UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Desarrollo de un sistema de monitorización y control de un robot simulador de diabetes

TRABAJO FINAL DE MÁSTER: MÁSTER EN AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

Antonio Bengochea Carrasco
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DIRECTOR: Juan Francisco Blanes Noguera

Tabla de contenido

Tabla de ilustraciones

1. Introducción y objetivos

1.1. Marco del proyecto

El termino robot ya no resulta desconocido en estos tiempos. La mayoría de la gente lo suele asociar al área de la industria, donde se encuentran los brazos robóticos o grandes maquinarias automatizadas, pero el concepto de robot abarca mucho más, como son los robots humanoides, los cuales llevan mucho tiempo en desarrollo con el fin de conseguir que sean capaces de realizar todo tipo de tareas.

Aunque aún no se ha logrado esa meta, las grandes compañías llevan utilizando este tipo de robots (Honda y su robot ASIMO, Boston Dynamics y el robot ATLAS, etc.), para experimentar y desarrollar nuevos avances tecnológicos, como la interacción con herramientas o entornos con humanos, nuevas técnicas de navegación, robótica educativa, etc.

En estos últimos años, hemos visto un fuerte aumento de los dispositivos conectados a internet con el objetivo de ofrecer más servicios y mejorar los ya existentes. Esta tendencia se la conoce con el término “internet de las cosas”, basado en la filosofía, todo conectado y todo accesible desde cualquier lugar.

La gran mayoría de estos dispositivos están basados en Sistemas empotrados o embebidos, pequeños ordenadores que se encuentran dentro de los dispositivos y realizan todas las gestiones necesarias. Este tipo de sistemas se suelen diseñar para cumplir con necesidades específicas, pero cada vez más se están empezando a utilizar ordenadores de propósito general gobernados por sistemas operativos GNU/Linux, dotando a estos dispositivos de una mayor capacidad para abarcar más necesidades y facilidad a lo hora del diseño.

Estas nuevas tendencias y avances han permitido el desarrollo de robots humanoides basados en sistemas empotrados GNU/Linux, con gran conectividad, accesibilidad y costes de fabricación y producción mucho menores, permitiendo que este tipo de robots sean accesibles a más profesionales (incluso al público general), dando lugar a más desarrollos e investigaciones y no limitando el uso solo a grandes empresas del sector.

1.2. Motivación

Este proyecto se presenta como un trabajo final del máster de automática e informática industrial (MAII) de la Universidad Politécnica de Valencia y ha sido desarrollado en los laboratorios del Instituto de Automática e Informática Industrial (ai2).

En particular este proyecto une las áreas de la informática, redes de comunicación y programación de robots, áreas las cuales son de mi mayor interés.

Además, permite trabajar diversos temas tratados en el master como:

- Programación de sistemas empotrados
- Programación concurrente y en tiempo real
- Programación para el control de robots
- Sistemas distribuidos de comunicación
- Interacción persona-robot

1.3. Objetivos del proyecto

El objetivo final de este proyecto es diseñar y desarrollar una arquitectura de soporte para la ejecución y monitorización de un simulador de páncreas artificial embebido en un robot humanoide. Esta arquitectura permitirá realizar la monitorización y control vía internet de todo el estado del nuestro robot humanoide, además de ofrecer una serie de herramientas para poder programar nuevas funcionalidades al sistema de una manera sencilla, al tiempo que proporciona facilidad de modificación y flexibilidad en su uso.

También se desarrollará una aplicación para ordenador que nos permita conectarnos al robot vía internet con la que se pueda visualizar toda la información que nos proporciona durante una sesión de uso y controlar las diferentes funciones o comandos programados en él.

El software del robot nos permitirá controlar el estado de diferentes funcionalidades (hilos de ejecución) de forma simultánea, como un simulador de diabetes, diferentes escenarios de interacción humano-robot o cualquier otra funcionalidad programada. A su vez el software se encargará de enviar de forma periódica toda información que se considere importante y estará a la escucha de las peticiones que pueda realizar el cliente.

Anteriores trabajos desarrollados en el instituto de Automática e Informática Industrial ya habían trabajado con el robot NAO y el simulador de diabetes (Desarrollo de un sistema de simulación on-line de procesos metabólicos y su aplicación en robots humanoides) aunque adolecían de limitaciones. Se busca avanzar en dichos proyectos implementado una base software más sólida y general la cual permita la gestión de hilos (y por tanto de las actividades a ejecutar en el robot), comunicación vía internet (para monitorización continua del estado del robot y la simulación) y proporcione herramientas para crear nuevas funcionalidades en el sistema.

1.3.1 Objetivos específicos

- Desarrollo de una arquitectura software para el robot NAO que:
    - Permita la creación y gestión de hilos de ejecución, mediante una interfaz de desarrollo sencilla.
    - Ofrezca un sistema de intercambio de información entre los hilos que conforman el sistema robusto y eficiente
    - Permita tener un sistema de comunicación vía internet para el intercambio bidireccional de información con una aplicación cliente
    - Facilite el uso de las funcionalidades ofrecidas por el robot NAO
- Implementar (incorporar) el simulador de diabetes y los escenarios de interacción persona-robot (desarrollados en trabajo anteriores) en la arquitectura desarrollada.
- Desarrollo de una aplicación cliente para ordenador que permita:
    - Mostrar la información recibida por el servidor
    - Realizar peticiones al servidor

2. Estado del arte

En esta sección se definirá el concepto de robot humanoide y se hablará sobre los sistemas empotrados y su tendencia actual. Además, se comentará el papel de los sistemas basados en GNU/LINUX en la robótica actual.

2.1. Robots Humanoides

El termino robot humanoide se asocia al de robot cuya forma o estructura intenta asemejarse los más posible al cuerpo humano, y cuyos movimientos y acciones intentan replicar nuestro comportamiento [1].

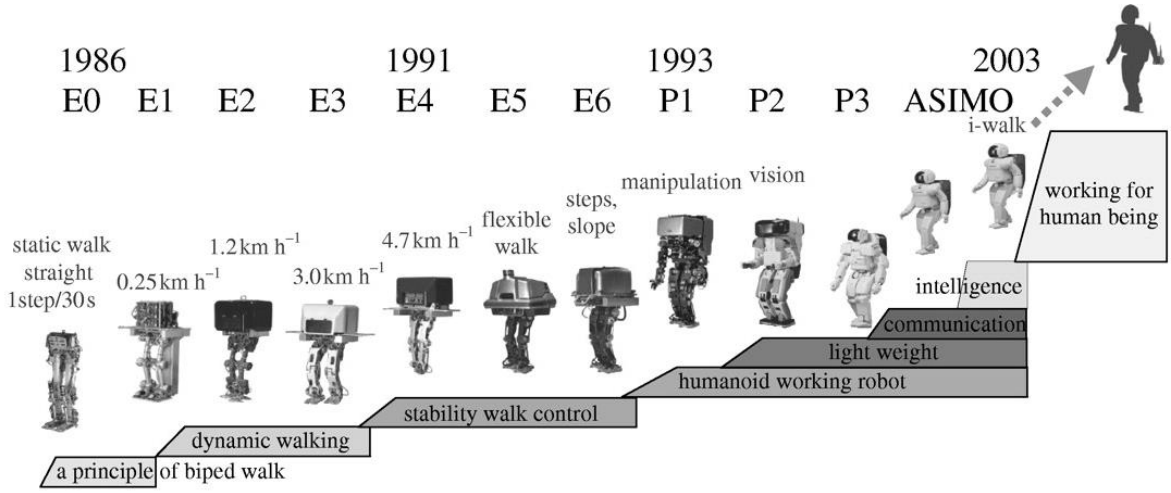

Ilustración 1 Evolución robot humanoide de Honda

En general suelen tener una cabeza, un torso, dos piernas y dos brazos (llegando a encontrarse robots que emulen partes más concretas como los ojos o la boca) [2][3], aunque pueden darse situaciones en los que algunos robots solo repliquen alguna de las partes (como robots humanoides con ruedas en vez de piernas) ya sea por conveniencia ante el entorno o limitaciones técnicas.

Ilustración 2 Robot Humanoide industrial Hiro

Dependiendo del robot humanoide y de su construcción, sus características y capacidades pueden cambiar, pero las más comunes de encontrar son:

- Alta sensorización por todo el cuerpo del robot (cámaras, micrófonos, giroscopios, sensores de presión, temperatura, encoders, etc).
- Alto número de actuadores, que permitan al robot interactuar con el entorno.
- Número elevado de grados de libertad, que le permita asemejar sus movimientos a los de los humanos.
- Reconocimiento de voz y capacidad de habla que le permita al robot interactuar con los humanos en su medio más común, el habla.
- Capacidad de procesamiento de imagen y visión por computación que le faciliten identificar objetos, disposición del entorno, etc.

El diseño de estos robots, así como sus capacidades y características suelen estar centradas en conseguir fines funcionales (dotarle de más de una capacidad al robot y que sea fácilmente adaptable entre tareas). Con propósitos y funcionalidades tales como:

- Usados de forma muy común como herramientas para la investigación científica en diferentes áreas.
- Desarrollo de un comportamiento similar al humano.
- Desarrollo y estudio de nuevos sistemas de locomoción.
- Remplazar a humanos en entornos de trabajo peligrosos.
- Asistencia en el trabajo.
- Entretenimiento.
- Enseñanza.
- Interacción con herramientas.
- Etc.

Como podemos ver, las aspiraciones para estos robots son muy altas, exigiéndoles tareas muy complejas y de muy alto nivel de abstracción (en algunos casos) para un robot. Por ello este campo de la robótica humanoide está muy ligado al campo de la inteligencia artificial [4], el cual les permite a los robots aprender cada vez que interactúan con los humanos ya sea en entornos de trabajo o desarrollo, consiguiendo cada vez comportamientos más parecidos y siendo capaces de ir mejorando en las tareas que se les asignen.

2.2. GNU/LINUX en la robótica

GNU/Linux es el termino por el que se conoce al sistema operativo GNU de tipo Unix (conjunto de software y herramientas que da funcionalidad al sistema) junto al kernel Linux (encargado de que el software y hardware del equipo trabajen juntos de manera correcta), comúnmente conocidos solo por Linux [5].

Este sistema operativo se caracteriza por ser un software libre (con licencia GPL) y de venir acompañado del código fuente, el cual todo el mundo puede modificar para que cumpla sus necesidades particulares, creando paquetes o incluso modificando el propio kernel del sistema para acceder al hardware (o hardware adicional) del sistema de maneras distintas.

Esto ha llevado a la creación y publicación de una multitud de distribuciones diferentes [6] cada una enfocada y optimizada para diferentes usos (sistemas multimedia, sistemas de tiempo real, sistemas de propósito general, etc) e incluso con tamaños más livianos capaces de entrar en memorias USB o sistemas con poca capacidad de almacenamiento.

Ilustración 3 Distribuciones GNU/LINUX

Cada vez se está utilizando más el sistema operativo Linux en el mundo de la robótica, esta tendencia tiene mucho que ver con la proliferación de micro controladores y ordenadores embebidos con núcleos ARM y x86 que son capaces de correr distribuciones especialmente diseñadas para controlar un robot. Convirtiendo a los sistemas Linux en opciones baratas, eficientes y muy rápidas.

Encontrando ya robots comerciales funcionando con sistemas operativos Linux (como Ava 500, Pepper, etc) y frameworks dedicados (con mantenimiento y soporte) al control de robots:

- Player/stage
- ROS (Robot Operating System)
- Orocos

- Rock Robotics
- JAUS
- Ad Hoc

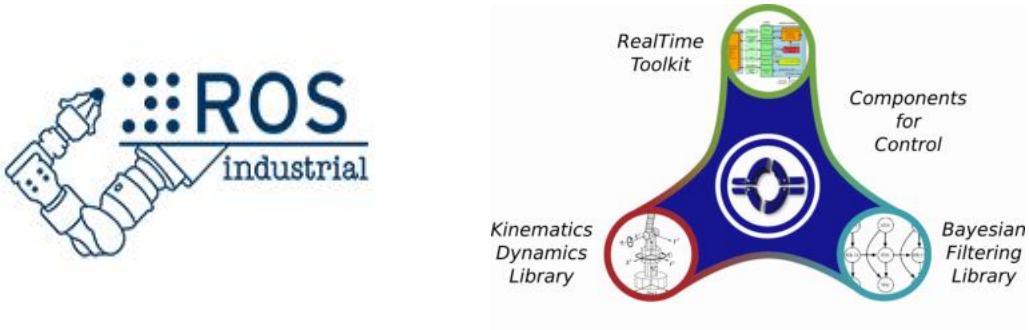

Ilustración 4 Logos de los softwares ROS y Orocos

Estas distribuciones proporcionan herramientas muy potentes a la hora de programar el robot, siendo el propio sistema operativo el que se encargue de hacer las gestiones generales con el hardware y gestionando los recursos de memoria, dejando solo la tarea de programar el comportamiento del robot.

Facilitan la interacción con hardwares específicos (como todo tipo de sensores o actuadores), proporcionando drivers bien definidos y probados (muchas veces realizados por las propias compañías que desarrollan el hardware). Además, permiten utilizar herramientas y paquetes ya desarrollados (muy probadas y testeadas) para ordenadores de propósito general como, por ejemplo, librerías de visión artificial, librerías para cálculos matemáticos, librerías para el cálculo de cinemáticas, librearías de inteligencia artificial etc.

Sin olvidar los beneficios que tiene trabajar sobre un sistema operativo GNU/Linux como:

- Sistema de archivos, bien conocido por la mayoría de programadores y con soporta de varios sistemas de archivos (ext2, ext3, etc).
- Sistema multi-tarea, pudiendo tener varios programas funcionando a la vez gestionados por el propio sistema.
- Memoria virtual, puede utilizar una parte de la memoria del disco duro como memoria ram, aumentado la eficiencia del sistema al mantener los procesos activos en memoria física y el resto en virtual.
- Repositorios de software propios.
- Seguridad, mediante el uso de autenticación, autorización de usuarios y encriptación de ficheros.
- Muy orientado a la red, con las capacidades de networking bien implementadas en el núcleo.

Además, la utilización de software libre beneficia en gran medida la robótica para aficionados, puesto que los avances, desarrollos y construcciones pueden ser utilizadas por toda la comunidad (casi siempre de forma gratuita).

2.3. Sistemas empotrados

Los sistema empotrados o embebidos hacen referencia a sistemas de computación que forman parte de un sistema mayor y están diseñados para realizar una o varias funciones dedicadas a controlar el sistema global [7].

Por lo general los sistemas embebidos se pueden programar directamente en el lenguaje ensamblador del microcontrolador o microprocesador incorporado sobre el mismo (cada vez menos), o también, si se está trabajando sobre un sistema operativo, pueden utilizarse lenguajes como C, C++, Java o cualquier otra del que se disponga de un compilador.

Hoy en día la mayoría de dispositivos electrónicos que se pueden encontrar en el mercado llevan integrado al menos un sistema empotrado llegándose a encontrar productos con múltiples sistemas empotrados trabajando de forma simultánea.

Ilustración 5 Sistema empotrado y ejemplos de uso

En general los productos con sistemas empotrados integrados suelen contar con las siguientes partes en su estructura:

- Unidad de control, que se encargar de la lógica del sistema global y recae en manos del sistema computacional.
- Entrada/salida, pines e interfaces para conectar los componentes que constituirán el producto.
- Sensores, que se encargaran de recoger información de su entorno para su posterior procesado e interpretación.
- Alimentación de corriente, encargada de proporcionar y adecuar los niveles de corriente necesarios a cada componente.
- Actuadores, encargados de realizar movimientos, mandar mensajes por pantalla, encender leds, es decir, realizar las acciones que el usuario pueda interpretar.
- Unidad de comunicación, encargada de proporcionar interfaces de comunicación estándar para conectarse con otros equipos o dispositivos.

