UNIVERSIDAD POLITÉCNICA DE VALENCIA

Departamento de Informática de Sistemas
y Computadores

ARQUITECTURA PARA EL CONTROL DE ROBOTS MÓVILES MEDIANTE DELEGACIÓN DE CÓDIGO Y AGENTES

TESIS DOCTORAL

Presentada por:
D. Juan Luis Posadas Yagüe

Dirigida por:
D. José Enrique Simó Ten

Valencia, 2003

TESIS DOCTORAL

11 de Julio de 2003

AGRADECIMIENTOS

Quiero dejar constancia de mi agradecimiento a las tres personas que han hecho posible la consecución de esta tesis: a mi madre, quien me inspiró el deseo por el saber, a mi padre, quien siempre me ha apoyado proporcionándome todo aquello que he necesitado, y a mi director de tesis, quien me ha enseñado y orientado en todo el proceso de desarrollo.

Por supuesto, no olvido al resto de personas que también me han ayudado con su esfuerzo y a quienes les estoy totalmente agradecido.

Compañeros del grupo de investigación y proyectos con quienes comparto el trabajo desarrollado: Alfons, Paco, Pascual, Ginés, José Luis y Pedro.

Mi compañero de despacho Juan Carlos por sus consejos.

Y a mi novia Patricia por el tiempo que le he quitado y la paciencia que ha tenido.

RESUMEN

Esta tesis se enmarca en el estudio de arquitecturas híbridas para el control de robots móviles inteligentes, así como en los sistemas de comunicaciones necesarios para su diseño e implementación.

El principal objetivo de este trabajo ha sido la definición de una estructura modular y portable que permita el desarrollo rápido de sistemas de control de robots.

Los robots móviles son agentes físicos que deben estar en continua interacción con el entorno dinámico en el que se mueven. Para ello, disponen de distintos tipos de sensores distribuidos en diferentes nodos cuya información debe utilizarse en el cálculo de las acciones a realizar en cada instante, de esta manera se obtiene un funcionamiento reactivo que consiste en realizar una correspondencia entre los valores de los sensores y las acciones a realizar por los actuadores (correspondencia que da lugar a los denominados patrones de comportamientos). Por otro lado, dicha información sensorial también debe ser integrada en el sistema distribuido para la obtención de un modelo simbólico del entorno que permita realizar procesos de planificación y predicción que tengan en cuenta tanto la información actual como la pasada, obteniéndose de esta forma un funcionamiento deliberativo o basado en conocimiento.

La tendencia actual en el diseño de arquitecturas para robots móviles está basada en la combinación de ambos tipos de funcionamiento (arquitecturas híbridas) separándolos en dos niveles diferentes dentro de la arquitectura: nivel reactivo y nivel deliberativo. El nivel deliberativo debe encargarse de obtener los patrones de comportamientos cuya ejecución permita alcanzar los objetivos del robot, y el nivel reactivo debe encargarse de ejecutar dichos comportamientos asegurando la interacción en tiempo real con el entorno. En este sentido, el sistema de comunicaciones constituye la base para la interacción necesaria entre los niveles reactivo y deliberativo, debiendo permitir el acceso a la información sensorial que se encuentra distribuida en el sistema tanto por parte de las tareas reactivas como por parte de las tareas deliberativas.

El diseño del sistema de comunicaciones también deberá contemplar las diferentes restricciones temporales entre ambos niveles. Mientras que las tareas reactivas deben reaccionar en tiempo real estricto ante los cambios del entorno, por ejemplo para evitar un obstáculo, las tareas deliberativas no precisan de cotas temporales tan estrictas, pudiendo no cumplir con sus periodos de ejecución sin que ello lleve al sistema a una situación catastrófica.

Las tareas deliberativas, para la construcción del modelo simbólico del entorno, deben realizar procesos de fusión espacio-temporal basados en los valores de los sensores. Para ello, será necesario que la información proporcionada por el sistema de comunicaciones esté caracterizada temporalmente mediante la antigüedad de la misma. De esta forma, los datos requeridos por los procesos podrán validarse a través de su antigüedad y utilizarse en función de la misma.

Las diferentes tareas de los niveles deliberativo y reactivo suelen estructurarse de manera natural mediante componentes *software* independientes. Así, el diseño de las arquitecturas híbridas multinivel se basa en la modularidad *software* y ejecución independiente de sus componentes. Los comportamientos, proporcionados por el nivel deliberativo y ejecutados en el nivel reactivo, son procesos independientes que se ejecutan concurrentemente. La implementación de los comportamientos puede basarse en la "*teoría de esquemas*" según la cual un comportamiento es la combinación de al menos un *esquema de percepción* y un *esquema motor*. Modularidad e independencia son características innatas de los agentes *software*, lo cual les hace adecuados para el diseño e implementación de este tipo de arquitecturas.

Teniendo en cuenta estos requerimientos, como tema central de la tesis se propone una arquitectura denominada SC-Agent para el control de robots móviles. Esta arquitectura es híbrida y distribuida, los niveles deliberativo y reactivo están compuestos por agentes software que pueden moverse entre los nodos del sistema con el objetivo de reducir los requerimientos de comunicación en el acceso a los datos. La arquitectura aporta un nuevo nivel semántico donde se especifican los componentes de la misma pero no su ubicación física. Un agente podrá decidir su ubicación óptima en función de índices de medida no convencionales basados en la antigüedad de los datos que reciba. La arquitectura también permite mediante el uso de los agentes móviles aplicar técnicas de delegación de código para realizar el control de los robots minimizando los problemas relacionados con la latencia de la red. Dicho control se basa en separar el tiempo necesario de las comunicaciones del tiempo necesario de procesamiento. En primer lugar se realiza el envío del código de control necesario a los nodos del robot y en segundo lugar se realiza la ejecución del mismo de forma local. Al ejecutarse localmente dicho código, en lugar de hacerse de forma remota, se facilita el cumplimiento de las restricciones de tiempo real estricto.

El diseño de esta arquitectura se sustenta sobre un sistema de comunicaciones basado en una estructura de pizarra distribuida de objetos. El sistema de comunicaciones permite la interacción entre los distintos agentes de la arquitectura mediante una interfaz común para el acceso a los objetos de la pizarra. Los agentes

podrán transmitir mensajes o enviar información al resto de agentes mediante la escritura y lectura de los objetos de la pizarra. Los datos o valores de dichos objetos están caracterizados temporalmente por el sistema a través de su antigüedad.

El sistema de comunicaciones emplea un bus de tiempo real estricto para conectar los nodos que forman parte del nivel reactivo y un bus de tiempo real no estricto para conectar los nodos que forman parte del nivel deliberativo. El bus de tiempo real estricto ofrece al nivel reactivo predicción temporal en la transmisión de los datos y el bus de tiempo real no estricto ofrece al nivel deliberativo la transmisión de estructuras de datos más complejas como implica la transmisión de los agentes. Estos buses permiten asegurar las distintas restricciones temporales de ambos niveles. Mediante el empleo de objetos pasarela, el sistema de comunicaciones realiza las conversiones de datos necesarias entre los buses para permitir el acceso a la información sensorial e interacción con los actuadores por parte de ambos niveles de la arquitectura.

La arquitectura y sistema de comunicaciones propuestos han sido implementados en diferentes aplicaciones industriales obteniéndose resultados satisfactorios. Además, se han realizado diferentes prototipos software que validan las distintas características ofrecidas.

Finalmente, destacar que se ha obtenido una arquitectura de control general cuya modularidad *software,* basada en componentes de ejecución independientes que interaccionan a través de una pizarra de objetos distribuidos, permite su aplicación e implementación rápida sobre diferentes tipos de sistemas particularmente aquellos basados en el control de robots móviles.

RESUM

Aquesta tesi s'emmarca en l'estudi d'arquitectures híbrides per al control de robots mòbils intel·ligents, així com en els sistemes de comunicacions necessaris per al seu disseny i implementació.

El principal objectiu d'aquest treball ha sigut la definició d'una estructura modular i portable que permeta el desenvolupament ràpid de sistemes de control de robots.

Els robots mòbils són agents físics que han d'estar en contínua interacció amb l'entorn dinàmic en el que es mouen. Per això, disposen de distints tipus de sensors distribuïts en diferents nodes, on la informació del qual ha d'utilitzar-se en el càlcul de les accions a realitzar en cada instant, obtenint-se d'aquesta manera un funcionament reactiu que consisteix a realitzar una correspondència entre els valors dels sensors i les accions a realitzar pels actuadors (correspondència que dóna lloc als denominats patrons de comportaments). D'altra banda, la informació sensorial també ha de ser integrada en el sistema distribuït per a l'obtenció d'un model simbòlic de l'entorn que permeta realitzar processos de planificació i predicció que tinguen en compte tant la informació actual com la passada, obtenint-se d'esta forma un funcionament deliberatiu o basat en coneixement.

La tendència actual en el disseny d'arquitectures per a robots mòbils està basada en la combinació d'estos dos tipus de funcionament (arquitectures híbrides) separant-los en dos nivells diferents dins de l'arquitectura: nivell reactiu i nivell deliberatiu. El nivell deliberatiu ha d'encarregar-se d'obtenir els patrons de comportaments l'execució dels quals permeta aconseguir els objectius del robot, i el nivell reactiu ha d'encarregar-se d'executar els dits comportaments assegurant la interacció en temps real amb l'entorn. En aquest sentit, el sistema de comunicacions constitueix la base per a la interacció necessària entre els nivells reactiu i deliberatiu, permetent l'accés a la informació sensorial que està distribuïda en el sistema tant per part de les tasques reactives com per part de les tasques deliberatives

El disseny del sistema de comunicacions també haurà de contemplar les diferents restriccions temporals entre estos dos nivells. Les tasques reactives han de

reaccionar en temps real estricte davant dels canvis de l'entorn, per exemple per a evitar un obstacle, en canvi les tasques deliberatives no precisen de cotes temporals tan estrictes, podent no complir amb els seus períodes d'execució sense que això porte al sistema a una situació catastròfica.

Les tasques deliberatives, per a la construcció del model simbòlic de l'entorn, han de realitzar processos de fusió espai-temporal basats en els valors dels sensors. Per a això, serà necessari que la informació proporcionada pel sistema de comunicacions estiga caracteritzada temporalment per mitjà de la seua antiguetat. D'esta forma, les dades requerides pels processos podran validar-se a través de la seua antiguetat i utilitzar-se en funció de la mateixa.

Les diferents tasques dels nivells deliberatiu i reactiu solen estructurar-se de manera natural per mitjà de components *software* independents. Així, el disseny de les arquitectures híbrides multinivell es basa en la modularitat del *software* i execució independent dels seus components. Els comportaments, proporcionats pel nivell deliberatiu i executats en el nivell reactiu, són processos independents que s'executen concurrentment. La implementació dels comportaments pot basar-se en la "*teoria d'esquemes*" segons la qual un comportament és la combinació de al menys un *esquema de percepció* i un *esquema motor*. Modularitat i independència són característiques innates dels agents *software*, la qual cosa els fa adequats per al disseny i implementació d'aquest tipus d'arquitectures.

Tenint en compte aquestos requeriments, com a tema central de la tesi es proposa una arquitectura denominada SC-Agent per al control de robots mòbils. Esta arquitectura és híbrida i distribuïda, els nivells deliberatiu i reactiu estan compostos per agents *software* que poden moure's entre els nodes del sistema amb l'objectiu de reduir els requeriments de comunicació en l'accés a les dades. L'arquitectura aporta un nou nivell semàntic on s'especifiquen els components de la mateixa però no la seua ubicació física. Un agent podrà decidir la seua ubicació òptima en funció d'índexs de mesura no convencionals basats en l'antiguetat de les dades que reba. L'arquitectura també permet per mitjà de l'ús dels agents mòbils aplicar tècniques de delegació de codi per a realitzar el control dels robots minimitzant els problemes relacionats amb la latència de la xarxa. El control es basa en separar el temps necessari de les comunicacions del temps necessari de processament. En primer lloc es realitza l'enviament del codi de control necessari als nodes del robot i en segon lloc es realitza l'execució del mateix de forma local. Com el codi s'executa localment, en compte de fer-se de forma remota, facilita el compliment de les restriccions de temps real estricte.

El disseny d'aquesta arquitectura es sustenta sobre un sistema de comunicacions basat en una estructura de pissarra distribuïda d'objectes. El sistema de comunicacions permet la interacció entre els distints agents de l'arquitectura per mitjà d'una interfície comuna per a l'accés als objectes de la pissarra. Els agents podran transmetre missatges o enviar informació a la resta d'agents per mitjà de l'escriptura i lectura dels objectes de la pissarra. Les dades o valors dels objectes estan caracteritzats temporalment pel sistema a través de la seua antiguetat.

El sistema de comunicacions empra un bus de temps real estricte per a connectar els nodes que formen part del nivell reactiu i un bus de temps real no estricte per a connectar els nodes que formen part del nivell deliberatiu. El bus de temps real estricte ofereix al nivell reactiu predicció temporal en la transmissió de les dades i el bus de temps real no estricte ofereix al nivell deliberatiu la transmissió d'estructures de dades més complexes com implica la transmissió dels agents. Estos dos bussos permeten assegurar les distintes restriccions temporals de estos dos nivells. Per mitjà de l'ocupació de les passarel·les, el sistema de comunicacions realitza les conversions de dades necessàries entre estos dos bussos que permeten l'accés a la informació sensorial i la interacció amb els actuadors per part d'estos dos nivells de l'arquitectura.

L'arquitectura i sistema de comunicacions proposats han sigut implementats en diferents aplicacions industrials obtenint-se resultats molt satisfactoris. També s'han realitzat diferents prototipus *software* que validen les distintes característiques oferides.

Finalment, destacar que s'ha obtingut una arquitectura de control general on la modularitat *software*, basada en components d'execució independents que interaccionen per mitjà d'una pissarra d'objectes distribuïts, permet la seua aplicació i implementació ràpida sobre diferents tipus de sistemes, particularment aquells basats en el control de robots mòbils.

ABSTRACT

This thesis deals with hybrid architectures that control intelligent mobile robots, and the communications systems that are necessary to design and implement hybrid architectures.

The main objective of this work is to define a modular and portable structure that enables the quick development of robot control systems.

Mobile robots are physical agents that move and interact continuously while embedded in a dynamic environment. Robots have different kinds of sensors which are distributed in various nodes. Sensorial information is used to calculate actions that the robot will execute. Robots behave reactively by making couplings between sensor values and actuator actions, and these couplings are known as behavior patterns. Sensorial information is also used to obtain a symbolic internal model of the environment. This model enables planning and prediction processes to be made which work by considering the current information and the past information of the system. This behavior is known as deliberative behavior.

Current mobile robot control architectures are based on both models of behavior – reactive and deliberative. This combination is known as hybrid architecture and it has two levels: reactive and deliberative. The deliberative level has to obtain and offer the reactive level those behavior patterns that are necessary for the robot to achieve its objectives. The reactive level has to execute these behavior patterns by guaranteeing real time restrictions. The communications system is the base that enables the necessary interaction between reactive and deliberative levels and it sends distributed sensorial information to tasks in both levels.

Communications system design has to consider the various temporal constraints between levels. The reactive level executes tasks which must respond to the changes in the environment. These tasks have associated hard real time constraints, for example, a task that avoids obstacles must react to proximity sensor signals in a time limit determined by the speed of the robot. The deliberative level executes tasks which have soft real time constraints. When a deliberative task does not achieve its execution period, the system is not exposed to a risk of catastrophic malfunction.

Deliberative tasks have to execute spatial-temporal fusion processes in order to build a symbolic internal model of the environment by using the values of the sensors. A communications system must be able to provide all the data with its temporal information. Therefore, tasks can validate useful data by using their time properties.

Deliberative and reactive tasks can be structured in a natural way by means of independent software components. Multilevel hybrid architecture design is based on the modularity of the software and the independent execution of the components. Pattern behaviors obtained from the deliberative level and executed in the reactive level are independent processes which are executed concurrently. The implementation of behavior patterns can be based on *schema theory*. According to this theory, a behavior is composed of at least one *motor schema* and at least one *perceptual schema*. Modularity and independence are innate characteristics of software agents. So, software agents are useful for designing and implementing this kind of architectures.

In this thesis, architecture for controlling mobile intelligent robots is proposed. This architecture, known as SC-Agent, is hybrid and distributed. Reactive and deliberative levels are composed of software agents that can move through the nodes of the system. This mobility can be used to reduce communications overload while obtaining data that agents need. The proposed architecture provides a new semantic level where processes are specified without a fixed location. Agents can move to where environmental conditions are better. They choose their location by using specific measurement indexes based on the age of the data they receive. The architecture applies code delegation techniques to control robots by means of mobile software agents. Delegation techniques reduce networking latency problems by separating communication time from processing time. The architecture makes all off-line communications for delegating code by moving necessary agents to the nodes of the robot, and then executes the code by using the real time local infrastructure.

The design of this architecture is based on a communications system that takes the form of an object distributed blackboard. The communications system allows interaction among all agent components by using a common interface to access blackboard objects. Agents can send messages, or data, to other agents by writing and reading blackboard objects. The system attaches temporal information to the distributed objects, this information being the age of the data.

The communications system connects reactive nodes through a hard real time bus, and connects deliberative nodes through a soft real time bus. Both buses are necessary because of the time characteristics of the reactive and deliberative levels are different. The reactive level needs a communication bus that guarantees access time and data frame transmission in order to be able to respond to environmental change in a bounded time. The deliberative level can use a bus without these constraints and this enables more complex data structures – such as agents. By means of gateway software, the communications system performs specific translations

between both buses. As a result, reactive and deliberative agents can interact with sensors and actuators and access all data in the system.

The proposed architecture and communications system has been implemented in various industrial applications with good results. Software prototypes have been developed to validate the characteristics of the architecture.

In conclusion, the main contribution is a modular and general control architecture. This modularity is based on independent software components that interact through an object distributed blackboard. The architecture can be applied and implemented quickly in order to control different types of systems, especially mobile robot control systems.

CONTENIDOS

AGRADECIMIENTOS **5**

RESUMEN **7**

CONTENIDOS **19**

INTRODUCCIÓN **25**

Motivación 25

Objetivos 27

Estructura del Documento 28

Capítulo 1
ARQUITECTURAS DE CONTROL PARA ROBOTS MÓVILES **31**

1.1. Introducción 31

1.2. Paradigma Deliberativo 33

1.3. Paradigma Reactivo 35

1.4. Paradigma Híbrido 37

1.4.1. Teoría de Esquemas 40

1.4.2. Arquitecturas 41

1.4.2.1. Arquitectura AURA 41

1.4.2.2. Arquitectura SFX 43

1.4.2.3. Arquitectura 3T 45

1.4.2.4. Arquitectura GLAIR 47

1.4.2.5. Arquitectura SAPHIRA *49*
1.4.3. Necesidad del control temporal de las comunicaciones 50
1.5. Conclusiones 51

Capítulo 2
COMUNICACIONES EN SISTEMAS DISTRIBUIDOS 55

2.1. Introducción 55
2.2. Modelo Cliente-Servidor 56
2.3. Modelo de Grupos 57
2.4. Modelo de Llamadas a Procedimientos Remotos 59
2.5. Modelo de Objetos Distribuidos 60
2.6. Arquitectura CORBA 62
2.6.1. Interoperatividad entre ORB's 65
2.6.1.1. Puentes *65*
2.6.1.2. Protocolo de comunicación entre ORB's: GIOP e IIOP *66*
2.6.2. Tiempo Real en CORBA: RT-CORBA 70
2.7. Procesamiento distribuido en Java 72
2.7.1. Java RMI 72
2.7.2. Java IDL 73
2.8. Microsoft .NET 74
2.9. Modelo de Código Móvil 75
2.10. Modelo Multiagente 77
2.11. Conclusiones 77

Capítulo 3
AGENTES SOFTWARE 79

3.1. Introducción 79
3.2. Concepto de Agente Inteligente 81
3.3. Sistemas Basados en Agentes 82
3.4. Dominio de Aplicación de los Agentes 83
3.5. Clasificación de los Agentes 85
3.5.1. Agentes Móviles 85
3.5.1.1. Ventajas de los Agentes Móviles *86*
3.5.2. Agentes Deliberativos 88

3.5.3. Agentes Reactivos 88
3.6. Taxonomía de un Agente 89
3.7. Estándares 90
3.7.1. MAF 92
3.7.2. FIPA 94
3.7.2.1. ACL 96
3.7.3. Integración de FIPA y MAF 97
3.8. Implementación de Sistemas de Agentes Móviles 99
3.9. Conclusiones 99

Capítulo 4
ARQUITECTURA SC-AGENT 101

4.1. Introducción 101
4.2. Descripción de la Arquitectura 102
4.2.1. Acoplamiento entre los Niveles Deliberativo y Reactivo 105
4.2.2. Marco del Control Distribuido 106
4.2.3. Movilidad de los Componentes 107
4.2.4. Procesamiento vs. Comunicaciones: Control por Delegación de Código 108
4.2.5. Requerimientos del Sistema de Comunicaciones 110
4.2.6. Necesidad del Control Temporal 111
4.2.7. Índice de Comodidad 115
4.2.8. Conclusiones 117
4.3. Sistema de Comunicaciones 118
4.3.1. Interfaz FSA 120
4.3.1.1. Especificación IDL 121
4.3.1.1.1 Estructura IDL de los agentes 124
4.3.1.2. Espacio de Nombres Lógicos 125
4.3.1.3. Metodología 128
4.3.2. Control Temporal 132
4.3.3. Comparación 135
4.4. Conclusiones 137

Capítulo 5
ARQUITECTURA SC-AGENT: AGENTES 139

5.1. Introducción 139
5.2. Agentes 140
5.2.1. Tipos de Objetos 141
5.2.2. Componentes de la Arquitectura 142
5.2.3. Interfaces 144
5.3. Nivel Reactivo 146
5.3.1. Componentes y Agentes 146
5.3.2. Conexión entre los agentes 148
5.4. Nivel Deliberativo 152
5.4.1. Agentes y Conexión con el Sistema 152
5.4.2. Sistema Multiagente 153
5.4.2.1. Interoperatividad 154
5.4.2.2. Mensajes ACL 158
5.5. Conclusiones 161

Capítulo 6
ARQUITECTURA SC-AGENT: IMPLEMENTACIÓN 163

6.1. Introducción 163
6.2. Sistema de Comunicaciones SC 164
6.2.1. Aplicación del Sistema de Comunicaciones SC 168
6.2.1.1. Planta de Producción de Objetos Decorativos de Porcelana 168
6.2.1.2. Terminal Marítima de Contenedores 170
6.2.1.3. Arquitectura de Control para Robots Móviles 172
6.3. Prototipo Desarrollado 172
6.3.1. Componente Robot Simulado 175
6.3.2. Componente Entorno Simulado 176
6.3.3. Componente Nivel Deliberativo 177
6.3.4. Componente Nivel Reactivo 178
6.3.4.1. Agente Esquema de Percepción de Posición 179
6.3.4.2. Agente Esquema de Percepción Detector de Obstáculos 179

6.3.4.3. Agente Esquema Motor para Evitar Obstáculos 180
6.3.4.4. Agente Esquema Motor para Ir a Destino 180
6.3.4.5. Agente Esquema Motor Explorador 180
6.3.4.6. Agente Compositor por Suma 180
6.3.5. Pruebas Realizadas y Resultados 181
6.4. Robot YAIR 186
6.4.1. Pruebas Realizadas y Resultados 190
6.5. Conclusiones 194

CONCLUSIONES 195
Contribuciones 195
Trabajos Futuros 198
Publicaciones Relacionadas 200

ÍNDICE DE FIGURAS 203

ÍNDICE DE TABLAS 209

REFERENCIAS BIBLIOGRÁFICAS 211

INTRODUCCIÓN

Esta tesis se ubica en el estudio de arquitecturas híbridas distribuidas para el control de robots móviles basadas en sistemas multiagente y de delegación de código, así como en los sistemas de comunicaciones necesarios para su diseño e implementación. Como resultado principal, se propone una arquitectura denominada SC-Agent cuyos componentes software móviles interaccionan a través de un sistema de pizarra distribuida, el cual proporciona el acceso a los datos caracterizándolos temporalmente.

