Capítulo 2

Arquitectura General del Sistema

2. ARQUITECTURA GENERAL DEL SISTEMA

Tanto para de trabajar con un único robot, como para realizar determinados tipos de tareas en la que intervengan varios, es necesario definir una nueva arquitectura común entre todos ellos que permita, de una forma más efectiva, satisfacer dos aspectos fundamentales:

- *En un único robot*: debe existir una organización en la arquitectura y en el código que facilite al programador, de una forma sistemática y sencilla, la implementación de nuevos módulos que doten al robot de una mayor funcionalidad.

- *En un sistema de robots*: debe existir un nivel de abstracción común a todos ellos que permita a un usuario tratarlos de la misma forma, independientemente de las características de cada uno. A su vez, debe facilitar la adición de nuevos robots al sistema, permitiendo que se produzca una cooperación entre los mismos de una forma más eficiente.

En este capítulo se hace una descripción general de la arquitectura software antes mencionada. Para ello, la información que aquí se expone ha sido recopilada del documento creado por Antidio Viguria e Iván Maza *"Arquitectura para múltiples robots heterogéneos"*, el cual se recomienda leer para una información más detallada.

En lo sucesivo, se estudiará la arquitectura del sistema completo, incidiendo en la forma en la que se establecen las comunicaciones, y se realizará también una descripción de la arquitectura desde el punto de vista del robot, profundizando en las distintas capas y módulos que la conforman, así como las distintas interfaces que los comunican entre si.

De esta forma, la arquitectura del sistema completo está formada por tres partes fundamentales:

1. **Control Center**: proporciona una visión completa de los distintos robots bajo una misma interfaz en la que todos se tratan de la misma forma, independientemente de las características individuales de cada uno.

2. **Time Server**: proporciona una base de tiempo común a todos los robots, que permita la sincronización de los mismos.

3. **Arquitectura de los robots**: está formada por varias capas, dependiendo de la cercanía al hardware del robot, las cuales están formadas por varios módulos que, a su vez, contienen los hilos necesarios para llevar a cabo su tarea. Los robots se distinguen entre sí mediante el uso de identificadores, cuya numeración va del 1 en adelante, estando el 0 asignado al *Control Center*.

Cada robot se comunica con el *Control Center*, con el *Time Server* y con el resto de robots mediante la *General Robot Interface* (GRI), como se puede apreciar en la siguiente figura:

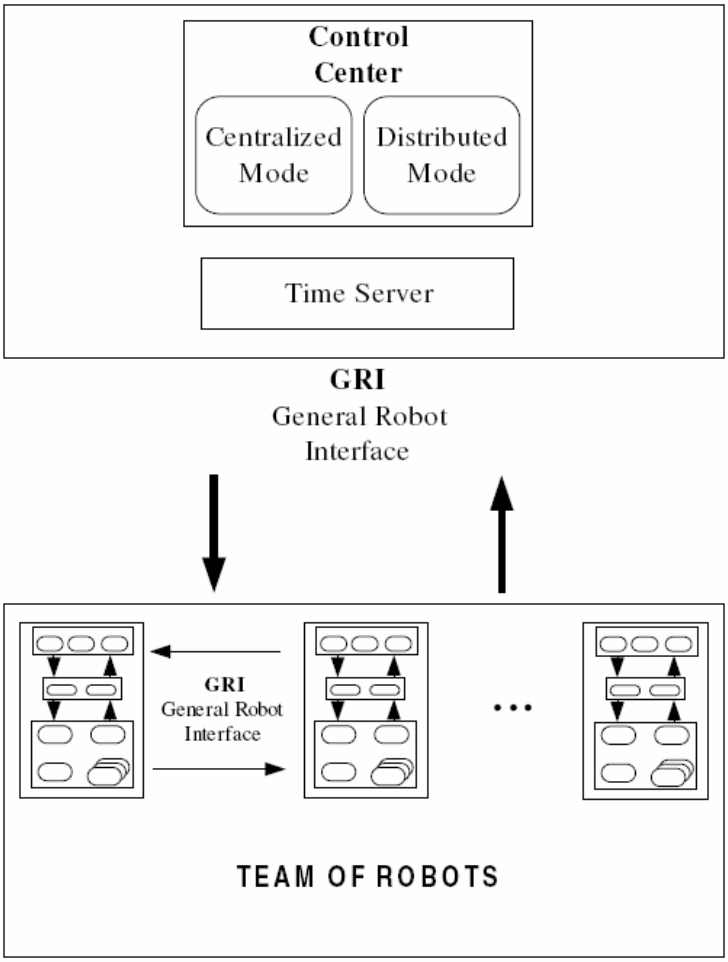

Figura 2-1: Arquitectura del sistema

Una vez visto los elementos que componen esta arquitectura, conviene introducir los cimientos que hacen posible su implementación. Para ello se debe mencionar dos partes fundamentales, entidades y comunicaciones, que se definen a continuación:

- **Entidades:** pueden estar formadas por una o varias capas, las cuales a su vez pueden contener uno o varios módulos, de forma que tanto una entidad, como una capa o un módulo pueden ser procesos independientes, todos ellos con una arquitectura estándar.