Una de las partes más importantes de los sistemas empotrados es la capacidad de comunicación que necesitan ya que suelen estar conectados con multitud de hardware y dispositivos de terceros. Por ello los sistemas suelen venir con gran cantidad de interfaces estandarizadas de comunicación, como:

- RS-232
- SPI
- I2C
- CAN
- USB
- WI-FI
- …

Se pueden diferenciar dos grandes tipos, los sistemas empotrados con microprocesadores que utilizan circuitería integrada separada, comunicada con el controlador mediante buses. Y los que utilizan microcontroladores donde la mayoría del hardware está integrada en el mismo chip reduciendo el tamaño y consumo de energía.

Mayormente un sistema empotrado es un sistema cuyo hardware y software están diseñados y optimizados para resolver las necesidades del sistema global volcando todos los recursos disponibles con el propósito de conseguir ese fin.

Hoy en día, a los sistemas empotrados actuales se les exigen aún más funcionalidades, se requieren más conexiones, se pide más personalización para el usuario, etc. Llevando a desarrollos de softwares empotrados de millones de línea que requieren hardware más avanzado para funcionar de forma fluida, dando lugar a la creación de placas con aceptación mundial y capacidades más extensas (aumentado con cada nueva versión), como Arduino, Raspberry o BeagleBone [8], ofreciendo algunas de ellas la capacidad de instalación de sistemas operativo.

Ilustración 6 Placas BeagleBone black y Raspberry pi

Lo que lleva a una gran cantidad de posibilidades a la hora de elegir una plataforma. La elección del hardware para nuestro sistema o producto final dependerá totalmente de las especificaciones técnicas, necesidades del producto, tamaño que se desee, consume energético, y diferentes características del sistema global que se quiera desarrollar.

3. Materiales y métodos

En esta parte de la memoria se hablará del robot humanoide utilizado en la realización de este trabajo de fin de master, centrándose en el hardware del que dispone y del software que proporciona. Además, se expondrán los lenguajes de programación que se utilizaran para la realización del servidor que correrá en el robot y del programa de escritorio a desarrollar.

3.1. Robot NAO

El robot humanoide que se va a utilizar para desarrollar este trabajo de fin de master es el robot NAO (versión del robot V4), desarrollado y distribuido por la empresa francesa Aldebaran Robotics, subsidiaria del grupo SoftBank [9].

El desarrollo de este robot comenzó en 2004, sustituyendo en 2007 al robot Aibo de Sony como plataforma estándar para la Robocup (concurso internacional de robótica [10]). Además, se realizó una edición académica desarrollada específicamente para universidades y laboratorios con fines de investigación y educación. Lanzada en 2008 para estas entidades comentadas y más tarde en 2011 para un público general, siendo uno de los primeros robots humanoides autónomo y programable accesibles al público general.

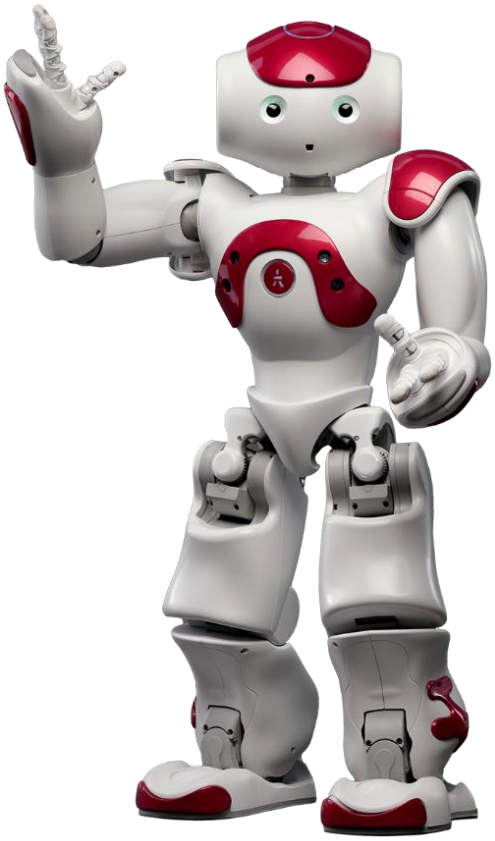

Ilustración 7 Robot NAO

El principal propósito del robot NAO recae en la investigación y el uso para la educación, esto se debe a que es totalmente programable permitiendo explorar las áreas que cada usuario necesite. Además, ofrece unas herramientas software y hardware muy potentes y en constante desarrollo. Teniendo un software (Choregraphe) de alto nivel para programar comportamientos

de forma rápida en el NAO sin tener que escribir código especifico o un framework de más bajo nivel que nos permite realizar programas más avanzados.

3.1.1 Robot NAO

- **Construcción**

Las dimensiones del robot en una posición estándar (de pie con los brazos extendidos) son, 573mm de altura, 311mm de profundidad y 275mm anchura, como se puede ver en la ilustración 8. Con un peso de unos 5kg.

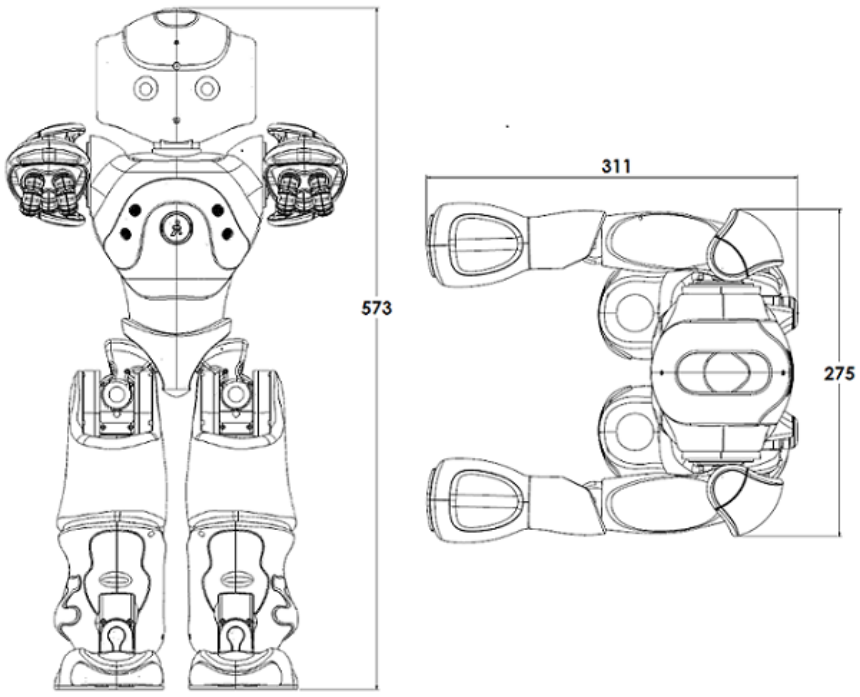

Ilustración 8 Dimensiones del robot NAO

El robot está dotado de motores de corriente continua sin escobillas, utilizados para mover todas sus articulaciones, que se dividen en:

- Cabeza: con 2 grados de libertad.
- Brazos: con 6 grados de libertad cada uno.
- Piernas: con 5 grados de libertad cada uno.
- Pelvis: con 1 grado de libertad:

Dando un total de 25 grados de libertad que le permiten al robot realizar movimientos complejos y caminar de forma bípeda sin perder estabilidad. En la figura 9 se pueden ver de forma general colocación y posición de los motores y articulaciones, para ver de forma más detalla se puede consultar la documentación de Aldebaran [11].

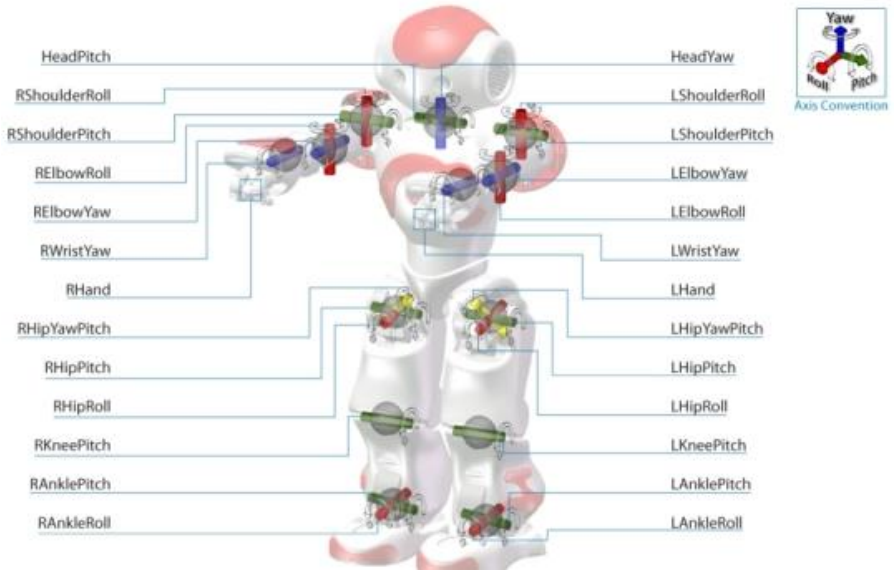

Ilustración 9 Articulaciones NAO

- **Sensores y elementos de interacción**

Nao cuenta con un nivel de sensorización completo repartido por toda su estructura, mayormente utilizado para el control motriz y de estabilidad y para la interacción con las personas y el entorno.

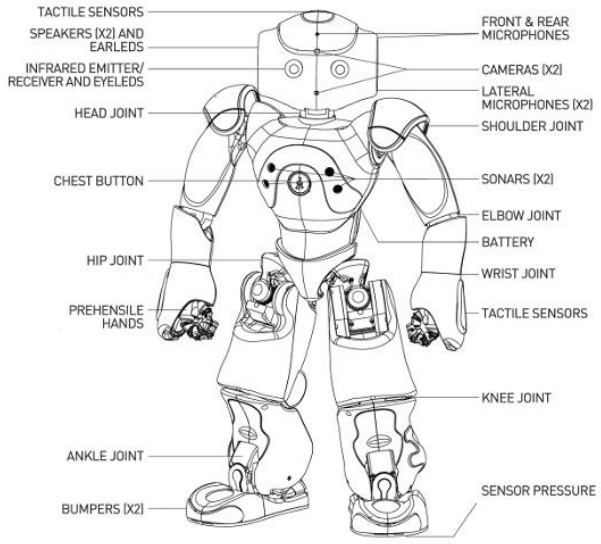

Ilustración 10 Posición de algunos sensores y actuadores

La lista de sensores con la que está dotado el robot es la siguiente (más detalles sobre cada elemento en la documentación online [12] en los apartados de “Sensors” e “Interaction”):

- 8 sensores de fuerza (resistivos), colocados 4 en cada pie (en la planta).
- Una unidad inercial localizada en el torso, la cuan cuneta con su propio procesador.
- Un sonar compuesto por dos sensores de ultrasonidos (con dos emisores y dos receptores), colocados al frente del torso.

- 36 encoders magnéticos, para el control de posición de las articulaciones (32 bits de precisión).
- 3 sensores capacitivos situados en la cabeza.
- 3 sensores capacitivos en cada mano.
- 2 bumpers, colocados uno en cada pie (parte delantera).

También cuenta con una serie de actuadores y sensores mayormente destinados a la interacción con las personas.

- 2 altavoces, que le proporcionan un sistema estéreo (situados en las orejas).
- 4 micrófonos, colocados alrededor de la cabeza, capaces de captar sonidos en el rango de 150Hz a 12KHz.
- 2 video cámaras idénticas, con una resolución de 1280x960 y una capacidad de captura de 30 frames por segundo. Situadas las dos en la cabeza, una apuntado al frente con un propósito más general y la otra apuntado al suelo dedicada más a detectar obstáculos cercanos.
- 2 infrarrojos, con propósito académico.
- Conjuntos de leds repartidos por toda la estructura del robot. En la cabeza, en cada ojo, en cada oreja y los pies.

- **CPU y conectividad**

Cuenta con una CPU Intel Atom Z530 (situada en la cabeza) a 1.6 GHz, 1 GB de memoria RAM, 2GB de memoria flash y 8GB de Micro SDHC, que le permiten correr sin ningún problema una versión modificada del sistema operativo Linux del que se hablara más adelante. Además, cuenta con una segunda CPU situada en el torso destinada a la gestión de señales de las articulaciones y propósitos de control.

El robot cuenta con una conexión para ethernet de tipo RJ45 10/100/1000 baseT, cuyo principal propósito es configurar la conexión Wifi del robot. La conexión Wifi utiliza el estándar IEEE 802.11 b/g/n y permite seguridad WEP, WPA, WPA2. Esta conexión es la destinada a la comunicación con las distintas aplicaciones.

Por último, el robot tiene un USB, destinado a labores de actualización de software y para conectar dispositivos externos como una placa Arduino, una Kinect o un sonar Asus 3D.

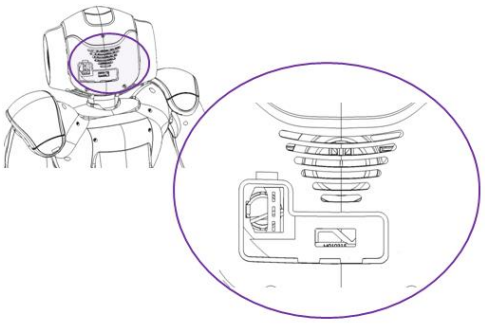

Ilustración 11 Conexiones USB y ethernet

- **Sistema operativo y Software**

El robot viene con una serie de software embebido, corriendo en la CPU situada en la cabeza, que le permite realizar comportamientos autónomos y ejecutar código creado por los usuarios.

El software que trae se puede diferenciar en dos partes, la primera y base del robot es OpenNAO, que es el sistema operativo del robot y no es más que un sistema GNU/Linux basado en la distribución Gentos y modificado específicamente por la compañía para que se ajuste a las necesidades del robot.

La segunda es el software NAOqi, el cual corre sobre OpenNAO y controla todo el robot, además de proporcionar un framework para poder programar el robot.

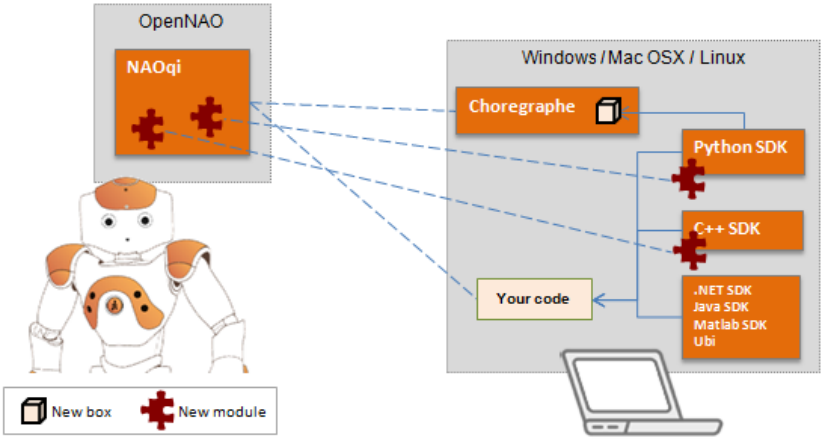

Ilustración 12 Diagrama de arquitectura

- **Framework NAOqi**

El framework al ser una combinación de un sistema operativo GNU/Linux y el software NAOqi ofrece todo lo necesario para la programación del robot (concurrencia, eventos, mecanismos de sincronización, acceso a recursos, …). Ya que OpenNAO nos ofrece todas las funcionalidades de los sistemas operativos convencionales como, creación de thread, llamadas al sistema o llamadas bloqueantes que son herramientas de programación muy conocidas. A la vez el software NAOqi encapsula todas las llamadas especificas a actuadores, sensores y demás características del robot.

El framework es multiplataforma, lo que quiere decir que es posible desarrollar en Windows, Linux o Mac (siempre que el desarrollo no sea para ejecutarse dentro del robot en cuyo caso no). Además de ser multilenguaje con APIs de programación idénticas tanto para C++ como para Python.

El conjunto de estas herramientas nos permite realizar módulos que se pueden ejecutar tanto en el robot consiguiendo que el código se ejecute mucho más rápido (módulos locales), como en otro robot u ordenador comunicándose mediante la red (módulos remotos).

Cuando se enciende el robot el software NAOqi carga todas las librerías indicadas en un fichero interno del sistema (autoload.ini), las librearías ofrecen una serie de clases (módulos) con diferentes servicios para utilizar.