Motivación

Existen diferentes líneas de investigación que centran sus esfuerzos en la idea de conseguir los denominados robots inteligentes. Sin embargo, actualmente esta meta todavía se encuentra demasiado lejos de alcanzar, sobre todo, si se tiene en cuenta que ni tan siquiera se conoce con exactitud el funcionamiento del cerebro humano, el cual nos proporciona el conocimiento e inteligencia deseado en los robots. De hecho, dicha meta podría ser físicamente imposible debido a la propia naturaleza de la inteligencia [Penrose, 1989].

Este trabajo intenta aportar un paso más encaminado a la obtención de arquitecturas de control de robots móviles que permitan exhibir a los robots comportamientos aparentemente inteligentes por "composición" de comportamientos básicos. Para ello, se realiza un enfoque basado en el estudio de los sistemas de comunicación necesarios entre los distintos componentes *software* de las arquitecturas.

Los robots móviles son un caso particular de los sistemas distribuidos de tiempo real, donde han evolucionado diferentes tipos de arquitecturas debido al aumento de la complejidad *hardware* de los sensores y actuadores, a la interacción entre diferentes componentes inteligentes, y a las restricciones de tiempo real existentes. Actualmente, las arquitecturas híbridas de control forman la solución más extendida

para la implementación de robots. Todo ello por varias razones [Murphy, 2000]: primero, el uso de técnicas de procesamiento asíncrono ha permitido que las funciones deliberativas se ejecuten independientemente de los comportamientos reactivos, segundo, la modularidad software ha permitido a los subsistemas o componentes en las arquitecturas híbridas ser combinados y reutilizados para diferentes aplicaciones específicas.

Ejecución independiente y modularidad son características innatas de los agentes *software*, un paradigma actual de programación para el desarrollo de aplicaciones [Jennings, 1998]. Debido a sus características, los agentes son adecuados para la especificación e implementación de arquitecturas de control híbridas. Actualmente, los agentes son el centro de interés de muchos campos de la ingeniería del computador y de la inteligencia artificial, y están usándose en un amplio y creciente conjunto de aplicaciones (aplicaciones orientadas a la industria, al comercio, a la medicina, al entretenimiento, etc.).

Los datos con los que tienen que operar las arquitecturas de robots provienen de distintos tipos de sensores cuya ubicación se encuentra distribuida a través de diferentes nodos de adquisición y procesamiento. Dicha información debe proporcionarse a todos los componentes de la arquitectura que la requieran, lo cual estará condicionado por la forma de interconexión de los nodos y el sistema de comunicaciones empleado. Igualmente, las acciones generadas por las diferentes tareas y comportamientos ejecutados en la arquitectura, tienen que llevarse a cabo en los distintos actuadores también situados de forma distribuida. La distribución y ubicación de las tareas y comportamientos en los nodos de la arquitectura dependerá de varios factores, como de los datos que requieran y de las restricciones temporales de ejecución.

Por todo ello, la especificación e implementación de una arquitectura, adecuada a los sistemas híbridos, tiene que contemplar la propia naturaleza distribuida tanto de la información sensorial como de los comportamientos. Las diferentes restricciones temporales que existen entre las tareas reactivas y deliberativas tienen que reflejarse en el diseño del sistema de comunicaciones. Por un lado, las tareas reactivas deben cumplir limitaciones o cotas temporales estrictas en su ejecución, mientras que las tareas deliberativas no precisan de un control temporal tan estricto. Cuando una tarea deliberativa no cumple sus tiempos de ejecución, el sistema no tiene que quedar expuesto a ningún riesgo catastrófico debido a que el conjunto de tareas reactivas tiene que encargarse de llevar el sistema bajo control.

En este sentido, en esta tesis se presenta una arquitectura de control aplicada a robots móviles inteligentes basada en el paradigma híbrido reactivo/deliberativo. Esta arquitectura es multinivel y distribuida, sus principales componentes son agentes software móviles que interactúan a través de un sistema de comunicaciones de pizarra distribuida. Estos agentes pueden ejecutarse de forma local en el propio robot o de forma remota en computadores ajenos basándose en los requerimientos temporales de los mismos.

Los aspectos desarrollados en la tesis se enmarcan en la línea de investigación de Arquitecturas para Sistemas Autónomos, perteneciente al grupo de investigación de Informática Industrial del cual forma parte el autor de este trabajo. Esta línea de investigación trata principalmente los temas de interfaces de comunicaciones para sistemas distribuidos, protocolos y distribución del procesamiento y de la información, modelos de objetos distribuidos, y diseño e implementación de arquitecturas de control para robots móviles.

Los resultados del trabajo están implicados en varios proyectos:

- CRUDA (CDTI): proyecto en colaboración con la empresa constructora de buques Unión Naval de Levante y la Escuela de Ingenieros Navales de la UPM para el desarrollo de la tecnología de "puente inteligente". Los temas de interés en el proyecto son: distribución de la información, protección frente a incendios e inundaciones y planificación de rutas con índices de coste no convencionales.

- CERAMICUS (CDTI): proyecto en colaboración con la empresa de objetos decorativos LLADRÓ para la implantación de un sistema distribuido de comunicaciones en un almacén automatizado de fabricación de figuras de porcelana.

- Desarrollo de un prototipo de robot multi-sensor autónomo, con capacidad de percepción y reacción en tiempo real para entornos industriales (CICYT): proyecto para el desarrollo de un robot autónomo denominado YAIR con capacidad de maniobra dentro de un entorno industrial. Los objetivos en el proyecto son: establecer modelos del entorno mediante fusión sensorial, interconexión de los módulos sensores y actuadores al sistema multiprocesador mediante buses de campo, implementación del sistema de comunicaciones local del robot.

- Informática de Sistemas para la Operación de una Terminal Marítima de Contenedores (FEDER): proyecto en colaboración con la empresa Marítima Valencia S.A. para la implantación de un sistema de gestión automática en una terminal de contenedores. Los temas de interés en el proyecto son: monitorización de la terminal, integración de diferentes sistemas heterogéneos mediante el diseño, implementación e implantación de una interfaz e infraestructura de comunicaciones común, gestión automática de eventos y actualización de la base de datos, control distribuido mediante componentes software móviles basados en la tecnología de agentes.

Objetivos

Desde un punto de vista general, el principal objetivo de esta tesis consiste en la definición de una estructura modular y portable que permita el desarrollo rápido de sistemas de control de robots.

Los objetivos concretos planteados son:

- Estudio y análisis de las necesidades de comunicaciones existentes en el diseño de arquitecturas híbridas para el control de robots móviles.

- Especificación y diseño de un sistema de comunicaciones multinivel que permita la interacción entre todos los componentes de una arquitectura. Las diferentes restricciones de tiempo real existentes entre los niveles reactivo y deliberativo de las arquitecturas híbridas deben considerarse en dicho diseño.

- Caracterización temporal de la información distribuida. El sistema de comunicaciones, además de proporcionar a los componentes de la arquitectura todos los datos del sistema, también tiene que proporcionar la antigüedad de dichos datos.

- Propuesta de una arquitectura híbrida, multinivel y distribuida para el control de robots móviles inteligentes. La arquitectura debe basarse en el sistema de comunicaciones obtenido y en la técnica de delegación de código. El objetivo es permitir la separación del tiempo empleado para las comunicaciones del tiempo empleado para el procesamiento:

    - En primer lugar el código necesario se delega o se mueve a los nodos distribuidos y una vez allí se ejecuta localmente. De esta forma, el código necesario de control puede ejecutarse bajo restricciones de tiempo real estricto evitando los problemas de latencia de la red.

- Ubicación automática de los componentes software de la arquitectura distribuida mediante el uso de sistemas de agentes móviles. Los agentes se mueven allí donde las condiciones del entorno son más favorables (por ejemplo, donde la sobrecarga temporal que suponen las comunicaciones para el acceso a los datos sea menor).

Estructura del Documento

El trabajo está organizado en varios capítulos (fig.1).

En el capítulo uno se realiza una revisión del estado del arte correspondiente a las arquitecturas para el control de robots móviles centrándose en las características y necesidades de comunicación de aquellas denominadas híbridas.

En el capítulo dos se analiza la evolución de los distintos modelos de comunicación existentes para sistemas distribuidos estudiando las características de tiempo real ofrecidas.

En el capítulo tres se realiza una revisión del estado del arte de los sistemas multiagente y agentes móviles relacionando sus características con las necesidades de diseño e implementación de una arquitectura híbrida para el control de robots móviles.

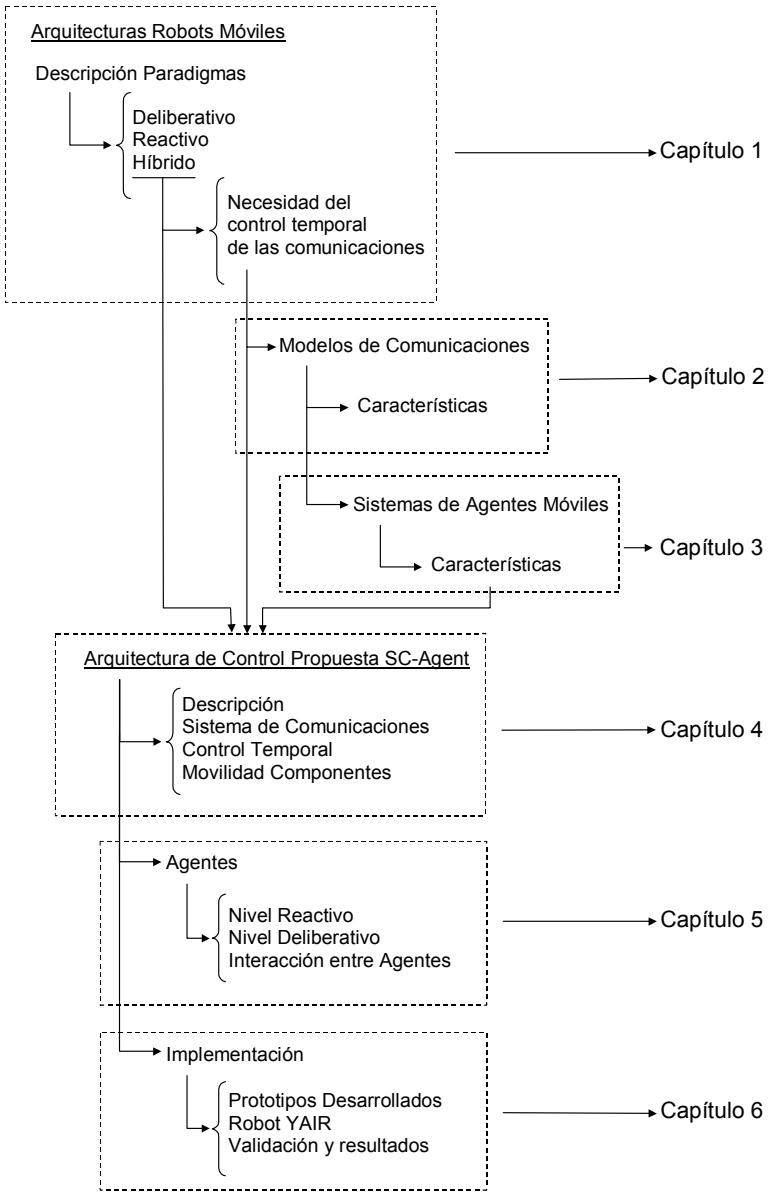


Figura 1: *Relación entre los distintos capítulos de la tesis*

El capítulo cuatro es la contribución principal de esta tesis, donde se propone una arquitectura de control híbrida denominada SC-Agent y un sistema de comunicaciones denominado SC como base para el diseño de dicha arquitectura. El sistema SC se basa en una estructura de tipo pizarra distribuida que permite la interacción entre los distintos componentes de la arquitectura independientemente del nivel reactivo o deliberativo al que pertenezcan ofreciendo, además, la caracterización temporal de los datos del sistema. Los componentes de la

arquitectura son agentes móviles que pueden decidir su ubicación física en el sistema distribuido utilizando para ello índices de medida no convencionales como el denominado "índice de comodidad".

En el capítulo cinco se realiza una descripción detallada del diseño de los niveles reactivo y deliberativo de la arquitectura propuesta. Para ello, se describen las relaciones e interacciones existentes, a través del sistema de comunicaciones SC, entre los agentes que componen dichos niveles.

El capítulo seis se dedica a presentar los resultados obtenidos en la implementación y aplicación del sistema de comunicaciones y arquitectura de control propuestos. Se describe un prototipo *software* que se ha desarrollado para validar las características de la arquitectura, y la implementación de la misma sobre un robot real denominado YAIR.

Finalmente se exponen las conclusiones y trabajos futuros.

Capítulo 1

ARQUITECTURAS DE CONTROL PARA ROBOTS MÓVILES

En este capítulo se realiza un análisis de los distintos paradigmas que abordan la inteligencia en los robots. Con este objetivo, se dedica un primer apartado a introducir los factores utilizados en la descripción de los mismos. A continuación, se incluyen varias secciones destinadas a presentar las características e inconvenientes de los distintos paradigmas concluyendo con la adecuación del paradigma híbrido en el diseño de arquitecturas de control distribuidas. Finalmente, se estudian las arquitecturas híbridas más significativas en la actualidad y se presenta como conclusión principal la importancia del control temporal de las comunicaciones así como que éstas permitan la transmisión de estructuras complejas de datos como supondría la transmisión de agentes software, objetivo abordado en la arquitectura que se propone en esta tesis.

1.1. Introducción

La complejidad del mundo real dificulta la interacción inteligente con éste. Un agente interactuando en un entorno dado se enfrenta en todo momento con el problema de qué acción realizar a continuación. Para tomar una decisión el agente dispone de dos fuentes principales de información:

- Un modelo interno del mundo basado en su experiencia previa.
- Información sensorial actual.

Durante las dos últimas décadas, en Inteligencia Artificial ha quedado establecido que cualquier robot autónomo tiene que poseer las siguientes tres funciones elementales o primitivas [Murphy, 2000]:

- Un sistema de PERCEPCIÓN: capacita al robot para disponer de la información de los sensores y producir, a partir de ésta, información útil para los otros elementos funcionales.

- Un sistema de RAZONAMIENTO o PLANIFICACIÓN: capacita al robot para producir las tareas a ejecutar (ir a la entrada, girar a la derecha, moverse 2 metros y parar, etc.) en base a la información sensorial y conocimiento disponible del mundo que le rodea o entorno.

- Un sistema de EJECUCIÓN o ACCIÓN: capacita al robot para la ejecución de tareas mediante la interacción con los actuadores.

Actualmente, la organización de la autonomía o inteligencia en robots móviles se aborda desde tres paradigmas diferentes: DELIBERATIVO, REACTIVO e HÍBRIDO. Cada una de estas filosofías puede describirse atendiendo a dos factores:

- La relación existente entre las tres funciones primitivas comunes para cualquier robot: PERCEPCIÓN, RAZONAMIENTO y EJECUCIÓN. En la tabla 1 se describen estas tres primitivas en términos de los posibles valores de entrada y de salida. En función de los valores de entrada y salida considerados en cada uno de los paradigmas, se establecen distintas relaciones entre las primitivas.

Primitivas	**Entradas**	**Salidas**
PERCEPCIÓN	Valores de los sensores	Información sensorial
RAZONAMIENTO	Información (sensorial y/o cognitiva)	Directivas-acciones
EJECUCIÓN	Información sensorial o directivas-tareas	Órdenes de actuación

Tabla 1: *Definición de las tres funciones elementales o primitivas de un robot en términos de entradas y salidas*

- El mecanismo de procesamiento y distribución de la información sensorial a través del sistema. Hace referencia a cuánto o cómo los sentidos influyen en una persona, robot o animal. En algunos paradigmas, la información sensorial se usa de una manera específica para cada función del robot y además de forma local. Por el contrario, en otros paradigmas, la información sensorial se usa para la creación de un modelo global del mundo que luego es utilizado de forma distribuida por las distintas funciones del robot.

A continuación se realiza una descripción de los tres paradigmas atendiendo a estos factores. De este modo, se presentan los inconvenientes de los paradigmas deliberativo y reactivo para concluir con la adecuación del paradigma híbrido en el diseño de arquitecturas de robots inteligentes. Seguidamente, se realiza un estudio de las arquitecturas híbridas más significativas en la actualidad y, finalmente, se concluye con la importancia en cualquier arquitectura distribuida del control temporal de las comunicaciones, objetivo abordado por la arquitectura que se propone en la presente tesis y que constituye uno de los aspectos centrales de la misma.

1.2. Paradigma Deliberativo

El paradigma deliberativo [Nilsson, 1980] es el más antiguo, su filosofía predominó principalmente desde 1967 hasta 1990. Según el paradigma deliberativo un robot autónomo primero tiene que percibir su entorno, después planear la acción a realizar y finalmente ejecutar dicha acción (S-P-A: sentir-planear-actuar). De esta forma, el ciclo tiene que repetirse hasta completarse cada una de las tareas del robot (fig. 2).

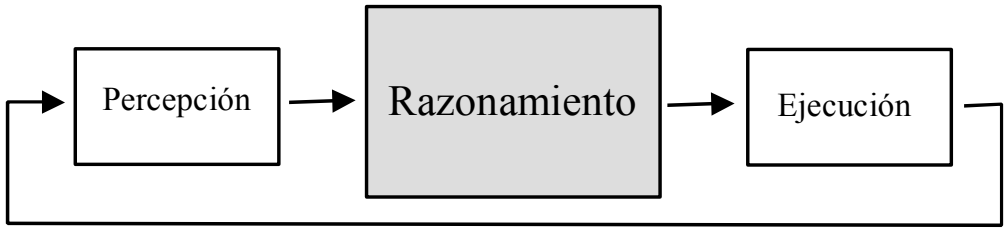


Figura 2: *Filosofía del paradigma deliberativo: Sentir-Planear-Actuar (S-P-A)*

Primitivas	Entradas	Salidas
PERCEPCIÓN	Valores de los sensores	Información sensorial
RAZONAMIENTO	Información (sensorial y/o cognitiva)	Directivas-acciones
EJECUCIÓN	Información sensorial o directivas-tareas	Comandos de actuación

Tabla 2: *Relación de las tres funciones elementales o primitivas de un robot según el paradigma deliberativo*

El paradigma deliberativo implica la planificación explícita de cada una de las acciones del robot, donde el razonamiento ha suplantado ampliamente el comportamiento reactivo comúnmente observado en los animales. En la tabla 2,

donde se relacionan las primitivas funcionales de un robot según el paradigma deliberativo, puede observarse que se ha suprimido el hecho de que la información sensorial pueda provocar de forma directa una acción.

Como base de razonamiento para todos los planes y acciones, el paradigma deliberativo se caracteriza por intentar obtener y utilizar un modelo interno del mundo real lo más exacto y completo posible (basado en representaciones simbólicas). De manera que el procesamiento de la información sensorial se orienta a la consecución de este objetivo.

Este fundamento es al mismo tiempo el punto fuerte y débil del paradigma deliberativo. Como punto fuerte, si se asume la existencia de un modelo interno que sea totalmente completo y correcto, también puede asumirse que el procesador de dicho modelo podrá planear y razonar sobre el entorno a cualquier nivel deseado. Sin embargo, el hecho de intentar conseguir y utilizar representaciones completas y precisas del mundo real como base de razonamiento, enfrenta al paradigma deliberativo con una serie de problemas que pueden resumirse en los siguientes puntos:

- Hipótesis del mundo cerrado. Se refiere a la suposición de un modelo interno totalmente completo y correcto, lo cual implica varios problemas:
    - El problema de cómo desarrollar las estructuras de conocimiento [Bates, 1992]. Debido a la complejidad que supone la percepción y reconocimiento de los objetos, uno de los principales objetivos de la Inteligencia Artificial ha sido la representación del conocimiento. Sistemas con mínimas capacidades de percepción necesitan modelos internos de representación al no poder extraer eficientemente la información que necesitan del entorno. Por ejemplo, un robot con sensores con mucho ruido pero con un mapa exacto del entorno podría moverse correctamente confiando casi por completo en su modelo interno. En estos casos, el modelo interno tiene que ser lo más completo y exacto posible.
    - El conocimiento no puede ser completo. Todas las características relevantes del mundo real no pueden reflejarse en el modelo interno usado, por lo que pueden planificarse acciones incorrectas debido a la falta de información.
    - El conocimiento en sistemas bajo dominio temporal tampoco puede ser correcto. Existe un problema de incertidumbre, el mundo es incierto, lo que es verdad en un momento dado puede no serlo después. Además, información a la que normalmente se accede puede estar degradada u oculta en determinadas ocasiones debido a cambios en las condiciones del entorno. Al planificar no se conoce el estado que el mundo tendrá en el futuro.

Incluso aunque se lograra un modelo interno totalmente completo y correcto, éste sería intratable con la tecnología existente [Chapman, 1987] debido al coste y complejidad necesarios para producirse el razonamiento en dicho modelo.

Las dificultades expuestas en la obtención de un modelo interno que sea completo y correcto se tratan de resolver centrándose más en el aprendizaje, lo cual implica realizar menos suposiciones acerca del conocimiento perfecto, creándose una balanza entre el aprendizaje y el conocimiento.

- Problema del marco. El efecto de las acciones del robot no es posible conocerlo a priori. Esto dificulta la planificación a largo plazo puesto que se producen diferencias entre el estado real del robot y el supuesto estado en el momento de la planificación.

- Problema de operación en tiempo real. La planificación de una tarea se ve como una búsqueda, a través del espacio del problema, de una secuencia de operadores que transformarán el estado inicial en el estado objetivo. Sólo cuando esta secuencia es encontrada se ejecuta el plan que probablemente, debido a los problemas descritos, tenga que volverse a replantear de forma continuada. Según esta filosofía, los datos leídos de los sensores tienen que "viajar" a lo largo de la arquitectura simbólica antes de provocar una reacción. Todo esto hace que las restricciones de tiempo real sean difíciles de cumplir. Por ejemplo, cuando es detectado un obstáculo es necesaria una reacción rápida, lo cual es difícil de conseguir si el sistema, para poder actuar, tiene que esperar todo el proceso de planificación descrito.

1.3. Paradigma Reactivo

El paradigma reactivo, inspirado en la observación del comportamiento reactivo de los animales, surgió en 1986 [Brooks, 1986] como una reacción frente a los problemas del paradigma deliberativo. Presenta una organización Sentir-Actuar (S-A) (fig.3). Mientras que el paradigma deliberativo asume que la entrada que origina una acción es siempre el resultado de un razonamiento o plan, el paradigma reactivo asume que la entrada que origina una acción siempre es la salida directa de un sensor (tabla 3).

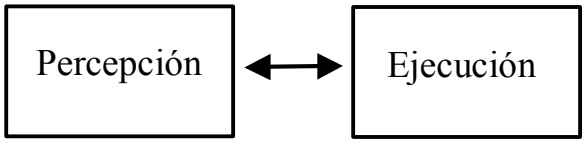


Figura 3: *Filosofía del paradigma reactivo: Sentir-Actuar (S-A)*

Se basa en la existencia de múltiples instancias de procesos Sentir-Actuar, denominados patrones de comportamientos, que se ejecutan concurrentemente. Un

patrón de comportamiento toma la salida de un sensor y genera una acción a realizar independientemente de lo que realicen los otros comportamientos. Por ejemplo, puede haber un comportamiento que dirija al robot en línea recta para alcanzar un objetivo y al mismo tiempo otro que lo haga girar 90º para esquivar un obstáculo, el resultado final podría consistir en que el robot hiciera una combinación de ambos comportamientos resultando un movimiento hacia adelante con giro de 45º. El resultado final emerge de la combinación de ambos comportamientos, lo que se denomina comportamiento emergente.