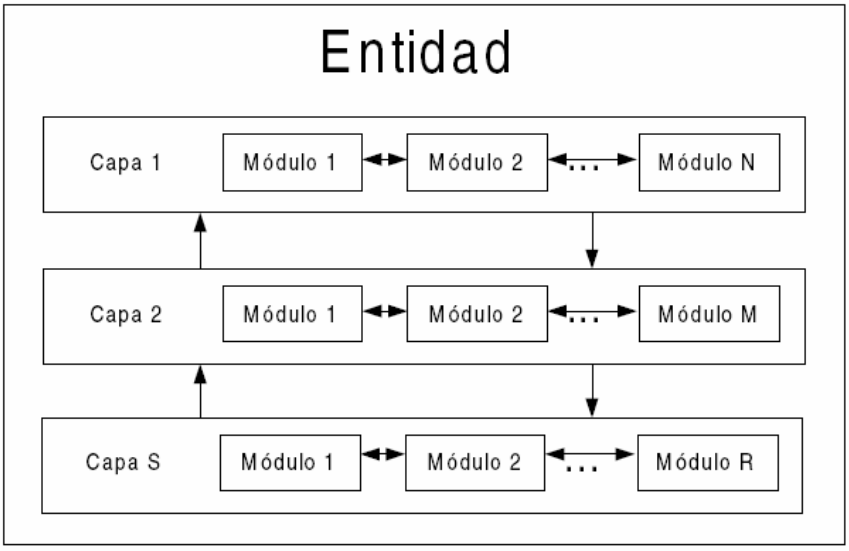

Figura 2-2: Esquema de una entidad

A su vez, cada proceso estará formado por un hilo de comunicaciones y por uno o varios hilos de procesamiento, de forma que cada hilo de procesamiento se comunicará con el hilo de comunicaciones y con el resto de hilos de procesamiento (en caso de que proceda) a través de la sección crítica.

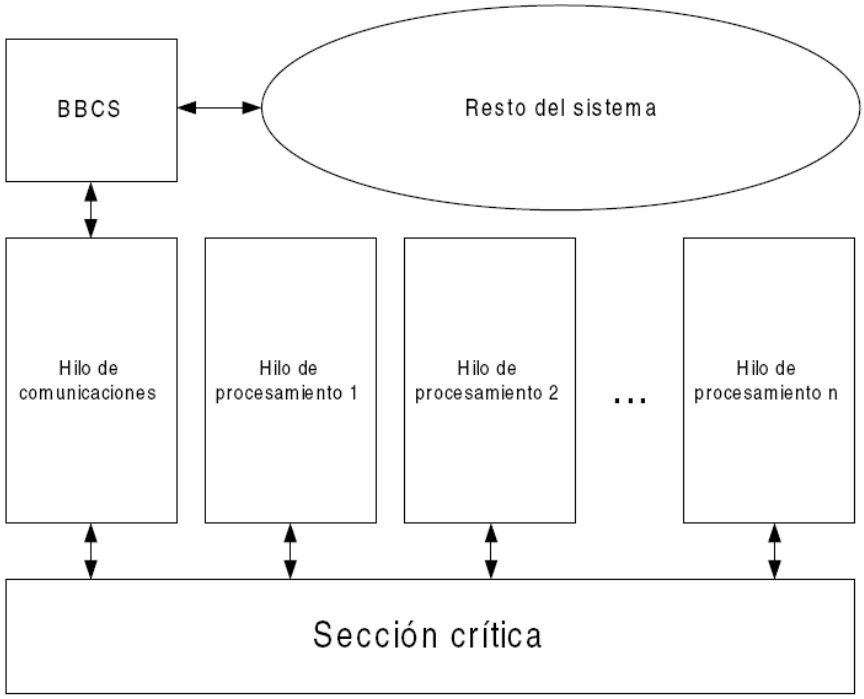

Figura 2-3: Esquema de un proceso estándar

Por lo tanto, el programa principal *main* de un proceso *XXX*, se encargará de levantar tanto el hilo de comunicaciones como los de procesamiento:

1. Hilo de comunicaciones: se encarga de las comunicaciones con el resto de procesos dentro de la misma capa o de capas adyacentes. Actualmente, estas comunicaciones se implementan utilizando el protocolo de comunicaciones BBCS (*Black Board Communication System*), aunque éste se podría cambiar fácilmente (cambiando la clase correspondiente) al ser independiente del resto del sistema. Este hilo está compuesto por los siguientes archivos:

    - *XXXcommunication.cpp* y *XXXcommunication.h*: implementan la clase que se encarga de las comunicaciones con el resto del sistema.

    - *XXXcommunicationthread.cpp*: implementa la ejecución del hilo de comunicaciones.

2. Hilos de procesamiento: se encargan de implementar el proceso en sí, comunicándose con el resto de hilos mediante una sección crítica que está implementada dentro de una clase (*XXXcriticalsection.cpp* y *XXXcriticalsection.h*). Al igual que en el hilo de comunicaciones, existe un fichero *XXXthread.cpp* donde se implementa el ciclo de ejecución del hilo, y una clase (*XXX.cpp* y *XXX.h*) donde se implementan las funciones necesarias para dicha ejecución.

Para una mayor información acerca del protocolo de comunicaciones BBCS que aquí se emplea se recomienda la lectura del documento basado en el proyecto *COMETS* "*Functional Specification of the Real-Time Communications API: Real-time coordination and control of multiple heterogeneous unmanned aerial vehicles*".

- **Comunicaciones:** las entidades poseen un proceso independiente o hilo, denominado *Relay Node*, los cuales se conectan entre sí permitiendo establecer las comunicaciones entre ellas mediante el uso compartido de los slots de datos de la GRI. A su vez, todos los módulos pertenecientes a una misma entidad se conectan al *Relay Node* para poder comunicarse entre ellos mediante el uso compartido de los slots de datos de las interfaces internas.

  La conexión de todos los *Relay Nodes* del sistema entre sí (conexión en malla), permite hacer al sistema más tolerante a fallos, ya que permite al *Relay Node* modificar el enrutamiento de los datos en caso de que se produzca un fallo en un camino. Sin embargo, por simplificación, la conexión de los *Relay Nodes* pertenecientes a los módulos internos de una entidad tiene una topología en estrella.