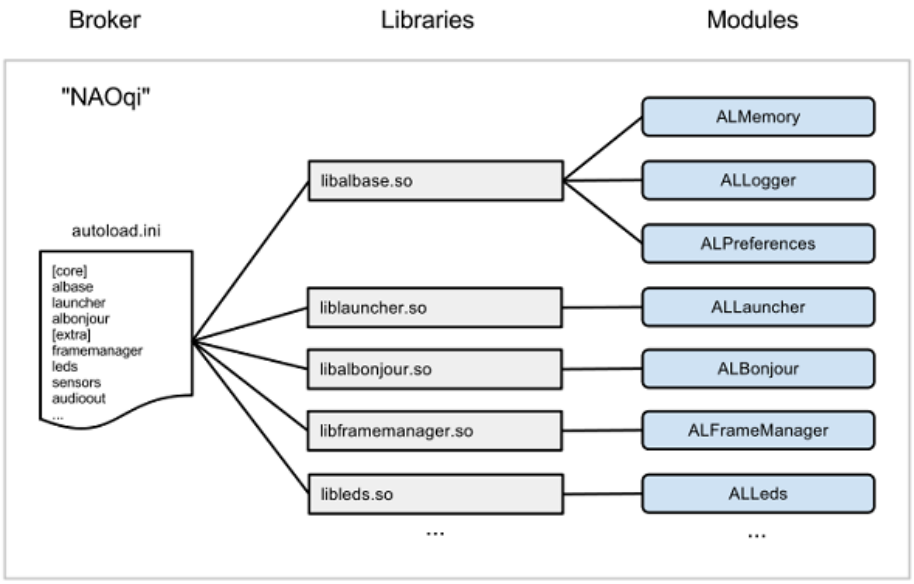

Ilustración 13 Esquema de inicio de NAOqi

El acceso a todos estos módulos instanciados en el inicio del sistema se hace a través de un bróker. El cual tiene dos roles principales, permitir encontrar módulos y métodos y proveer acceso a la red, permitiendo que los módulos que no estén ejecutándose en el robot tengan acceso desde fuera.

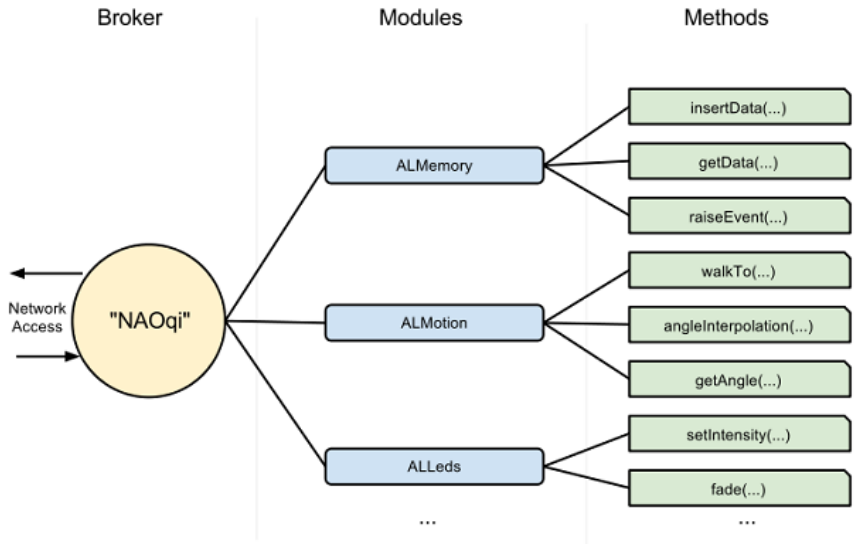

Ilustración 14 Esquema acceso a broquer

Los modulo normalmente son una clase (o más de una) asociada a una librería o biblioteca, estas librerías se cargan en el arranque del sistema y se instancia su clase asociada, permitiendo el acceso al resto de módulos a las funcionalidades que ofrezcan y/o ejecutando su propio código en caso de que no permita acceso a sus métodos.

Dentro del módulo no todas las funciones de la clase se comparten, las que se deseen compartir para el resto de módulos tienen que ser especificadas de forma manual para que NAOqi se dé cuenta y las ponga a disposición.

A la hora de desarrollar un nuevo módulo la principal característica y la que más marcara el desarrollo es el tipo de modulo, local o remoto.

- Módulos locales: son compilados como bibliotecas (.so) y solo pueden ser ejecutadas en el robot. Un módulo local es uno o más módulos que se ejecutan en el mismo proceso, comunicándose entre ellos solo por un bróker. El hecho de estar en el mismo proceso permite que puedan compartir datos sin tener que pasar por la red, lo que hace a los módulos locales mucho más eficientes (todo desarrollo que se necesite tiempo real tendrá que ser local).
- Módulos remotos: son compilados como ejecutables, suelen ser más fáciles de programas y debbugear, pero menos eficientes. Los módulos remotos son módulos que se comunican utilizando la red, y necesitan un broquer específico para comunicarse con otro modulo. La conexión entre módulos remotos puede darse de bróker a bróker donde los dos módulos pueden hablar entre ellos, o de proxy a bróker donde solo existe una línea de comunicación.

3.2. C++ y bibliotecas

Para programar la parte del servidor (el cual será un módulo local que se ejecutara en el robot), el grupo Aldebaran nos proporcionar la posibilidad de elección entre solo dos lenguajes de programación (Python y C++), esto se debe a que se necesita realizar una compilación cruzada y las herramientas para ello solo están disponibles para estos dos lenguajes.

En este proyecto se va a utilizar el lenguaje de programación C++, ya que es un lenguaje bien conocido que ofrece buenas características, además de que se utilizaran una serie de librerías externas realizadas en este lenguaje.

Como se comentó antes se necesita realizar una compilación cruzada, ya que el S.O del robot está utilizando el lenguaje C++ junto a unas bibliotecas aparte, conocidas como Boost las cuales el software NAOqi utiliza de manera muy amplia, por lo que se necesita herramientas que puedan compilar nuestro código con las herramientas mencionadas.

- **C++**

Se considera a C++ un lenguaje híbrido, puesto que nos permite elegir entre diferentes paradigmas de programación, de entre los cuales se destacan la programación estructurada y la programación orientada a objetos.

Es totalmente compatible con el paradigma de programación orientada a objetos incluyendo los cuatro pilares en los que se basa:

- Encapsulación.
- Ocultación de datos.
- Herencia.
- Polimorfismo.

Considerado lenguaje de nivel medio, ya que ofrece características de lenguajes de bajo nivel (operaciones más cercanas al manejo directo de la CPU), como por ejemplo el uso de punteros o el manejo de memoria, y a su vez también ofrece características de lenguajes de alto nivel (nos abstraen de los detalles de la CPU), como por ejemplo plantillas (templates) o el manejo de excepciones (try-catch).

Las principales características del lenguaje C++ son las siguientes:

- Lenguaje estandarizado (ISO-abierta): en 1988 C++ fue estandarizado por un comité de la ISO (Organización Internacional de Normalización).
- Lenguaje compilado: traducido directamente a código nativo de la máquina, consiguiendo con esto llegar a ser uno de los lenguajes más rápidos del mundo (siempre que el código esté optimizado).
- Soporta la revisión de tipos de forma estática y dinámica: permite conversiones de tipos de variables que deben controlarse ya sea en tiempo de compilación o en tiempo de ejecución.
- Ofrece distintos paradigmas para trabajar: apoyo a diferentes paradigmas de programación, procedimental, orientado a objetos, genérico, etc.
- Es portable: dispone de una amplia gama de compiladores que se ejecutan en diferentes plataformas con pocos o ningún cambio en el código.

- Es compatible con C: al basarse directamente en C es compatible con casi todo el código escrito en C, permitiendo el uso de bibliotecas escritas en C, con pocos o ningún cambio en el código.
- Cuenta con gran apoyo: al ser un lenguaje muy popular cuenta con mucha gente trabajando con él, dando lugar a una amplia cantidad de bibliotecas destinadas a todo tipo de usos.

- **Bibliotecas Boost**

Las bibliotecas boost son un conjunto de bibliotecas separadas de software libre [13], preparadas para extender las capacidades del lenguaje C++, se distribuyen bajo una licencia de tipo BSD y ofrecen:

Ilustración 15 Logo bibliotecas Boost para C++

- Procesamiento de cadenas y texto.
- Contenedores de datos.
- Iterados.
- Algoritmos.
- Programación de funciones de orden superior.
- Programación genérica.
- Plantillas.
- Programación concurrente.
- Métodos matemáticos y numéricos.
- Etc.

- **Biblioteca Alglib**

Además, se utilizará la biblioteca Alglib [14] (utilizada en los proyectos anteriores), para la implementación del simulador de diabetes, la cual será la que nos proporcione las herramientas matemáticas que necesita el simulador.

Ilustración 16 Logo biblioteca Alglib

Biblioteca enfocada al análisis matemático y procesamiento de datos, que ofrece:

- Análisis de datos.
- Optimizadores y solucionadores no líneas.
- Interpolación y ajuste lineal/no lineal de mínimos cuadrados
- Algebra lineal (solucionadores lineales directos e indirectos, transformada de Fourier).
- Etc.

3.3. Java

Para programar la parte del cliente la cual estará centrada en realizar y gestionar una comunicación con el robot, se ha seleccionado el lenguaje de programación Java.

Ilustración 17 Logo Java

En gran medida se ha elegido este lenguaje debido a la facilidad que proporciona la máquina virtual Java a la hora de ejecutar una aplicación en diferentes sistemas operativos y diferentes configuraciones de hardware.

Cuando se tiene el programa desarrollado, este se compila y se transforma en instrucción conocidas con el nombre de bytecodes que se guardan como ficheros ".class". Estas instrucciones son independientes del tipo de ordenador. La máquina virtual, junto al interprete ejecutaran las instrucciones que componen el programa independientemente del ordenador.

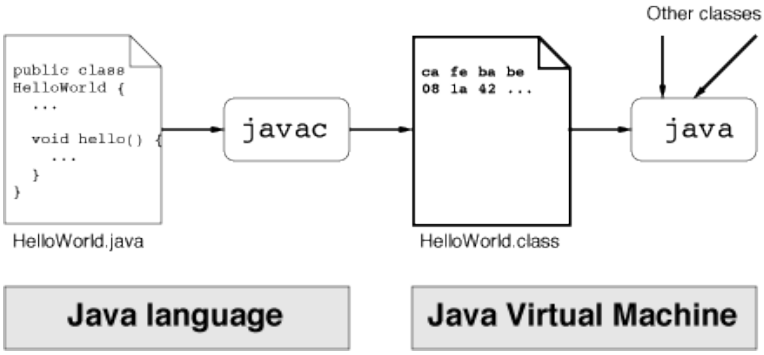

Ilustración 18 Esquema funcionamiento de la máquina virtual de java

Además de por esta gran ventaja también se ha seleccionado por:

- Ser orientado a objetos
- Multi hilo
- Robusto
- Seguro
- Simple

Se utilizaría una biblioteca externa para facilitarnos la creación de distintos componentes gráficos como gráficos de barras, series temporales etc. La biblioteca es Jfreechart y se puede encontrar en [15].

Como entorno de desarrollo se utilizará eclipse ya que proporciona herramientas para facilitarnos la creación de interfaces gráficos de usuario.

4. Arquitectura y Servidor

En esta sección se hablará de las características de las arquitectura y funcionalidades del servidor desarrollado, como también de una breve descripción de las clases que lo componen (más información en el anexo Manual del programador).

4.1. Funcionalidad

Las funcionalidades que se desarrollaran en esta parte del proyecto se pueden diferenciar en dos tipos: funciones que ofrezca el servidor al estar ejecutándose y funciones que proporcionara la arquitectura software:

- Con el desarrollo de la arquitectura software, se pretende ofrecer unas herramientas de programación que proporcionen ayuda en futuros desarrollos a la hora de implementar nuevos comportamientos y funcionalidades.
- El servidor en ejecución estará siempre a la espera de comandos entrantes para procesar y enviado información de estado de forma periódica, a la vez que ejecuta todos los hilos programados.

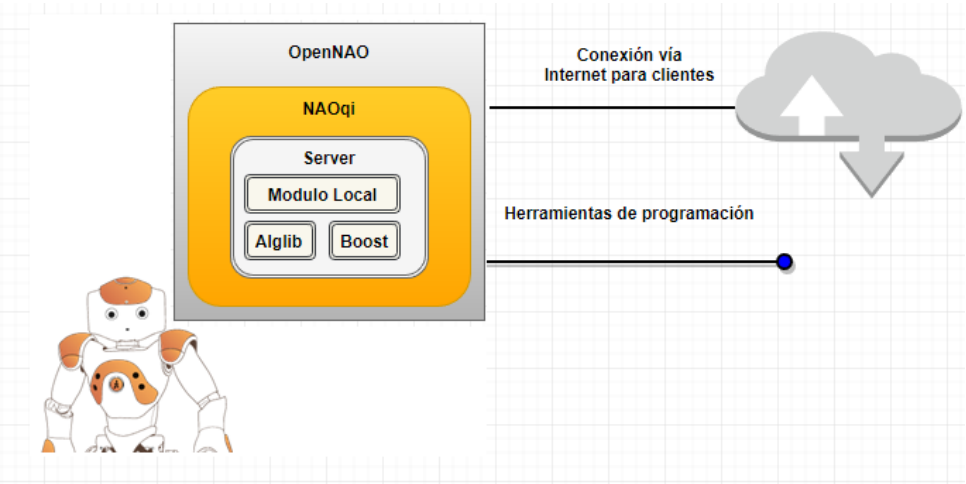

Ilustración 19 Esquema arquitectura software e interfaces proporcionadas

Entrando más en detalle, la arquitectura software a desarrollar nos ofrecerá las siguientes características:

- Herramientas para la creación de hilos.
- Ofrecerá una clase que se encargará de la gestión del estado de los hilos.
- Facilitará el intercambio seguro de información entre hilos.
- Facilitará el envió de información por la red a la aplicación cliente.
- Proporcionará herramientas para ampliar los comandos del servidor.
- Ofrecerá una clase que encapsulará la interacción con el robot NAO.

4.2. Clases del sistema

4.2.1 Disposición

Para poder ofrecer todas la funcionalidades y características comentadas en el apartado anterior, la arquitectura contara con las siguientes clases:

- AccioensNAO
- MyModule
- ThreadManager
- Dispatcher
- Runnable
- TCPServer
- TCPsender
- TCPreciver
- DatosCompartidos
- Escenario
- Interaccion
- Simulador

Conectadas entre si como se puede ver en el diagrama de clases del sistema en la ilustración 20.