Primitivas	Entradas	Salidas
PERCEPCIÓN	Valores de los sensores	Información sensorial
RAZONAMIENTO	Información (sensorial o cognitiva)	Directivas-tareas
EJECUCIÓN	Información sensorial o directivas-tareas	Comandos de actuación

Tabla 3: *Relación de las tres funciones elementales o primitivas de un robot según el paradigma reactivo*

La aproximación basada en comportamientos, en su forma más pura, no utiliza modelos internos simbólicos de representación del mundo real. Según Brooks [Brooks, 1991a y 1991b], el problema de incertidumbre no puede resolverse mediante el uso de modelos. La solución consiste en la utilización de sistemas basados en comportamientos donde, en lugar de modelos, es el propio entorno el que proporciona cualquier información necesaria.

La ventaja de estos sistemas, debido a que no utilizan modelos internos simbólicos y a que son totalmente reactivos (no existe la planificación o razonamiento), consiste en que no necesitan realizar accesos y procesamiento de estructuras de representación por lo que son más rápidos y predecibles que los sistemas que sí tienen que hacerlo y pueden utilizarse para cumplir restricciones de tiempo real estricto, a diferencia de los sistemas deliberativos donde la operación en tiempo real se presenta como un problema. Además, en los sistemas reactivos tampoco existen los problemas de hipótesis del mundo cerrado y problema del marco, característicos del paradigma deliberativo, puesto que reaccionan directamente a todo aquello que es perceptible y no confían en creencias y predicciones. Sin embargo, en el paradigma reactivo existe el problema sensorial, considerar únicamente la información sensorial actual para generar las acciones de control puede ser problemático debido a los ruidos en los sensores y actuadores y a factores físicos como rozamientos y deslizamientos.

Los sistemas realizados han tenido gran éxito, construyéndose robots que imitan conductas aparentemente inteligentes de animales e insectos. Sin embargo, estos sistemas no permiten el desarrollo de robots que realicen tareas complejas al no

disponer de ningún sistema de razonamiento. Un sistema sin ningún modelo interno que no es capaz de planificar operaciones de forma eficiente, está en desventaja frente al mismo sistema con dicho modelo.

Por un lado, almacenar toda la información disponible no es una opción viable para un sistema inteligente [Chown, 1999]. Además, es necesario economizar dado el tamaño de memoria disponible en comparación con la complejidad del mundo. Por otro lado, actuar sin almacenar ningún tipo de información sería un grave error, puesto que los beneficios de poder disponer de algún tipo de información almacenada de una forma coherente o modelo interno son varios, principalmente la habilidad de predecir sucesos en el entorno antes de que éstos ocurran usando la información de las experiencias pasadas y permitiendo ir por delante de la información sensorial actual. Además, el uso de representaciones internas puede reducir el cómputo necesario cuando, por ejemplo, se reconoce un lugar familiar, en dicho caso podría no ser necesario volver a realizar su inspección.

Actualmente el paradigma reactivo sigue utilizándose, pero desde 1992 la tendencia ha sido el uso del paradigma híbrido.

1.4. Paradigma Híbrido

El paradigma híbrido deliberativo/reactivo, que surgió en 1990 [Arkin, 1990], se basa fundamentalmente en el paradigma reactivo por su rápida velocidad de ejecución y consiguiente adecuación a los sistemas de tiempo real estricto, pero también incorpora un nivel de razonamiento basado en modelos internos.

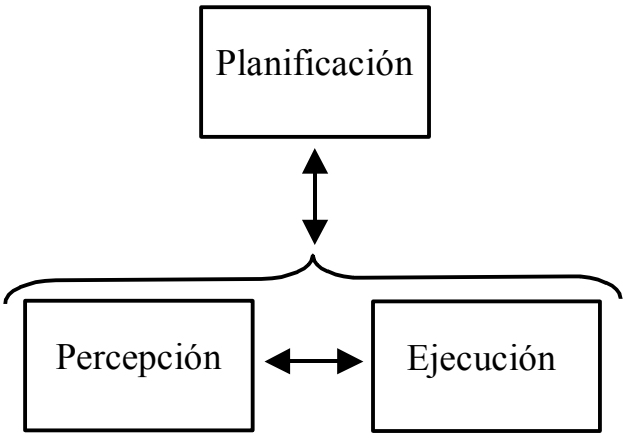


Figura 4: *Filosofía del paradigma híbrido: Planear, Sentir-Actuar (P, S-A)*

Según el paradigma híbrido, el robot primero planea (razonamiento) la mejor forma de descomponer la tarea a realizar en varias subtareas, y selecciona los comportamientos adecuados para alcanzar cada subobjetivo. Posteriormente, el robot empieza con la ejecución de la misma forma que en el paradigma reactivo. Este tipo

de organización (fig.4) puede denotarse por "Planear, Sentir-Actuar" (P, S-A), donde la coma indica que en un primer paso se realiza la planificación y posteriormente se ejecutan los patrones de comportamientos (Sentir-Actuar).

La organización y procesamiento de la información sensorial en el paradigma híbrido también es una mezcla de los paradigmas deliberativo e híbrido (tabla 4). Esta información es utilizada tanto a nivel de planificación, para la construcción de un modelo interno del entorno del robot que se utiliza para la descomposición de tareas y selección de comportamientos, como a nivel de ejecución de dichos comportamientos, el valor de cada sensor es utilizado por cada comportamiento que lo necesite.

Primitivas	**Entradas**	**Salidas**
RAZONAMIENTO	Información (sensorial y/o cognitiva) →	Directivas-acciones
"PERCEPCIÓN & EJECUCIÓN" (comportamientos)	← Valores de los sensores	→ Comandos de actuación

Tabla 4: *Relación de las tres funciones elementales o primitivas de un robot según el paradigma híbrido*

Dependiendo del nivel en el que se procese la información, los tiempos requeridos son distintos. En el nivel de ejecución (reactivo) es necesario una respuesta en tiempo real rápida (por ejemplo para poder reaccionar ante la presencia de un obstáculo), mientras que en el nivel de planificación (deliberativo), donde las acciones a realizar requieren un mayor coste temporal, no se precisa de tiempos de respuesta rápidos aunque sí es necesario un conocimiento temporal de la información, por ejemplo para constatar la validez de dicha información en el momento actual o para realizar de forma adecuada la construcción de los modelos internos. Estas características requieren de un sistema de comunicaciones que soporte las restricciones temporales de la información dependiendo del lugar donde se procese, siendo uno de los temas centrales de esta tesis.

El valor potencial de los modelos internos puede resumirse en las siguientes características [Chown, 1999]:

- Los modelos tienen que ser a pequeña escala. Demasiada información puede ser en algunas ocasiones tan ineficiente como poca información. Un modelo útil tiene que poseer únicamente información relevante.

- Los modelos tienen que ser sintéticos. Los conocimientos y experiencias del pasado tienen que almacenarse de forma que puedan ser utilizados para tratar acciones del presente y del futuro.

- Los modelos tienen que ser predictivos. Pueden predecirse los resultados de las acciones antes de tomarlas evitando de esta forma aquellas que fueran peligrosas.

En cuanto al papel que puede desempeñar el nivel deliberativo o de razonamiento en el paradigma híbrido, hay básicamente dos opciones [Simó, 1997a y b]:

- Utilizar el nivel deliberativo para razonar sobre qué comportamiento debe exhibir el robot móvil. Las conclusiones de este nivel son la conmutación de comportamientos. La utilización del nivel deliberativo como conmutador de comportamientos es una idea antigua y actualmente se ha visto rechazada por la mayor parte de los investigadores quienes han adoptado el modelo de comportamientos emergentes.

- Utilizar el nivel deliberativo en la planificación de operaciones abstractas y dar como resultado objetivos más simples, alcanzables por la capa de procesamiento reactivo. Las conclusiones de este nivel son el modelo de comportamientos emergentes, el cual considera que el desempeño de un comportamiento por parte del robot, no implica la existencia de un proceso interno orientado a imprimir dicho comportamiento al robot. El comportamiento que exhibe el robot es, en todo momento, una combinación o composición de comportamientos básicos. Todos los comportamientos básicos se considera que se ejecutan en paralelo y su contribución al comportamiento exhibido viene dada por unos coeficientes llamados "factores de motivación", que también se calculan en paralelo, de esta forma, no existe conmutación de comportamientos, sino en todo caso, transiciones suaves entre comportamientos emergentes.

En ambas tendencias la operación de tiempo real se ve asegurada por el nivel reactivo, pero el nivel deliberativo debe tener en cuenta explícitamente el tiempo en su razonamiento ya que con frecuencia el problema más importante es la sincronización entre los niveles.

En la actualidad, debido a que los sistemas híbridos tratan de aprovechar las ventajas de los paradigmas deliberativo y reactivo minimizando los problemas planteados en los mismos, el paradigma híbrido es la opción más empleada en la especificación de arquitecturas distribuidas para el control de robots móviles [Dorigo, 1998]. Por ello, conviene describir algunas de las arquitecturas más representativas destacando sus características principales.

A continuación, por su importancia en la descripción e implementación modular de varias de las arquitecturas híbridas más significativas en la actualidad [Murphy, 2000], se presentan los conceptos de la "teoría de esquemas" en los cuales además se basa la arquitectura que se propone en esta tesis. Posteriormente, se comentan diferentes tipos de arquitecturas híbridas existentes mediante una descripción breve de las más representativas.

1.4.1. Teoría de Esquemas

El término comportamiento se ha definido como una correspondencia de los valores de los sensores con el conjunto de acciones a realizar para alcanzar un objetivo, siendo el componente principal de la inteligencia en la mayoría de los sistemas.

La "teoría de esquemas" se ha utilizado en psicología desde comienzos del siglo XX, pero no ha sido hasta hace unos años [Arbib, 1986; Arkin, 1998] cuando se ha utilizado en el campo de la robótica móvil. Los esquemas han sido utilizados por los psicólogos como un medio de expresar la unidad básica de actividad. Un esquema consiste en el conocimiento de cómo actuar y/o percibir así como del procesamiento necesario para realizar una actividad.

En la "teoría de esquemas" aplicada a la robótica móvil, un patrón de comportamiento se compone de al menos un esquema motor y un esquema de percepción, además del conocimiento necesario para coordinar múltiples esquemas. Esta separación entre percepción y acción, implica una organización de los comportamientos mediante la combinación de esquemas motores y esquemas de percepción (fig.5), dando lugar a un modo de implementación. La percepción en comportamientos sigue dos reglas, bien como activadora de los comportamientos, bien como guía de los mismos.

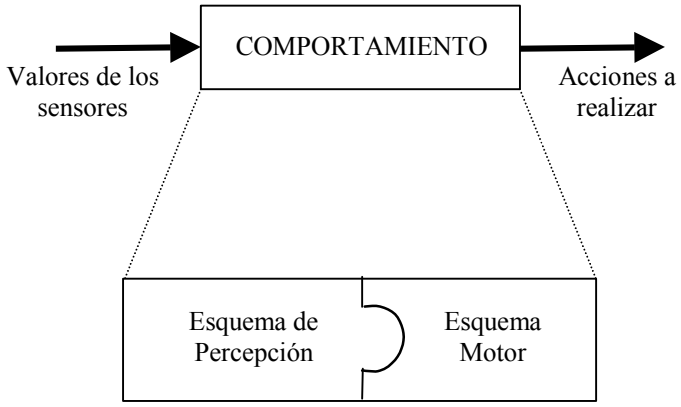


Figura 5: *Teoría de Esquemas: descomposición de los patrones de comportamientos en esquemas de percepción y esquemas motores*

La "teoría de esquemas" permite la representación orientada a objetos de los comportamientos, donde los esquemas motores y los esquemas de percepción son objetos que actúan independientemente y se combinan formando los comportamientos. Un mismo esquema de percepción podría combinar con distintos esquemas motores. Distintos comportamientos también pueden ser combinados, secuenciados o inhibidos entre ellos.

1.4.2. Arquitecturas

En esta sección se realiza una descripción general de las arquitecturas híbridas más representativas inspiradas en los robots autónomos y que sirven como marco de referencia para contextualizar el presente trabajo.

Los criterios [Arkin, 1998] que pueden seguirse para comparar y evaluar diferentes arquitecturas son los siguientes:

- Modularidad: acerca de la independencia de los distintos componentes de la arquitectura así como su capacidad de reutilización y escalabilidad.
- Adaptabilidad: cómo se adapta la arquitectura a la aplicación para la cual ha sido pensada.
- Portabilidad: facilidad de portabilidad a otros dominios, es decir, cómo se adapta la arquitectura a otras aplicaciones.
- Robustez: acerca de la vulnerabilidad de la arquitectura.

1.4.2.1. Arquitectura AURA

La arquitectura Aura (*Autonomous Robot Architecture*) es la más antigua de las arquitecturas híbridas para el control de robots autónomos. Diseñada e implementada por Ronald Arkin [Arkin, 1990 y 1998], está basada en la "teoría de esquemas". Está formada por cinco subsistemas que implementan tanto el nivel de control reactivo como el nivel deliberativo. El nivel deliberativo lo componen el subsistema de planificación y el subsistema cartográfico. El control reactivo se basa en el modelo de comportamientos emergentes, existiendo un conjunto limitado de comportamientos primitivos (esquemas), que por composición pueden dar lugar a comportamientos complejos. Los esquemas de Arkin se caracterizan por los siguientes puntos:

- Los esquemas forman una red dinámica de procesos.
- Cada esquema es un agente computacional independiente.
- La selección de los patrones de comportamientos se realiza en el nivel deliberativo de planificación, usando información sobre la misión, el entorno y el estado interno del robot.
- En base a la lista de patrones de comportamientos proporcionada por el nivel deliberativo, el nivel reactivo selecciona los esquemas de percepción y esquemas motores que formarán los comportamientos primitivos.
- El comportamiento exhibido por el robot se obtiene por la composición de los comportamientos primitivos.

- Su diseño está motivado por el estudio de sistemas biológicos y enfoques psicológicos y neurofisiológicos de comportamientos.

Los subsistemas de la arquitectura son los siguientes (fig.6):

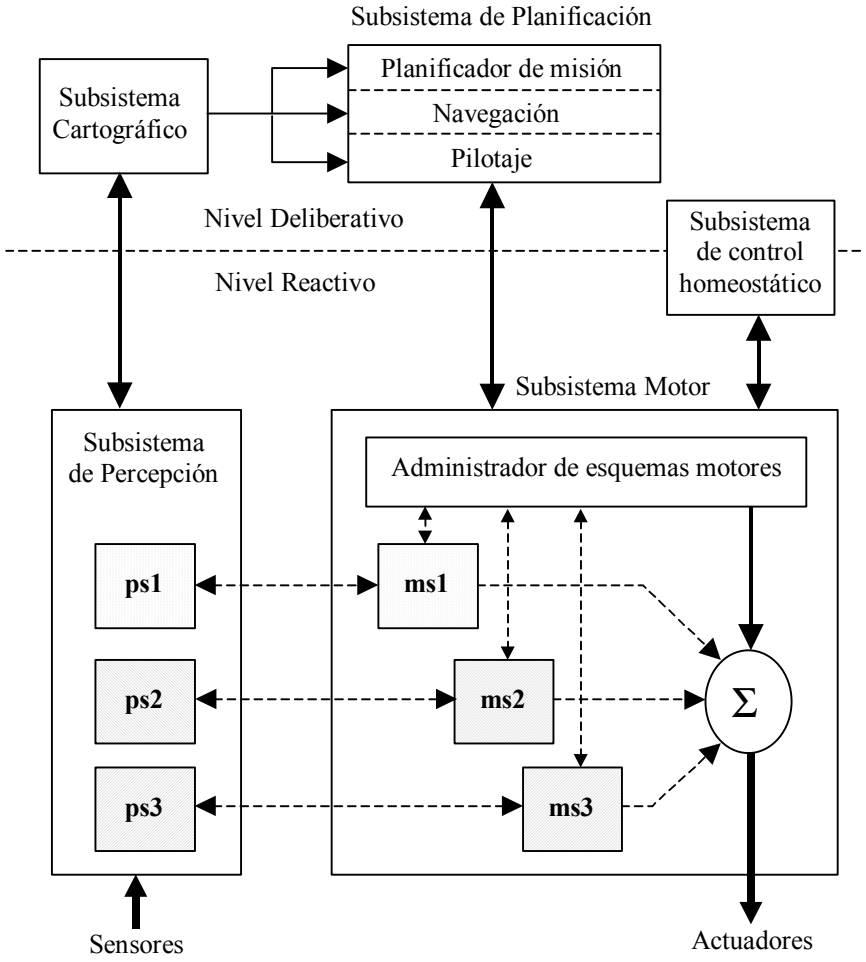


Figura 6: *Arquitectura híbrida AURA: estructura de dos niveles deliberativo y reactivo constituidos por cinco subsistemas y generación del comportamiento por composición de esquemas*

- Subsistema de percepción: se encarga de adquirir a través de los esquemas de percepción (ps) la información suministrada por los sensores, filtrarla y enviarla a los subsistemas cartográfico y motor. Dispone de una librería con todos los esquemas de percepción disponibles.

- Subsistema cartográfico: gestiona dos zonas de memoria. La memoria a largo plazo (LTM), en la que se encuentra información inicial y fija del entorno, y la memoria a corto plazo (STM), en la que se crea un modelo dinámico del entorno basado en la información adquirida por los sensores.

- Subsistema de planificación: tiene una estructura jerárquica de tres niveles, en su nivel inferior se hace uso de los comportamientos primitivos:

    - Planificador de misiones: tiene la responsabilidad de la planificación de alto nivel ofreciendo una interfaz con el usuario que permite definir las tareas a realizar.

    - Módulo de navegación: realiza la confección de una trayectoria inicial interactuando con el subsistema cartográfico.

    - Módulo de pilotaje: recibe del módulo de navegación una trayectoria e interactúa con el subsistema motor y administrador de esquemas motores (ms) para realizar la selección de los comportamientos primitivos apropiados que permitan al robot llevar a cabo la trayectoria especificada. Una vez seleccionados los comportamientos, se envían parametrizados al subsistema motor que se encarga de ejecutarlos. Los esquemas elegidos llevan asociados esquemas de percepción que proporcionan la información necesaria para calcular su salida. La elección del esquema de percepción que lleva asociado un esquema motor no es fija, y se realiza en función del entorno, las condiciones internas y los requerimientos del robot.

- Subsistema motor: dispone de una librería con todos los esquemas motores disponibles. El administrador de esquemas motores accede a las librerías de los subsistemas de percepción y motor para obtener los esquemas de los comportamientos seleccionados en el nivel deliberativo. Un mismo comportamiento puede estar formado por varios esquemas motores coordinados mediante máquinas de estados finitos. Además, dentro de los esquemas también se permite el uso de memoria, conocimiento y representaciones internas. En el subsistema motor se calcula la contribución, al movimiento total, de cada uno de los comportamientos seleccionados y realiza la composición de los mismos basándose en campos potenciales.

- Subsistema homeostático: involucra a las tareas de supervivencia del robot en circunstancias peligrosas. El control homeostático se define como la capacidad de mantener un estado de equilibrio entre los diferentes elementos o grupos de elementos de un organismo. Para obtener una verdadera autonomía en el robot, éste debe ser capaz de alterar su comportamiento en función de la evolución de su entorno y de su estado interno.

1.4.2.2. Arquitectura SFX

La arquitectura SFX (*Sensor Fusion Effects*), desarrollada por Murphy [Murphy, 2000], es una extensión de la arquitectura AURA que incorpora procesos de fusión sensorial y respuesta ante fallos de sensores. La robustez es una de las características principales de la arquitectura. La información sensorial está disponible tanto en el nivel reactivo como en el deliberativo mediante el uso de estructuras de datos tipo

pizarra [Nii, 1989], comunes en numerosos sistemas de Inteligencia Artificial para formar estructuras de conocimiento global [Mensch, 1993].

En la figura 7 se presenta la estructura de la arquitectura y la conexión mediante pizarras de sus componentes deliberativos y reactivos.

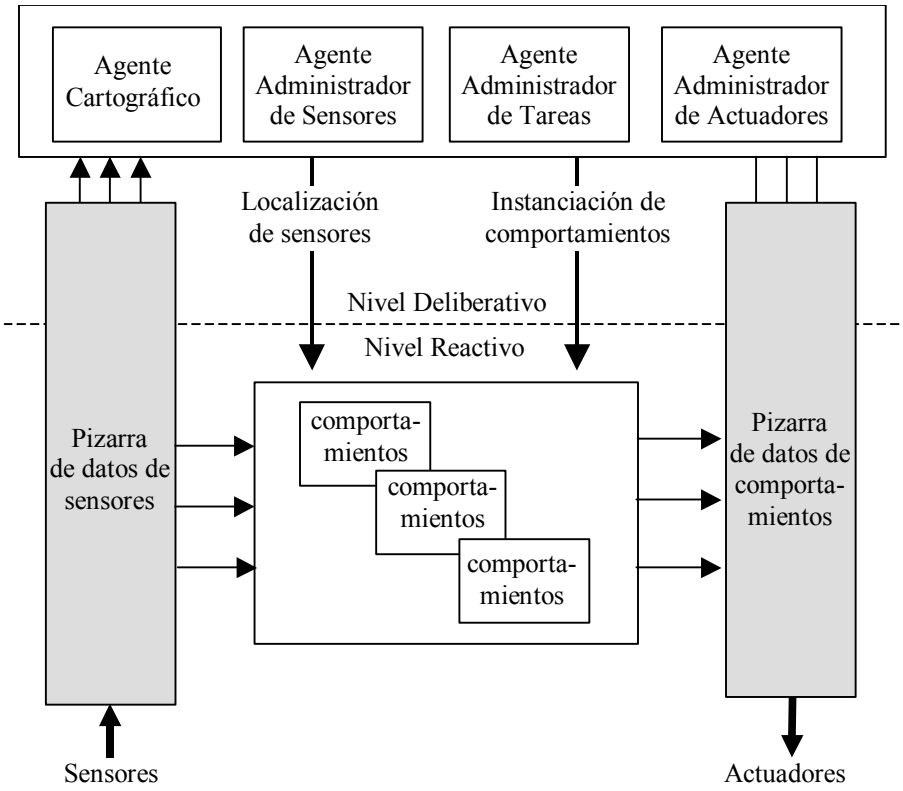


Figura 7: *Arquitectura híbrida SFX: estructura de pizarra con agentes independientes y generación del comportamiento por inhibición*

El nivel deliberativo está formado por componentes software independientes denominados agentes que están especializados en determinadas funciones y pueden interactuar entre ellos. El agente principal es el Planificador de misiones, interactúa con el usuario y especifica las directrices de la misión a los otros agentes del nivel deliberativo. El objetivo de los agentes es encontrar los comportamientos necesarios que aseguren la realización de la misión cumpliendo con las restricciones establecidas. Asimismo, existen agentes para determinar los sensores que utilizarán los esquemas de percepción y esquemas motores de los distintos comportamientos. Si un sensor fallara, estos agentes podrían averiguar la causa del fallo e intentar solucionarlo o podrían decidir el uso de algún sensor alternativo mediante la incorporación, en el comportamiento, del esquema de percepción correspondiente.

El nivel reactivo se divide en dos capas identificadas por los comportamientos estratégicos y por los comportamientos tácticos. A diferencia de la arquitectura AURA, donde se utilizan campos potenciales para combinar los comportamientos,

SFX utiliza un método de filtrado donde algunos comportamientos fundamentales inhiben a otros comportamientos. La idea es similar al proceso realizado en la arquitectura reactiva Subsumption de Brooks [Brooks, 1986] pero en este caso los comportamientos de la capa inferior (comportamientos tácticos) inhiben a los comportamientos de la capa superior (comportamientos estratégicos). Por ejemplo, la velocidad emergente en la arquitectura AURA es siempre el resultado de la suma ponderada de todos los comportamientos activos, mientras que en la arquitectura SFX existe un control de la velocidad máxima a través de un comportamiento táctico específico (fig.8). Esto quiere decir que los comportamientos estratégicos de la capa superior proporcionan una velocidad al igual que ocurre en la arquitectura AURA y, después, el comportamiento táctico de la capa inferior filtra este valor dependiendo de si es superior al valor máximo o no. La velocidad resultante siempre será el valor menor de ambos. FSX basa esta técnica en la idea de que la mayoría de las tareas pueden expresarse mediante un solo comportamiento estratégico y uno o varios tácticos que actúen de filtros del estratégico.