  La principal ventaja de este sistema de comunicaciones es que en el momento en que se inserta un módulo nuevo en la entidad, lo único que hay que hacer es conectarlo al *Relay Node* e incluir los slots de dicho módulo, sin

tener que tocar las comunicaciones del resto de módulos internos de la entidad. Por lo tanto, se han programado los *Relay Nodes* de forma que crean automáticamente sus propios ficheros de conexiones, a partir de un fichero general de configuración de la red (*./robot_architecture/relay_node/ NetConnections.conf*). En este fichero sólo hay que especificar las direcciones IP de cada entidad y, a partir de esta información, se crean las conexiones en malla de los *Relay Nodes* y las conexiones con los módulos internos.

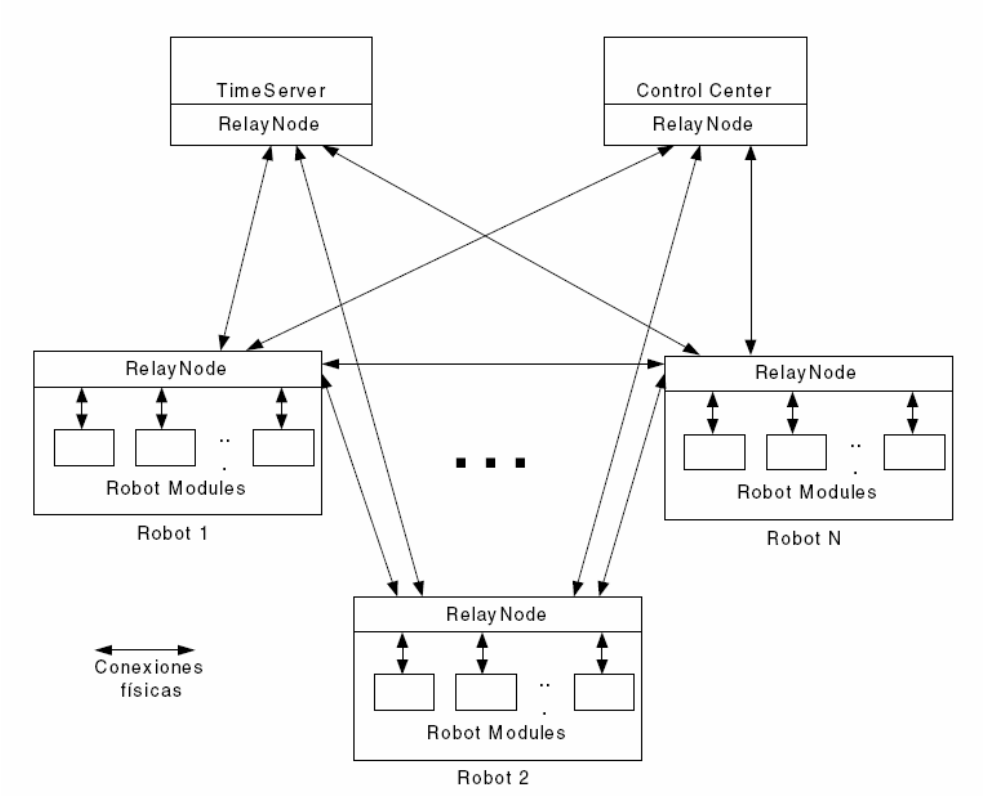

Figura 2-4: Sistema de comunicaciones

Un *Relay Node*, al no realizar procesamiento, estará formado por los siguientes hilos:

1. Programa principal *main*: se encarga de lanzar el hilo de comunicaciones y esperar la finalización del programa por parte del usuario o del propio programa.

2. Hilo de comunicaciones: se encarga de crear el fichero de conexiones a partir del fichero *NetConnections.conf*, de añadir los slots de la interfaz GRI, para que las distintas entidades se puedan comunicar entre sí, y los slots de los módulos internos de cada entidad, para que se puedan comunicar entre ellos, y de sincronizar el *Relay Node* cada cierto tiempo para transmitir los datos y mantener actualizada la tabla de enrutamiento.

Por último, conviene indicar que todos los robots tienen un proceso independiente que hace las funciones de *Relay Node*, mientras que en el *Control Center* y en el *Time Server* son los respectivos hilos de comunicaciones los que hacen dicha función.

A continuación, se ofrece una descripción detallada de las tres partes fundamentales que componen la arquitectura del sistema.

2.1. Control Center

Esta entidad es la encargada de que sea posible tratar a todos los robots de una misma forma, estando formada por un único proceso, y éste a su vez por varios hilos, los cuales se describen a continuación:

- **Programa principal *main*:** se encarga de lanzar el resto de hilos que conforman esta entidad, y de permanecer a la espera de que se finalice el programa por el usuario o por el propio programa.

- **Hilo de comunicaciones:** se encarga de las comunicaciones del *Control Center* con el resto de robots y el *Time Server*. Además crea automáticamente el fichero de conexiones a partir del fichero de configuración (*./robot_architecture/comms/NetConnections.conf*), donde sólo hay que indicar la dirección IP de cada robot, del *Control Center* y del *Time Server*.

- **Hilo de terminal:** se encarga de implementar un menú para que el usuario pueda mandar las tareas a ejecutar, siendo por tanto la interfaz existente entre el usuario y los robots. También se encarga del logeo de dichas tareas.

- **Hilo de representación gráfica en 2D:** se encarga de representar la posición de cada robot en un mapa de 2D.

- **Hilo del *CNP Manager*:** se encarga de publicar las tareas necesarias para el algoritmo de negociación distribuida, el cual resuelve el problema de asignación de las mismas.

- **Hilo de logeo del estado de los robots:** se encarga de logear los estados de los robots que le llegan al *Control Center*. Este estado es estándar, es decir, todos los robots transmiten la misma estructura de datos y son los mismos robots los encargados de transformar su estructura particular del estado a la estándar.

- **Hilo de logeo del estado de las tareas:** se encarga de comprobar el estado de las tareas que son transmitidas por los robots. En el momento que detecta que hay una tarea nueva o que ha cambiado de estado, imprime la tarea y su estado por la pantalla, y la logea en el fichero correspondiente, indicando el robot al que pertenece.

