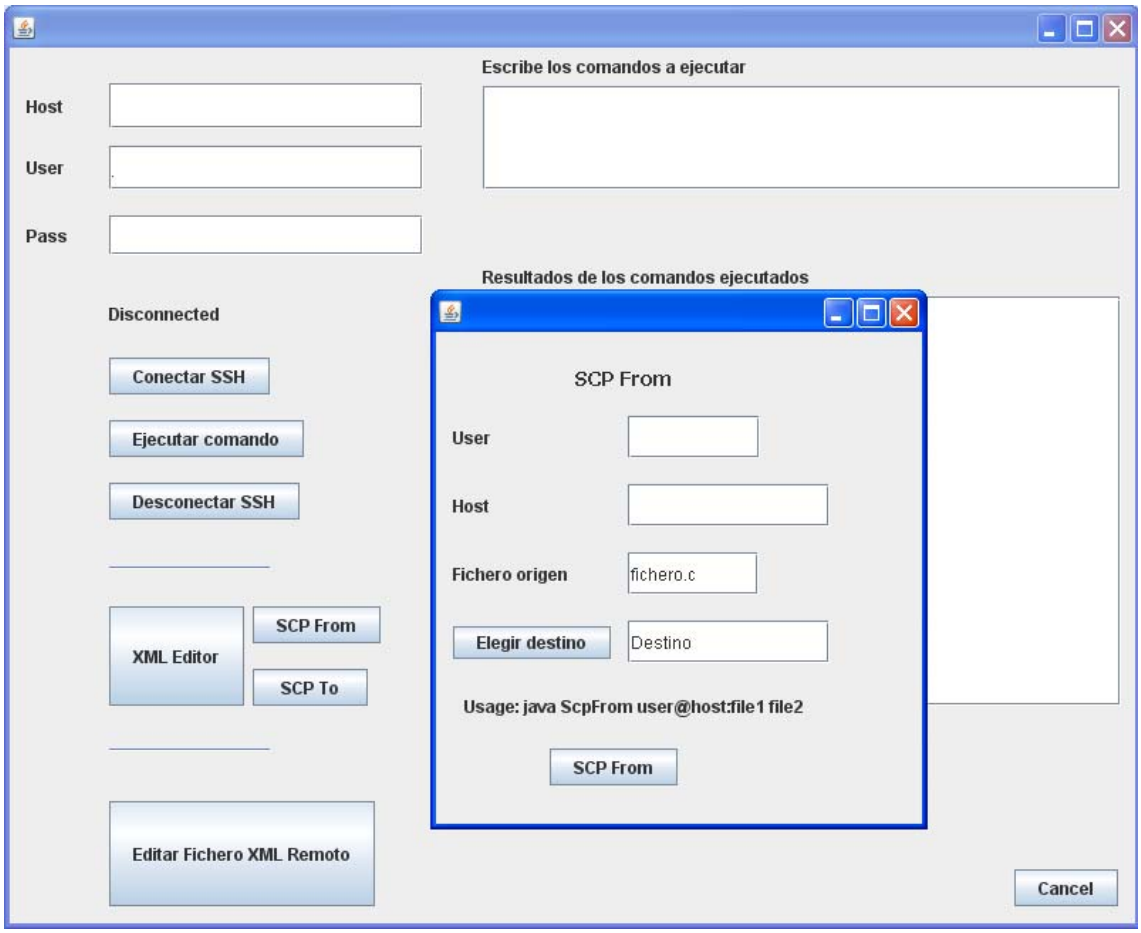

Figura 6.1.7

En ella podemos observar los campos que deberemos rellenar para en este caso, traer un fichero del equipo remoto a nuestro PC local. Para ellos deberemos indicar el host al que queremos conectarnos y el usuario con el que nos queremos autenticar. Luego deberemos indicar el fichero origen de la maquina remota que queremos traer a nuestro PC, para ello deberemos indicar la ruta completa de donde se encuentra el fichero origen. Para guardar el archivo en nuestro PC local le deberemos indicar un directorio donde guardarlo, para ello nos vamos a ayudar de la ventana elegir destino, donde vamos a examinar y elegir el directorio mediante una ventana de gestor de directorios. Nos será mucho más fácil de esta forma elegir el lugar donde queremos guardar el fichero origen.