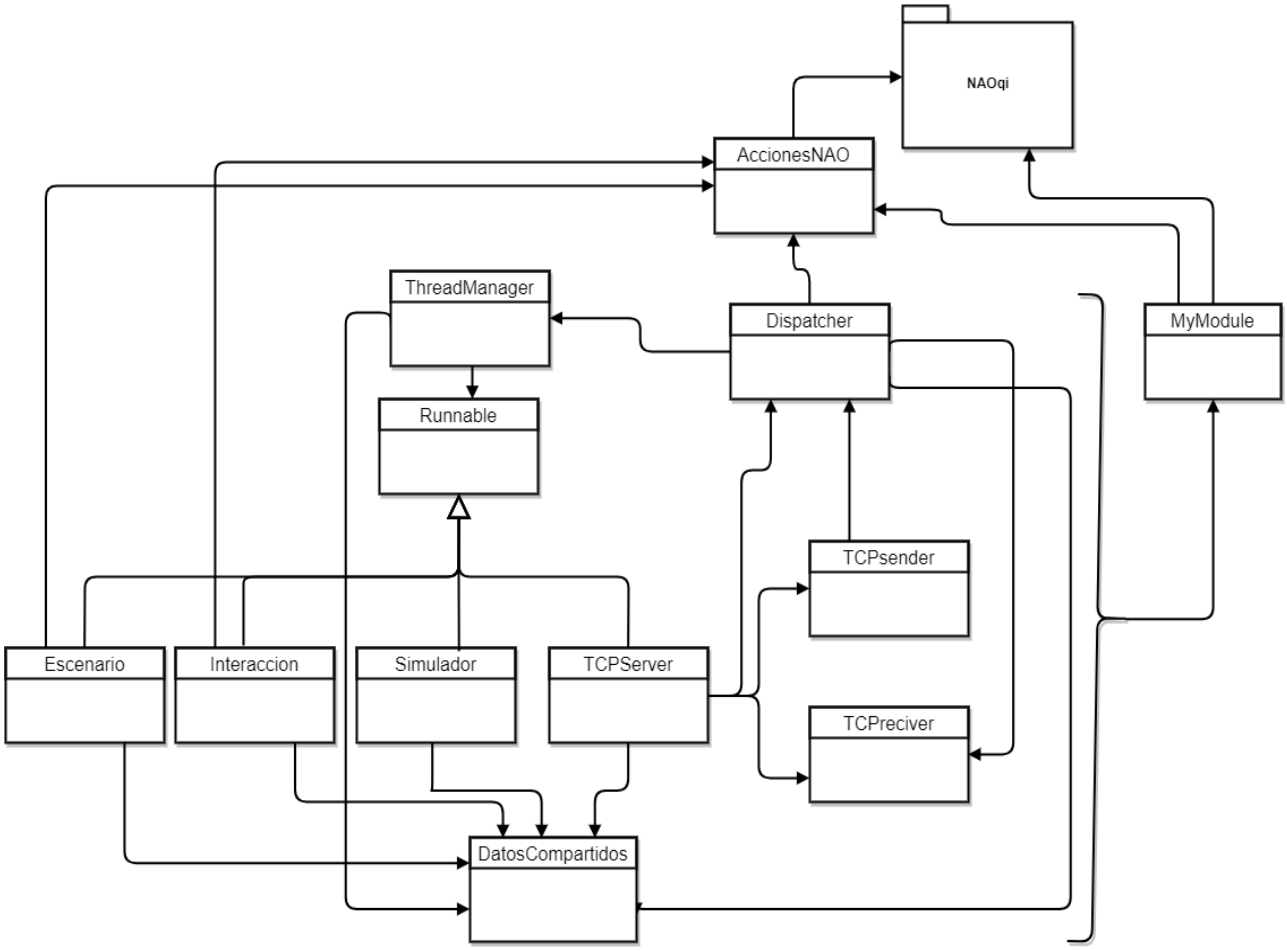

Ilustración 20 Diagrama de clases servidor

4.2.2 Explicación de las clases

- **Runnable**

Esta clase ayuda a crear y añadir hilos al sistema sin tener que hacer cambios grandes en la estructura del programa. Las clases que necesiten tener un hilo encapsulado (y necesiten ser controladas por el ThreadManager) heredarán de Runnable y tendrán que sobrescribir una serie de métodos que permitirán a la clase ThreadManager gestionar el estado del hilo asociado a la clase.

Los métodos para sobrescribir serán:

- run: este encapsulara en comportamiento específico del hilo.
- pausar: esto método será llamado siempre que se quiera pausar el hilo.
- desPausar: este método será llamado siempre que se quiera des-pausar el hilo.
- pararThread: este método será llamado siempre que se quiera parar el hilo.

Estos métodos proporcionan una interfaz para el manejo de los hilos del sistema, permitiéndonos manejar el estado de los hilos sin tener que preocuparnos de la implementación particular de cada uno. Cada hilo se preocupará de cómo debe realizar de forma correcta las acciones, pero con el mismo punto de acceso para todos.

Además de estos métodos ofrecerá otros métodos para el control del estado, como un método start o un método esperarCondicion.

Todos estos métodos definen el comportamiento básico que se le quiere dar a los hilos del sistema, los cuales pueden estar parados, arrancados o pausados (dentro de cada uno de estos estados podrán tener sub-estados definidos por cada hilo). En caso de querer que un hilo especifico no se pueda parar o pausar se sobrescribirá el método en blanco.

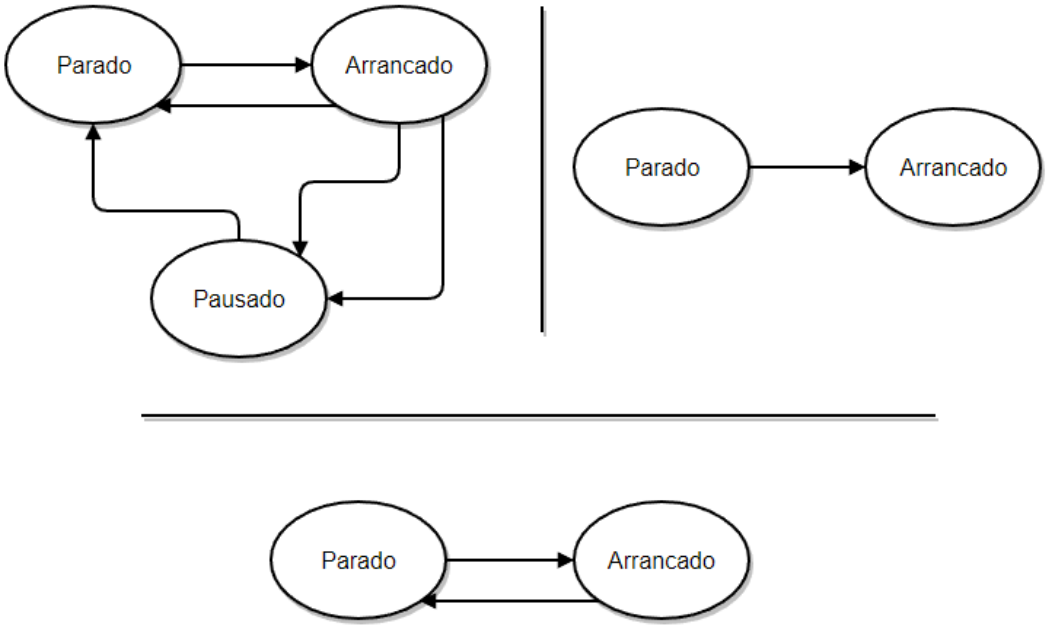

Ilustración 21 Diferentes esquemas de estados para los hilos

Un hilo parado simboliza un hilo el cual sub objeta ha sido instanciado y añadido al ThreadManager, pero no se ha llamado a la función start, un hilo arrancado será un hilo que este dentro de su bucle o secuencia de acción y un hilo parado será un hilo dentro de su bucle o secuencia de acción, pero parado en algún punto del código.

Las funciones pausar y parar, no pausaran o pararan el hilo justo cuando sean llamadas ya que un hilo no se puede parar en cualquier punto aleatorio (el lenguaje no lo permite), estas deberán utilizarse como un aviso de que se necesita pausar o parar, para que en las partes del código donde se pueda y quiera realizar la pausa o parada lo sepan.

- **ThreadManager**

En esta clase se pretende encapsular todo el manejo del estado básico de los hilos presentes en el sistema. Teniendo que pasar por ella siempre que se quiera cambiar el estado de un hilo, utilizando los métodos que obliga a sobrescribir la clase Runnable.

La clase ofrecerá una serie de métodos que permitirán añadir los hilos (las instancias de las clases) a una lista otorgándoles un ID único para que así posteriormente se pueda modificar el estado (indicando ID y acción).

Solo podrá manejar el estado de los hilos que hereden de la clase Runnable, ya que un hilo creado de forma manual sin heredar esta clase no ofrecerá los métodos básicos y obligatorios en la clase Runnable.

Por lo que con esta clase se pretende ofrecer una forma fácil de gestionar los hilos dentro del servidor, así como ofrecer la posibilidad de añadir nuevos hilos que puedan ser manejados por esta clase de una forma cómoda. Los pasos para añadir nuevos hilos serian:

1. Creación de una clase que herede de Runnable y sobrescriba los métodos abstractos de la clase.
2. Añadir el hilo creado al ThreadManager dándole una ID única
3. Gestionar el estado del hilo utilizando el ThreadManager y la ID indicada.

Esta clase no se ocupará de los aspectos particulares de cada hilo, sino que solo se encargará del estado básico de los hilos entendidos como:

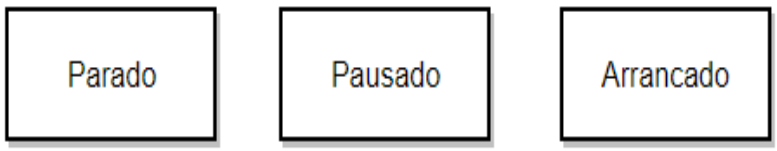

Ilustración 22 Estados básicos de los hilos manejados por el ThreadManager

Dentro de cada uno de los estados dependiendo de la implementación realizada podrá tener sub-estados o no.

Esta clase ofrecerá funciones adicionales como la posibilidad de añadir hilos excluyentes entre si, es decir, solo uno de todos los hilos que se añadan como excluyentes podrá estar activo a la vez. Esta funcionalidad está enfocada a controlar los hilos destinados al trabajo completo con el robot, los cuales no se pueden solapar en el tiempo ya que se podrían realizar llamas simultaneas de movimiento, habla, etc.

También mantendrá en DatosCompartidos un dato que hará referencia al número de hilos presentes, al ID de cada hilo y a su estado actual (separando los hilos excluyentes con el carácter '%') marcados para enviar vía internet y así el cliente sepa en todo instante el estado de los hilos del sistema.

- **DatosCompartidos**

En esta clase se encapsulará la gestión e intercambio de información entre los diferentes hilos y clases que se encuentran en el sistema. Siempre que un hilo quiera compartir algún dato con el resto de hilos o buscar un dato en particular acudirá esta clase.

Esta clase deberá instanciarse una única vez al inicio de nuestro modulo y ser pasada (su referencia) a todo hilo o clase que necesite compartir información.

Puesto que esta clase estará manejando datos y puede ser accedida por múltiples hilos de forma simultánea para modificarlos o leerlos, tendrá que ser una clase "threadSafe", es decir, todos los accesos ya sean para lectura o modificación de cualquier dato tendrán que estar protegidos por zonas de exclusión mutua (porción de código donde solo se puede ejecutar un hilo a la vez), consiguiendo que un hilo no lea un dato que se está modificando en ese mismo instante.

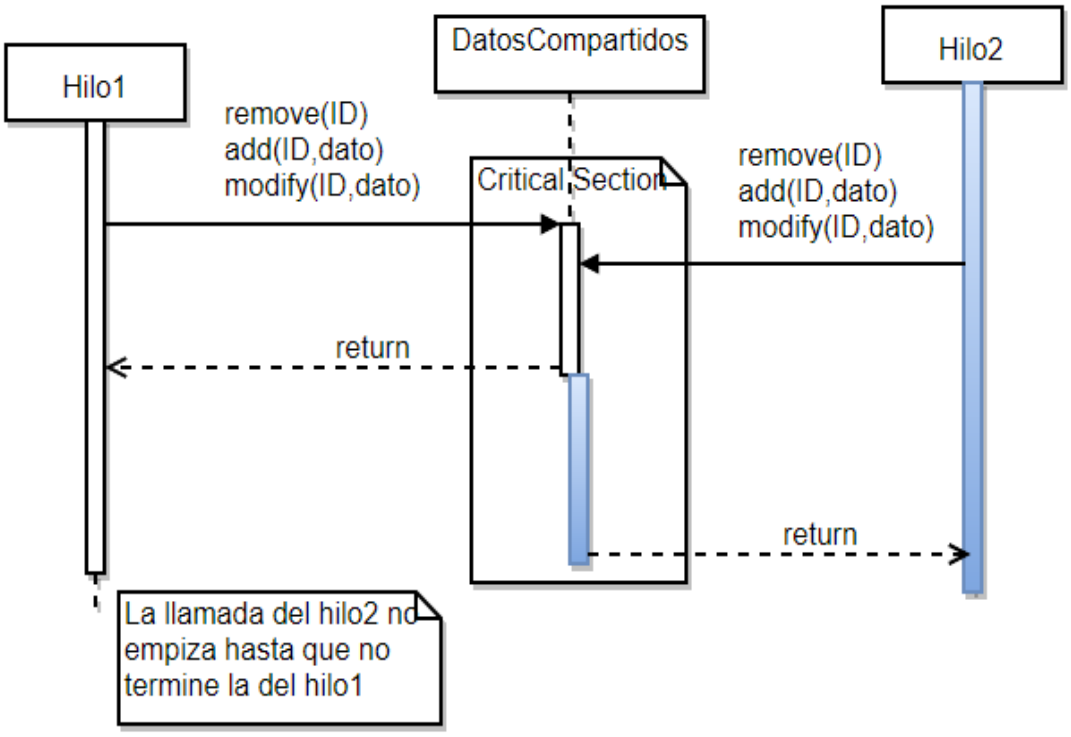

Ilustración 23 Diagrama secuencia llamada a sección critica de dos hilos simultáneamente

Ofrecerá herramientas para añadir, modificar y eliminar datos de forma dinámica (datos de tipo int, double y string). Cada dato almacenado estará ligado a un identificador único, el cual nos permitirá modificar y acceder al dato que queremos.

Además, permitirá marcar cualquier dato (cuando se haga su inserción) como enviable, formando una lista de datos perteneciente a los datos compartidos que deberá ser enviada al cliente que se conecte. Permitiendo así modificar de forma dinámica el envió de datos al cliente.

Aparte de los datos genéricos ya comentados, la clase gestionara también el acceso a un tipo de dato especifico (conjunto de valores doubles y booleans definidos dentro de una estructura) el cual es destinado a ser usado por el hilo Simulador. Este dato tendrá un valor por defecto y siempre que el hilo Simulador lo lea el dato se actualizará al valor por defecto.

- **AccionesNAO**

La función de esta clase es encapsular todo el comportamiento que tenga el sistema relacionado con acciones de movimiento e interacción del robot y que utilicen el framework NAOqi, es decir, todas las llamadas a primitivas de movimiento, leds, habla, etc.

Ofrecerá unas funciones de más alto nivel al sistema, ya que se encargará de hacer configuraciones previas necesarias, ajuste de parámetros y control de errores.

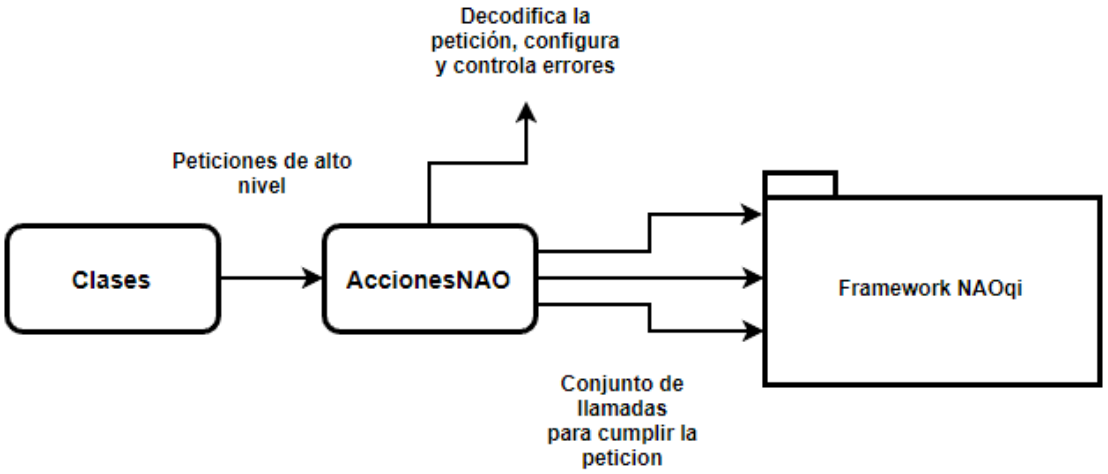

Ilustración 24 Esquema peticiones de interacción

Además de los métodos de interacción también proporcionara un método para solicitar al robot que escuche palabras, este método será bloqueante, es decir, todo hilo que llame a este método parara su ejecución hasta que el robot reconozca una palabra o se pase un periodo de tiempo especificado.