Por lo tanto, las funciones del *Control Center* son las siguientes:

- Representación gráfica de la posición de los distintos robots en un mapa en 2D.

- Mandar las tareas, para su ejecución, a los distintos robots de forma centralizada o distribuida.

- Abortar una tarea para que deje de ejecutarse.

- Suprimir el estado de una tarea para que se termine la transición del estado de ésta.

- Logeo del estado de cada robot, de las tareas que se le han enviado y del estado de éstas en cada momento.

Por último, cabe mencionar que el funcionamiento centralizado o distribuido del *Control Center* es independiente de la arquitectura, por lo que se puede considerar a éste como un *centro de control mixto*. De esta forma, sin tener que reiniciar ningún módulo, el *Control Center* puede mandar las tareas directamente a un robot o comenzar el algoritmo de negociación para que se las repartan automáticamente entre ellos.

2.2. Time Server

Esta entidad se encarga de transmitir por el sistema de comunicaciones, BBCS, una base de tiempo común para todas las entidades (robots y *Control Center*). Actualmente, se está utilizando en los archivos de *logs* y en el protocolo de negociación distribuido.

2.3. Arquitectura de un robot

Está formada por varias capas, cada cual en un nivel de abstracción mayor a medida que la dependencia con las particularidades de cada robot son menores, es decir, las capas superiores son independientes del robot, mientras que las inferiores no. Dichas capas están formadas por varios módulos, cada uno con una funcionalidad específica. A su vez, estos módulos contienen los hilos de procesamiento necesarios para llevar a cabo su tarea, los cuales estarán comunicados entre sí mediante una sección crítica, existiendo también un hilo de comunicaciones que permitirá la comunicación entre dichos módulos.

Entre las capas existe una interfaz para comunicarlas entre sí, las cuales se pretende que sean independientes del robot en el que se encuentren, aunque esto solo es posible actualmente en la interfaz GRI, que se encarga de la comunicación con el resto de robots, el *Control Center* y el *Time Server*.

En los apartados posteriores se describirán las distintas capas e interfaces que conforman la arquitectura de un robot.

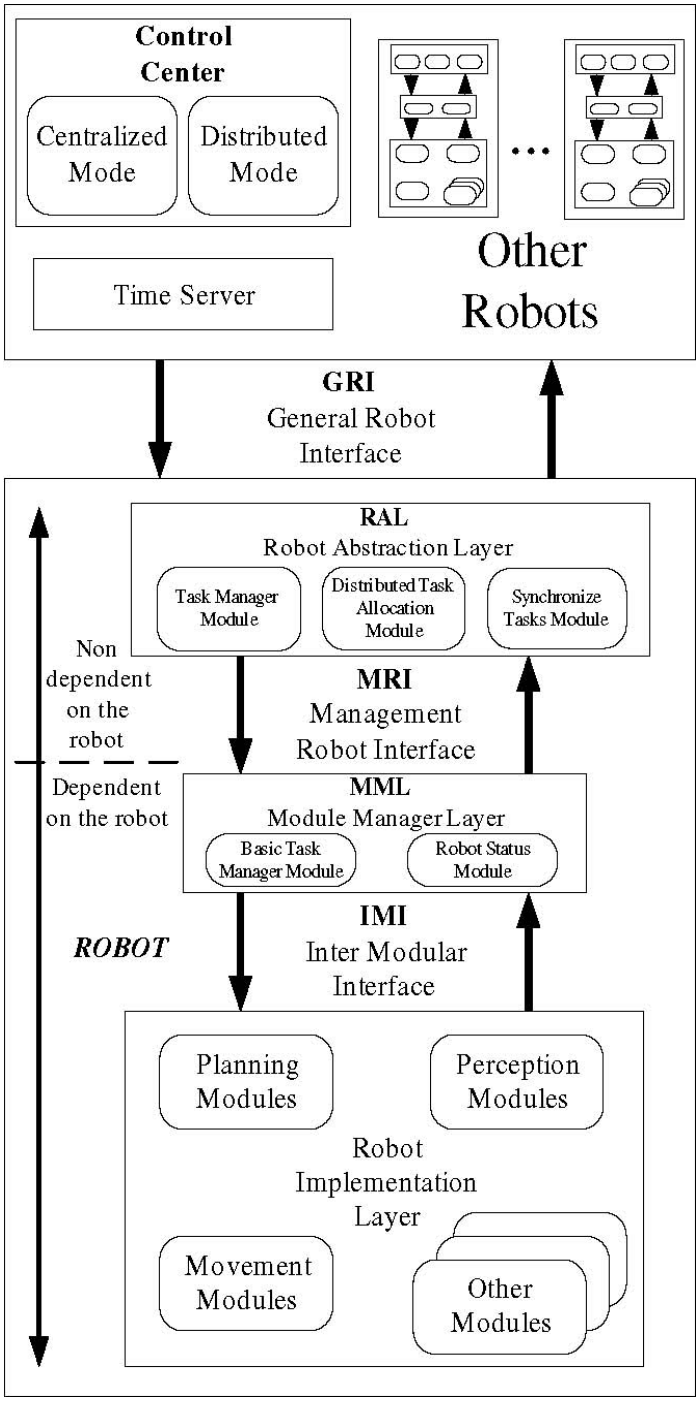

Figura 2.3-1: Arquitectura de un robot

2.3.1. Capas de la arquitectura

En la arquitectura de un robot aparecen tres capas bien diferenciadas (ver Figura 2.3-1): la *Robot Abstraction Layer* (RAL), la *Module Manager Layer* (RAL) y la *Robot Implementation Layer* (RIL).

Estas capas pueden estar formadas por uno o varios procesos:

- Si la capa está formada por un único proceso, como es el caso de la RAL y la MML, cada módulo será un hilo de procesamiento, y el hilo de comunicaciones será compartido.