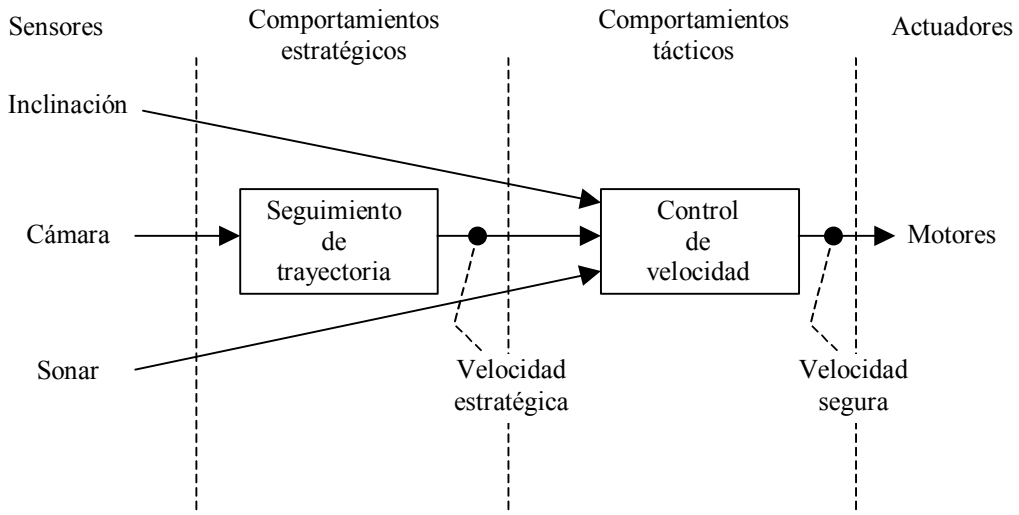


Figura 8: *Inhibición de comportamientos en la arquitectura FSX: los comportamientos tácticos inhiben a los comportamientos estratégicos*

1.4.2.3. Arquitectura 3T

La arquitectura 3T (*3-Tiered*) ha sido desarrollada en la NASA [Bonasso, 1997] y está dividida en tres niveles: uno claramente reactivo, otro claramente deliberativo y otro que sirve de interfaz entre los dos (fig.9). Los tres niveles pueden ejecutarse en computadores diferentes. Los componentes de la arquitectura se clasifican dentro de un nivel u otro dependiendo de sus necesidades temporales de ejecución.

El nivel superior o deliberativo está formado por el planificador, éste genera las tareas a realizar en base a los objetivos establecidos. Se basa en distintos módulos: planificador de misiones, cartográfico y monitor.

El nivel interfaz está formado principalmente por el secuenciador, recibe del nivel deliberativo las tareas a realizar y selecciona desde una librería los comportamientos primitivos (denominados skills) necesarios para ejecutarlas.

El conjunto de comportamientos o skills seleccionados forman el nivel inferior o reactivo denominado controlador de skills. Una característica importante del nivel reactivo consiste en que los skills tienen asociados eventos que cuando se producen activan a los correspondientes comportamientos.

La ubicación de los componentes de la arquitectura se realiza en función de los tiempos necesarios de ejecución. Los algoritmos con velocidades lentas se sitúan en el planificador mientras que los algoritmos rápidos se ubican en el controlador de skills. Este planteamiento determina el diseño de la arquitectura en función de las prestaciones de computación existentes. Por ejemplo, debido a su condición de algoritmos lentos, se situaron procesos de sensores de visión en el nivel de planificación en lugar de hacerse en el nivel reactivo por su condición sensorial.

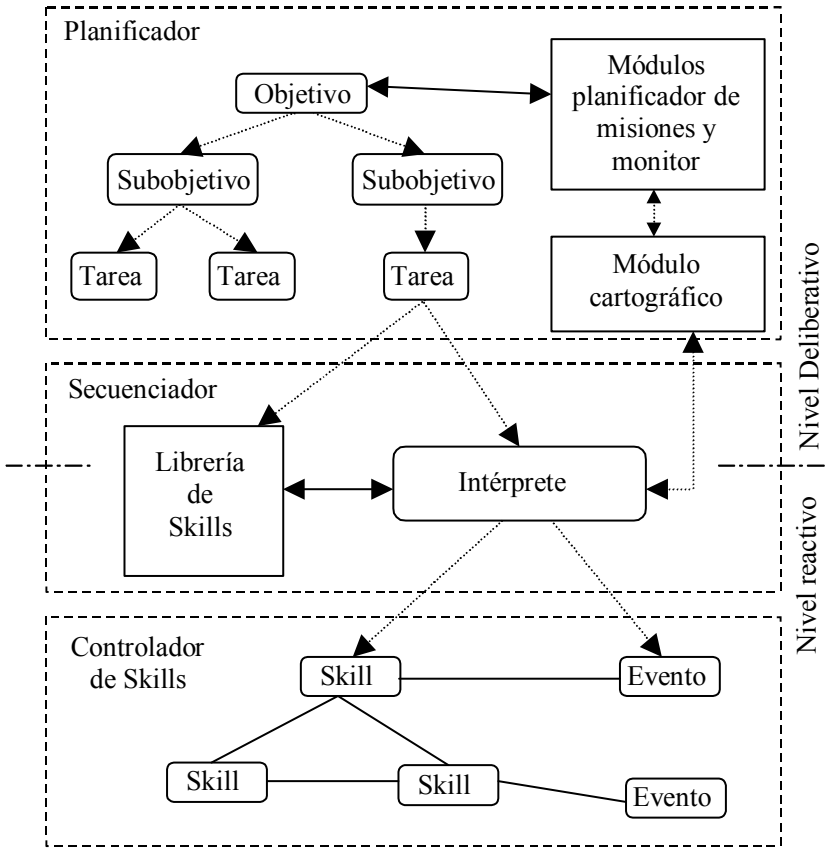


Figura 9: *Arquitectura híbrida 3T: estructura de tres niveles según restricciones temporales y generación del comportamiento por activación a través de eventos de los skills o comportamientos primitivos*

1.4.2.4. Arquitectura GLAIR

La arquitectura GLAIR [Lammens, 1993] está organizada en tres niveles (fig.10): el nivel de conocimiento (KL), el nivel Perceptuo-Motor (PML), y el nivel Senso-Actuador (SAL). GLAIR integra un sistema simbólico tradicional con un sistema físico en una arquitectura basada en comportamientos. Su característica principal es la presencia de tres niveles distintos con diferentes representaciones y mecanismos de implementación, en particular, la presencia explícita de un nivel de conocimiento. La representación, el razonamiento (incluida la planificación), la percepción y la generación de comportamientos se distribuyen a lo largo de los tres niveles.

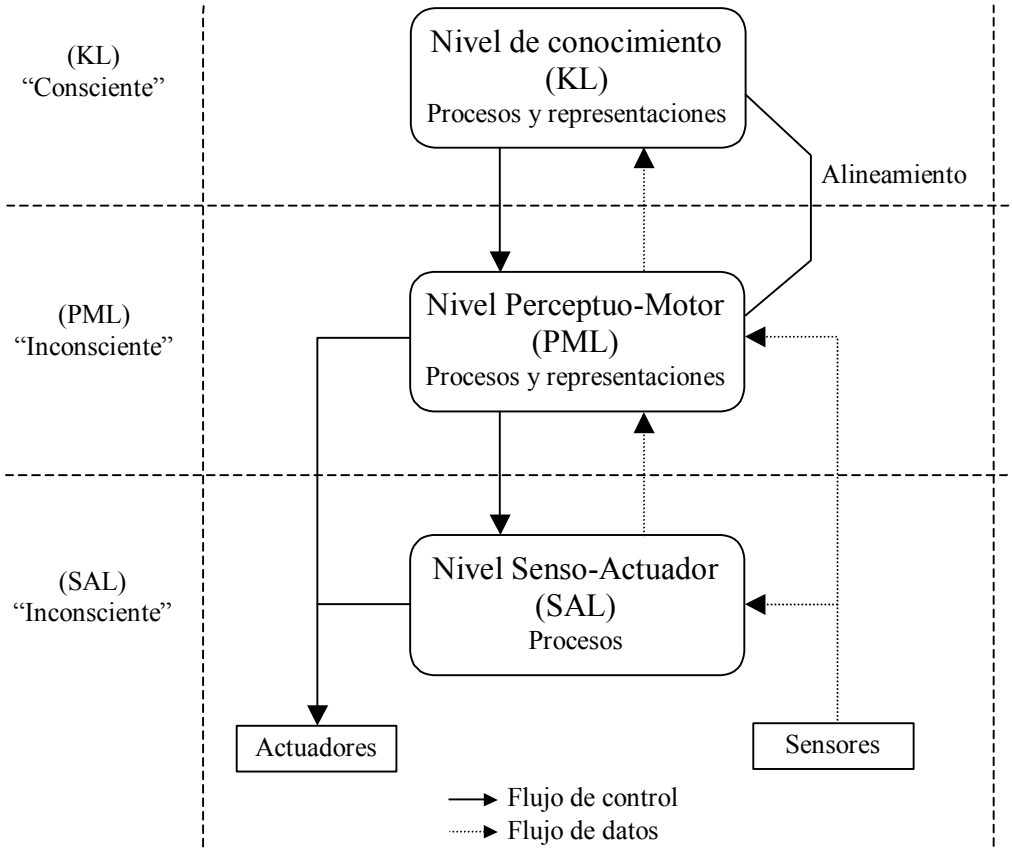


Figura 10: *Arquitectura híbrida GLAIR: estructura de tres niveles según grado de representación y generación del comportamiento por inhibición*

El interés se centra en los robots que se encuentran empotrados en un entorno dinámico con el que necesitan interactuar y reaccionar continuamente exhibiendo un comportamiento inteligente. Las características de GLAIR son:

- Se distingue entre el razonamiento consciente y el procesamiento inconsciente de los niveles Perceptuo-Motor y Senso-Actuador.

- Los niveles de la arquitectura son semiautónomos y se procesan en paralelo.

- El razonamiento consciente tiene lugar a través del razonamiento y la representación explícita de conocimiento.

- El razonamiento consciente guía el comportamiento inconsciente, y los niveles inconscientes, que están continuamente ocupados en el procesamiento perceptual y motor, pueden advertir al nivel consciente de los eventos importantes, tomando el control si fuera necesario. El control y la generación de comportamientos están distribuidos por capas pero no necesariamente de arriba hacia abajo.

- Los mecanismos de bajo nivel pueden expulsar a los de alto nivel. Esto es una forma de inhibición que depende de la capa en la que se encuentren los comportamientos.

- Hay una correspondencia entre los términos en el sistema de razonamiento y representación del conocimiento por una parte, y las sensaciones y percepciones de los objetos, propiedades, eventos y estado del mundo, y capacidades motoras, por otra. A esta correspondencia se le denomina alineamiento.

A continuación se realiza una breve descripción de los niveles que constituyen la arquitectura GLAIR:

- Nivel de conocimiento (KL): el nivel de conocimiento contiene un sistema de planificación y utiliza una representación selectiva de los objetos, eventos y estado del mundo según el curso de acción. El objetivo es modelar sólo las entidades relevantes para la interacción del robot con el mundo. Las representaciones en el nivel de conocimiento son necesarias para razonar sobre las entidades, mientras que las representaciones en el nivel Perceptuo-Motor son necesarias para la interacción física entre entidades.

- Nivel Perceptuo-Motor (PML): el nivel Perceptuo-Motor usa una representación más fina de los objetos, eventos y estados del mundo. En este nivel se debe proporcionar suficiente detalle para hacer posible el control preciso de los actuadores, y los sensores deben ser capaces de suministrar este nivel de detalle para situaciones u objetos determinados. El PML está parcialmente alineado con el KL, en el sentido en que hay una correspondencia entre los identificadores de los objetos en el KL y los objetos del PML. La representación en el PML está englobada con el robot, es decir, depende de la estructura física del robot, sus dimensiones y características particulares. En este nivel se generan varios comportamientos.

- Nivel Senso-Actuador (SAL): el nivel Senso-Actuador es el nivel de las acciones motoras y sensoriales primitivas. En este nivel no hay ninguna representación declarativa explícita de los objetos, sólo hay declaraciones

procedurales (para los actuadores) y datos (en el caso de los sensores). En este nivel se sitúan los "reflejos" que se consideran como bucles de bajo nivel entre los sensores y los actuadores, operando independientemente de los niveles superiores y capaces de expulsar las acciones de éstos.

1.4.2.5. Arquitectura SAPHIRA

La arquitectura Saphira [Konolige, 1998] es un sistema integrado de sensorización y control para las aplicaciones robóticas. Como componente central se encuentra el LPS (espacio de percepción local), una representación geométrica del entorno existente alrededor del robot. Debido a que diferentes tareas necesitan distintas representaciones, el LPS está diseñado para acomodarse a varios niveles de interpretación de la información sensorial.

El LPS proporciona al robot conocimiento sobre su entorno y es fundamental para las tareas de fusión de información, planificación del movimiento y la integración de la información del mapa. La arquitectura de control y percepción hace constante referencia al LPS, el cual proporciona a la arquitectura coherencia en la representación.

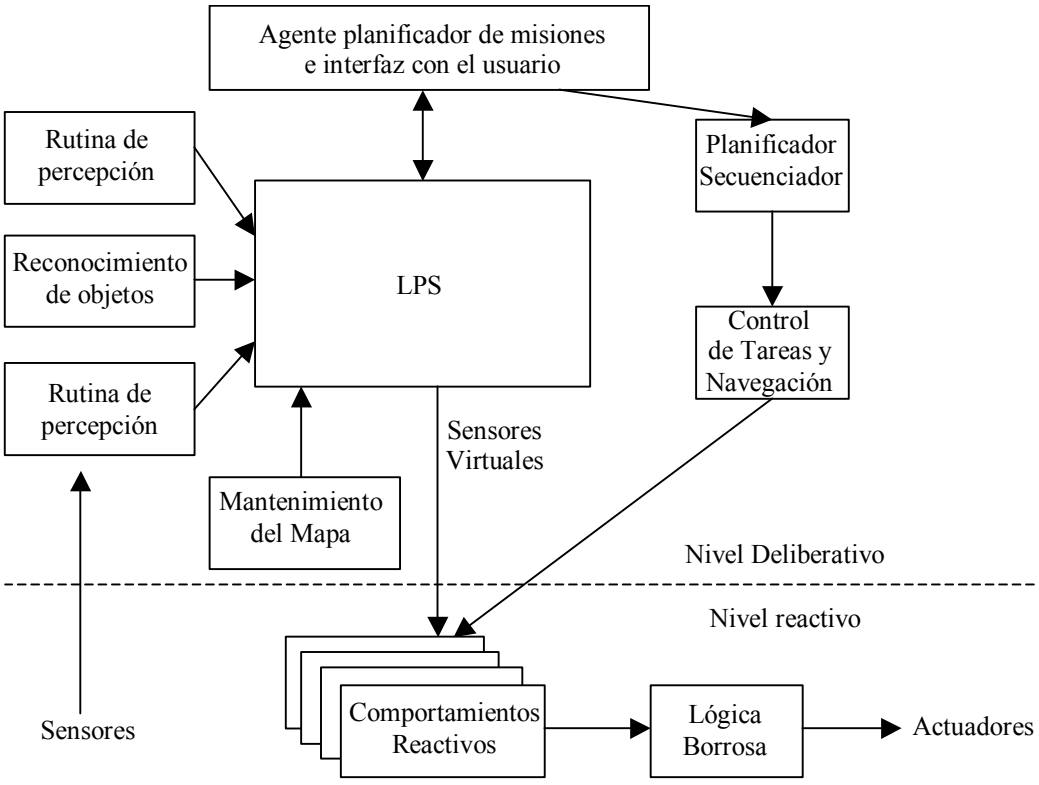


Figura 11: *Arquitectura híbrida Saphira: estructura de datos común con componentes independientes y generación del comportamiento por secuenciación*

La organización, como se puede observar en la figura 11, es parcialmente vertical y parcialmente horizontal. La organización vertical se puede observar tanto en la percepción (lado izquierdo), como en la acción (lado derecho). Varias rutinas de percepción se encargan de añadir información sensorial al LPS y de procesarla para producir información que puede ser utilizada para el reconocimiento de los objetos y la navegación. Los comportamientos más complejos que realizan acciones dirigidas por objetivos son utilizados para guiar el comportamiento reactivo. En el nivel de tareas, los comportamientos son secuenciados y su progreso es monitorizado a través de eventos en el LPS. La organización horizontal aparece porque los comportamientos pueden elegir la información apropiada del LPS. Los comportamientos críticos en el tiempo, como evitar obstáculos, confían más en un procesamiento simple de los sensores al estar disponible rápidamente.

En el nivel de control, la arquitectura Saphira se basa en el comportamiento: el problema de control se descompone en pequeñas unidades de control llamadas comportamientos básicos (como evitar obstáculos o seguir un pasillo). Una de las características distintivas de Saphira es que los comportamientos están escritos y combinados utilizando técnicas basadas en la lógica borrosa.

Los diferentes componentes de la arquitectura son independientes y no tienen por qué ejecutarse en el mismo nodo correspondiente al robot que están controlando.

1.4.3. Necesidad del control temporal de las comunicaciones

En las arquitecturas híbridas para el control de robots móviles, los comportamientos son procesos paralelos y distribuidos cuya ejecución es dirigida por las tareas deliberativas. Los requerimientos temporales entre las actividades reactivas y deliberativas son diferentes. Además, los requerimientos temporales de los comportamientos son variables y dependen del entorno y/o parámetros de aplicación. Mientras que las tareas reactivas presentan restricciones de tiempo real estricto, en las tareas deliberativas las restricciones son de tiempo real no estricto. Cuando alguna tarea deliberativa no cumple algún tiempo de entrega, el sistema no debe llegar a ninguna situación catastrófica porque las tareas reactivas tienen que asegurar el control del mismo.

El diseño de una arquitectura basada en el paradigma híbrido deberá soportar la naturaleza distribuida de la información sensorial y de los comportamientos. El sistema de comunicaciones será la base para la construcción de los esquemas de percepción y motores, y deberá ser extensible al nivel de computación deliberativo.

Todos estos requerimientos implican la necesidad de un sistema de comunicaciones que permita la disponibilidad de la información en todos los niveles asegurando las restricciones temporales de cada uno de ellos [Posadas, 2000 y 2002].

Las comunicaciones en tiempo real estricto son necesarias para la ejecución de los comportamientos. En este caso, el sistema o infraestructura de comunicaciones

debe caracterizarse por la predecibilidad que permita asegurar los tiempos de transmisión de los datos, lo cual suele implicar una representación rígida de sus estructuras. Por otro lado, las comunicaciones en tiempo real no estricto son necesarias en la computación deliberativa. En este caso, el sistema o infraestructura de comunicaciones debe caracterizarse por permitir la transmisión de estructuras de datos más complejas con diferentes representaciones aunque ello implique que no pueda asegurar los tiempos de transmisión. Como se ha comentado anteriormente, la información sensorial debe estar presente en ambos tipos de infraestructura. Esta información es utilizada en el nivel deliberativo para la construcción de un modelo global del mundo orientado a tareas. La información de este modelo necesita información temporal asociada a los objetos distribuidos para permitir la fusión de datos tanto espacial como temporal. De este modo, la información temporal de los objetos distribuidos debe estar disponible en todas las infraestructuras de comunicación, a pesar de las diferencias existentes en predecibilidad temporal.

1.5. Conclusiones

En este capítulo se ha realizado un estudio de los distintos paradigmas que abordan la inteligencia en los robots.

El paradigma deliberativo se basa en el razonamiento aplicado sobre una representación simbólica del conocimiento, mientras que el paradigma reactivo está basado en los comportamientos los cuales reaccionan (proporcionando valores para los actuadores) directamente y sin una planificación previa ante los valores de los sensores. Las arquitecturas deliberativas no son apropiadas ante restricciones de tiempo real mientras que las reactivas pueden acotarse temporalmente. A la hora de especificar una arquitectura para el control de robots móviles, se concluye que lo ideal es realizar una combinación de ambas.

Tras la descripción realizada de algunas de las arquitecturas híbridas mas representativas en la actualidad, se identifican varias ideas comunes en todas ellas:

- Separación de un nivel reactivo de otro deliberativo en función de las características de respuesta o reacción en tiempo real que poseen los distintos componentes que forman los niveles.

- El nivel deliberativo se caracteriza por estar formado por componentes software independientes que realizan las tareas de planificación, monitorización, generación de mapas de entorno, etc. Estos componentes no precisan de tiempos de respuesta acotados para la realización de sus funciones. Un mayor retardo en la respuesta de los mismos implica una degradación de las prestaciones pero no conducen al sistema a un estado catastrófico.

- El nivel reactivo se caracteriza por estar formado por los sensores, actuadores y comportamientos, los cuales sí requieren una respuesta en

tiempo real ante determinadas situaciones (por ejemplo, tomar las acciones necesarias para evitar un obstáculo). Sin embargo, esto no implica que no pueda haber componentes reactivos que estén ejecutándose con otras restricciones de tiempo real no estricto. Los niveles reactivos se diferencian por la forma en que realizan la combinación de los distintos comportamientos. Los comportamientos se ven afectados por el concepto de motivación que determina la importancia de éstos. Dependiendo de los valores de las motivaciones, que pueden ser dinámicos, el sistema exhibirá uno u otro comportamiento (conmutación) o una composición de los mismos (composición que resulta en un comportamiento denominado emergente).

- Tanto los componentes deliberativos como reactivos necesitan acceder a la información de los sensores para poder realizar su función. Esto implica la necesidad de un sistema de comunicaciones adecuado que conecte ambos niveles respetando las características temporales de los mismos.

No se ha encontrado en la bibliografía ninguna arquitectura donde no haya que especificar la ubicación física de los componentes, lo cual implica una fuerte dependencia del diseño con el hardware y los recursos disponibles. La ubicación predeterminada de los componentes de la arquitectura no puede cambiarse y adaptarse dinámicamente a los cambios que puedan producirse en el sistema. Por otro lado, los comportamientos que forman el nivel reactivo también están predefinidos a priori y en la mayoría de las arquitecturas éstos se obtienen mediante el acceso a librerías específicas.

Lo ideal sería que los componentes pudieran decidir su propia ubicación dependiendo de sus necesidades o restricciones temporales. En este sentido, los componentes reactivos o comportamientos tampoco tendrían por qué conocerse a priori a la hora de especificar o implementar la arquitectura, sino que éstos podrían crearse e incorporarse al sistema de forma dinámica.

En las arquitecturas basadas en los sistemas de tiempo real clásicos [Musliner, 1993], antes de la puesta en marcha del sistema, se realiza un estudio previo de la planificabilidad de los comportamientos existentes que van a tener que ejecutarse bajo restricciones de tiempo real estricto. En un sistema donde no se conozcan a priori los comportamientos que van a tener que ejecutarse, la visión del tiempo real cambia en el sentido que será el sistema quien tenga que adaptarse a los nuevos comportamientos en función de los recursos disponibles para asegurar la integridad del mismo. Por ejemplo, si el sistema no pudiera cumplir los periodos de ejecución de los comportamientos necesarios para que el robot vaya a una determinada velocidad esquivando los obstáculos que encuentre en su camino, podría optar por reducir la velocidad [Hassan, 2001] (con la consiguiente degradación de prestaciones pero asegurando la integridad del sistema mediante el uso de periodos de ejecución mayores) o podría optar por delegar a otros nodos la ejecución de comportamientos no críticos (en caso de existir) para así asegurar la ejecución de los estrictamente necesarios en ese momento.

La idea planteada implica la necesidad del diseño de una arquitectura que proporcione un nuevo nivel semántico, donde se especifiquen los componentes pero no su ubicación. Este ha sido uno de los objetivos propuestos en esta tesis.