También ofrecerá un par de métodos para solicitar y mirar si un hilo está usando el robot en modo exclusivo. Por ejemplo, un hilo que esté realizando un escenario de interacción usando todas las funcionalidades del robot solicitara el modo exclusivo, cuando otro hilo quiera realizar una acción puntual (por ejemplo, cuando se reciba una orden por la red) mirara si el robot esta en modo exclusivo y en caso afirmativo no realizara la acción.

Para que esta clase pueda funcionar de forma correcta se le tendrá que pasar una vez se instancie el nombre de nuestro modulo (necesario para poner al robot en espera de palabras) y un puntero al parentBroker, si no la clase no realizaría las peticiones y dará error. La necesidad de pasarle el parentBroker es que la clase AccionesNAO pueda solicitar los proxys que necesita para realizar las peticiones de interacción. Los proxys que se utilizarán serán:

- ALTextToSpeech
- ALAutonomousMoves
- ALRobotPosture
- ALMotionProxy
- ALAutonomousLifeProxy
- ALLedsProxy
- ALSpeechRecognition
- ALAnimatedSpeech
- ALBasicAwarenessProxy

- **Dispatcher**

Clase encargada de procesar los comandos entrantes al servidor. Para ello esta clase, como se puede ver en el diagrama de clases de la ilustración 20, tendrá acceso a casi todas las funcionalidades que proporciona el servidor.

- Acceso a datos compartidos.
- Envió de respuesta.
- Control de las acciones del robot.
- Control de hilos.

El procesamiento de un comando entrante se hará en tres fases distintas:

1. Se verificará que el formato del comando (separadores, prefijo, final de comando, etc.) sea el correcto, en otro caso se desechará la petición.
2. Se identificará el tipo de comando (el cual le permitirá saber a la clase como procesarlo) y los argumentos asociados.
3. Con el comando identificado y validado, se procederá a realizar la petición, dependiendo de cada comando podrá tener más verificaciones (de erros, de argumentos o comprobaciones de que la clase AccionesNAO no tenga solicitado uso exclusivo).

Esta clase será utilizada mayoritariamente por el hilo que se encargará de estar a la espera de comando entrantes (TCPreciver), cuando el hilo reciba un comando utilizará las funciones que proporciona la clase para procesarlo.

Además de los comandos que ya están implementados en el servidor, esta clase permite que se sigan agregando más comandos a necesidad del desarrollador. Para poder añadir nuevos comandos se deberán seguir los siguientes pasos:

1. Añadir el tipo de comando a la lista de DEFINES.
2. Añadir la búsqueda del tipo en el método identificarTipoComando().
3. Añadir en la función ejecutarComando un nuevo “case” con el tipo del comando y la llamar a la función específica que ejecutara el comando pasándole los argumentos.
4. Crear la función específica que ejecutara el nuevo comando

- **TCPserver**

Esta clase heredera de la clase Runnable y será el hilo encargado de ofrecer funcionalidad de conexión al servidor.

La clase se encargará de crear y configurar un socket (punto de conexión entre dos programas vía internet) de tipo TCP en el puerto que se le especifique por parámetro (en nuestro caso 6666).

Como esta clase hereda de Runnable tiene que implementar los métodos que exige la clase, este hilo en particular se quiere tener siempre corriendo, que pase de parado a corriendo y solo termine al cerrar todo el servidor, por ello los métodos que proporcionan la funcionalidad de pausa y parada se sobrescribirán en blanco.

Una vez el hilo sea lanzado, inmediatamente se pondrá a la espera de un cliente, bloqueando la ejecución de ese hilo sin consumir recursos del sistema. Esta acción se realizará en un bucle infinito, si un cliente pide conexión mientras otro está conectado se le rechazará la petición y volverá a la espera, en el caso de que no tengamos cliente se aceptará la conexión y se volverá a la espera de otro cliente.

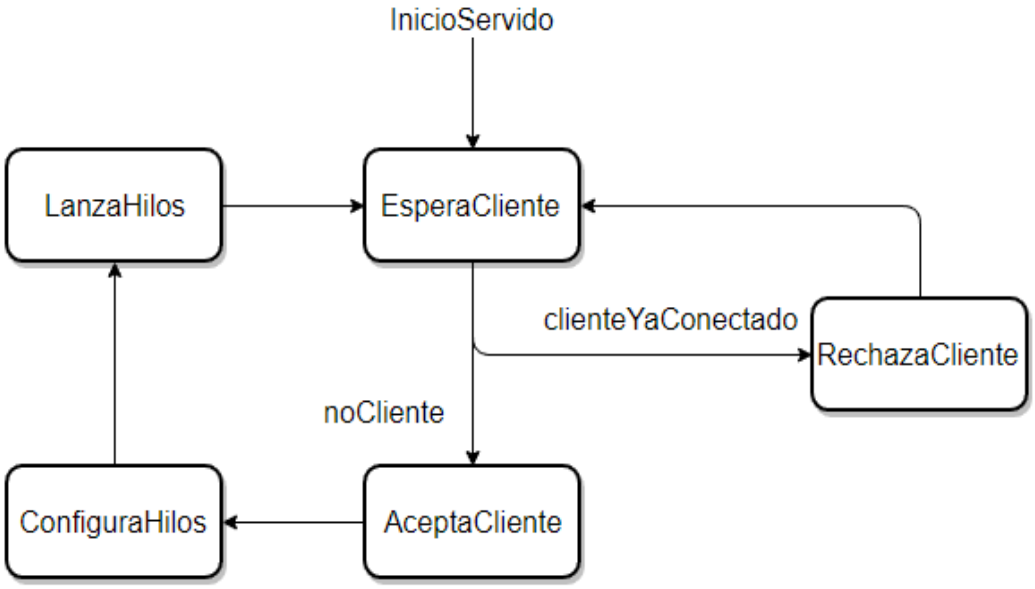

Ilustración 25 Esquema conexión de un cliente

Como vemos en la ilustración 25, cuando un cliente pida la conexión y se acepte, la clase configurara y creara dos hilos nuevos. Uno de ellos destinado a controlar el envió de datos al cliente (TCPreciver) y el otro destinado a la recepción de comandos (TCPsender) por parte del cliente.

Estos dos nuevos hilos no heredaran de Runnable ya que será la propia clase TCPserver la que se encarga de gestionar el estado de los hilos y su creación por lo que no se tiene necesidad de añadirlos al ThreadManager.

- **TCPsender**

Esta clase encapsulara la gestión y control del hilo que se encargara de enviar de forma periódica (cada segundo) información del robot al cliente. Como ya se comentó esta clase no heredara de Runnable. Necesitará que se le pase una referencia a los datos compartidos para que pueda recolectar los datos a enviar.

El comportamiento básico del hilo se puede ver en la ilustración 26, en cada periodo obtendrá los datos a enviar con los cuales formará un mensaje que enviará al cliente utilizando el socket creado por la clase TCPserver.

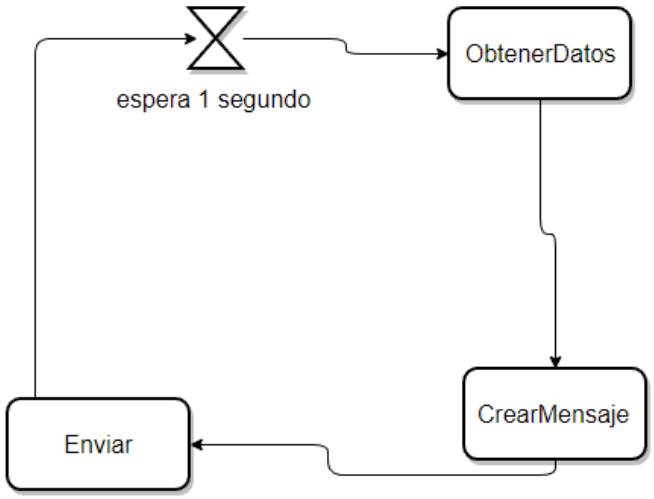

Ilustración 26 Esquema etapas de funcionamiento del hilo TCPsender.

Cada vez que necesite enviar datos, obtendrá todos los datos marcados como enviables dentro de la clase DatosCompartidos y formara una cadena de texto en la que se identificara cada dato con su ID y se indicara su valor (en siguientes apartados se concretara más el formato del mensaje).

Además, ofrecerá un método aparte (mandarRespuesta) que proporcionara un punto por el que se pueden mandar mensajes no periódicos al cliente, este método será usado por la clase Dispatcher para realizar envíos de respuestas (indicaciones de error o confirmaciones) a comandos entrantes.

Este hilo siempre se detendrá cuando se desconecte el cliente o se termine de trabajar con el robot y se apague.

- **TCPreciver**

Esta clase encapsulara la gestión y control del hilo que se encargara de estar a la espera de los posibles comandos enviados por el cliente, procesados por el mismo hilo una vez lleguen. Como ya se comentó esta clase no heredara de Runnable.

Se puede ver en la ilustración 27 un esquema básico de funcionamiento del hilo, donde se refleja como utiliza los métodos proporcionados por la clase Dispatcher comentada anteriormente para validar, identificar y ejecutar el comando.

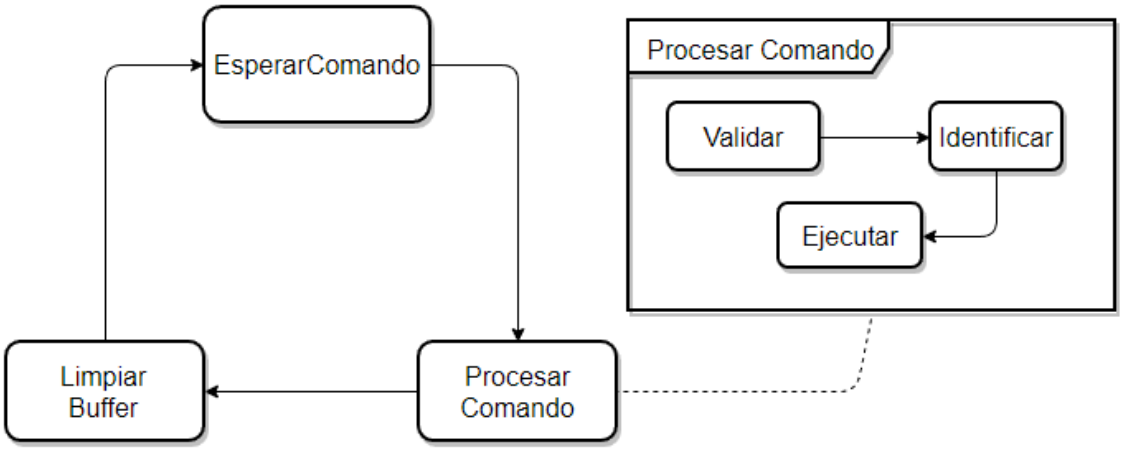

Ilustración 27 Esquema funcionamiento TCPreciver

El buffer de datos se limpiará después de procesar cada comando, con el fin de evitar posibles comandos indeseados, repetidos o acumulaciones, por ejemplo, si desde el cliente se pulsa de forma repetida (sin darse cuenta) una petición de movimiento, el servidor tendría que realizar todas las peticiones hasta quedarse libre de nuevo.

Este hilo siempre se detendrá cuando se desconecte el cliente o se termine de trabajar con el robot y se apague.

- **Simulador**

Esta clase heredara de la ya comentada clase Runnable y encapsulara el hilo que se encarga de realizar la simulación de diabetes en el robot. Al heredar de Runnable tendrá que sobrescribir los métodos de parada, pausa y des-pausa. Como para este hilo se quiere que ofrezca estas funcionalidades, los métodos se sobrescribirán para que se pueda solicitar al simulador que se pause, des-pause o que termine su ejecución.

Todo hilo que necesite avisar a este de realizar una simulación nueva deberá modificar en datos compartidos los datosSim indicando los valores para la nueva simulación.

El simulador podrá estar en tres modos distintos: rápido, lento o normal. Dependiendo el modo los valores de glucosa se actualizarán de forma más rápida o más lenta. Para cambiar el modo de simulación se tendrá que modificar una variable compartida con el ID “MODOSIMU”, la cual la creara la clase Simulador en su constructor. Cada vez que se inicie el hilo simulador este realizara una configuración previa en la que entre otras cosas mirara esta variable para poner un modo u otro.

Cada segundo el hilo mirará (comprobara si se han modificado los valores asociados al hilo en datos compartidos) si necesita realizar una simulación nueva, en caso afirmativo la realizara y actualizara los valores del simulador y datos compartidos. En caso negativo actualizara el valor de la glucosa y mirara si pasaron 5 minutos en cuyo caso realizara una simulación nueva con parámetros estándar. Para ver más en detalle se puede ver en la ilustración 27 el diagrama de estados.

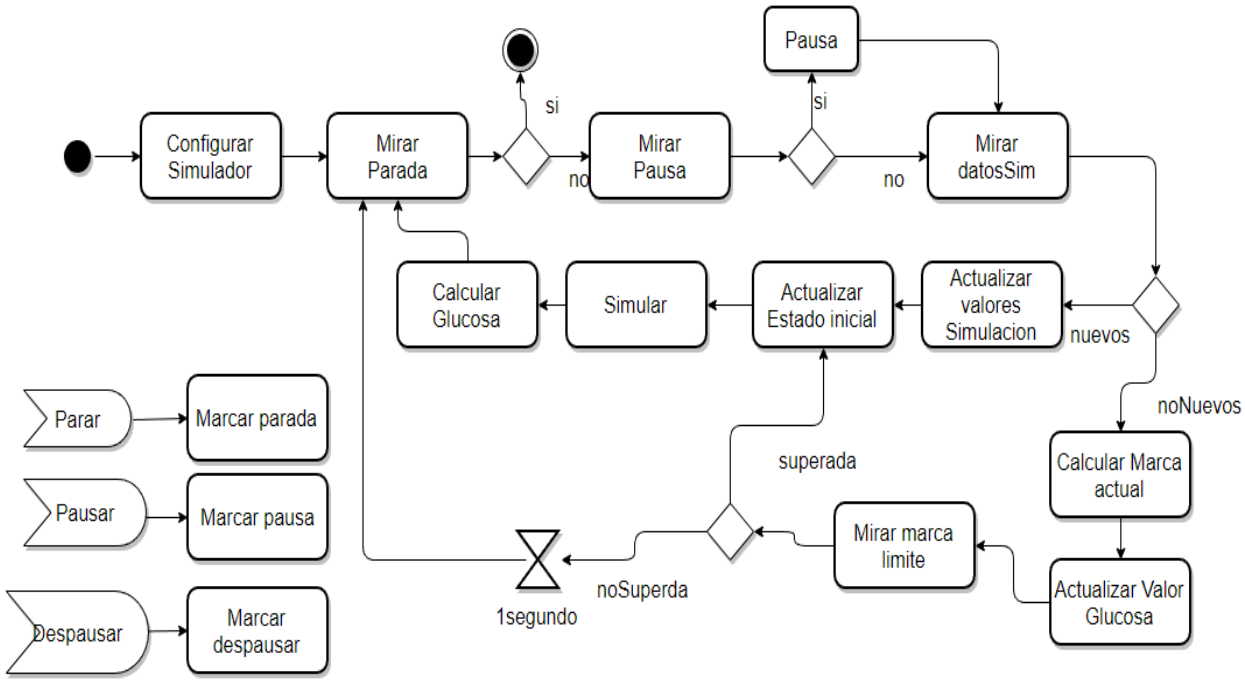

Ilustración 28 Diagrama de estados hilo Simulador

Si la simulación se pone en pausa se actualizará el valor de glucosa a -999 para que todo aquel mirando el valor de la glucosa sepa que el hilo está en pausa, además contara el tiempo que permanece pausado para tenerlo en cuenta a la hora de ver si pasaron los 5 minutos entre simulación. Si el hilo pasa al estado parado se actualizará el valor a -1000 por la misma razón.

- **Interacción**

Esta clase heredara de la clase Runnable y encapsulara el hilo que se encargara de realizar una integración básica con los usuarios del robot. Esta interacción estará orientada a probar la forma que tiene el robot de interactuar con las personas (como una primera toma de contacto), podrá responder a algunos comandos de voz, realizará algunas acciones y tendrá una pequeña interacción dependiendo de los niveles de glucosa.