- Si la capa está formada por varios módulos, los cuales son procesos independientes, cada uno de ellos estará formado por un hilo de comunicaciones y uno o varios hilos de procesamiento, como es el caso de la RIL.

A continuación se describen las distintas capas de la arquitectura.

2.3.1.1. Robot Abstraction Layer (RAL)

Esta capa se encarga de la gestión de las tareas enviadas por el *Control Center*, del protocolo para la asignación distribuida de tareas y de la sincronización de las mismas. Además, permite abstraer al robot sobre el que se implementa, de forma que todos se vean igual desde el *Control Center*. Es independiente del robot y por lo tanto la misma implementación debe servir para todos ellos. Por último, destacar que estos módulos no tienen control de estado (dormir el módulo, despertarlo, etc.) porque deben estar siempre funcionando, ya que es una parte fundamental de la arquitectura.

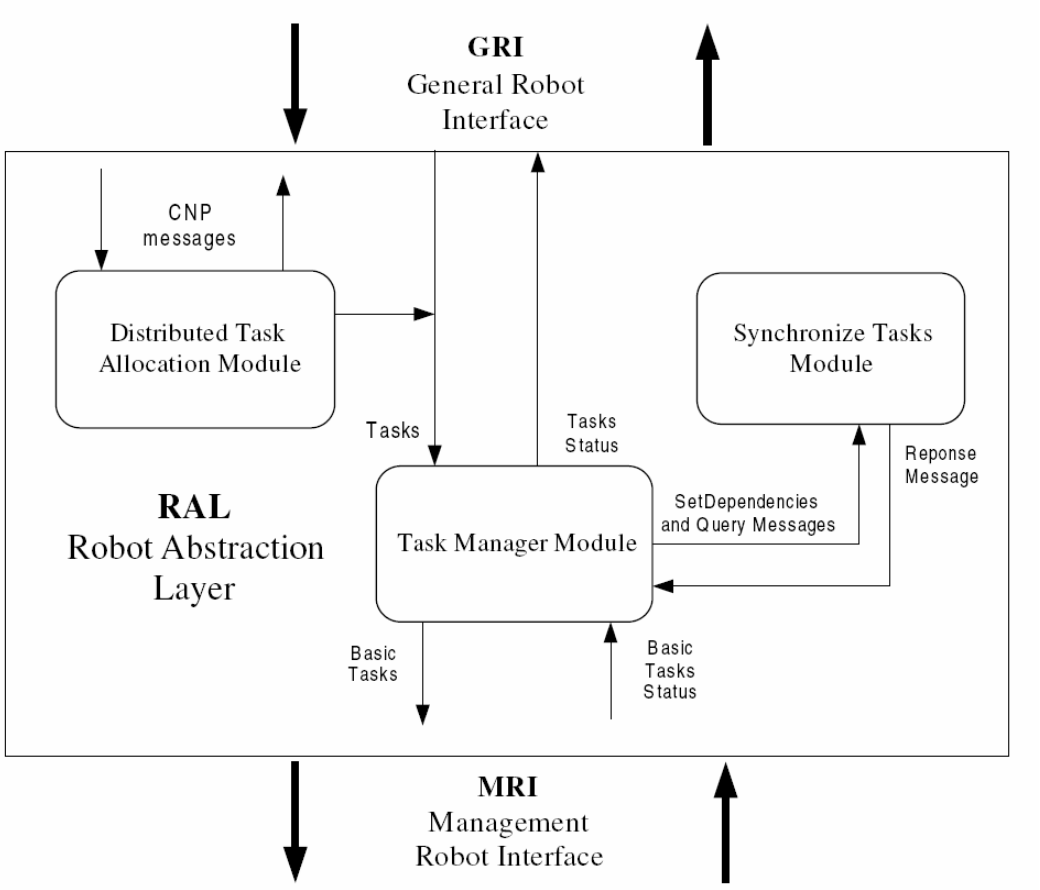

Figura 2.3.1.1-1: La Robot Abstraction Layer (RAL)

La *Robot Abstraction Layer* está formada por los 3 módulos que se describen a continuación:

- **CNP Manager Module:** se encarga de la implementación del algoritmo de negociación para la asignación de tareas distribuidas, basado en el protocolo CNP (*Contract Net Protocol*). Una vez que consigue una tarea, la envía al *Task Manager Module* para que la gestione.

  Las entradas y salidas de este módulo son:

  - Entradas: los mensajes del protocolo CNP (GRI).

  - Salidas: los mensajes del protocolo CNP (GRI), y las tareas obtenidas al finalizar el algoritmo de asignación, las cuales se envían al *Task Manager Module*.

- **Task Manager Module:** se encarga de gestionar las tareas que debe ejecutar el robot, las cuales le pueden llegar directamente del *Control Center* (modo de funcionamiento centralizado) o del *CNP Manager Module* (modo de funcionamiento distribuido). Este módulo no tiene conciencia del modo de funcionamiento que se está utilizando, simplemente recibe las tareas y las gestiona, por lo que también se podría utilizar un modo de funcionamiento mixto. Actualmente existen tres tipos de tareas que se pueden gestionar en paralelo (movimiento, percepción y *pan&tilt*), las cuales son traducidas a un conjunto de tareas básicas, de forma que el *Task Manager Module* sólo comunica a la MML una tarea por cada tipo de tarea que se puede gestionar en paralelo a la vez, por lo que hasta que no concluye una tarea no se manda la siguiente. Por otra parte, este módulo se comunica internamente por medio de la sección crítica con el *Synchronize Task Module*, que se encarga de la sincronización de tareas.

  El funcionamiento de este módulo, básicamente, es el siguiente:

  1. Cuando le llega una nueva tarea, si ésta tiene precondiciones, comunica al *Synchronize Task Module* el identificador de la misma y sus precondiciones por medio de un mensaje del tipo *SET_ DEPENDENCIES*, almacenado dicho módulo la información.