Para la consecución de dicho objetivo, el diseño de la arquitectura requiere de la existencia de una infraestructura de comunicaciones adecuada, que permita no sólo la transmisión de estructuras de datos simples sino también la transmisión de estructuras más complejas como supondría la transmisión de agentes software permitiéndose la movilidad de su código.

En resumen, las características del sistema de comunicaciones que ofrezca la infraestructura necesaria para el diseño de la arquitectura planteada en los objetivos de esta tesis son:

- Acceso a todos los datos del sistema desde cualquier componente de la arquitectura independientemente del nivel al que pertenezca, respetando las características temporales de cada nivel.

- Caracterización temporal de dichos datos proporcionando la antigüedad de los mismos. Con esta información pueden realizarse procesos de fusión de datos espacio-temporal o decidirse la ubicación de los componentes de la arquitectura en función de dónde obtienen los mismos datos con menor coste temporal de comunicaciones.

- Transmisión y ejecución de agentes entre los diferentes nodos de la arquitectura. Los agentes podrán decidir su ubicación dependiendo de sus restricciones temporales y necesidades de comunicación para el acceso a los datos.

Con este fin, en el siguiente capítulo se realiza un repaso a las distintas tendencias de comunicación a lo largo de la historia haciendo hincapié en las tendencias actuales orientadas al diseño e implementación de componentes comunicacionales modulares (tan importantes en las especificaciones de las arquitecturas híbridas) y móviles (que permitirán desacoplar a los componentes de una ubicación fija). En concreto, a los sistemas de agentes y más específicamente a los sistemas de agentes móviles, por su importancia en la obtención de las necesidades planteadas, se les dedica un capítulo describiéndose mediante el estado del arte sus características más relevantes.

Capítulo 2

COMUNICACIONES EN SISTEMAS DISTRIBUIDOS

En este capítulo se realiza un breve repaso de los diferentes patrones seguidos en las comunicaciones en sistemas distribuidos. El objetivo es destacar las características básicas y principales de los modelos de comunicación tradicionales para así, posteriormente, poder abordar el estudio de modelos más modernos como es el empleo de agentes inteligentes.

2.1. Introducción

La evolución de los computadores personales (PC's) en términos de coste, potencia y robustez, y la evolución de las redes (Internet) en términos de seguridad, velocidad y fiabilidad, ha influido de forma muy significativa en el desarrollo y aplicación de nuevos modelos de comunicación, así como en la manera en que actualmente se diseñan y se desarrollan los sistemas de automatización y los sistemas de robótica [Brugali, 2002]. En los últimos años, grandes compañías como General Motors han iniciado acciones para reemplazar en el sector industrial los clásicos controladores lógicos programables (PLCs) por sistemas basados en PC ejecutando *software* de emulación de PLC. El uso de las redes e Internet está invadiendo sectores que estaban fuera de su alcance hasta hace algunos años: control de tiempo real, tele-manipulación y tele-operación de vehículos, acceso a información sensorial e integración en robótica móvil. La revolución de los PC's, las redes y los modelos de comunicación promete la integración sencilla de periféricos, el acceso universal a datos y recursos compartidos, y la reconfiguración a bajo coste de los sistemas distribuidos [IEEE 1451]. Realmente, todavía se requiere un gran esfuerzo en investigación y desarrollo. Los sistemas distribuidos son la línea de investigación

donde se estudian y experimentan las nuevas técnicas, modelos y metodologías para alcanzar dichos objetivos.

Fundamentalmente, existen dos modelos a seguir en las comunicaciones en sistemas distribuidos: el modelo Cliente-Servidor y el modelo de Grupos. Partiendo de estos patrones se derivan otros modelos y arquitecturas que constituyen un nivel de abstracción superior:

- Modelo RPC
- Modelo de Objetos Distribuidos: CORBA, Java, .NET
- Modelo de Código Móvil
- Modelo Multiagente

Los sistemas multiagente constituyen un nuevo paradigma de programación para el desarrollo de aplicaciones *software*. Actualmente los agentes son el centro de interés en muchas ramas de la ingeniería informática e inteligencia artificial, y están usándose en una amplia y creciente variedad de aplicaciones. Las ventajas que se persiguen con la consolidación de la tecnología de agentes son la resolución de problemas de una forma significativamente mejor que la forma utilizada hasta el momento, e incluso la resolución de problemas que no habían podido resolverse por no disponerse de la tecnología adecuada o porque la utilización de la existente implicaba un coste excesivo.

En las siguientes secciones se describen las características de cada uno de los modelos de comunicación citados.

2.2. Modelo Cliente-Servidor

El modelo Cliente-Servidor se caracteriza por asociar un "proceso servidor" a los recursos que quieren compartirse o hacerse visibles a otros nodos. Existen dos patrones de comportamiento diferenciados:

- Servidores: gestionan un recurso y ofrecen servicios relacionados con este recurso a otros procesos. Reciben peticiones de "procesos clientes" vía mensajes, las ejecutan en su nombre y devuelven una respuesta.

  Los servidores pueden se secuenciales o multi-hilo. Un servidor secuencial no atiende una nueva petición hasta que no ha finalizado de servir la actual, mientras que un servidor multi-hilo crea un nuevo hilo de ejecución para servir cada una de las peticiones recibidas de forma que éstas pueden servirse en paralelo e independientemente.

- Clientes: formulan peticiones de servicio a los servidores.

La comunicación entre los clientes y servidores se realiza mediante el envío de mensajes punto a punto a través de conexiones (fig.12).

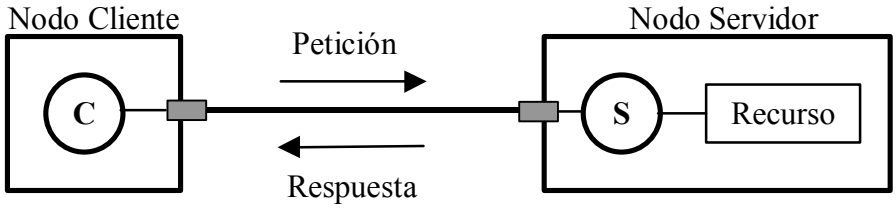


Figura 12: *Modelo de comunicaciones Cliente-Servidor: asociación de procesos servidores a recursos compartidos, comunicación mediante mensajes punto a punto entre los clientes y servidores*

Uno de los problemas en el modelo de comunicación Cliente-Servidor es obtener la dirección de un servidor a partir del nombre del servicio que se desea invocar. Para resolver esta tarea conocida como vinculación (*binding*) existen varios enfoques:

- Vinculación estática: la dirección del servidor se conoce en tiempo de compilación.
- Vinculación dinámica: la dirección no se conoce en tiempo de compilación, en su lugar, los clientes manejan nombres simbólicos. Para resolver los nombres simbólicos se utilizan varias técnicas:
  - Difusión: los servidores tienen asociados un identificador único y los clientes realizan una difusión (*broadcast*) para localizarlos.
  - Servidor de Nombres: existe un servidor (*broker*) que posee un catálogo de interfaces que relacionan un nombre simbólico con su dirección.

2.3. Modelo de Grupos

En el modelo de Grupos existen grupos de procesos que colaboran en la consecución de una tarea común o se abastecen de flujos de información comunes. Los grupos pueden estructurarse de varias formas:

- Grupo de colegas (*peer group*): toda la comunicación tiene su origen en procesos del grupo y el destino es también el grupo. Los miembros conocen a sus colegas.
- Grupo servidor: un proceso cliente pide un servicio a un grupo de procesos servidores. Puede dirigir su petición a un representante del grupo o

difundirla a todo el grupo. (Ejemplo: servidores de ficheros distribuidos, servidores redundantes).

- Grupos jerárquicos: los grupos pueden organizarse en subgrupos con líderes. Los líderes forman, a su vez, un grupo raíz. La gestión es más eficiente. (Ejemplo: servidores de nombres).

- Grupo de subscriptores: los procesos que se afilian reciben las mismas fuentes de información. Los miembros no suelen conocerse entre sí. (Ejemplo: listas de correo).

Los grupos también pueden tener carácter:

- Estático: los componentes nunca cambian.

- Dinámico: los componentes se adhieren, dejan el grupo o incluso fallan.

Las comunicaciones se realizan mediante difusión de mensajes. Las primitivas de comunicación en grupos son:

- *Broadcast*: difusión a todos los nodos o procesos.

- *Multicast*: difusión selectiva a grupos.

Ambos tipos pueden implementarse mediante mensajes punto a punto. Una implementación eficiente requiere apoyo hardware.

Algunas aplicaciones de la difusión a grupos son:

- Tolerancia a fallos basada en replicación de servicios.

- Localización de objetos en servicios distribuidos.

- Aumento de prestaciones mediante replicación de servicios.

- Notificación de eventos a grupos de procesos.

En un sistema con réplicas las primitivas de difusión tienen que cumplir las siguientes propiedades:

- Atomicidad: un *multicast* atómico garantiza que, o bien todos los componentes operativos del grupo reciben el mensaje, o bien ninguno de ellos los recibe.

- Ordenación: establece el orden relativo en que dos difusiones son recibidas en todos los destinos.

2.4. Modelo de Llamadas a Procedimientos Remotos

El modelo de llamadas a procedimiento remoto o RPC *(Remote Procedure Call*) es una forma de comunicación en sistemas distribuidos basada en extender el uso de llamadas a procedimiento para invocar servicios sobre objetos o recursos remotos. Sus características principales son:

- Hace transparente la utilización de mensajes.
- Hace transparente la invocación de servicios locales y remotos.
- Integra el modelo Cliente-Servidor con los lenguajes de programación procedurales y orientados a objetos mediante la ocultación del sistema de mensajes.
- Realiza el empaquetado (*marshalling*) de los datos en mensajes para su transmisión en las comunicaciones.

En la implementación del modelo RPC existen dos nuevos conceptos a destacar (fig.13):

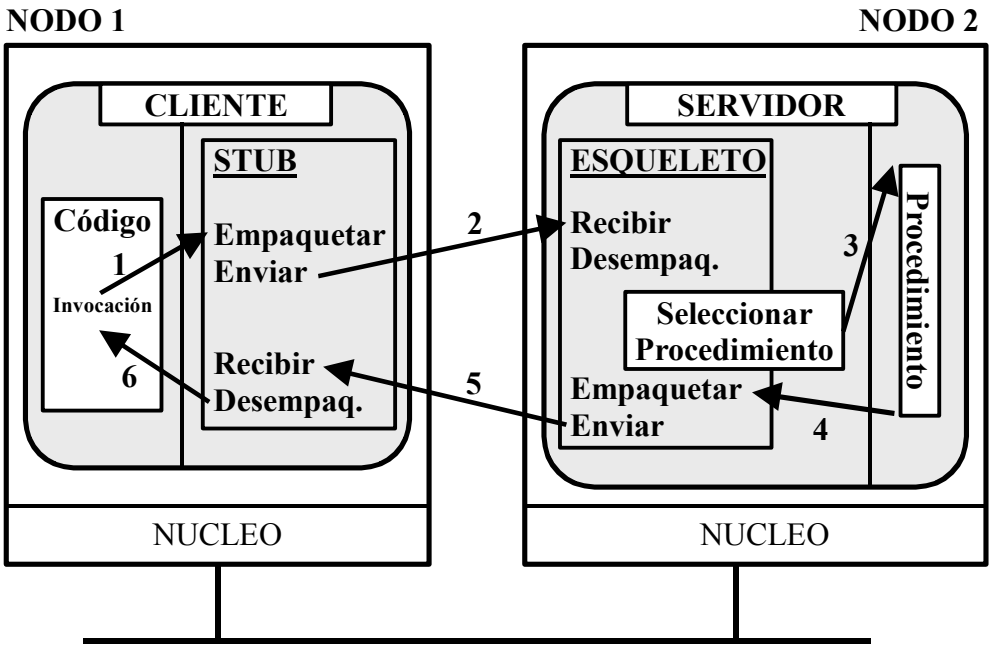


Figura 13: *Modelo de comunicaciones RPC: comunicación mediante invocación de servicios, ocultación del sistema de mensajes y empaquetado de los datos*

- Componente denominado "stub": procedimiento de biblioteca que transforma la invocación a un procedimiento remoto realizada por un cliente en mensajes. Principalmente, las funciones de un stub son obtener la

dirección del servidor (*binding*), empaquetar los datos en mensajes (*marshalling*) y convertir los datos a una representación externa estándar que sea independiente de la arquitectura.

- Componente denominado "esqueleto": proceso servidor multi-hilo asociado a un procedimiento que puede invocarse de forma remota. Recibe los mensajes procedentes de los clientes, realiza el desempaquetado y conversión de los datos, e invoca a los procedimientos como representante de los clientes.

Cada servidor con procedimientos que pueden ser invocados de forma remota tiene un esqueleto y un stub asociados a dichos procedimientos. El esqueleto debe residir en el mismo nodo que el servidor y el stub en los nodos de los clientes.

Tanto los stubs como los esqueletos pueden implementarse de forma manual o automáticamente por el compilador del lenguaje de desarrollo.

2.5. Modelo de Objetos Distribuidos

El modelo de Objetos Distribuidos se caracteriza por encapsular los recursos del sistema en objetos y extender el paradigma de objetos a sistemas distribuidos.

Los objetos remotos son transparentes [Guerraoui, 1999], es decir, no es necesario conocer la ubicación física del objeto para utilizarlo y se invocan de igual forma los objetos remotos que los locales. Además, los objetos pueden referenciarse como argumentos en los métodos.

La arquitectura del modelo de objetos está basada en los siguientes conceptos (fig.14):

- Proxy: objeto que proporciona una interfaz local a un objeto remoto (fig.15). Redirige las peticiones al objeto remoto. Para ello realiza las funciones de resolver la vinculación con el objeto remoto y ocultar el sistema de mensajes.

- Referencia a objeto remoto: puntero al objeto remoto. Contiene la información necesaria para ubicarlo.

- Realiza el empaquetado (*marshalling*) en mensajes tanto de los datos como de las referencias a los objetos para su transmisión en las comunicaciones.

Se conoce como empaquetado de parámetros al proceso de conversión de los parámetros de una invocación de un método remoto a un formato serie (secuencia de bytes) adecuado para su transmisión en un mensaje. Cuando el parámetro es una referencia a un objeto no se pasa el objeto serializado sino un proxy del mismo.

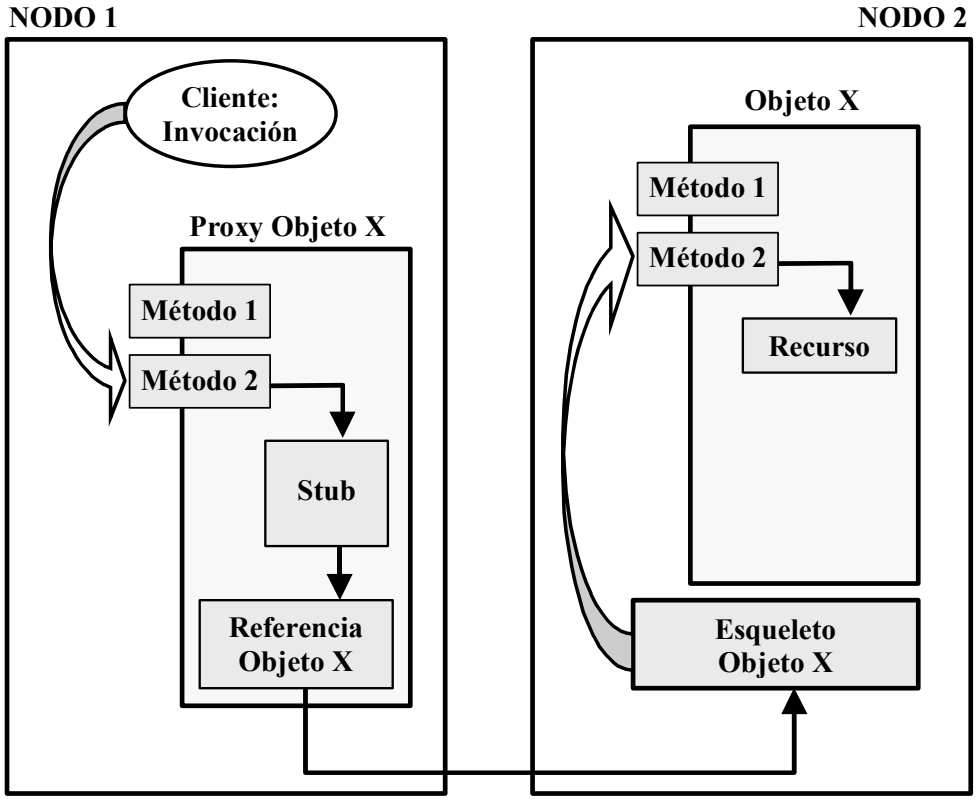


Figura 14: *Modelo de comunicaciones de objetos distribuidos: comunicación mediante objetos remotos, empaquetado de los datos y de las referencias a los objetos*

Figura 15: *Nivel del proxy o interfaz local del objeto remoto en el modelo de comunicaciones de objetos distribuidos*

Existen dos tipos de sistemas:

- Homogéneos: todos los componentes están implementados con la misma tecnología.

- Heterogéneos: los componentes están realizados con tecnologías diferentes. En estos sistemas tiene que obtenerse la interoperatividad entre componentes.

Para conseguir la interoperatividad entre los componentes de los sistemas heterogéneos hay que resolver principalmente los siguientes problemas:

- Normalizar la representación de los datos básicos. Existen dos formas de abordar la heterogeneidad en la representación de datos:
  - Los valores son convertidos a una representación externa consensuada para transmitirlos y vueltos a convertir a representación interna al recibirlos (ejemplo: XDR *eXternal Data Representation* de SUN).
  - Los valores son transmitidos en su representación original, junto con una etiqueta de la arquitectura y, en caso necesario, son convertidos por el receptor (ejemplo: s.o. Mach).
- Normalizar un lenguaje de especificación de interfaces. Permite especificar interfaces estándar entre aplicaciones escritas en distintos lenguajes de programación (ejemplo: IDL *Interface Definition Language*, estandarizado por CORBA).
- Normalizar la representación de objetos.

Existen diversos enfoques para conseguir el objetivo de la interoperatividad:

- CORBA: define una arquitectura software estándar.
- Java: define una máquina virtual idéntica para todas las arquitecturas.
- Microsoft .NET: tecnología nueva que intenta aprovechar las ventajas tanto de CORBA como de java.

2.6. Arquitectura CORBA

La arquitectura CORBA (*Common Object Request Broker Architecture*) [CORBA, 2002] es una especificación estándar de un modelo o protocolo de comunicación entre objetos distribuidos. Está basada en un gestor de peticiones a objetos comunes y permite la interoperatividad entre aplicaciones cliente/servidor ubicadas en máquinas remotas en un entorno distribuido completamente heterogéneo [Seetharaman, 1998].

La especificación CORBA ha sido desarrollada por la Organización Internacional OMG (*Object Management Group*) [OMG] fundada en 1989 y formada por un consorcio de 800 empresas tales como: Digital Equipment

Corporation, Hewlett Packard Company, HyperDesk Corporation, NCR Corporation, Object Design Inc., SunSoft Inc., IBM, etc. Los objetivos principales de OMG son la reutilización, portabilidad e interoperatividad del *software* orientado a objetos en sistemas distribuidos y heterogéneos. Con ese fin ha desarrollado varias especificaciones además de CORBA.

CORBA permite el desarrollo de *software* para entornos distribuidos heterogéneos separando la especificación de las aplicaciones de su implementación.

La especificación de una aplicación u objeto constituye su interfaz, que es independiente del lenguaje de desarrollo e indica el conjunto de servicios ofrecidos al resto de aplicaciones u objetos y la forma de invocarlos. CORBA proporciona un lenguaje estándar para definir interfaces independiente del lenguaje de programación llamado OMG IDL o CORBA IDL (*Interface Definition Language*), siendo uno de los componentes principales de la arquitectura. Una característica importante a destacar es que el procesamiento de interfaces IDL en CORBA permite generar automáticamente stubs y esqueletos.

Por otro lado, la implementación de una aplicación u objeto depende del lenguaje de desarrollo, sin embargo, para poder interoperar con otras aplicaciones u objetos sólo es necesario disponer de sus interfaces. En el desarrollo de la implementación, la interoperatividad entre las aplicaciones se realiza mediante referencias a los objetos implicados y utilizando sus interfaces. La arquitectura software que da soporte a esta interoperatividad mediante el uso de referencias es otro componente fundamental en CORBA llamado ORB (*Object Request Bus*). ORB es un objeto que gestiona las peticiones a los objetos comunes (fig.16), su implementación depende del lenguaje de desarrollo utilizado aunque ofrece una funcionalidad común en todas las plataformas.

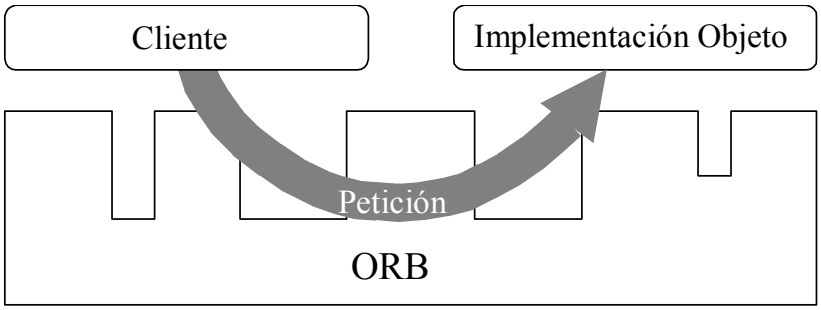


Figura 16: *CORBA: especificación estándar del modelo de comunicaciones de objetos distribuidos basada en el componente gestor de peticiones ORB*

El resto de componentes definidos en la arquitectura CORBA son los siguientes (fig.17):

- Interfaz para invocación dinámica o DII (*Dynamic Invocation Interface*): es una interfaz entre los objetos clientes y el ORB que permite invocar

dinámicamente peticiones a objetos. Permite, de una forma estándar, descubrir objetos, recuperar su interfaz y realizar una petición a través del ORB indicando el nombre del objeto y servicio que se desea invocar.

DII es una alternativa al uso estático de stubs también disponible en la arquitectura.

- Interfaz de esqueletos dinámica o DSI (*Dynamic Skeleton Interface*): interfaz entre el ORB y los métodos de los objetos. Permite ejecutar métodos mediante una única función DIR (*Dynamic Implementation Routine*). Los servidores sólo tienen que implementar esta función y el componente "adaptador de objetos" OA (*Object adapter*) invoca a la DIR asociada al servidor pasando como parámetro la petición de servicio del cliente correspondiente.

  DSI es una alternativa al uso estático de esqueletos también disponible en la arquitectura.

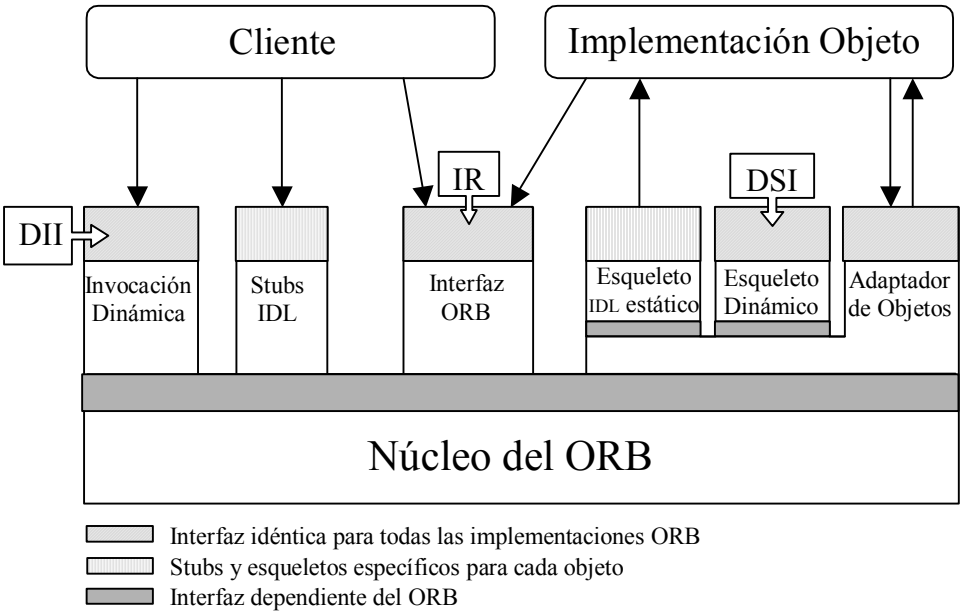


Figura 17: *Componentes de la arquitectura CORBA*

- IR (*Interface Repository*): catálogo de las interfaces de todos los objetos disponibles en el sistema distribuido.