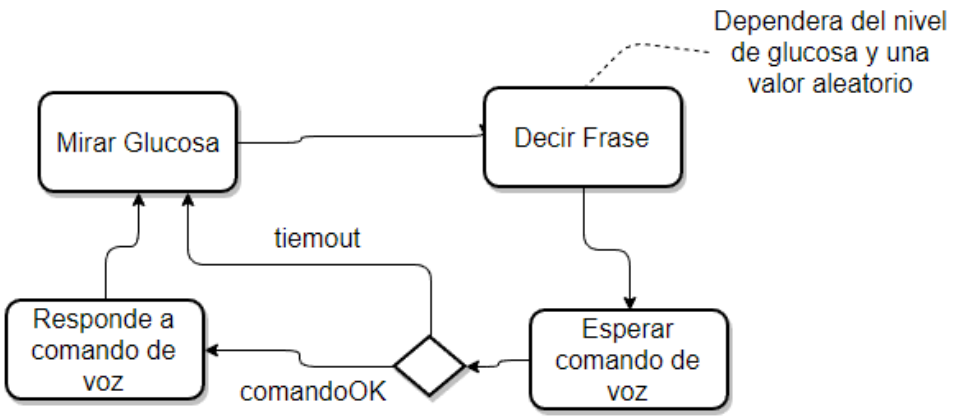

Ilustración 29 Diagrama estados básico hilo Interacción

Podemos ver el comportamiento básico del hilo en la ilustración 29, como se ve el hilo entra en un bucle infinito del cual saldrá cuando se solicite su parada (no se le permitirá al hilo poder pausarse).

El NAO cuando reconozca una palabra o frase, proporciona un nivel de exactitud (cuanto cree el NAO que es la frase correcta), este valor servirá para aceptar o no los comandos de voz, el umbral para aceptar el comando estará definido en un dato compartido con el ID "EXACPALABRA", pudiendo así cambiarlo de forma dinámica durante una interacción.

Como este hilo utilizara la mayoría de funciones que proporciona NAOqi (el habla, reconocimiento de voz, funciones motrices), por lo que será añadido al ThreadManager marcado como excluyente y al inicio del hilo se marcara el flag de uso exclusivo en la clase AccionesNAO.

- **Escenario**

Esta clase también heredera de Runnable y encapsulara el hilo encargado de realizar los dos escenarios principales de interacción con el simulador de diabetes y los usuarios. También utilizara la mayoría de funciones que proporciona NAOqi, por lo que será añadido al ThreadManager marcado como excluyente y al inicio del hilo se marcara el flag de uso exclusivo en la clase AccionesNAO.

La dinámica general del hilo se puede ver en la ilustración 30, el hilo estará en bucle infinito hasta que se le solicite parada, realizando pregustas y acciones dependiendo de la palabra reconocida y el valor de glucosa en ese instante.

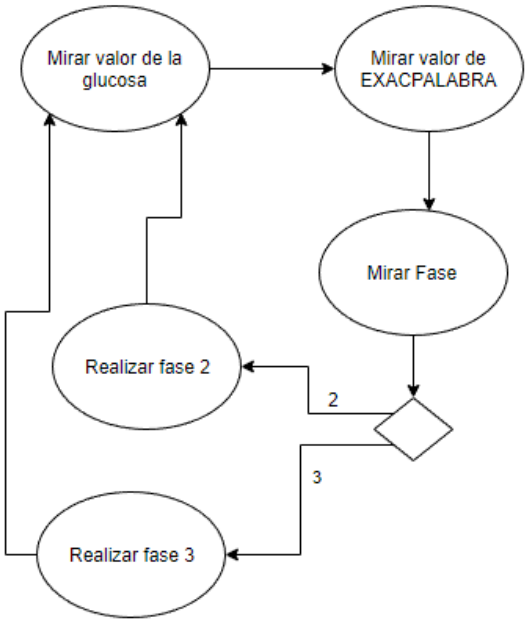

Ilustración 30 Diagrama de funcionamiento general del hilo Escenario

Las diferencias entre la fase 2 y la fase 3 radica sobre todo en los diferentes comandos de voz de los que estará pendiente el robot y las respuesta y acciones con las que responderá ante ellos. Pero la estructura general del hilo, que se puede ver en la ilustración 31, será prácticamente igual.

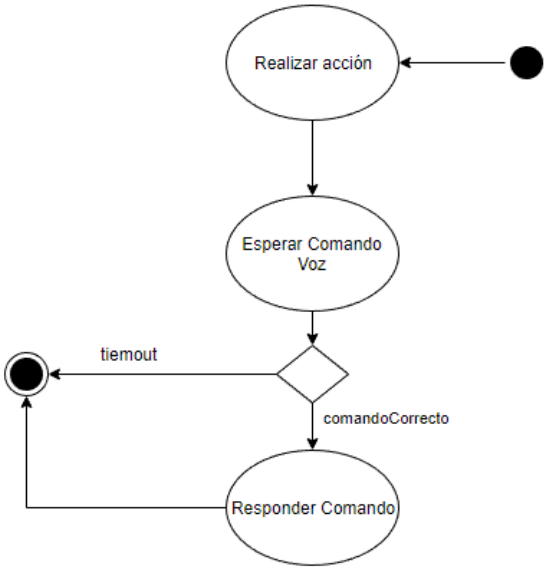

Ilustración 31 Secuencia hilo Escenario que hará para las fases

En la fase 2 se ha implementado (para ver cómo se puede añadir y eliminar datos de forma dinámica al envió periódico) él envió del estado de la fase (un array con información de cómo va la fase). Al inicio de hilo añadirá a datos compartidos un dato con el ID "ESCENARIO", cada vez que entre a ejecutar la fase 2 formará una cadena con la información de estado relevante a la misma y modificara el dato "ESCENARIO", cuando se pare este hilo el dato será eliminado de los datos compartidos.

- **Server**

Esta clase será la entrada del sistema, el módulo local que NAOqi deberá instanciar al encender el robot.

Tendrá que heredar de la clase AL::ALModule (clase proporcionada por NAOqi), necesario para que en el encendido del robot el framework sepa cómo tiene que lanzar el módulo. Instanciara y configurara todas las clases comentadas e iniciara algunos hilos básicos como el TCPserver o el Simulador.

Como esta clase hereda de AL::ALModule necesitara sobrescribir algunos métodos, uno de ellos y el más importante es el método init(), este método será llamado una vez NAOqi instancie el módulo, por lo que tendrá que ser en este método donde se configure la estructura de clases que hemos visto.

Si nuestro modulo proporciona alguna funcionalidad que pueda ser llamada por otro modulo, será en el constructor de esta clase donde se tenga que especificar de forma manual. En nuestro caso tendremos que especificar un método "onSpeechRecognized", que será invocado cada vez que se le pida al robot escuchar alguna palabra y de resultado positivo.

```
functionName("onSpeechRecognized",getName(),"Called by ALMemory when a word is recognized.");
BIND_METHOD(Server::onSpeechRecognized);
```

Ilustración 32 Fragmento de código para indicar a naoqi la función a llamar al reconocer una palabra

4.3. Conexión a internet

Para realizar la conexión vía internet, tenemos dos alternativas entre las que elegir, UDP/IP y TCP/IP. En este caso se va a utilizar el conjunto de protocolos orientados a conexión TCP/IP ya que a diferencia del UDP, en este conjunto de protocolos se asegura la llegada de tramas, así como la llegada ordenada de las mismas, por lo que nos evitamos tener que realizar nosotros esta funcionalidad.

El conjunto de protocolos que componen el dúo TCP/IP brinda de las siguientes características:

- Orientado a conexión: dos ordenadores establecen una conexión para el intercambio de datos. Los dos sistemas se sincronizan con el otro para manejar el flujo de paquetes y adaptarse a la gestión de la red.
- Operación full-dúplex: una conexión TCP es un par de circuitos virtual, cada uno en una dirección, permitiendo el envío y recepción por el mismo canal de conexión.
- Revisión de errores: una técnica de checksum es usada para verificar que los datos en la recepción no están corruptos.
- Acuse de recibo: de uno o más mensajes, el receptor regresa un acuse de recibo indicando que recibió los datos. Si los paquetes no son notificados en un periodo de tiempo establecido, el transmisor reenvía los paquetes.
- Control de flujo: si el transmisor está desbordando el buffer del receptor, el receptor descarta los paquetes.
- Servicio de recuperación de paquetes: el receptor puede pedir la retransmisión de un paquete de datos concretos.

La transmisión de los datos se hará en tres fases:

1. Conexión.
2. Intercambio de datos, con el protocolo especifico por el desarrollador.
3. Desconexión.

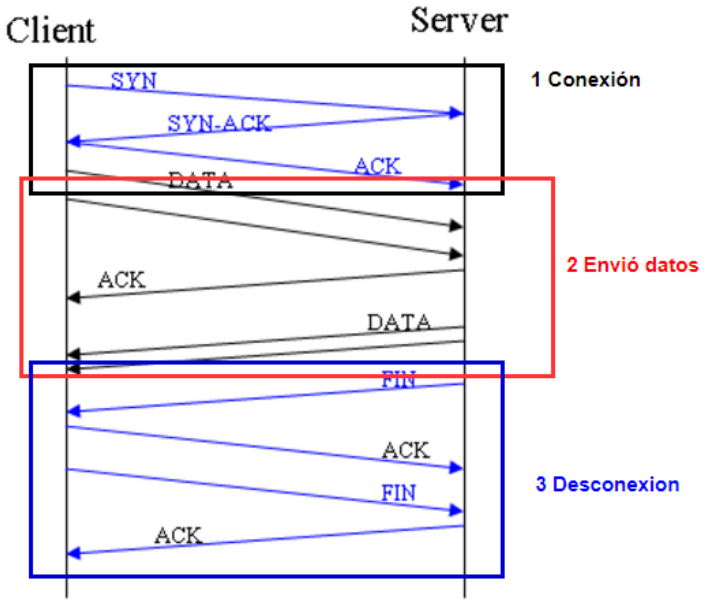

Ilustración 33 Fases comunicación TCP/IP

En nuestro caso el envió de datos se realizará mediante cadenas de texto con formato bien definido y conocido por los dos integrantes de la comunicación. Dependiendo del prefijo del mensaje cada extremo de la comunicación sabrá cómo tiene que manejar el mensaje.

El servidor estará siempre a la espera de comandos provenientes del cliente, a la llegada de un comando lo procesará y responderá si lo necesita. Y el cliente estará siempre escuchando los datos que le envía el servidor, los cuales analizara para saber si son periódicos o puntuales y los tratara como necesite.

4.3.1 Formato de los comandos

Todos los comandos enviados entre los extremos de la comunicación serán cadenas de caracteres, con formato especifico delimitado por una serie de caracteres reservados (que no podrán ser enviados como información), serán:

- '#',
- '~'
- '/'
- ';'
- ':'
- ','

Todos los comandos empezaran por un prefijo de identificación, seguido por los datos dependiendo del tipo de prefijo y terminando por el carácter de fin de comando ';'. Se distinguen tres tipos de comandos intercambiados:

- Enviados de forma periódica por el servidor.
- Enviados como respuestas por el servidor.
- Enviados como peticiones por el cliente.

-Periódicos-

Los mensajes enviados de forma periódica por el servidor tendrán el prefijo "PER#", seguidos por el conjunto de datos identificados de la forma "ID~VALOR" separados por el carácter '/', hasta encontrar el fin de comando ';'.

"PER#ID~VALOR/ ID~VALOR/ ID~VALOR/ ID~VALOR/ ID~VALOR;"

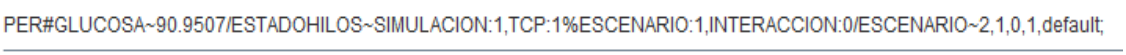

Ilustración 34 Muestra datos periódicos

-Respuesta-

EL prefijo para los datos de respuesta a comandos será "RES#" seguido del mensaje de respuesta que toque, enviados siempre después de recibir una petición por parte del cliente.

"RES#respuesta;"

```
RES#Error thread utilizando robot;
```

```
RES#Modo simulacion cambiado;
```

Ilustración 35 Ejemplo mensajes de respuesta

-Comandos cliente-

Lo comandos tendrán el prefijo “NAO#” seguidos del tipo “TIPO:” de comando los argumentos que acompañan al comando (separados por ‘,’ en caso de ser mas de un argumento), acabando con ‘;’.

“NAO#TIPO:arg1,arg2,arg3;”

```
String aux = "NAO#SIMULAR:0,0,true,0,50,60;";

String aux = "NAO#MODOSIMU:1;";

String aux = "NAO#THREAD:"+hilo.getNombre()+",1;";
```

Ilustración 36 Ejemplos comandos enviados por el cliente

Los tipos de comandos implementados son:

- DECIR: que tomara como argumento la frase o palabra que se desee que el robot reproduzca.
- SIMULAR: tomara como argumentos los parámetros para nuestra simulación.
- MOVER: tendrá un único argumento que será el nombre de la acción que se desea realizar.
- THREAD: este comando servirá para el control de los hilos del sistema, tomando como argumentos el ID del hilo que se quiere modificar y el estado (será un numero) al que se le quiere pasar.
- LED: este comando cambiara el color de los leds situados en los ojos, tomara el color que se quiere modificar y el estado (apagado o encendido).
- MODOSIMU: tomara como argumento el modo de simulación que sea desea para nuestra simulación (número del 1 al 3).
- EXACPALABRA: tomara como argumento el nivel de exactitud a la hora de admitir una palabra.

5. Cliente NAO

En esta sección se comentarán las funciones del cliente desarrollado para comunicarse con el servidor anteriormente comentado. También se hablará sobre las clases que componente al cliente y del diseño de las interfaces graficas de usuario.

5.1. Funciones del cliente

El cliente se desarrollará con el fin de que sea capaz de conectarse al servidor (utilizando una conexión TCP/IP) para recibir los datos que tenga que mandar el servidor y mostrarlos de una forma gráfica de forma que facilite la interpretación al usuario del programa y de facilidades a la hora de presentar datos ante un grupo de gente.

Además, ofrecerá una parte destinada al control, en la que se podrán realizar distintas peticiones al servidor utilizando los comandos que ofrece para poder modificar el comportamiento durante una sesión de uso del robot.

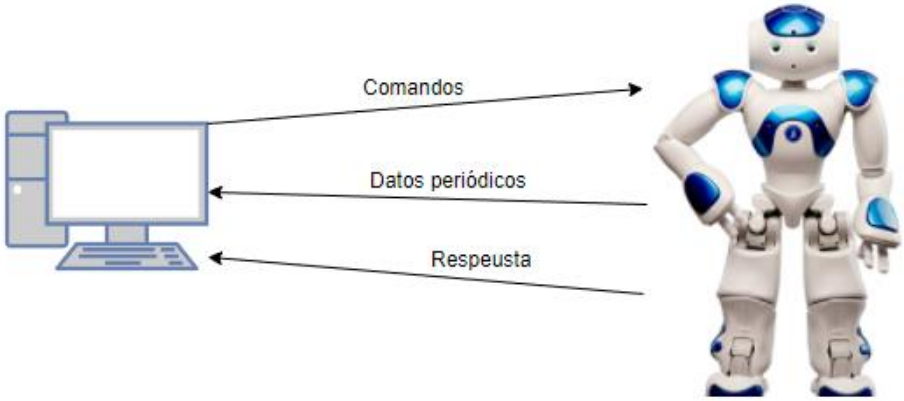

Ilustración 37 Esquema intercambio información cliente servidor

5.2. Diseño de interfaz

Se busca que el programa cliente ofrezca una interfaz de usuario donde se tengan dos ventanas separadas, una en la que se englobara el control del robot y visualización de la información de estado y la otra que se encargara de la visualización de niveles de glucosa y toda la información que no sea de control. Además, se añadirán ventanas para funciones específicas en caso de necesitarse.