  2. El *Synchronize Task Module* esta continuamente leyendo, a través del sistema de comunicaciones, el estado de las tareas del propio robot y del resto. En el momento que una tarea aparece como finalizada (*ENDED*), elimina todas las precondiciones asociadas a dicha tarea.

  3. Antes de que el *Task Manager Module* comunique a la capa inferior la ejecución de una nueva tarea, le pregunta al *Synchronize Task Module* si ya se han cumplido todas las precondiciones, y por lo tanto si dicha tarea se puede ejecutar. Esta acción se realiza

mediante un mensaje tipo *QUERY*, donde el único parámetro es el identificador de la tarea en cuestión.

4. El *Synchronize Task Module* responde con un mensaje del tipo *RESPONSE*, donde únicamente se comunica si una tarea con un identificador determinado se puede ejecutar o no, para que, en caso positivo, sea transmitida a la capa inferior.

Las entradas y salidas de este módulo son:

- Entradas: las tareas que debe ejecutar el robot (GRI), el estado de las tareas básicas (MRI) y la respuesta del *Synchronize Task Module*, por medio de un mensaje del tipo *RESPONSE*, para saber si puede ejecutar una tarea o no (comunicación interna).

- Salidas: el estado de las tareas del robot (GRI), los mensajes del tipo *SET_DEPENDENCIES* y *QUERY* al *Synchronize Task Module* (comunicación interna) y las tareas básicas que se envían a la MML (MRI).

- **Synchronize Tasks Module:** se encarga de llevar un registro de las tareas que se van a ejecutar en el robot, y de sus respectivas precondiciones, comprobando constantemente cuales de dichas precondiciones se han cumplido, para que cuando sea preguntado, responda si una tarea se puede ejecutar o no. Debido a que este módulo solo se comunica con el *Task Manager Module*, y que ambos van a estar en el mismo proceso, la comunicación va directamente por medio de la sección crítica compartida por ambos hilos.

  Las entradas y salidas de este módulo son:

  - Entradas: los mensajes del tipo *SET_DEPENDENCIES* y *QUERY* (comunicación interna).

  - Salidas: los mensajes del tipo *RESPONSE* (comunicación interna).

2.3.1.2. Module Manager Layer (MML)

Esta capa se encarga de configurar y conectar los distintos módulos que componen la última capa de la arquitectura (la *Robot Implementation Layer*) para que el robot ejecute la tarea que se le ha asignado. Además, se encarga de traducir la telemetría del robot a un estado genérico, común a todos los robots, y transmitirla al *Control Center*. Esta capa ya depende del tipo de robot sobre el que se implemente, aunque dicha dependencia no es tan fuerte como en el caso de la siguiente (RIL).

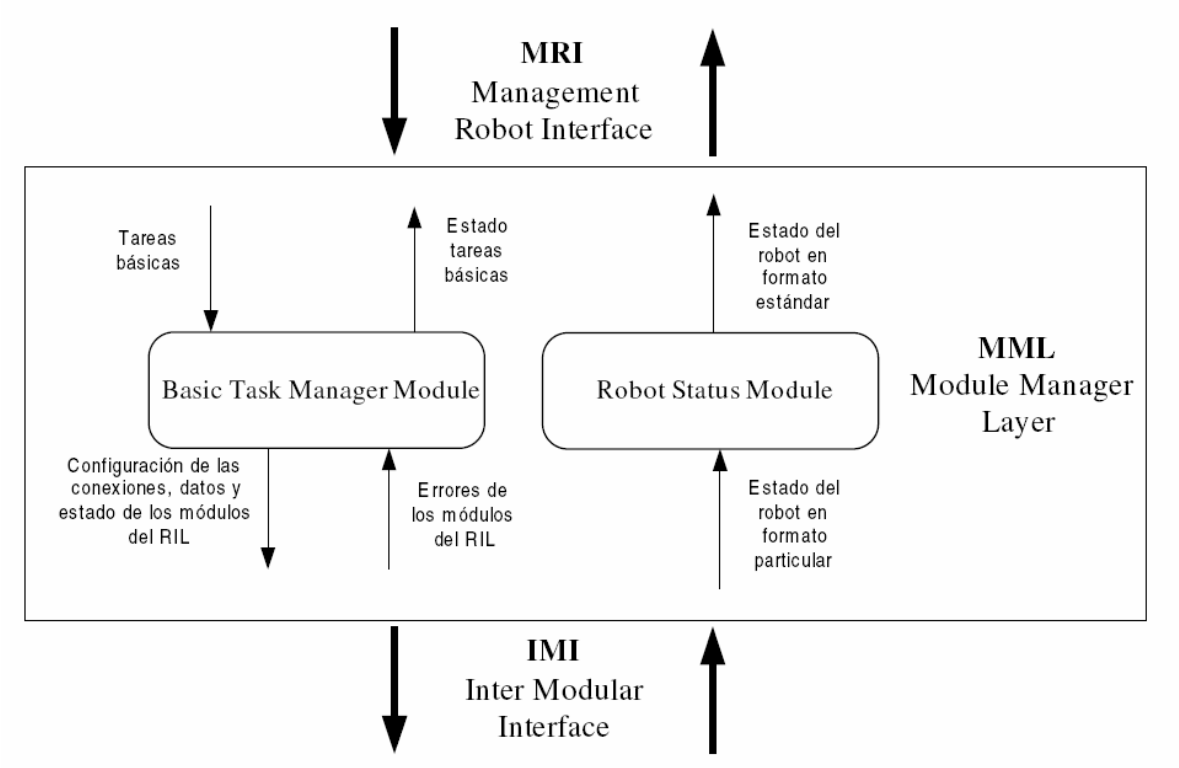

Figura 2.3.1.2-1: La Module Manager Layer (MML)

La *Module Manager Layer* está formada por los 2 módulos, independientes entre si, que se describen a continuación:

- **Basic Task Manager Module:** se encarga de gestionar las tareas básicas que le envía el *Task Manager Module*, despertando los módulos necesarios de la RIL, configurando las conexiones de los mismos y transmitiendo los datos necesarios a los módulos que los necesitan. Además, transmite el estado de las tareas básicas a partir de los errores que le llegan de los distintos módulos de la RIL, y comprueba que la tarea que se vaya a ejecutar sea coherente con el tipo de robot y el estado de éste, de forma que si no lo es, aborta la tarea indicando el error correspondiente asociado al estado de la misma.