- Núcleo de ORB: responsable de localizar el objeto, prepararlo para recibir la petición y enviar la petición y la respuesta.

- Adaptador de objetos OA (*Object Adapter*) o POA (*Portable Object Adapter*): vía de acceso de la implementación de un objeto a los servicios de ORB. Responsable de crear e interpretar referencias a objetos, invocación de métodos y seguridad.

2.6.1. Interoperatividad entre ORB's

La interoperatividad entre ORB's resuelve el problema de cómo un objeto cliente con un ORB invoca operaciones de un objeto servidor con otro ORB (fig.18).

Los clientes y servidores con sus ORB's pueden encontrase en la misma máquina o distribuidos, al mismo tiempo pueden estar implementados con el mismo lenguaje de programación o con diferentes (C++, Java, ADA, etc.). La implementación del ORB y de su interfaz con los clientes y servidores (así como la implementación de éstos) es dependiente del lenguaje empleado.

Es preciso una interfaz y protocolo de comunicación entre ORB´s independiente de su implementación. Para ello CORBA proporciona los puentes (*bridges*) y el protocolo GIOP (fig.19).

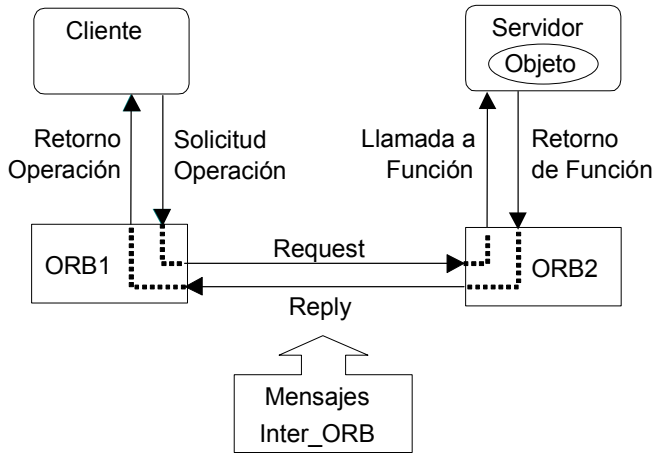


Figura 18: *Interoperatividad entre ORB's: invocación de operaciones entre cliente/servidor con diferentes ORB's*

2.6.1.1. Puentes

Los puentes (*bridges*) son aplicaciones interfaces entre los ORB´s. Un *bridge* transforma información de un formato a otro dependiendo de los formatos utilizados en las implementaciones de los ORB´s implicados. Existen dos tipos de puentes:

- *Full-Bridge*: recibe peticiones de clientes de un determinado ORB e invoca a los objetos correspondientes de otro ORB en la misma máquina. Sólo puede usarse entre ORB's no distribuidos.

- *Half-Bridge*: recibe peticiones de clientes de un ORB y las transmite mediante un formato y protocolo específico (GIOP) a otro half-bridge de otro ORB (fig.19).

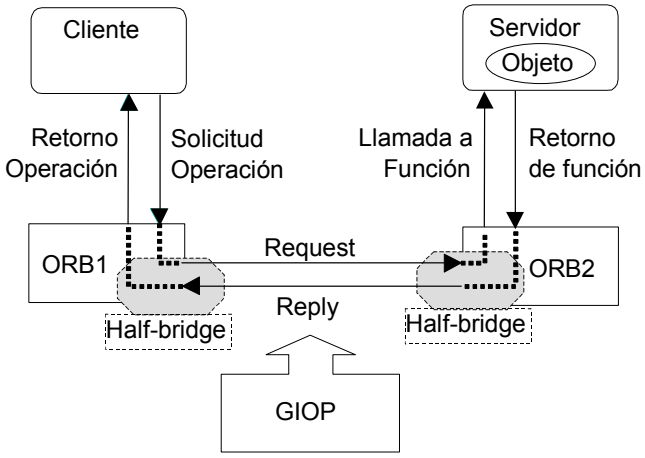


Figura 19: *Interoperatividad entre ORB's: uso de puentes (bridges) y del protocolo GIOP*

2.6.1.2. Protocolo de comunicación entre ORB's: GIOP e IIOP

GIOP (*General Inter-ORB Protocol*) es el protocolo de comunicación estándar entre ORB's establecido por la especificación CORBA. En la especificación de GIOP se establecen las siguientes características:

1. Representación común de datos o CDR *(Common Data Representation*).

2. Código específico para cada tipo de datos (*TypeCodes*).

3. Formato de las referencias a objetos o IOR (*Interoperable Object Reference*).

4. Protocolo entre ORB's o IOP (*Inter-ORB Protocol*): especifica el contenido, formato y semántica de los mensajes.

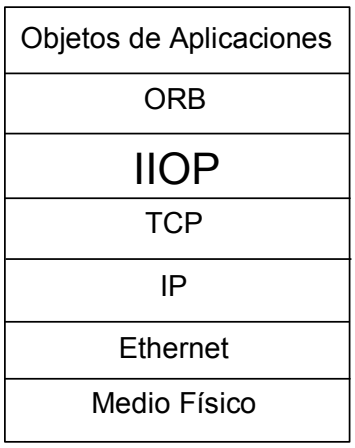


Figura 20: *Nivel del protocolo IIOP: implementación del protocolo GIOP según TCP/IP*

IIOP (*Internet Inter-ORB Protocol*) es la implementación según TCP/IP de GIOP (fig.20).

La implementación de GIOP según IIOP implementa las características especificadas por GIOP de la siguiente forma:

1. Representación común de datos o CDR *(Common Data Representation*). Define una sintaxis o codificación (*TypeCodes*) para la transferencia de todos los tipos de datos IDL: básicos, estructurados y referencias a objetos. Toda la información se codifica usando CDR para enviarse a través de una conexión IIOP, incluidas las estructuras que representan los mensajes IIOP.

   La codificación CDR consiste en convertir la información en una serie de filas de octetos (bytes) formando una estructura denominada *Octet Stream* que puede manejarse como un bloque o *buffer* de memoria.

   En la codificación CDR se especifica el orden de los bytes (*big endian* o *little endian*) y el alineamiento a partir del comienzo del *Octet Stream*.

   Existen dos clases de *Octet Streams*:

   - Mensajes: son las unidades básicas de intercambio de información.
   - Encapsulaciones: son las estructuras de datos IDL codificadas, donde el primer octeto indica el orden de los bytes. Pueden representarse como el tipo de dato IDL: ***sequence<octec>*** y posteriormente incluirse en un mensaje o volverse a encapsular.

2. Formato de las referencias a objetos o IOR (*Interoperable Object Reference*). Los ORB´s, dependiendo del lenguaje de desarrollo, pueden usar diferentes especificaciones e implementaciones para las referencias a objetos. Para la interoperatividad se necesita la especificación de una estructura para la referencia a objetos independiente del lenguaje. La solución adoptada en la especificación CORBA es el formato IOR definido mediante IDL.

   Con IOR un ORB obtiene toda la información necesaria para acceder al ORB con el objeto en cuestión. IOR se codifica según CDR para ser transmitido. Un IOR codificado también puede serializarse (convertirse en una cadena de caracteres hexadecimales) para su almacenamiento y uso, por ejemplo, en proxys.

3. Protocolo entre ORB's o IOP (*Inter-ORB Protocol*).

   IIOP establece el formato o estructura de los mensajes mediante IDL. En el protocolo de mensajes intervienen dos ORB´s, uno actuando de cliente y otro de servidor (fig.21).

- Cliente: inicia la conexión mediante TCP/IP obteniendo la dirección IP y el puerto a partir del IOR del objeto con el que se desea conectar. Los mensajes que puede enviar son: Request, LocateRequest, CancelRequest.

- Servidor: acepta conexiones a través de un puerto y dirección IP. Los mensajes que puede enviar son: Reply, LocateReply, CloseConnection. Las conexiones no son totalmente simétricas, tanto clientes como servidores pueden enviar mensajes de forma asíncrona.

Los mensajes se envían a través del socket de la conexión establecida.

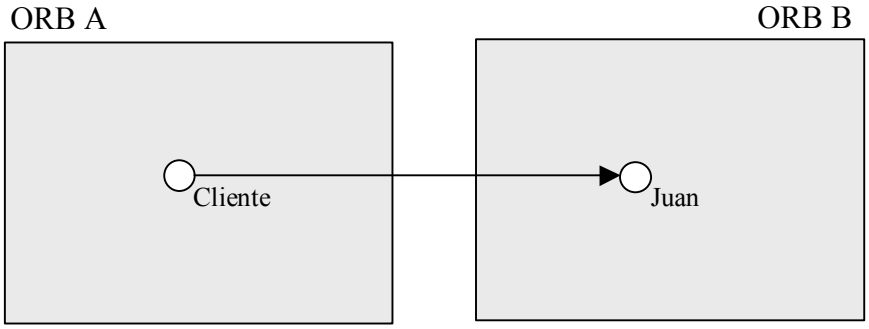


Figura 21: *Ejemplo de interoperatividad entre ORB's (1)*

Un ejemplo de interoperatividad entre dos objetos de diferentes ORB's es el representado en la figura 21, donde existe un objeto cliente que quiere operar con un objeto denominado *Juan* situado en otro ORB. En la figura 22 puede observarse que el ORB del objeto cliente posee un proxy del objeto *Juan* denominado *JL* y una tabla donde se asocia dicho proxy con su IOR. En la figura 23 el cliente invoca un método del objeto *Juan* a través del proxy *JL* pasando como parámetro otro objeto llamado *María*.

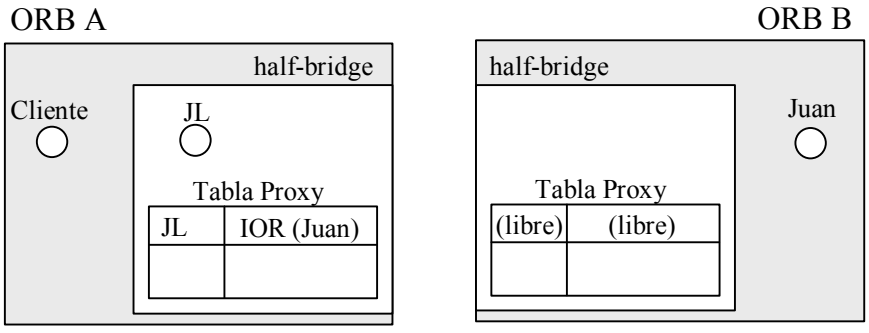


Figura 22: *Ejemplo de interoperatividad entre ORB's (2)*

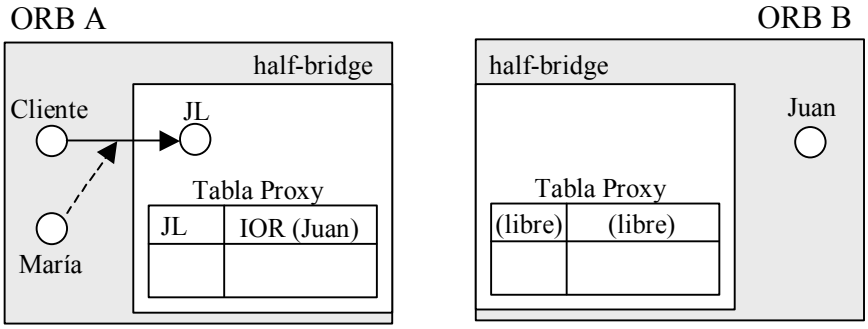


Figura 23: *Ejemplo de interoperatividad entre ORB's (3)*

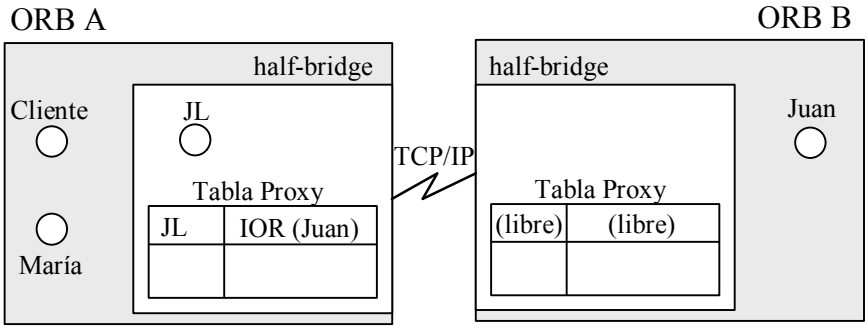


Figura 24: *Ejemplo de interoperatividad entre ORB's (4)*

Después, en la figura 24, se realizan todas la comunicaciones IIOP entre los ORB´s, para ello el ORB del cliente tiene que acceder a la tabla de proxys y obtener el IOR del objeto *Juan* correspondiente, obtener la dirección IP y puerto para establecer la conexión con el otro ORB, codificar según CDR toda la información a transmitir incluido el IOR del objeto *María* e iniciar la comunicación con un mensaje *Request*, por otro lado, el ORB destino recibe la información correspondiente a la petición efectuada y crea un proxy denominado *Mar* asociado al IOR recibido del objeto *María*. Finalmente, en la figura 25, el objeto *Juan* recibe la invocación correspondiente del objeto cliente y puede acceder al parámetro enviado que es otro objeto mediante el proxy *Mar*.

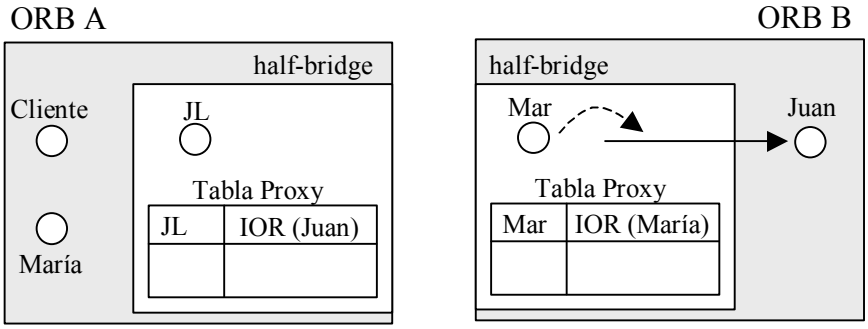


Figura 25: *Ejemplo de interoperatividad entre ORB's (5)*

2.6.2. Tiempo Real en CORBA: RT-CORBA

Recientemente, OMG ha desarrollado una extensión del estándar CORBA para aplicaciones en tiempo real conocida como RT-CORBA [RT-CORBA, 2002; Schmidt, 2000; Natarajan, 2002].

El principal objetivo de RT-CORBA es proporcionar la posibilidad de especificar y mantener requerimientos de tiempo real en la ejecución de aplicaciones que se encuentran distribuidas en varios nodos. Para dar soporte al cumplimiento de las restricciones de tiempo real, la especificación RT-CORBA presenta las siguientes características:

- RT-CORBA define un conjunto de extensiones para el estándar CORBA. En la figura 26 pueden observarse las nuevas entidades u objetos de tiempo real que han sido especificados.

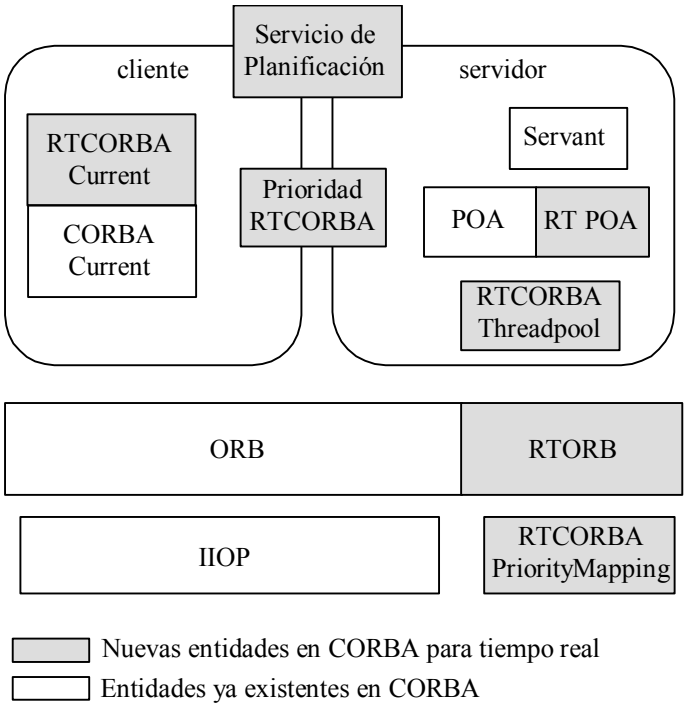


Figura 26: *Arquitectura RT-CORBA: tiempo real en CORBA*

- La implementación de RT-CORBA debe estar sustentada por un Sistema Operativo de tiempo real (estándar *IEEE POSIX 1003.1-1996 Real-Time Extensions*) y por un sistema de comunicaciones que garantice la exactitud temporal.

- Define el término "actividad" como una entidad planificable. La implementación de esta entidad se basa en la utilización de los hilos (un hilo por petición y un hilo por objeto) proporcionados por el Sistema Operativo sobre el que se sustente la implementación de CORBA. También define un

servicio de planificación basado en prioridades fijas para ayudar al programador en la planificación de las actividades.

- Define un conjunto de interfaces para el control de la prioridad de las entidades planificables. Define un tipo de prioridad universal independiente de la plataforma denominada prioridad RTCORBA. La prioridad RTCORBA se asocia a los hilos en ejecución mediante el atributo prioridad del objeto RTCORBA::Current. También se define la interfaz PriorityMapping para la correspondencia y conversión de las prioridades CORBA a las prioridades equivalentes en el sistema de planificación del Operativo (prioridades nativas) y viceversa.

- Establece un mecanismo para la propagación de prioridades que determina la prioridad con la que un servidor atenderá las solicitudes de los clientes. Soporta dos modelos:

  - CLIENT_PROPAGATED: el servidor hereda la prioridad del cliente.

  - SERVER_DECLARED: el servidor atiende las solicitudes de los clientes con una prioridad establecida cuando fue creado.

- Establece mecanismos para evitar el problema de inversión de prioridades y bloqueos en los servidores: propagación de la prioridad de los clientes a los servidores (CLIENT_PROPAGATED), interfaz de semáforos para la gestión del acceso a los recursos, políticas para especificar y configurar protocolos de comunicación, abstracción de una cola o almacén de hilos (threadpool) para la gestión de la ejecución de los hilos en el servidor.

En la arquitectura RT-CORBA (fig.26) se definen nuevas entidades u objetos como la interfaz RTORB, que es una extensión para tiempo real de la interfaz ORB de CORBA, y RT POA que es otra extensión para tiempo real del adaptador de objetos POA existente en CORBA.

Aunque en la actualidad ya existen algunas implementaciones de RT-CORBA [TAO], el estándar es muy reciente y todavía se encuentra bajo desarrollo [Becker, 2002]. Por otro lado, las características de tiempo real que proporciona RT-CORBA están basadas en la predicción de la ejecución de los procesos mediante una planificación previa basada en los sistemas de tiempo real clásicos donde se identifica cada uno de los procesos que intervendrán en el sistema con una prioridad determinada. Sin embargo, tal como se concluyó en el capítulo primero, las necesidades de tiempo real en las arquitecturas híbridas para el control de robots móviles no tienen que limitarse a este estudio de planificabilidad, pues, debido a las condiciones dinámicas de los entornos donde se implementan dichas arquitecturas, la posibilidad de incorporar dinámicamente nuevos agentes en el sistema así como la movilidad de los mismos son características deseables que no ofrecen los sistemas clásicos de tiempo real. Todo ello, hace necesario la especificación de nuevos modelos y metodologías para el control distribuido [Brennan, 2002].

2.7. Procesamiento distribuido en Java

Java es un lenguaje de programación presentado por SUN Microsystems en 1995. Debido a sus características [Singhal, 1998] se ha convertido rápidamente en uno de los lenguajes más utilizados actualmente en la implementación de las nuevas tecnologías de la información, teniendo especial importancia en INTERNET y en la programación distribuida.

El modelo Java define una Máquina Virtual (intérprete de código Java) que es independiente de la plataforma e igual en todos los sistemas, lo cual permite que una vez desarrollado código en Java éste pueda ejecutarse en cualquier lugar.

Java ha desarrollado un modelo propio de objetos distribuidos, llamado Java-RMI, que proporciona invocación transparente a métodos de objetos remotos. Actualmente, Java también proporciona una implementación de la arquitectura CORBA llamada Java IDL.

Uno de los inconvenientes de Java es que al ser un lenguaje interpretado (la Máquina Virtual interpreta el código Java traduciéndolo a código directamente ejecutable por la máquina) la velocidad de ejecución es lenta si se compara con otros lenguajes no interpretados como C++. La solución consiste en crear máquinas capaces de ejecutar código Java directamente sin interpretarlo, tecnología conocida como PicoJava [McGhan, 1998].

En cuanto a la adecuación de Java para sistemas de tiempo real, la solución consiste en la extensión de la máquina virtual mediante una interfaz de aplicación conocida como *Real-Time Java* [Nilsen, 1998; Bollella, 2000; PERC].

2.7.1. Java RMI

JAVA-RMI (*Java Remote Method Invocation*) es un sistema de invocación de métodos remotos que, a diferencia de CORBA, asume un entorno homogéneo de Máquina Virtual Java.

Proporciona un modelo de objetos distribuidos similar al modelo de objetos de Java, integrándolo en el lenguaje de manera natural. Permite principalmente invocar métodos de interfaces remotas, pasar una referencia a un objeto remoto en una invocación (local o remota) y la movilidad de código (objetos). Esta última característica adquiere una gran importancia en las metodologías actuales de programación basadas en agentes, tal como se verá más adelante.

La arquitectura del sistema RMI puede observarse en la figura 27.

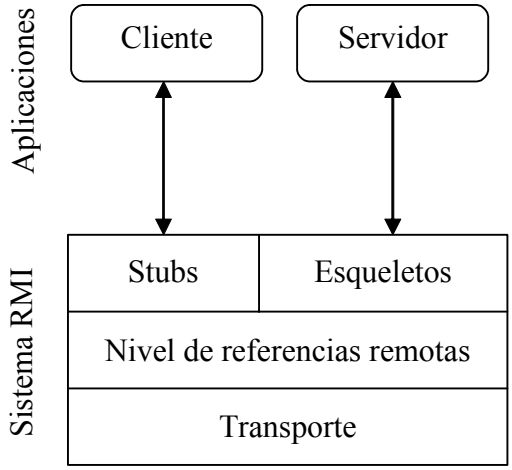


Figura 27: *Modelo de objetos distribuidos en Java: arquitectura de Invocación de Métodos Remotos RMI*

Los stubs y esqueletos no tratan con transportes específicos sino con una interfaz que emplea un mecanismo denominado serialización de objetos (*object serialization*) que permite a Java transmitir datos y objetos entre espacios de direcciones remotos. La serialización de Java no es solamente el empaquetado de parámetros, sino que consiste en convertir cualquier tipo de estructura (clases, objetos, parámetros, referencias, etc.) a un formato de cadenas ordenadas de bytes adecuado para ser transmitido por la red y para posteriormente poder reconstruir de nuevo dichas estructuras (deserialización). Este mecanismo es equivalente a la codificación CDR descrita en CORBA.

El nivel de referencias a objetos implementa un protocolo específico de referencias a objetos remotos que es independiente de los stubs y esqueletos. Trata con el nivel de transporte. Cada implementación de objeto remoto elige su propia subclase de referencia remota que opera en su nombre.