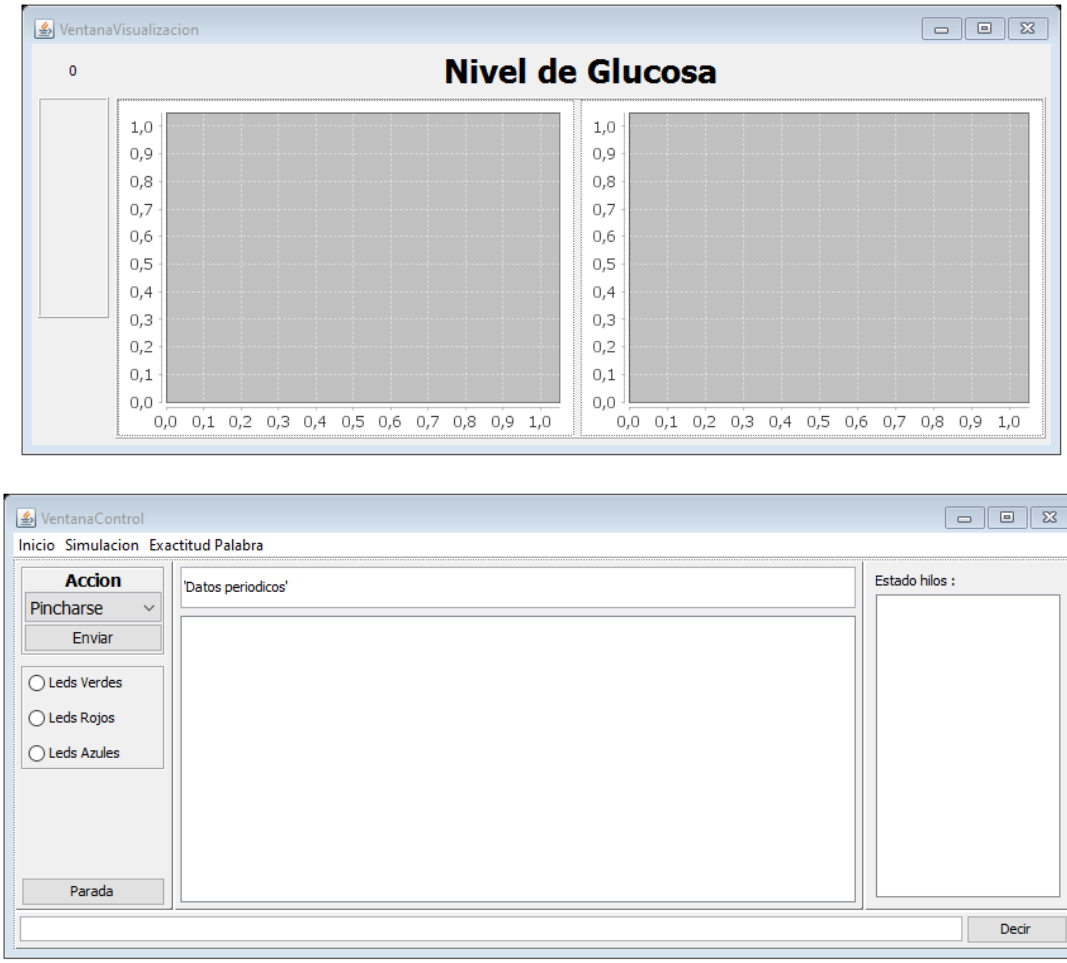

Ilustración 38 Ventanas principales del cliente

La razón para separar estas dos ventanas es poder mover la ventana de visualización a una segunda pantalla o incluso proyector mientras que la de control queda solo a los ojos de la persona que este gestionando el robot mientras que la de visualización podría verla el resto de personas.

En siguientes apartados se vera de forma más detalla cada ventana que conforma la aplicación.

5.3. Clases del cliente

5.3.1 Disposición

El cliente estará compuesto por las siguientes clases, algunas de las cuales tendrán asociada la gestión de la interfaz gráfica de usuario:

- MainWindow
- HiloServidor
- ClienteTCP
- VentanaControl
- VentanaVisualizacion
- Manager
- VentanaEscenario
- VentanaSimulacion

Relacionadas entre si de manera que se pueda realizar el intercambio rápido de información entre las clases controladoras y las que se encargan de manejar las vistas (se puede ver en el diagrama de clases de la ilustración 39).

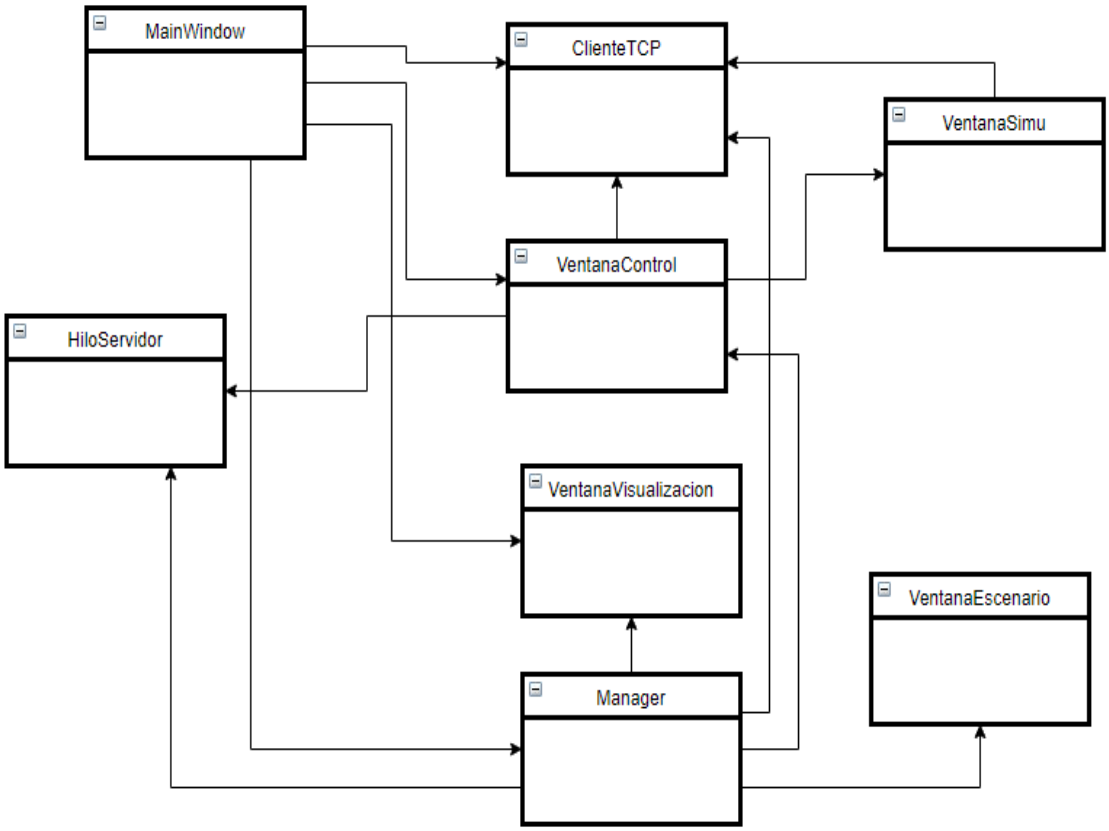

Ilustración 39 Diagrama de clases cliente

5.3.2 Explicación de las clases

- **ClienteTCP**

En esta clase se encargará de encapsular las operaciones para la comunicación con el servidor. Ofreciendo métodos para realizar la conexión e intercambio de información. Solo se necesitará crear una instancia de la misma.

Permitiéndonos realizar una conexión a la IP y puerto que se le especifiquen sin tener que fijarlo en el código (por si el robot cambia de red no tener que modificar el código fuente). Además, ofrecerá un método que nos permitirá validar el formato de la IP especificada.

- **HiloServidor**

Esta clase será un contenedor de datos que nos servirá para guardar la información asociada (nombre, estado y excluyente) de cada hilo que se encuentra en el sistema.

- **VentanaVisualizacion**

La clase encapsulara el funcionamiento asociado a la interfaz de usuario destinada a la visualización de niveles de glucosa, utilizando diferentes métodos de representación para el dato.

En esta clase se tendrá un par de gráficos a los cuales se les irán añadiendo los valores de la glucosa según lleguen del servidor, este grafico mantendrá la información de los valores de glucosa anteriores. También se tendrán un par de componentes destinados a mostrar la glucosa actual en diferentes formatos.

Si el usuario cierra esta ventana, el programa entero se cerrará y se desconectará del servidor.

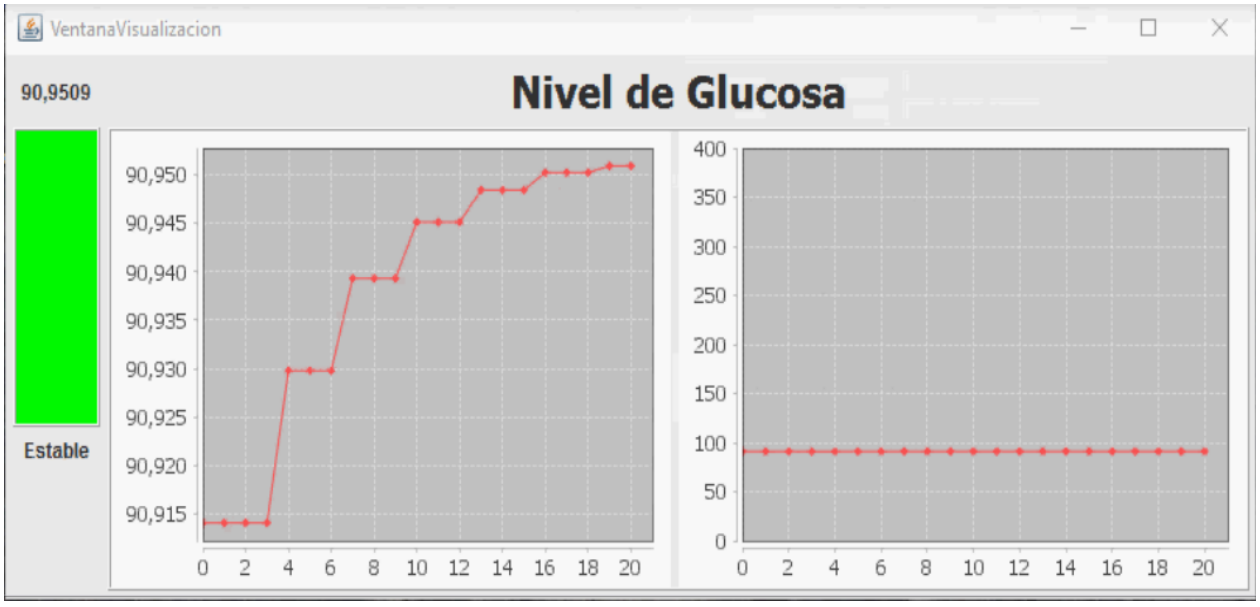

Ilustración 40 Ventana Visualización en funcionamiento

Como vemos en la ilustración 40, los gráficos abarcaran la mayor parte del espacio dentro de la ventana, son dos gráficos separados, pero muestran el mismo valor de glucosa solo que con diferentes escalas de visualización. El de la izquierda nos muestra la evolución temporal de forma detalla (con auto escalado), mientras que el de la derecha con una escala mayor nos muestra el comportamiento visto de forma global, consiguiendo con esto ver como la glucosa fluctúa en cada instante, aunque los cambios en ella no sean preocupantes a la hora de ver si se llegan o no a superar los niveles recomendados.

Además de los gráficos a la izquierda de la ventana tendremos 3 formas distintas de representar el valor de la glucosa, uno en modo texto indicando si el nivel de la glucosa es peligroso o no, otro en formato color indicando también si el valor de la glucosa es peligroso o no y el ultimo mostrando el valor numérico.

La ventana tendrá la capacidad de redimensionarse si el usuario lo desea, cada componente se redimensionará de manera proporcional.

- **VentanaSimulacion**

Esta clase encapsulará el comportamiento de una pequeña ventana que nos permitirá seleccionar los valores que queramos para solicitar una nueva simulación al servidor. Como esta ventana realizara el envió del comando tendrá acceso a la clase ClienteTCP.

En la ilustración 41 se puede ver el aspecto que tiene la interfaz, en este caso la interfaz no será redimensionable.

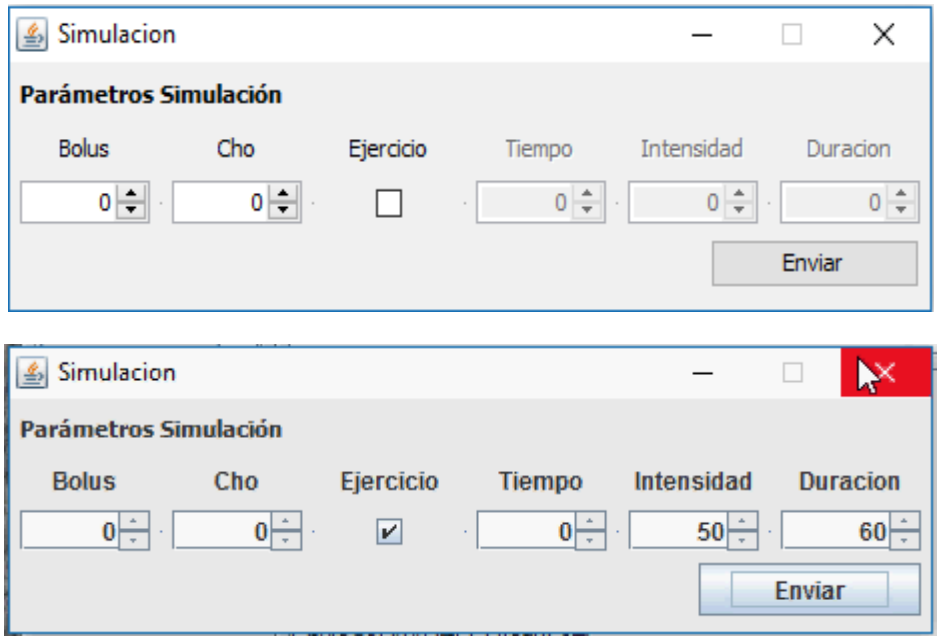

Ilustración 41 Aspecto ventana simulación

El envió del comando se realizará una vez se pulse el botón enviar, cada parámetro dispone de su propio componente para introducir el valor deseado, siendo el parámetro "Ejercicio" un checkbox que al pulsarlo nos permitirá modificar los 3 últimos parámetros.

Esta ventana cuando se cierre mantendrá los valores de cada parámetro para la siguiente vez que se quiera realzar la simulación.

- **VentanaEscenario**

Encapsulara la ventana que se encargara de mostrar la información que mandara el servidor siempre que este activo el hilo Escenario y se esté ejecutando la fase 2.

Esta ventana se abrirá de forma automática cuando al recibir la información periódica del servidor se detecte el dato con ID “ESCENARIO”. La ventana tendrá el aspecto que se ve en la ilustración 42.

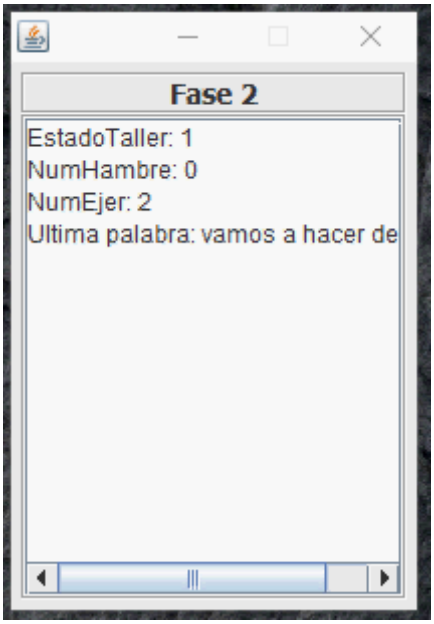

Ilustración 42 Aspecto ventana Escenario

Como se puede ver la ventana tiene un aspecto muy simple, compuesta por un label superior indicando la fase y un cuadro de texto que nos mostrara la información recibida. Esta ventana es un ejemplo de cómo tratar datos que nos mande el servidor de forma eventual.

- **VentanaControl**

En esta clase se encapsulará la gestión y control de la interfaz gráfica de usuario destinada a proporcionar los controles de usuario para modificar el estado de nuestro robot. Además, contara con algunos elementos de visualización para los datos de control enviados por el servidor, como los hilos presentes en el robot, las respuestas del servidor a los comandos enviados y los datos periódicos.