  Las entradas y salidas de este módulo son:

  - Entradas: las tareas básicas que le llegan del *Task Manager Module* (MRI), y los errores de los distintos módulos de la RIL (IMI).

  - Salidas: el estado de las tareas básicas (MRI) y los datos necesarios que se envían a los módulos correspondientes de la RIL para la ejecución de las mismas (IMI).

- **Robot Status Module:** se encarga de traducir el estado del robot de su formato particular a un formato estándar, que es común a todos los robots, y de transmitirlo al *Control Center*.

  Las entradas y salidas de este módulo son:

  - Entradas: el estado del robot en el formato particular (IMI).
  - Salidas: el estado del robot en el formato estándar (MRI).

2.3.1.3. Robot Implementation Layer (RIL)

Esta capa es la encargada de trabajar con el hardware del robot, por lo que es totalmente dependiente del mismo, y se encuentra en el nivel más bajo (menos abstracto) de esta arquitectura. Así, cada robot tendrá una capa RIL distinta, aunque siguiendo la misma estructura de programación, que dependerá del hardware exclusivo que posea cada uno.

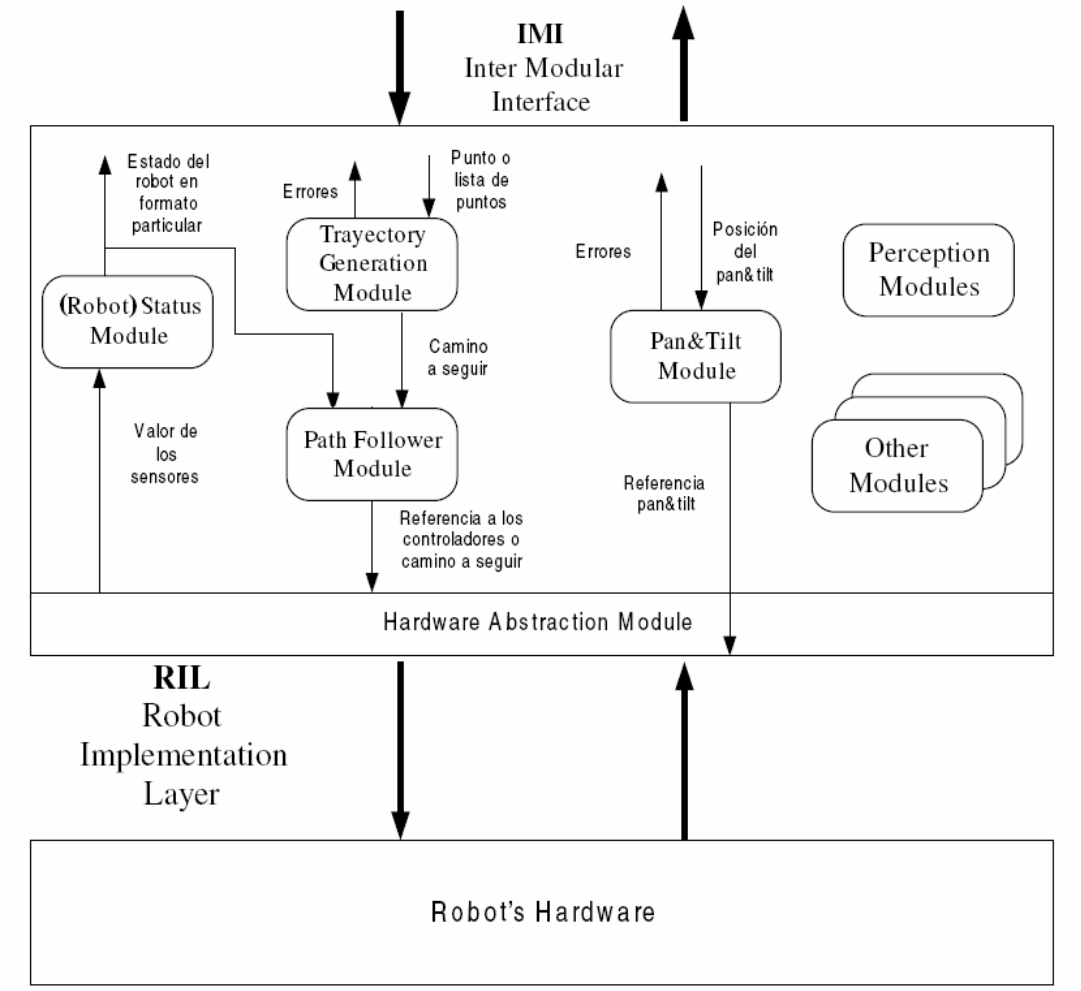

Figura 2.3.1.3-1: La Robot Implementation Layer (RIL)

La *Robot Implementation Layer* está formada actualmente por los 4 módulos que se describen a continuación:

- **Hardware Abstraction Module:** se encarga de toda la gestión de los dispositivos del robot, recibiendo las lecturas de datos de los sensores para, luego enviarlas al *(Robot) Status Module* (en nuestro caso será el *Romeo Status Module*), y enviando a los actuadores las referencias de control que le llegan del *Path Follower Module*. Este módulo no tiene control de estado porque debe estar siempre funcionando, ya que es una parte básica de la arquitectura.