2.7.2. Java IDL

Java IDL es la implementación Java de la arquitectura estándar CORBA. Proporciona un ORB para permitir la comunicación entre aplicaciones Java IDL y aplicaciones compatibles CORBA implementadas con otros lenguajes.

Java IDL también permite la conversión automática de las especificaciones realizadas en lenguaje IDL en código Java, obteniéndose, de esta forma, los ficheros Java con los stubs y esqueletos necesarios.

La comunicación entre los ORB (fig.28) se realiza mediante el protocolo IIOP.

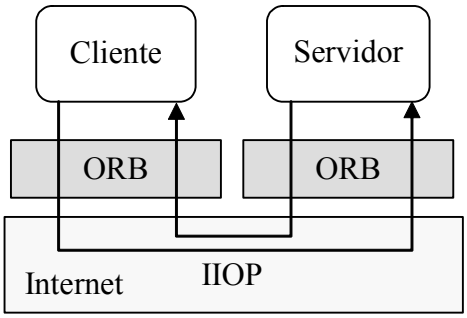


Figura 28: *Java IDL: implementación para java de la arquitectura CORBA*

También existe una extensión de Java llamada JOE que permite conectar sistemas Java con sistemas CORBA sin necesidad de instalar un núcleo CORBA en la máquina Java. Esta extensión es anterior a Java-IDL.

2.8. Microsoft .NET

.NET es la nueva tecnología de Microsoft para el desarrollo de sistemas distribuidos [Brugali, 2002]. Se basa en las características de Java y CORBA para proporcionar interoperatividad entre aplicaciones heterogéneas. Los siguientes cuatro aspectos caracterizan el marco .NET:

- Sistema de tipos común o CTS (*Common Type System*): es un modelo de objetos que extiende los modelos previos COM y DCOM con el objetivo de soportar múltiples lenguajes en el desarrollo de *software*.

- Lenguaje intermedio o IL (*Intermediate Language*): es un lenguaje orientado a objetos que se sustenta sobre CTS. Varios compiladores (de Microsoft y ajenos a Microsoft) de diferentes lenguajes (para C++, Java, etc.) generan código en el lenguaje IL (por ejemplo el VisualStudio .NET).

- Infraestructura de lenguaje común o CLI (*Common Language Infrastructure*): es un entorno de ejecución (similar a la máquina virtual de Java) que ejecuta código compilado en lenguaje IL.

- Kit de desarrollo software .NET o .NET SDK (*Software Development Kit*): es una colección orientada a objetos de componentes reutilizables que además facilita el desarrollo de nuevos componentes.

La esencia de la propuesta de Microsoft consiste en la posibilidad de compilar a lenguaje intermedio IL tanto el código existente como el código nuevo escritos en cualquier lenguaje de alto nivel. Las aplicaciones resultantes, desarrolladas a partir de librerías de clases y componentes de diferentes lenguajes, pueden interoperar a través del entorno de ejecución .NET.

El marco de desarrollo .NET simplifica la implementación de aplicaciones *software* mediante el uso extensivo del mecanismo de reflexión. Reflexión es la capacidad del entorno de ejecución CLI de permitir a las aplicaciones la exploración de su propia estructura en tiempo de ejecución. Esto es posible debido a que cada objeto basado en CTS tiene asociada información sobre el significado de su estructura y funcionalidad (meta-información). Los siguientes servicios implementan el mecanismo de reflexión:

1. .NET Remoting: permite a los objetos interactuar a través de Internet (incluida la movilidad de tales objetos) usando codificación binaria cuando el rendimiento es un parámetro crítico, o codificación XML (protocolo SOAP) [XML; SOAP] cuando la interoperatividad con otras aplicaciones es un parámetro esencial. Similar a Java RMI, este servicio se basa en el modelo de stubs/esqueletos: el stub (llamado proxy) se crea en tiempo de ejecución utilizando para ello la meta-información publicada por el servidor.

2. Servicios Web: son componentes reutilizables para el desarrollo de aplicaciones servidor. Combinan la capacidad de computación distribuida con los conceptos Web y pueden compararse a los Servlets de Java y JSP [JSP]. El ASP.NET es el entorno necesario para los servicios Web. Un ejemplo de servicio Web común es el e-mail. Los clientes encuentran los servicios Web mediante la utilidad "wsdl" (un servicio Web de directorio proporcionado por el .NET SDK).

3. Serialización: permite la conversión de los objetos en una secuencia lineal de *bytes* que puede ser almacenada en un disco o enviada a otros computadores a través de la red. Este mecanismo es la base del servicio de persistencia .NET que utiliza la meta-información de los objetos para automáticamente almacenarlos y reconstruirlos.

2.9. Modelo de Código Móvil

En la mayoría de los sistemas de objetos distribuidos, la información que puede transmitirse entre el cliente y el servidor se limita a un pequeño número de tipos de datos, referencias a otros objetos distribuidos o estructuras realizadas a partir de estos tipos. Lo lógico sería que los clientes y servidores pudieran intercambiar objetos completos mediante la transmisión de los datos y el código necesario para la ejecución de los mismos. Esto se hace posible mediante el modelo de comunicaciones distribuidas conocido como modelo de código móvil [Brugali, 2002]. El código móvil puede definirse como la capacidad de cambiar de forma dinámica las conexiones existentes entre los diferentes objetos distribuidos y el entorno o lugar de su ejecución (ubicación física). Este movimiento de código en tiempo de ejecución (*on-line*) ha dado lugar a un tipo de objetos software para programación distribuida conocidos como Objetos Móviles (*Mobile Objects*).

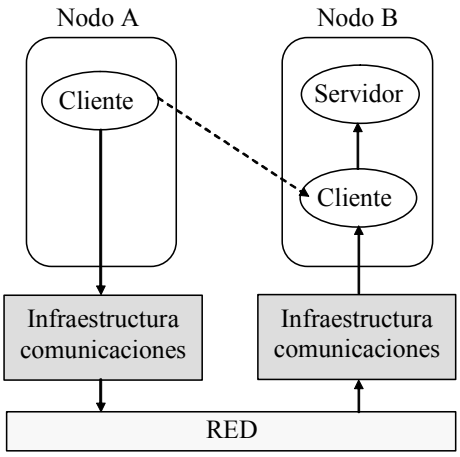


Figura 29: *Modelo de Código Móvil: migración de objetos (transmisión de datos y código) entre entornos cliente/servidor*

Un objeto móvil puede migrar en tiempo de ejecución desde el entorno de ejecución del cliente al entorno de ejecución del servidor o viceversa (fig.29). Atendiendo al sentido de la migración, el código móvil puede clasificarse como sigue:

- Código móvil bajo demanda (desde el servidor hacia el cliente): el cliente descarga desde el servidor los componentes *software*, objetos móviles, que necesita para la ejecución de sus tareas. Los objetos móviles se enlazan en tiempo de ejecución con la aplicación cliente y se ejecutan en su espacio de direccionamiento. Los objetos móviles podrían ser componentes pasivos o hilos que comienzan su ejecución cuando la descarga se ha completado. Se almacenan en el nodo o computador del cliente y se borran tras su utilización.

- Código móvil desde el cliente hacia el servidor: el cliente envía al servidor un objeto móvil que utiliza los recursos del servidor para ejecutar las tareas encomendadas por el cliente. El siguiente paso a esta aproximación consiste en la utilización del modelo multiagente, que se describe en el siguiente apartado, con agentes móviles. En el modelo multiagente no hay una clara diferencia entre clientes y servidores, y los objetos distribuidos, denominados agentes, cuando disponen de la capacidad de moverse a través de la red constituyen los denominados sistemas de agentes móviles.

El modelo de código móvil, comparado con los tradicionales modelos cliente/servidor, puede reducir el tráfico de la red y mejorar la fiabilidad de las comunicaciones. De hecho, la posibilidad de mover el código del cliente o del servidor al otro lado de la conexión permite la interacción entre ambos evitando la red y, por tanto, los problemas de comunicación que pudieran darse (por ejemplo en una red inalámbrica con fallos de cobertura). Con este modelo se obtiene una alternativa al incremento del ancho de banda de la red.

El código móvil hace al modelo cliente/servidor más flexible pues permite tanto al cliente como al servidor ampliar sus capacidades de forma dinámica. En concreto, simplifica las tareas de mantenimiento al permitir la actualización independiente de los clientes y servidores. Los clientes siempre descargan la última versión de los objetos móviles de los servidores; los servidores pueden actualizarse enviándoles objetos móviles con nuevas funcionalidades.

2.10. Modelo Multiagente

El modelo multiagente se caracteriza por el uso de objetos software denominados agentes que poseen cierta autonomía. En este modelo no hay una clara distinción entre servidores y clientes, sino que son los propios agentes quienes asumen una u otra funcionalidad dependiendo de las tareas asignadas al sistema.

Los agentes pueden interactuar entre ellos intercambiando información, actuar en beneficio de otros agentes, reaccionar ante eventos internos o externos, ejecutar funciones autónomas como la monitorización de su propia actividad o el entorno exterior.

En definitiva, los sistemas de agentes pueden considerarse como la evolución de los diferentes sistemas descritos previamente, donde los objetos *software* adquieren su propia independencia y encapsulan los mecanismos de comunicación descritos haciéndolos transparentes en el desarrollo e implementación de los agentes.

Cuando los agentes tienen la capacidad de moverse entre los nodos del sistema distribuido, reciben el nombre de agentes móviles y constituyen los denominados sistemas de agentes móviles.

Debido a su importancia y utilización en la especificación de la arquitectura híbrida para el control de robots móviles que se propone en esta tesis, se dedica el siguiente capítulo al estudio de los sistemas multiagente y en concreto a los sistemas de agentes móviles destacando sus características y tendencias actuales.

2.11. Conclusiones

En este capítulo se ha realizado un repaso de los modelos de comunicaciones existentes en los sistemas distribuidos.

Se ha ofrecido una visión global de dichos modelos centrándose en el modelo de objetos distribuidos. El objetivo ha sido describir de forma general los estándares establecidos (CORBA) para los sistemas de objetos distribuidos y resaltar características y conceptos concretos que actualmente son la base de metodologías más modernas de programación distribuida. En concreto, se han introducido los

modelos de código móvil y sistemas multiagente que se estudiarán con mayor profundidad en el capítulo siguiente.

Como se concluyó en el capítulo primero, es necesario un sistema de comunicaciones que permita enlazar los distintos niveles de las arquitecturas híbridas respetando las restricciones temporales de cada uno de ellos. Los estándares actuales relacionados con las comunicaciones distribuidas en tiempo real ofrecen unas características adecuadas a los sistemas clásicos de tiempo real pero no adecuadas a las necesidades de una arquitectura híbrida donde el nivel deliberativo además de poder interactuar con el nivel reactivo tiene que poder comunicar de forma esporádica estructuras de datos más complejas de tamaño variable, como implica la transmisión de agentes, los cuales además deben poder incorporarse al sistema de forma dinámica.

Aspectos como la transparencia en la localización de los objetos y la posibilidad de la movilidad de datos e incluso de código cobran mucha importancia en los sistemas actuales de comunicaciones. Concretamente, en los sistemas de agentes inteligentes y en su aplicación a las arquitecturas híbridas para el control de robots móviles que son objetivo de esta tesis.

Por ello, en el capítulo siguiente se realiza un estudio de las relaciones entre los sistemas de agentes *software* móviles y los agentes físicos (robots). Posteriormente, en los capítulos sucesivos, se abordará la descripción de la arquitectura híbrida para el control de robots móviles basada en agentes *software* móviles que se propone y la descripción del sistema de comunicaciones necesario sobre el que se sustenta dicha arquitectura.

Capítulo 3

AGENTES SOFTWARE

En este capítulo se realiza una descripción del modelo de comunicaciones y desarrollo de aplicaciones denominado modelo multiagente. El modelo multiagente puede considerarse como una evolución de los sistemas orientados a objetos o basados en componentes que se describieron en el capítulo anterior. Los componentes de los sistemas multiagente son los "agentes software inteligentes", los cuales se caracterizan por disponer de cierta autonomía. Dichos agentes, además, pueden poseer la capacidad de movilidad entre los nodos del sistema, constituyendo los sistemas de agentes móviles que se emplearán en los capítulos siguientes para la especificación de la arquitectura de control que se propone en esta tesis.

3.1. Introducción

Históricamente han ido apareciendo diferentes modelos de programación de forma que cada modelo nuevo ha representado un mayor grado de abstracción que el anterior. Dicha evolución puede resumirse de forma aproximada en la figura (fig.30).

Actualmente, la programación de sistemas basados en componentes es un hecho. El siguiente paso lógico o grado de abstracción es evolucionar a los denominados "sistemas multiagente" los cuales se basan en la programación de agentes inteligentes.

Existe una estrecha relación entre los modelos de programación y los modelos de comunicación, relación que se acentúa al hablar de entornos distribuidos. Debido a ello, la aparición de un nuevo modelo de programación de sistemas distribuidos suele implicar nuevas o diferentes necesidades de comunicaciones y por consiguiente

la aparición de nuevos modelos de comunicación. Es el caso del modelo de comunicación multiagente, resultado del modelo de programación basado en agentes.

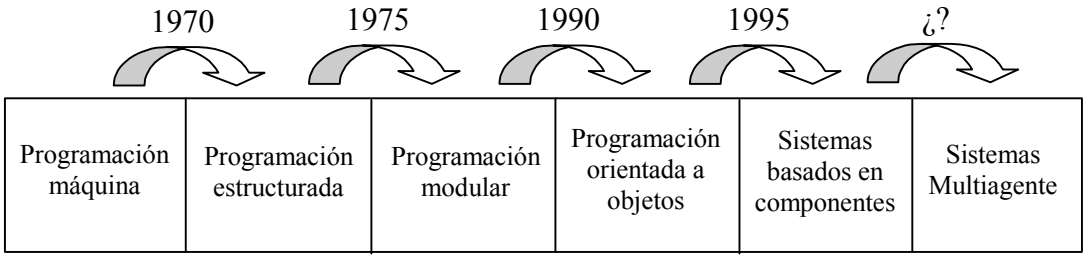


Figura 30: *Evolución de los diferentes modelos de programación hasta el modelo multiagente actual*

El modelo de programación de sistemas multiagentes o MAS, debido a sus características de modularidad y ejecución independiente de los agentes, resulta adecuado para la especificación e implementación de arquitecturas híbridas para el control de robots móviles. Es por ello, por lo que se dedica este capítulo al estudio de dichos sistemas haciendo hincapié en las relaciones existentes entre los agentes software, elementos principales de los MAS, y los agentes físicos propios de arquitecturas para robots. De hecho, se observa claramente una implicación directa de las características de los paradigmas existentes en robótica móvil con las características de los agentes móviles. Esto es debido a la autonomía que caracteriza a estos agentes y que permite estudiarlos como si se tratara de verdaderos agentes físicos. En este sentido, existen implementaciones de sistemas donde se asocia cada agente físico o robot a un agente software formando una sola entidad, de manera que el agente software proporciona la inteligencia y el robot la capacidad de interacción con el entorno real. Sin embargo, se puede realizar una abstracción mayor donde varios agentes móviles a través de la red pueden utilizar diferentes agentes físicos para la consecución de sus tareas como si se trataran de nuevos recursos disponibles en el entorno software distribuido. Desde este punto de vista, el diseño de una arquitectura general para la implementación de este tipo de sistemas, requiere la existencia de un sistema de comunicaciones que permita la ejecución y movilidad de los agentes software que además tendrán que poder acceder a la información de los sensores e indicar acciones a los actuadores. Si además se necesita reacción en tiempo real, como por ejemplo para poder evitar obstáculos, tendrán que coexistir diferentes tipos de agentes con diferentes restricciones temporales, lo cual conlleva a una estructura de la arquitectura en varios niveles. Estos niveles también tendrán que estar conectados a través del sistema de comunicaciones empleado. Por todo ello, en este capítulo se realiza un estudio de las características de los estándares existentes para sistemas multiagente que permitirá el diseño de una arquitectura y sistema de comunicaciones necesario en base, por un lado, a las necesidades planteadas en el capítulo uno tras el estudio del paradigma híbrido, y por otro lado, a las necesidades de los agentes.

A continuación se describen con detalle los fundamentos de los agentes inteligentes y se introduce el concepto de sistema multiagente. También se comentan

los estándares desarrollados que especifican las características de dichos sistemas. Estos estándares pueden considerarse como la evolución correspondiente de los estándares para sistemas orientados a objetos descritos en el capítulo anterior (CORBA, IIOP, etc.).

A lo largo del capítulo se estudiarán las relaciones existentes entre los sistemas de agentes *software* y los agentes físicos (robots).

3.2. Concepto de Agente Inteligente

El término agente se ha utilizado para denotar objetos físicos con la propiedad básicamente de controlar su propio destino (por ejemplo un robot). Sin embargo, actualmente este término también se utiliza para definir un determinado tipo de *software*, surgiendo como resultado el concepto de agente *software* frente al de agente físico. Este capítulo se centra en los agentes *software* y en concreto en aquellos denominados inteligentes, conceptos que a continuación van a tratarse de definir.

Los agentes inteligentes constituyen un paradigma de programación para el desarrollo de aplicaciones *software* [Jennings, 1998]. Actualmente, los agentes son el centro de interés en muchas ramas de la ingeniería informática e inteligencia artificial, y están usándose en una amplia y creciente variedad de aplicaciones.

El significado del concepto "agente inteligente" es una cuestión muy discutida donde no se encuentra un acuerdo general [Franklin, 1996]. Sin embargo, sí existen una serie de características asociadas a los agentes inteligentes a partir de las cuales puede ofrecerse una definición que sería aceptada por la mayoría de los investigadores [Wooldridge, 1995].

Primero, se puede decir que un agente *software* es un programa con capacidad de actuar de forma autónoma en el entorno donde se encuentre, con la finalidad de conseguir los objetivos para los cuales fue diseñado. Por autonomía se entiende la capacidad de actuar de forma independiente y autosuficiente, sin la intervención humana o de otros agentes y teniendo su propio control. Sin embargo, los sistemas autónomos no son nuevos, existen muchos programas con esta característica que pueden clasificarse como agentes, aunque no como agentes inteligentes. Por ejemplo, los sistemas de control de tiempo real que monitorizan el estado de entornos reales y realizan acciones de estabilización cuando las condiciones de éstos cambian, sistemas tan sencillos como los termostatos o tan complejos como el control de un reactor nuclear, e igualmente los sistemas *software* que de una forma automática, por ejemplo, avisan al usuario cada vez que le llega un nuevo correo. Estos sistemas actúan de forma autónoma (agentes *software*) pero no lo hacen de forma inteligente.

Para que un agente pueda llamarse inteligente además de autónomo tiene que ser flexible. El término flexible en la autonomía del agente engloba las siguientes características:

- Sensible o reactivo (*responsive*): el agente tiene que percibir su entorno (el cual puede ser el mundo físico, un usuario, una colección de agentes, Internet, etc.) y responder en un tiempo adecuado ante los cambios que se produzcan en dicho entorno.

- Oportunista o pro-activo (*proactive*): el agente no sólo ha de reaccionar frente a su entorno, sino que el agente tiene que aprovechar aquellas situaciones que puedan darse en dicho entorno si éstas favorecen la consecución de sus objetivos, ofreciendo una conducta de iniciativa propia.

- Social (*social*): el agente también tiene que ser capaz de interactuar, cuando lo considere oportuno, con otros agentes y humanos con el objetivo de completar sus actividades o ayudar a la realización de las tareas de los otros.

Con estas características queda definido el concepto de agente inteligente. Sin embargo, estos agentes pueden poseer, además, otras características adicionales como son la movilidad y adaptabilidad. Por movilidad se entiende la capacidad del agente de moverse de un entorno a otro por decisión propia [Karnit, 1998]. La adaptabilidad o aprendizaje puede definirse como la capacidad del agente de reaccionar ante una misma situación de formas diferentes con el objetivo de encontrar los comportamientos óptimos orientados a la resolución de cada uno de los diferentes problemas.

3.3. Sistemas Basados en Agentes

Un sistema basado en agentes o sistema multiagente (MAS *Multi-Agent System*) es aquel en el que el nivel de abstracción utilizado es el agente. En principio, un sistema basado en agentes podría especificarse en términos de agentes pero no implementarse con un entorno específico de desarrollo de software orientado a agentes. Sin embargo, lo ideal es realizar tanto el diseño como la implementación en términos de agentes. Para ello, en el campo de la robótica cognitiva, existen herramientas software de desarrollo basadas en la tecnología de agentes que permiten implementar sistemas de sociedades de agentes cooperativos [Sloman, 1998 y 1999].

Un sistema basado en agentes puede estar constituido por un único agente (*single-agent system*) o por múltiples agentes (*multi-agent system*)

El proceso de transformar un componente software en un agente se ha formalizado con el nombre de agentificación (*agentification*), [Shoham, 1993].

Un ejemplo de software para desarrollar agentes es el entorno de desarrollo de IBM basado en el lenguaje de programación JAVA. Dicho entorno ofrece una API para crear agentes móviles denominados AGLETS [AGLETS].

Otro ejemplo de tecnología para realizar aplicaciones basadas en agentes es COSY, una arquitectura de agentes modular basada en los conceptos de comportamientos, recursos e intenciones o motivaciones, y DASEDIS como el entorno de desarrollo para implementar dichos agentes [Burmeister, 1998].

3.4. Dominio de Aplicación de los Agentes

Los agentes nos ofrecen en un amplio dominio de aplicaciones [Jennings, 1998] tanto la habilidad de resolver problemas que hasta entonces no habían podido resolverse (por no disponerse de la tecnología adecuada o porque la utilización de la existente implicaba un coste excesivo a nivel de dificultad, tiempo, consumo, riesgo, etc.) como la habilidad de resolver problemas de una forma significativamente mejor (atendiendo al coste, sencillez, eficiencia, rapidez, etc.) que la forma utilizada hasta el momento.

Existen una serie de características que cuando forman parte de la especificación de un sistema se suelen citar para justificar la adopción de una tecnología basada en agentes. Dichas características son [Jennings, 1998]:

- Cuando el sistema es reactivo existiendo una continua interacción con el entorno [Pnueli, 1986]. A este tipo de sistemas pertenecen los denominados sistemas complejos (*complex systems*) y los sistemas abiertos (*open systems*).

    - Sistemas complejos. Son sistemas donde, debido a su complejidad, la modularidad y la abstracción constituyen las herramientas más adecuadas para el desarrollo natural de su diseño e implementación.

      Los agentes representan una poderosa herramienta para realizar sistemas modulares. Si existe un problema demasiado complejo o grande, entonces puede ocurrir que la única forma de abordar su diseño e implementación sea mediante su división en un número determinado de componentes modulares especializados cada uno de ellos en la resolución de una parte específica del problema. El objetivo final puede conseguirse mediante la colaboración entre todos los módulos. En tales dominios, un sistema basado en múltiples agentes facilita la tarea de descomponer el problema en un determinado número de componentes más pequeños y sencillos, los cuales son más fáciles de desarrollar, especializados en resolver subpartes de dicho problema. Esta descomposición permite a cada agente utilizar la técnica más apropiada para resolver su problema particular, en lugar de tener que adoptarse forzosamente una técnica global para todo el

sistema que podría no ser la óptima en la resolución de cada uno de los casos particulares. Desde este punto de vista, se aprecia la idoneidad del empleo de los agentes para la descomposición en subtareas de las tareas u objetivos asignados a un robot en una arquitectura híbrida para el control de robots móviles. Los agentes podrían constituir la base para la especificación e implementación de los distintos esquemas de percepción y motores que forman los comportamientos básicos.

El concepto de agente también proporciona una útil abstracción del sistema. De forma que el programador puede concebir el sistema complejo como una sociedad cooperativa de elementos autónomos especializados en la resolución de problemas concretos y sencillos. Mediante la cooperación de estos elementos autónomos se abordan y se resuelven objetivos y problemas cada vez más complejos. Las arquitecturas híbridas para el control de robots móviles suelen organizarse en diferentes niveles formados por componentes independientes que tienen que cooperar entre ellos. En este sentido, los agentes también resultan idóneos para especificar e implementar dichos componentes.