Cuando tengamos todo ello, simplemente debemos pulsar el botón “SCP From”, que nos pedirá la contraseña correspondiente al usuario para autenticarnos y de esta manera poder realizar la transferencia del archivo. Si existiera cualquier error con la contraseña o con la transmisión del fichero, nos indicará un mensaje de error, mientras que recibiremos un mensaje si todo ha ido correctamente.

La otra opción “SCP To” es análoga a la que acabamos de explicar, simplemente deberemos de elegir en nuestro PC local el archivo origen que queremos enviar, e indicar la ruta del equipo remoto donde queremos guardar el archivo.

Por último vamos a disponer de la última región de nuestra aplicación en la que solo se muestra un botón, “Editar Fichero XML Remoto”. Este botón se encarga de editar un fichero que está alojado en el equipo remoto, y volver a enviar el que hemos modificado de nuevo. Es una función que aglutina todas las funcionalidades anteriores, pudiendo hacerlo todo de un solo paso. La ventana que nos aparece es la que vemos en la Figura 6.1.8.

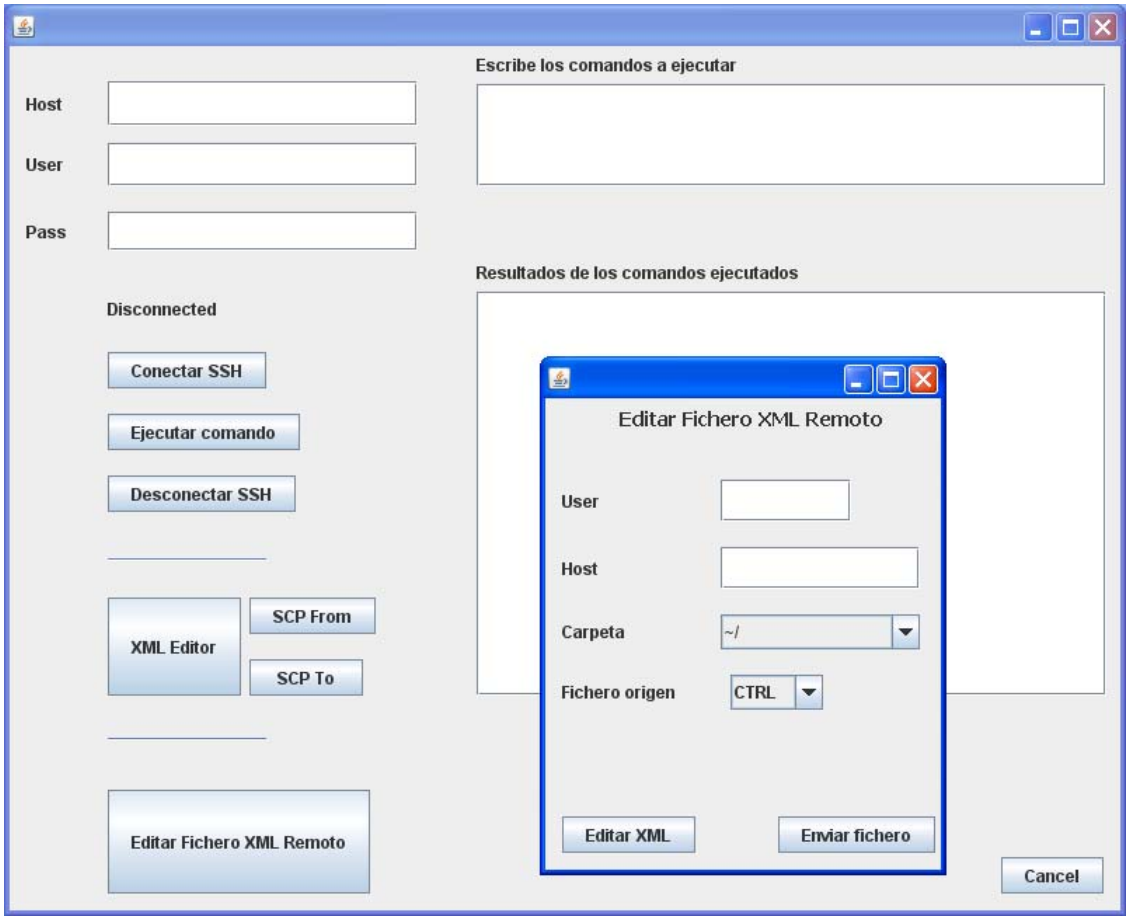

Figura 6.1.8

Podemos ver que existen los campos host y user, donde deberemos introducir el host al que queremos conectarnos y el usuario con el que queremos autenticarnos. Ahora debemos elegir la carpeta donde se aloja el fichero que queremos modificar, para ello disponemos de un desplegable con las rutas donde suelen estar alojados los ficheros de configuración, si bien también es editable por si queremos elegir una ruta diferente.