Como pasa con la ventana de visualización principal, si se cierra esta ventana el programa finalizará y se cerrará la conexión, también será totalmente redimensionable.

La clase contara con una serie de métodos que permitirán a otras clases añadir información y actualizar la vista, así como los métodos que se encargaran de controlar los eventos producidos en la misma (pulsación de botones, clic derecho, etc).

Se puede ver la interfaz gráfica de usuario en funcionamiento en la ilustración 43 para poder ver todo lo que ofrece.

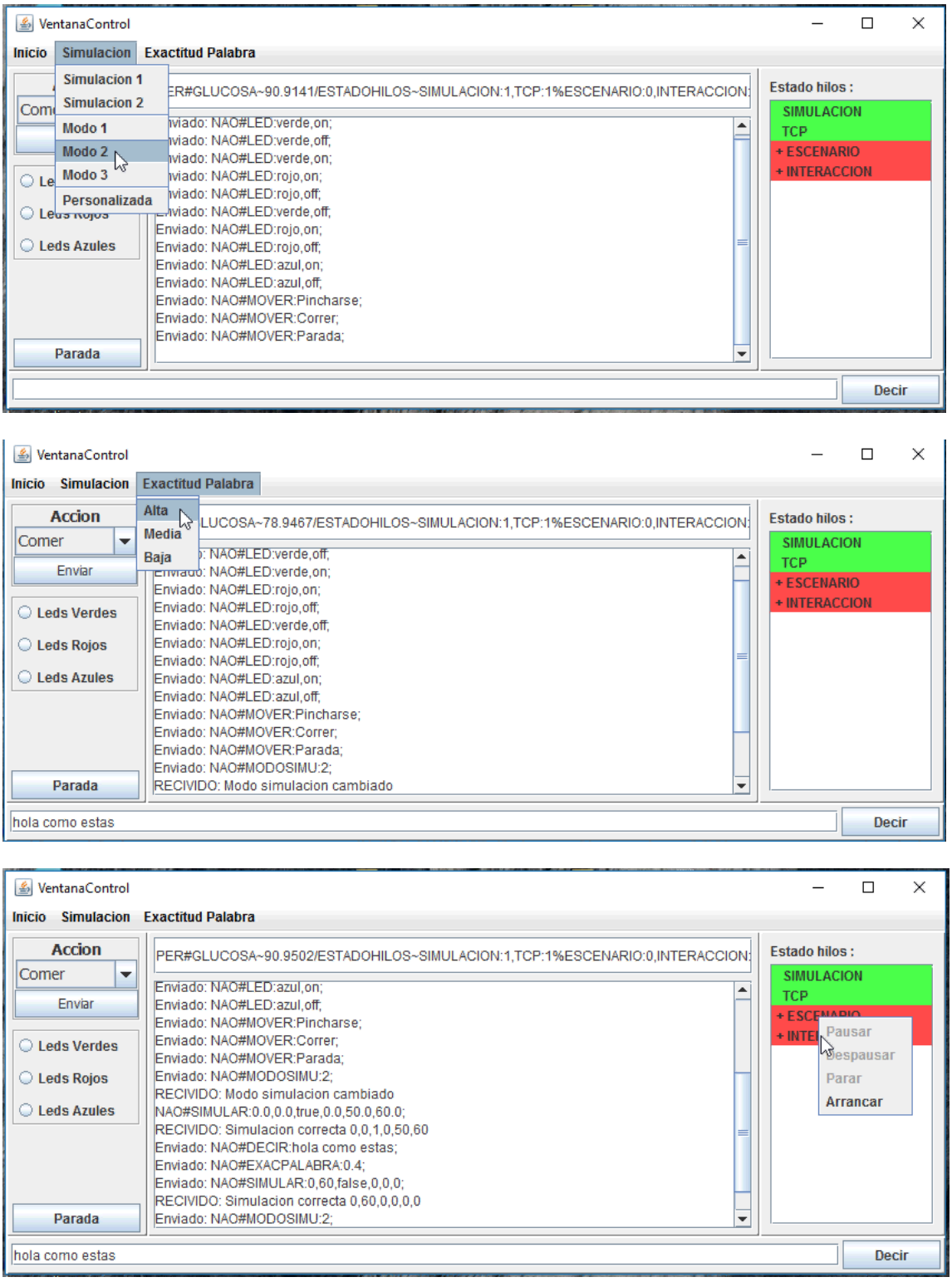

Ilustración 43 Ventana Control en funcionamiento

En el centro de la ventana tendremos dos cuados de texto, el mayor destinado a mostrar las peticiones de comandos y respuesta del servidor (quedando de forma permanente) y el superior a mostrar los datos recibidos por el servidor en el mismo instante.

A la derecha tenemos otro cuadro que permitirá visualizar y gestionar los hilos que se encuentran en el sistema. El estado del hilo se marcará mediante el uso de color, verde

corriendo, rojo parado y amarillo pausado. Para poder cambiar el estado tendremos que hacer clic derecho sobre el hilo que queramos y se abrirá un panel que nos permitirá mandar el comando para modificar el hilo.

Debajo de la ventana vemos una barra de texto y un botón, que nos permitirán introducir la frase o palabra que queramos que el robot reproduzca una vez pulsemos el botón.

A la izquierda tenemos un panel con diferentes opciones, como modificar el color de los leds o pedir que el robot realiza una de sus acciones o movimiento prefijados. Además del botón parada, el cual llevara al robot a una posición de seguridad.

Por último, tenemos el menú de opciones, en el cual se tienen dos submenús que nos permitirán, realizar cambios en el simulador ofreciendo la posibilidad de cambio de modo, simulaciones predefinidas y abrir la ventanaSimulación para realizar una simulación a medida. También tendremos el submenú para seleccionar el umbral de exactitud para que NAO reconozca una palabra.

- **Manager**

Esta clase encapsulara al hilo encargado de estar a la esperar de los datos enviados por el servidor (tanto los periódicos como las respuestas), de procesarlos y efectuar las acciones necesarias en cada caso. Se puede ver el comportamiento general de hilo en el diagrama mostrado en la ilustración 44.

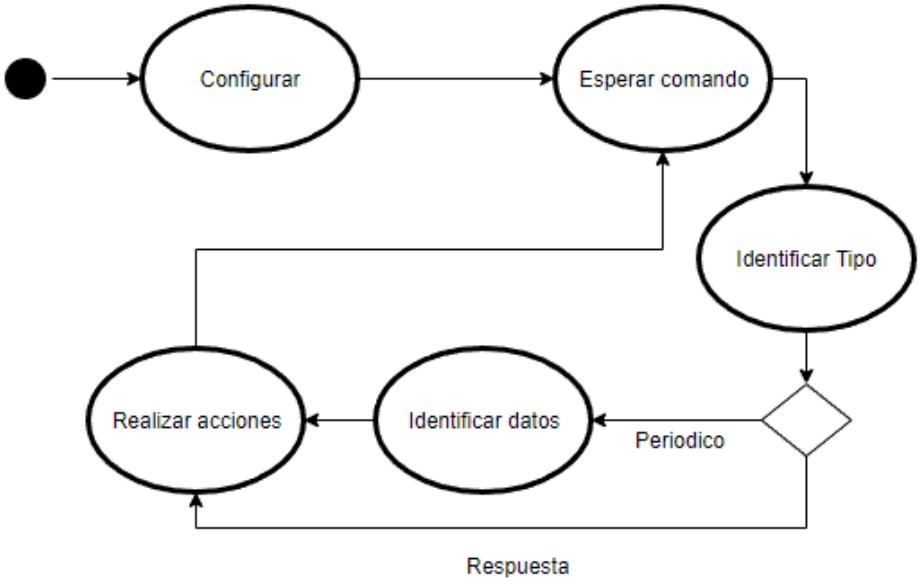

Ilustración 44 Diagrama estados clase Manager

En hilo será el encargado de identificar que tipo de comando ha llegado al cliente, en función de si el dato tiene el prefijo “PER” o “RES” realizaría unas acciones u otras. En caso de que los datos sean de tipo “PER” como el contenido no es fijo y puede variar de forma dinámica realizaría otra identificación para ver que datos han llegado.

Con los comandos identificados, así como los datos reconocidos, este hilo actualizara todas las vistas que necesiten ser actualizadas (ventanas de control, visualización y escenario).

- **MainWindow**

Esta clase es el punto de entrada a nuestro programa, encapsulara el comportamiento de la primera ventana que vera el usuario (se puede ver en la ilustración 45) e instanciara y mostrara el resto de ventanas.

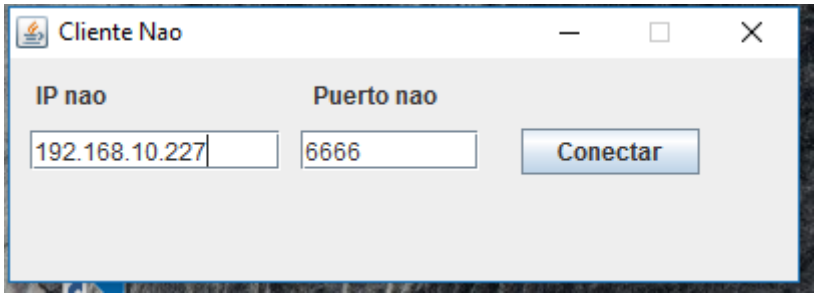

Ilustración 45 Ventana de conexión al NAO

Ofrece dos cuadros de texto para indicar la IP y puerto donde se encuentra esperando conexiones el servidor dentro del NAO. Estos cuadros no aparecerán el blanco, si no que tendrán una IP y un puerto por defecto. Para solicitar la conexión contamos con el botón “Conectar”.

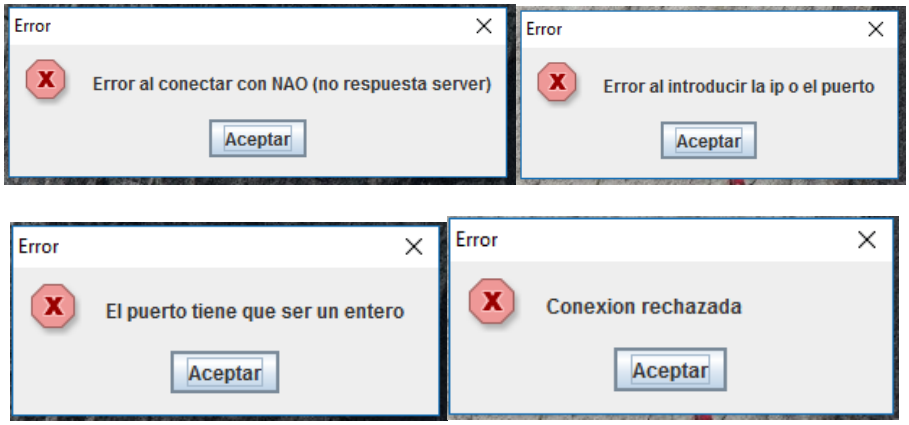

Ilustración 46 Posibles errores a la hora de realizar la conexión

Una vez que se especifique una IP y puerto correcto y se pulse el botón, se comprobará que los datos sean correctos en cuyo caso se conectara al servidor, la ventana se ocultara y mostrara las ventanas de control y visualización.

6. Conclusiones

6.1. Conclusiones

Se ha conseguido implementar una arquitectura software que proporciona herramientas y facilidades para el desarrollo y futuras ampliaciones del programa servidor, como hemos visto en el apartado 4 de la memoria, más en detalle se ha conseguido:

- Permitir la creación y gestión de hilos de ejecución mediante las clases Runnable y ThreadManager implementadas en el servidor.
- Se ofrece un sistema de intercambio de información para todos los hilos del sistema, facilitado con la creación de la clase DatosCompartidos.
- Crear una comunicación vía internet para el intercambio de informacion con un cliente, proporcionada por las clases TCPServer, TcpReciver y TcpSender.
- Se ha encapsulado el acceso al framework NAOqi para simplificar su uso con el servidor, mediante la clase AccionesNAO.
- Se han implementado el simulador de diabetes y los escenarios de interacción de trabajos anteriores utilizando las herramientas que ofrece la arquitectura en las clases Simulador, Escenario e Interaccion.

También se ha logrado desarrollar una aplicación Java que actúa como cliente de nuestro servidor, vista en el apartado 5 de la memoria. La cual proporciona una conexión con el robot mediante las clases Manager y ClienteTCP y una interfaz de control y visualización de toda la información enviada por el robot como se puede ver en las clases VentanaVisualizacion y VentanaControl.

A medida que se ha ido realizando el desarrollo con el framework del robot, se ha podido apreciar las muchas capacidades (como la detección automática de caídas) que ofrece el software NAOqi a la hora de realizar un módulo destinado a la ejecución en el robot. Permitiendo llegar a modificar cualquier aspecto del robot, pero siempre a nuestra cuenta y riesgo.

Uno de los puntos en contra de tener un framework tan potente y con tantas capacidades, es lo difícil que puede resultar encontrar las causas a algunos errores puntuales para poder corregirlos, si a esto le sumamos las dificultades de depuración (debido a que el módulo local se ejecuta en el robot y es el propio robot quien lo ejecuta no nosotros sin poder disponer de una consola convencional) se puede llegar a tardar bastante en encontrar el origen del error.

6.2. Líneas futuras

A continuación, se comentarán algunas mejoras con las que seguir mejorando e implementando las capacidades del servidor y el cliente:

- Separar la clase AccionesNAO en subclases, en las que tengamos una clase solo para las primitivas básicas de acceso a funciones del robot y una o varias clases destinadas a acciones o escenarios más preparados donde entren en conjunto varias acciones básicas.
- Aumentar los comandos que acepta el servidor, con comandos para cambiar el volumen del habla, comandos para pedir información como la batería, etc.
- Incrementar en el hilo Interacción las capacidades para interactuar con las personas.
- En el cliente crear una clase que haga de intermediario (controlador) entre las ventanas y el ClienteTCP que se ocupe de gestionar el formato de los comandos.

- Tanto en el servidor como en el cliente seguir implementando más comprobaciones de errores.

7. Referencias y bibliografía

1. “Honda humanoid robots development” Masato Hirose and Kenichi Ogawa http://lars.mec.ua.pt/public/LAR%20Projects/Humanoid/2011_RicardoGodinho/Pesquisa%20Disserta%C3%A7%C3%A3o/11.full.pdf
2. “Humanoid Robots” https://www.engineersgarage.com/articles/humanoid-robots?page=1
3. “All Robots Are Not Created Equal: The Design and Perception of Humanoid Robot Heads” Carl F. DiSalvo, Francine Gemperle, Jodi Forlizzi, Sara Kiesler http://www.cs.cmu.edu/~kiesler/anthropomorphism-org/pdf/DIS-Disalvo.pdf
4. “Is imitation learning the route to humanoid robots?” Stefan Schaal http://www.bcp.psych.ualberta.ca/~mike/Pearl_Street/PSYCO354/pdfstuff/Readings/Schaal1.pdf
5. GNI/LINUX https://www.gnu.org/gnu/linux-and-gnu.en.html
6. List of Linux distributions https://en.wikipedia.org/wiki/List_of_Linux_distributions
7. Sistemas Empotrados http://personales.upv.es/rmartin/elinux/tema1/sisempotrados.html
8. Top 4 Open Source Boards: Raspberry Pi, Arduino, BeagleBoard, and Intel. https://www.ema-eda.com/about/blog/top-4-open-source-boards-raspberry-pi-arduino-beagleboard-and-intel
9. Albelbaran Robotics https://www.ald.softbankrobotics.com/en
10. Robocup http://www.robocup.org/
11. Documentación NAO http://doc.aldebaran.com/2-1/family/robots/joints_robot.html
12. Documentación NAO http://doc.aldebaran.com/2-1/family/robots/index_robots.html#all-robots
13. Biblioteca Boost http://www.boost.org/doc/libs/1_55_0/
14. Biblioteca ALGLIB http://www.alglib.net/
15. Biblioteca JFreechart http://www.jfree.org/jfreechart/