  Las entradas y salidas de este módulo son:

  - Entradas: las referencias de control para los actuadores, provenientes del *Path Follower Module*, y la configuración de las conexiones con otros módulos (IMI).

  - Salidas: los datos de los sensores que son enviados al *(Robot) Status Module* (en nuestro caso será el *Romeo Status Module*) y los posibles errores (IMI).

- **(Robot) Status Module:** (en nuestro caso se denomina *Romeo Status Module*) se encarga de calcular el estado del robot en el formato particular, recibiendo la información de los distintos sensores que le llega del *Hardware Abstraction Module* y lo envía al *Path Follower Module* para que calcule las señales de control a partir de él, y al *Robot Status Module* de la MML para que lo transforme al formato estándar.

  Las entradas y salidas de este módulo son:

  - Entradas: la información de los distintos sensores que le llega del *Hardware Abstraction Module.*

  - Salidas: el estado del robot en el formato particular hacia el *Path Follower Module* y hacia el *Robot Status Module* de la MML (IMI).

- **Path Follower Module:** se encarga del seguimiento de caminos, recibiendo el mismo del *Trajectory Generation Module* y el estado del robot del *(Robot) Status Module*, y enviando las referencias de control al *Hardware Abstraction Module*.

  Las entradas y salidas de este módulo son:

  - Entradas: el camino a seguir procedente del *Trajectory Generation Module* y el estado del robot del *(Robot) Status Module*, además de la configuración y el estado del módulo (IMI).

- Salidas: las referencias de control para los actuadores al *Hardware Abstraction Module* (IMI).

- **Trajectory Generation Module:** se encarga de generar un camino a partir de un punto o lista de puntos.

  Las entradas y salidas de este módulo son:

  - Entradas: el punto, o lista de puntos, a partir del cual se genera el camino, además de la configuración y el estado del módulo (IMI).

  - Salidas: el camino a seguir por el robot, que es enviado al *Path Follower Module*, y los errores que se produzcan (IMI).

2.3.2. Interfaces de la arquitectura

Las capas anteriormente vistas se comunican entre sí mediante interfaces, de forma que las existentes entre las capas superiores son independientes del robot, mientras que las existentes entre las capas inferiores no lo son actualmente (aunque se está trabajando para ello).

Estas interfaces son bidireccionales, de forma que la capa superior se encarga de comunicar a la inferior las tareas que se deben realizar y los datos asociados a ellas, mientras la capa inferior se encarga de comunicar a la superior el estado de dichas tareas y los posibles errores. A medida que se asciende hacia capas superiores, se van generalizando las tareas y los tipos de errores que se puedan producir. Los errores que se transmiten a las capas superiores son aquellos que indican el fallo en la finalización de la tarea que se está ejecutando en las capas inferiores.

Las tres interfaces existentes son (ver Figura 2.3-1):

1. **Generic Robot Interface (GRI):** interfaz existente entre el *Control Center*, el *Time Server* y las distintas RAL de los robots. Está formada por las tareas de alto nivel y sus respectivos estados, los mensajes CNP (*Contract Net Protocol*) y el reloj del *Time Server*. Es independiente del robot en el que se use.

2. **Management Robot Interface (MRI):** interfaz existente entre la RAL y la MML. Está formada por las tareas básicas y sus respectivos estados. Depende del robot sobre el que se usa.

3. **Inter Modular Interface (IMI):** interfaz existente entre la MML y la RIL. Se encarga de la configuración de los módulos y sus respectivas conexiones. Depende del robot sobre el que se usa.

Por último, resaltar que cada interfaz está dividida en tres partes, cada una con un fichero asociado (situados en *./robot_architecture/comms*):

1. Comunicaciones: se configuran las comunicaciones utilizadas entre las capas para comunicarse utilizando el interfaz. En el fichero *XXX_slots.h* se especifican los distintos slots utilizados en el BBCS para la interfaz *XXX*.

2. Tipos de datos: se especifican los tipos de datos necesarios para la interfaz. En el fichero *XXX_data.h* se definen las estructuras de datos utilizadas en la interfaz *XXX*.

3. Códigos de error: se determinan los códigos de error que son transmitidos de la capa inferior a la capa superior. En el fichero *XXX_error_code.h* se especifican dichos errores correspondientes a la interfaz *XXX*.

2.4. Conclusiones

En este capítulo, se ha dado una visión general de la nueva arquitectura que se ha implantado para trabajar con un robot de una forma más sistemática, desde el punto de vista de la implementación de software, a la vez que se han visto las ventajas que conlleva dicha arquitectura a la hora de trabajar con múltiples robots, de forma que cooperen y se coordinen entre sí para el desarrollo de tareas más complejas.

De esta forma, se ha conseguido una estandarización en el código, que permitirá que los módulos y algoritmos desarrollados para un robot puedan ser traspasados a otros robots distintos sin tener que realizar cambios importantes. A su vez, el programador que va a desarrollar nuevos algoritmos solo necesitará conocer el funcionamiento del módulo sobre el que va a trabajar, evitando así el tener que conocer todo el código de la arquitectura, cosa que, en el caso del ROMEO-4R, era imposible con la anterior arquitectura (ver Apéndice A).

Esta arquitectura ha permitido una considerable reducción en el proceso de creación de nuevos robots (que permitirá a su vez el trabajo con el resto de robots que sigan esta arquitectura), para lo cual, básicamente habría que desarrollar el *Hardware Abstraction Module*, realizar las modificaciones oportunas en el resto de módulos la *Robot Implementation Layer* y adaptar la *Module Manager Layer* a las particularidades del robot.

Por último, comentar la existencia de un simulador para múltiples robots, que permite realizar las pruebas necesarias que aporten de una forma bastante aproximada el comportamiento de los mismos en la realidad, siendo la única diferencia con ésta la creación de un modelo cinemático y dinámico que permita la simulación del *Hardware Abstracción Module* (ver Capítulo 4).