En fichero origen debemos elegir, de la lista dada, que son los archivos de configuración que se suelen modificar, cuál es el fichero que queremos modificar. Una vez hecho todo esto, pulsaremos el botón “Editar XML”, y tras autenticarnos con nuestra contraseña de usuario, pasaremos a la ventana de edición y modificación de los ficheros XML, como la de la Figura 6.1.5. Hacemos las modificaciones pertinentes, y guardamos nuestro archivo modificado. En este caso se nos guardará en la carpeta donde están todos los archivos de la aplicación.

En este punto es donde volveremos a la pantalla de la Figura 6.1.8, y tendremos la opción de enviar archivo. Ahora podremos enviar el archivo modificado de nuevo al robot, o una de las funcionalidades de la aplicación es que podremos modificar el host, de forma que este archivo que hemos modificado, lo podremos mandar no solo al robot origen, sino también a otros robots que queremos que tengan el mismo comportamiento.

6.2 Manual de desarrollador

En este apartado vamos a describir cual es la estructura que siguen los ficheros dentro del proyecto, y las funciones que hemos utilizado para programar la aplicación. El objetivo de este apartado es ayudar a que cualquier usuario programador sea capaz de modificar la aplicación.

En la Figura 6.2.1 podemos observar la estructura de nuestro proyecto con los diferentes ficheros JAVA que lo componen. Podemos ver diferenciados los archivos JAVA que forman parte del proyecto, y en la parte inferior las librerías que hemos incluido para poder desarrollar ciertas funciones.

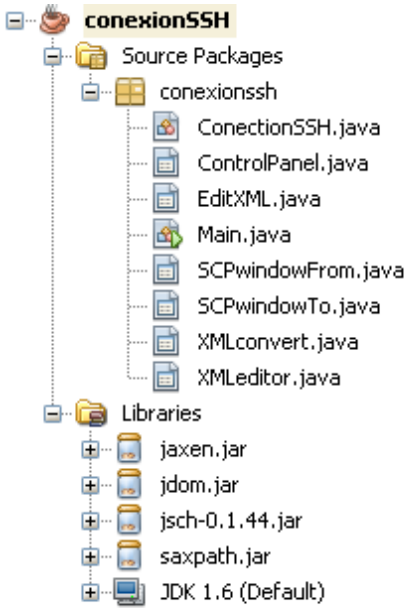

Figura 6.2.1

Como podemos ver se incluye una clase “Main” que va a ser el código que se va a ejecutar al iniciar la aplicación. A partir de este podemos ver el resto de clases que van a depender unas de otras dependiendo de las llamadas que se vayan ejecutando.

Vamos a ir describiendo una por una todas las clases de las que forma parte el proyecto y explicando cual es el funcionamiento y objetivo de cada una.

La clase “Control Panel” es la que da forma a la ventana principal de nuestro proyecto. En esta clase principalmente mostramos por pantalla los botones y cuadros de texto necesarios para ejecutar el resto de funcionalidades. En la Figura 6.2.2 podemos

observar las funciones que forman parte de la clase “Control Panel”. Todas ellas como hemos comentado son funciones que llevan asociadas una acción tras pulsar el botón correspondiente. Además de eso contiene la función “initComponents” que es donde por defecto se van a inicializar todas las variables.

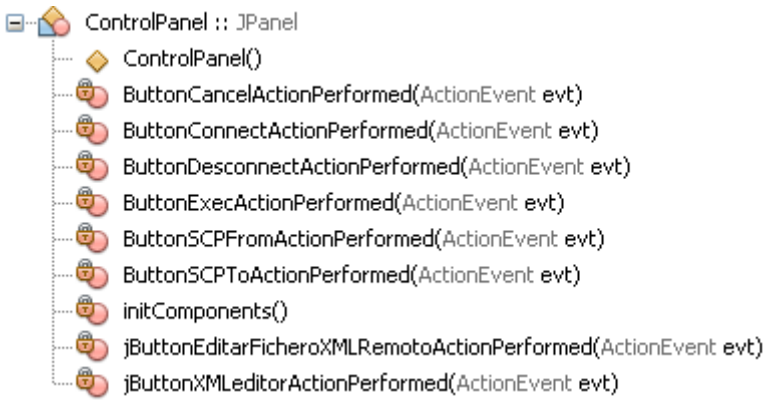

Figura 6.2.2

La siguiente clase que vamos a describir es “ConexionSSH”, esta clase se encarga de realizar la conexión con el equipo remoto y de ejecutar los comandos. Como podemos ver en la Figura 6.2.3 solamente tiene tres funciones muy simples. En este caso no disponemos de botones ni ventana que mostrar por lo que las funciones aquí detalladas serán llamadas desde el la ventana principal, pulsando los botones correspondientes.

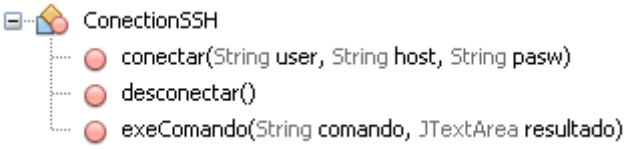

Figura 6.2.3

La clase “XMLconvert”, que podemos ver en la Figura 6.2.4, es la encargada de traducir los ficheros .INI en ficheros XML, y de cargar una plantilla XML para poder editar y modificar sus campos. Tenemos dos funciones donde vamos a seleccionar tanto el fichero origen .INI que queremos traducir como el destino donde queremos ubicar el fichero XML. Luego tenemos la función “convert” que es la que va a ejecutar la función para traducir el fichero .INI. Ya por otra parte tenemos la función de “cargar plantilla” y la de “editar XML” donde pasaremos a la nueva ventana donde vamos a poder modificar los valores de los atributos. Por último disponemos de la función “initComponents” donde por defecto se van a inicializar todas las variables.

Figura 6.2.4

Las clases “SCPwindowFrom” y “SCPwindowTo” son básicamente iguales. La necesidad de dos clases diferentes se basa en que son dos instrucciones diferentes y por tanto necesitan dos ventanas diferentes con distintos campos. Las funciones como podemos ver en la Figura 6.2.5 son idénticas, intercambiando el destino con el origen del fichero que queremos transferir, y la función SCP únicamente cambia la sintaxis de la instrucción.

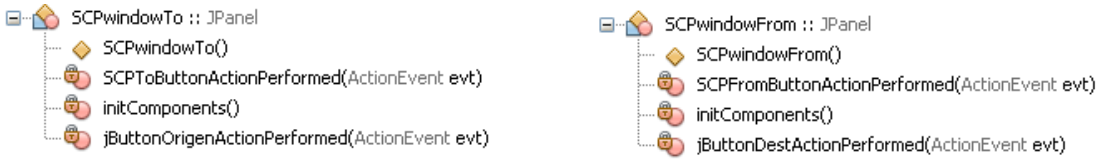

Figura 6.2.5

La clase “EditXML”, en la figura 6.2.6, es la encargada de la modificación de un fichero XML remoto. Para ello hemos hecho uso de otras funciones ya implementadas en el proyecto como son las de “EditarXML” que va a transferir el archivo remoto a nuestro PC local y vamos a modificarlo, y la función “EnviarFichero” que va a volver a transferir el archivo ya modificado al equipo remoto de nuevo. Estas dos funciones, como hemos dicho, están formadas por funciones ya implementadas y ya explicada anteriormente. Como en todas las clases anteriores, disponemos de “initComponents” que va a inicializar las variables.

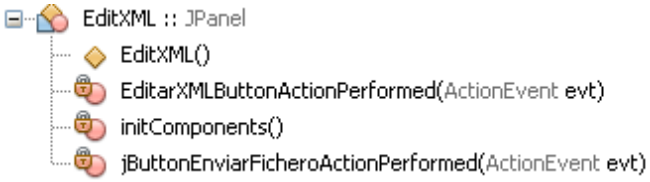

Figura 6.2.6

Por último nos queda comentar la clase “XMLeditor”, que quizás sea la que más funciones y variables tenga ya que es la encargada de hacer la función principal de la aplicación, que es la de modificar el archivo XML. Primero de todo comentar que la aplicación dispone de capacidad para mostrar y modificar 30 variables en la ventana del editor de XML. Lo que vamos a hacer es hacer visible o invisible los campos según la cantidad de variables que necesitemos mostrar, de esta forma no nos aparecerán casillas en blanco ni sin formato. Como podemos ver en la Figura 6.2.7, disponemos de 11 funciones en la clase, ahora comentaremos las más importantes.

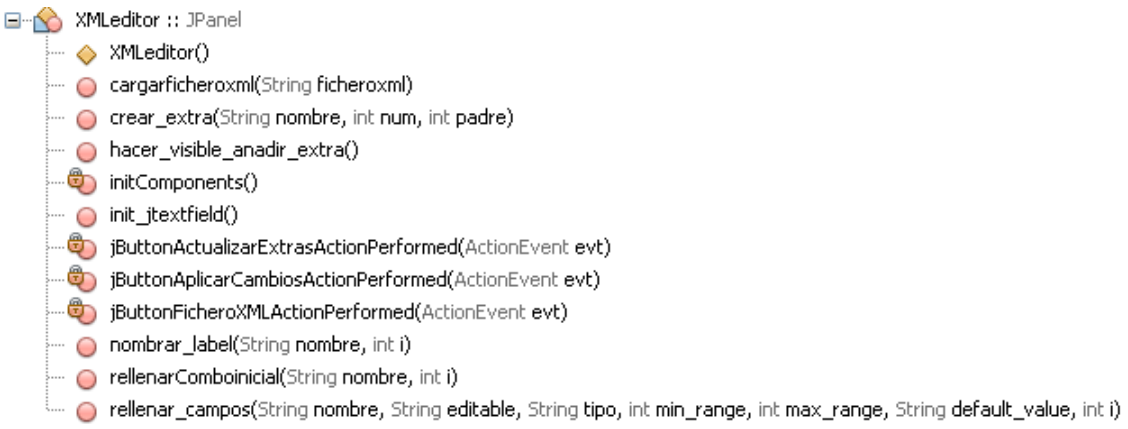

Figura 6.2.7

Comenzaremos con las más sencillas, las funciones que ejecutamos al pulsar un botón. Entre ellas podemos ver la función “AplicarCambios” que se va a encargar de primero de todo comprobar que todas las variables cumplan las restricciones definidas, luego según si el fichero es remoto o no, deberemos elegir la ubicación del fichero destino, y si es remoto se guardará en la carpeta de la aplicación para después poder transferirlo de nuevo al equipo remoto. Luego solo falta escribir el fichero en formato XML y guardarlo en la ubicación seleccionada.

La función “FicheroXML” es la encargada de cargar un fichero XML que ya tenemos definido, para de esta forma cargar todas las variables con los valores correspondientes a este fichero que ya teníamos definido.

Y por último tenemos la función “ActualizarExtras”, que como hemos dicho anteriormente se utiliza para un tipo de variables especiales, y que deben actualizarse según el número de extras indicados en el campo correspondiente. La función “crearextra” como hemos dicho implica hacer visibles las casillas correspondientes y inicializar todos los componentes con los valores que se van a mostrar.

Por otra parte tenemos las funciones de “rellenarComboinicial” que va a introducir todos los datos que se van a mostrar en las listas desplegables para luego poder elegir entre ellas. Esta parte se hace desde la lectura del fichero XML, introduciendo los valores en la estructura de datos del Combo que luego va a ser mostrado. Y la otra función “rellenarcampos” es la que va a introducir en los arrays de las variables todas las restricciones y valores que vamos a leer del fichero XML. Esta función es la que se encarga de dar los valores de los atributos y almacenarlos en los arrays para poder consultarlos más tarde.

Por último comentar las funciones “init_jtextfields” y “initComponents” que son las encargadas de inicializar las variables, y en el caso de los cuadros de texto de inicializar el valor que se va a mostrar por defecto al mostrar la ventana de edición.

Capitulo 8

BIBLIOGRAFÍA

[1] Aldebaran Robotics http://www.aldebaran-robotics.com/

[2] Standard Platform League Robocup www.tzi.de/**spl**/

[3] Robocup http://www.robocup.org/

[4] “Key Java: advances tipes and techniques”, Hunt John, London: Springer, 1998, ISBN 3540762590

[5] Netbeans 7.0 http://netbeans.org/

[6] “SSH, the secure shell”, Barret, Daniel J., Sebastopol: O´Reilly, 2005, ISBN 0596008953

[7] “XML: step by step”, Young, Michael J., Redmond: Microsoft Press, 2002, ISBN 0735614652

[8] “Servicios software en robots humanoides”, Juan José Alcaraz Jiménez, Trabajo fin de Máster 2008

[9] “Locomoción bípeda del robot humanoide Nao”, Samuel Fernández Iglesias, Proyecto Final de Carrera 2009