- Sistemas abiertos. Son sistemas cuya estructura cambia de forma dinámica. Se caracterizan porque los componentes que lo forman no son conocidos a priori, pueden cambiar a lo largo del tiempo y pueden ser heterogéneos (en el sentido en que pueden implementarse por diferentes personas, en diferentes momentos y usando diferentes técnicas y herramientas de software). En este sentido, los agentes son adecuados para este tipo de sistemas por sus características de autonomía y ejecución independiente que permiten el desarrollo individual de los mismos así como su integración de forma dinámica en el sistema. Las características de los sistemas abiertos se relacionan directamente con los objetivos planteados en las conclusiones del capítulo uno para la especificación de una arquitectura híbrida para el control de robots móviles, donde no hubiera que especificarse a priori ni todos los componentes software ni su ubicación física, sino que éstos se adaptaran dinámicamente a un entorno también dinámico.

- Cuando la información, el control o los recursos se encuentran distribuidos. En este caso, el uso de agentes permite su distribución en función de la ubicación de los datos o recursos. En el capítulo uno se concluyó que una arquitectura adecuada para el control de robots móviles tenía que ser híbrida y distribuida acorde con la propia naturaleza distribuida de la información sensorial y comportamientos.

- Cuando el sistema posee elementos que deben interactuar con otros elementos de otros sistemas. Los agentes pueden facilitar la interacción entre sistemas mediante el uso de lenguajes comunes de comunicación entre

agentes. En una arquitectura híbrida multinivel tiene que existir una interacción entre los componentes de los distintos niveles que la forman.

En estos casos, la utilización de agentes suele proporcionar una forma natural de modelar el sistema e implementarlo, resultando, por lo tanto, idóneos en la especificación e implementación de arquitecturas híbridas para el control de robots móviles. Los agentes constituirán los componentes de tales arquitecturas y dependiendo de la función a realizar o ubicación dentro de la arquitectura (nivel al que pertenezcan) tendrán que poseer determinadas características. Dichas características permiten realizar una clasificación de los distintos tipos de agentes existentes.

3.5. Clasificación de los Agentes

Los agentes se pueden clasificar [Nwana, 1996] atendiendo a diversos factores:

- Según su movilidad:
  - Estáticos: permanecen donde son creados.
  - Móviles: pueden moverse a través de la red.
- Según su forma de actuar:
  - Deliberativos: derivan del paradigma de pensamiento deliberativo. Los agentes procesan un modelo de razonamiento simbólico interno que les permite planificar sus actuaciones y negociar con otros agentes para alcanzar sus objetivos.
  - Reactivos: tienen su origen en las investigaciones llevadas a cabo por Brooks (1986) y Agre y Chapman (1987). Estos agentes no tienen ningún modelo simbólico interno de razonamiento, sino que actúan según el modo de comportamiento estímulo-respuesta.

En general un agente puede clasificarse por la característica que más destaque e influya en su comportamiento e implementación. El agente ideal es aquel que reúne todas las características que definen a un agente inteligente, sin embargo, hoy en día esto aún no es posible y es la aspiración de los investigadores en tecnología de agentes.

3.5.1. Agentes Móviles

Los agentes móviles son objetos móviles con autonomía flexible que pueden tomar decisiones en cuanto a su movilidad (dónde y cuándo viajar, etc.).

En base al modelo de movimiento empleado, pueden distinguirse dos tipos de agentes móviles:

- Agentes con sólo movimiento de código (*just code*). En el transporte de los agentes móviles sólo se realiza el transporte del código necesario para su ejecución en otro lugar, no se transportan los datos correspondientes al contexto o estado de ejecución. Esto implica que no es posible interrumpir la ejecución de un agente en un ordenador y continuar con dicha ejecución, tras realizarse el movimiento, en otro computador diferente. Ejemplos: TCL, JAVA con RMI (*Remote Method Invocation*).

- Agentes con movimiento de código y contexto de ejecución (*not just code*). En este caso puede interrumpirse la ejecución de un agente en un punto concreto y proseguir, tras realizarse el movimiento a otro ordenador, con la ejecución de dicho objeto desde el punto en el que se interrumpió. Ejemplos: TELESCRIPT, OBLIQ.

Los agentes móviles también se diferencian por el modelo de comunicación utilizado para interactuar entre ellos:

- Agentes con sólo comunicación local. Para que un agente se comunique con otro agente en otra máquina tiene que viajar a esa máquina. Ejemplo: TELESCRIPT.

- Agentes con comunicación por red. No es necesario que un agente se desplace al lugar dónde se encuentra el agente con el que tiene que comunicarse. En este caso el transporte de un agente implica, además del movimiento de código y contexto, el movimiento de las conexiones. Ejemplo: OBLIQ.

3.5.1.1. Ventajas de los Agentes Móviles

Las ventajas que pueden obtenerse con los sistemas de agentes móviles [Lange, 1999] son:

- Reducir la carga de la red. Permiten que los agentes se comuniquen en el mismo nodo y no a través de la red, reduciéndose de esta forma las transferencias de datos por la red. También, cuando hay que procesar una gran cantidad de información ubicada en nodos remotos, puede transferirse el código necesario allí donde se encuentren los datos y procesarse de forma local. La idea es mover el código en lugar de los datos si éstos ocupan un mayor volumen.

- Vencer la latencia de la red. Sistemas de tiempo real estricto, tales como robots, necesitan responder a cambios en su entorno en un tiempo determinado. Realizar un control de los robots a través de la red puede no ser aceptable por las latencias introducidas por ésta. Los sistemas móviles

ofrecen una solución al permitir la transmisión al robot, desde un nodo central, del código necesario para controlarlo de forma local.

- Encapsular protocolos. Para intercambiar información en un sistema distribuido es necesario que cada nodo disponga del código correspondiente al protocolo utilizado en las comunicaciones. Cuando surge la necesidad de cambiar o mejorar el protocolo, surge el problema, a veces insalvable, de actualizar el código de todos los nodos del sistema distribuido. El uso de los sistemas de agentes móviles ofrece una solución al permitir encapsular los protocolos mediante los agentes. De esta forma, para utilizar un determinado protocolo en los nodos, únicamente habrá que transmitirles el agente correspondiente que implementa dicho protocolo.

- Ejecución de tareas asíncronamente y autónomamente. Las conexiones para comunicaciones proporcionadas por los dispositivos móviles suelen ser caras e intermitentes, por ejemplo un ordenador portátil. Cuando se requiere la ejecución en el dispositivo móvil de alguna tarea que precisa una conexión permanente con una red fija, por ejemplo una tarea de búsqueda de información en Internet, su ejecución puede ser poco económica e incluso técnicamente imposible. El uso de los sistemas de agentes móviles permite la creación de un agente con la tarea a realizar y lanzarlo a la red, una vez allí el agente puede operar de forma autónoma y asíncrona permitiendo la desconexión del dispositivo móvil con la red. Cuando dicho dispositivo reanude la conexión, el agente puede volver al dispositivo con los resultados de la tarea realizada.

- Adaptación dinámica al entorno. Los agentes móviles pueden percibir su entorno de ejecución y reaccionar autónomamente a cambios. De esta forma, múltiples agentes móviles tienen la habilidad de distribuirse a través de los nodos de la red manteniendo siempre la configuración óptima para la resolución de un determinado problema. Cada agente puede moverse al nodo donde las condiciones del entorno le sean más favorables para su ejecución.

- Heterogeneidad. La computación en la red es fundamentalmente heterogénea, tanto en *software* como en *hardware*. Debido a que los agentes móviles son independientes de la plataforma física y nivel de transporte, únicamente dependen de su entorno de ejecución, proporcionan unas condiciones óptimas para su integración en el sistema heterogéneo.

- Robustez y tolerancia a fallos. La habilidad de los agentes móviles a reaccionar ante situaciones desfavorables del entorno, facilita la creación de sistemas distribuidos robustos y tolerantes a fallos. Por ejemplo, si un nodo se va a desconectar, puede avisarse y dar tiempo a todos los agentes ejecutándose en él para que puedan migrar a otros nodos y continuar allí su ejecución.

Todas estas características permiten la implementación de una arquitectura híbrida para el control de robots móviles cuyos componentes puedan, por un lado, incorporase al sistema de forma dinámica y, por otro lado, moverse a través de los diferentes nodos con el objetivo de acceder a la información del sistema reduciendo el coste temporal de las comunicaciones, objetivos planteados en la tesis y deducidos en las conclusiones del capítulo uno. Además, el uso de agentes móviles permite emplear técnicas de delegación de código para controlar los robots mediante el envío, en primer lugar, del código necesario a los nodos distribuidos y, posteriormente, la ejecución del mismo de forma local, evitando así los problemas de latencia de la red en la ejecución del código de control, y pudiéndose acotar su tiempo de ejecución para el estudio y cumplimiento de las restricciones temporales del sistema, objetivo también planteado en la tesis.

3.5.2. Agentes Deliberativos

Los agentes deliberativos o cooperativos enfatizan las características de autonomía y cooperación con otros agentes para la ejecución de tareas en entornos multiagentes. Este tipo de agentes es característico de la rama de Inteligencia Artificial Distribuida DAI (*Distributed Artifial Intelligence*) y su modo de operación está basado en un nivel de conocimiento mediante modelos simbólicos de representación interna del entorno.

Ejemplos de agentes deliberativos se encuentran en los sistemas: "ADEPT" [O'Brien, 1996], "MII" [Winter, 1996], "ARCHON" [Jennings, 1995] y "OASIS" [Rao, 1995].

Las críticas a este tipo de agentes se centran en el uso que realizan del paradigma de razonamiento deliberativo, en donde cada respuesta o acción del agente debe determinarse tras un análisis o procesamiento de la información disponible en el agente. Sin embargo, este procesamiento puede provocar tiempos de respuesta que ante determinadas situaciones sean excesivos por requerirse una reacción inmediata por parte del agente. En este sentido, los agentes reactivos ofrecen un punto de vista diferente dando soluciones a tales problemas.

Los agentes deliberativos son adecuados para implementar los componentes que forman parte del nivel deliberativo de las arquitecturas híbridas para el control de robots móviles. Las funciones de dichos componentes son: interacción con el usuario para la especificación de los objetivos, planificación y descomposición de tareas, generación de mapas de entorno, monitorización del sistema, selección de comportamientos básicos.

3.5.3. Agentes Reactivos

Los agentes reactivos representan una categoría especial de agentes que, a diferencia de los agentes deliberativos, no poseen ningún modelo de representación

interna del entorno, sino que responden a éste en forma de estímulo-respuesta. Las diferentes características del entorno donde se encuentre el agente reactivo son percibidas como diferentes estímulos que permiten elaborar una respuesta directa y rápida ante dicho entorno sin necesidad de elaborar una representación interna de éste. Su implementación está basada en la filosofía "la mejor representación del entorno es el propio entorno" [Brooks, 1991a y b].

Las tres ideas que caracterizan a los agentes reactivos son [Maes, 1991]:

- Comportamiento emergente (*emergent functionality*): los agentes reactivos son simples e interactúan con otros agentes mediante mecanismos básicos. El modelo de comportamiento complejo "emerge" cuando se contemplan de forma global todas las interacciones ocurridas entre los agentes. No existe un plan o especificación predeterminada para el comportamiento de los agentes reactivos.

- Descomposición de tareas (*task decomposition*): un agente reactivo puede verse como una colección de módulos que actúan de forma independiente responsabilizándose de tareas específicas. La comunicación entre módulos es mínima y suele ser de bajo nivel. No existe ningún modelo global dentro del agente sino que el comportamiento global tiene que emerger.

- Los agentes reactivos tienden a utilizarse en niveles próximos al nivel físico correspondiente a los dispositivos sensores del entorno.

Los agentes reactivos reaccionan directamente ante los sensores-estímulos sin necesidad de hacer un procesamiento y planificación previos por lo que obtienen tiempos de respuesta menores que los obtenidos por los agentes deliberativos.

Los agentes reactivos son adecuados para implementar los componentes del nivel reactivo de las arquitecturas híbridas para el control de robots móviles. Estos componentes se corresponden con los comportamientos básicos. Si la arquitectura se basa en la teoría de esquemas, los agentes reactivos constituirán los esquemas de percepción y esquemas motores. Componentes que por definición se ejecutan concurrentemente y de forma independiente.

3.6. Taxonomía de un Agente

Se han realizado estudios sobre el modelo conceptual de los agentes basándose en las tareas y habilidades humanas necesarias para la resolución de problemas [Laufmann, 1998]. En este sentido el modelo o taxonomía de un agente puede estructurarse en tres componentes (fig.31):

- Componente de interacción y comunicaciones: nivel de entendimiento. Proporciona al agente la capacidad de interactuar con otros agentes

mediante el uso compartido de protocolos y lenguajes de comunicación. Posibilita la realización de actividades coordinadas.

- Componente de tareas: nivel de recursos. Proporciona los recursos y habilidades del agente para especificar y ejecutar sus actividades o tareas particulares. Componente orientado al tipo de tareas a ejecutar.

- Componente de comportamiento y conocimiento general: nivel de inteligencia. Proporciona al agente comportamientos generales como la cooperación, planificación y priorización de tareas, razonamiento interno, predicción, negociación, etc. Componente orientado a la forma de ejecutar las tareas.

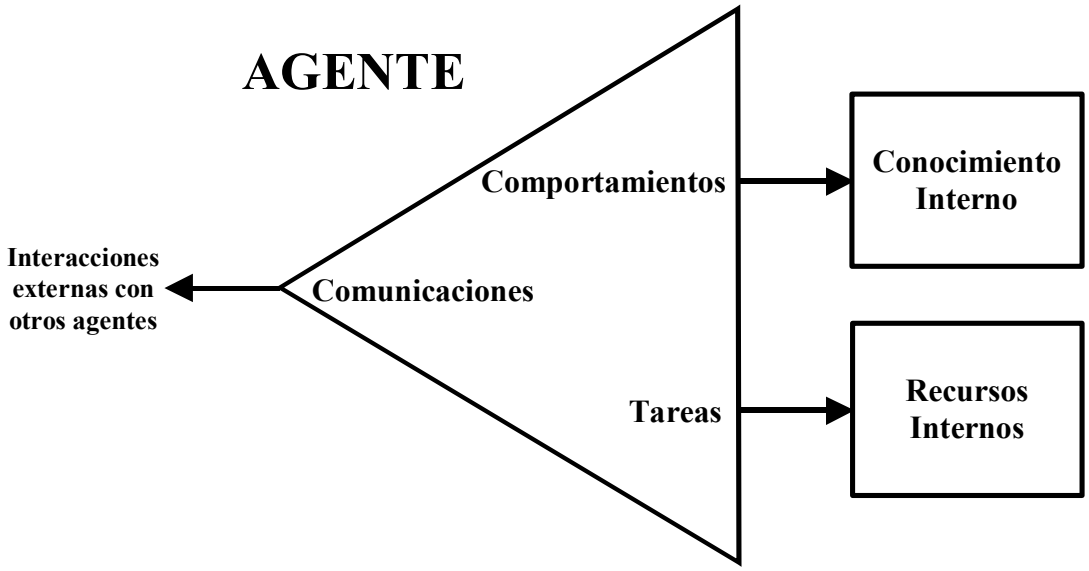


Figura 31: *Taxonomía de un agente: componente de conocimiento, componente de tareas y componente de comunicaciones*

El grado de desarrollo de estos componentes depende de la tecnología de implementación y determina el tipo de agente. De esta forma, según el modelo descrito, en los agentes reactivos tendrá especial importancia el componente de tareas, en los agentes deliberativos el componente de comportamiento y conocimiento general, y en los agentes móviles el componente de interacción y comunicaciones.

3.7. Estándares

En esta sección se realiza una revisión, basándose en los estándares actuales, de las características que han de poseer los sistemas multiagente. En concreto, el estudio se centra en los requerimientos necesarios para permitir la movilidad de los agentes,

característica innovadora en las más recientes metodologías de programación y de gran aplicabilidad en muchos tipos de sistemas como los sistemas de robots móviles, uno de los objetivos de esta tesis.

Los estándares más importantes actualmente desarrollados son: MAF (basado en el estándar CORBA) y FIPA. Ambos tratan de ser lo más generales posibles por lo que realizan una especificación de las características que tienen que poseer los sistemas de agentes sin entrar en detalles de implementación. Para ello, mediante la definición y caracterización de un conjunto de componentes *software*, proporcionan la especificación de una plataforma *software* necesaria para la ejecución y movimiento de los agentes.

FIPA se ha convertido en el estándar con más repercusión y aceptación social. Trata todos los temas relacionados con los sistemas multiagente proporcionando una amplia gama de documentos.

El lenguaje ACL, también desarrollado por el estándar FIPA, es una evolución del lenguaje KQML y se impone como el lenguaje estándar de comunicación entre los agentes. ACL se basa en el uso del envío y recepción de mensajes entre los agentes. Su característica principal consiste en que en los mensajes no sólo se encapsulan los datos sino también su significado, de forma que cualquier agente puede recibir un mensaje ACL y entenderlo. Mediante ACL se proporciona un idioma común a todos los agentes.

El protocolo de comunicación IIOP, establecido por la especificación CORBA para la interoperatividad entre distintas plataformas, se ha convertido en el protocolo por defecto utilizado en los estándares para asegurar la interoperatividad entre todos los sistemas de agentes.

Uno de los objetivos importantes en la tecnología de agentes móviles es la interoperatividad entre diferentes sistemas de agentes. Para facilitar dicha interoperatividad es necesario especificar de una forma estándar acciones como la transferencia de agentes, transferencia de clases y operaciones de control de los agentes. Cuando los sistemas de agentes origen y destino son similares (mismo lenguaje de implementación), la estandarización de estas acciones asegura la interoperatividad. Sin embargo, si los sistemas son diferentes sólo una mínima interoperatividad puede conseguirse, ya que el código que se transfiriera a un sistema con un lenguaje diferente no podría ejecutarse. La migración de agentes sólo puede efectuarse entre sistemas de agentes móviles que soporten el mismo lenguaje.

Las características comunes especificadas en los estándares como necesarias y que tienen que proporcionarse en todos los sistemas de agentes, independientemente del lenguaje de programación empleado, son:

- Operaciones sobre los agentes. Los estándares proporcionan la especificación de las operaciones que el sistema tiene que soportar y aplicar sobre los agentes. Para ello, MAF especifica las interfaces de los objetos

CORBA necesarios y FIPA especifica las acciones (mensajes ACL) correspondientes a los diferentes componentes de la plataforma. Se distinguen dos tipos de operaciones:

  - Operaciones generales de control de los agentes. Estas operaciones son: la creación y ejecución de un agente, suspensión y reanudación de su hilo de ejecución, y terminación del agente. Cada agente debe ser ejecutado de forma independiente al resto.

  - Operaciones de transferencia de agentes. En el caso de los sistemas de agentes móviles se debe permitir que un agente se mueva desde un sistema a otro. Tiene que proporcionarse un mecanismo que permita la captura del estado de ejecución del agente para así poder seguir con su ejecución después de la migración. Asimismo, también es necesario un mecanismo de serialización que permita transformar al agente o clases necesarias en un formato adecuado para ser transmitido a través de la red.

- Nombrado y localización de los agentes y de los sistemas de agentes. Tiene que proporcionarse la sintaxis y semántica de parámetros como el nombre y localización de los agentes o sistemas para así permitir que diferentes agentes se identifiquen a través del nombre, que un agente indique el sistema al que desea moverse y que un sistema pueda determinar si acepta o no la recepción de un determinado agente. Tanto MAF como FIPA especifican las estructuras necesarias para tales requerimientos. En cuanto a las direcciones de los agentes, ambos estándares utilizan direcciones de Internet (encapsuladas mediante formato URL). MAF, además, también ofrece la posibilidad del empleo de nombres CORBA (encapsulados mediante formato URI).

- Mecanismos de comunicación entre los agentes. Es necesario proporcionar un medio de comunicación entre los agentes, bien sea a través de mensajes o mediante invocación de métodos, situados en el mismo nodo o incluso de forma remota. MAF basa las comunicaciones en el empleo del estándar CORBA, mientras que FIPA se basa en el uso del lenguaje ACL.

- Mecanismos de seguridad. Los agentes pueden actuar como virus. Los sistemas de agentes tienen que proporcionar seguridad en la recepción y ejecución de agentes rechazando aquellos que no cumplan las especificaciones de seguridad establecidas. Tendrán que encargarse de la autentificación y verificación de los agentes.

3.7.1. MAF

MAF (*Mobile Agent Facility*) [MAF] es una especificación proporcionada por OMG (*Object Management Group*) para la interoperatividad entre sistemas de

agentes móviles escritos en el mismo lenguaje aunque desarrollados por diferentes compañías. Esta especificación también es conocida como MASIF (*Mobile Agent System Interoperability Facilities*). La interoperatividad de lenguajes no se trata en la especificación, se comenta que es técnicamente difícil de implementar y que tampoco es necesaria su especificación porque puede obtenerse dicha interoperatividad replicando el soporte para diferentes lenguajes en cada nodo.

MAF define interfaces para la transferencia y localización de agentes en sistemas de agentes.

La especificación MAF, desarrollada mediante el lenguaje estándar de definición de interfaces IDL, está formada por dos interfaces (fig.32) basadas en el estándar CORBA (*Common Object Request Broker Arquitecture*): el interfaz MAFAgentSystem y el interfaz MAFFinder.

El interfaz MAFAgentSystem define las operaciones de control de los agentes (creación, terminación, suspensión, transferencia y recepción de agentes) y el interfaz MAFFinder define las operaciones para registrar y localizar agentes y sistemas de agentes.

Las interfaces han sido definidas en el nivel de sistemas de agentes permitiendo la interoperatividad entre éstos. Tanto los sistemas de agentes como los agentes también pueden ser, aunque no necesariamente, objetos CORBA.

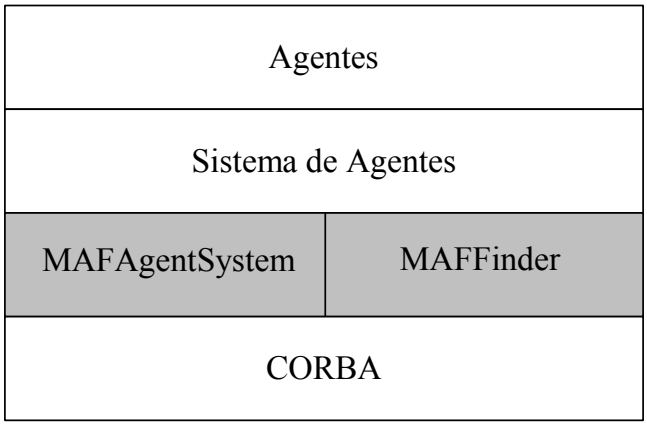


Figura 32: *Modelo Multiagente: estándar MAF para sistemas de agentes móviles basado en CORBA*

Actualmente existen varios sistemas de agentes implementados según el estándar MAF: AGLETS de IBM, Telescript de Magic o AgentTcl de Dartmouth College. Todos estos sistemas difieren entre ellos en los lenguajes de implementación, métodos de serialización y mecanismos de autenticación.

3.7.2. FIPA

El estándar FIPA (*Foundation for Intelligent Physical Agents*) [FIPA] documenta todos los temas característicos de un sistema de agentes (gestión, seguridad, movilidad, comunicación, etc.). De tal forma que para cualquiera de estos temas todos los esfuerzos se hagan en una única dirección.

El estándar FIPA se compone de dos partes claramente diferenciadas:

- Referencias Tecnológicas (*Normative Specification*). Esta primera parte (normativa) se ocupa de especificar una plataforma tecnológica que permita el desarrollo de sistemas multiagente abiertos. Presenta las normas que deben regir el comportamiento e interfaces externas de un sistema de agentes y asegura la interoperatividad con otros sistemas que cumplan dichas normas. El estándar no impone (aunque en algunas ocasiones da recomendaciones) cómo se deben implementar estos sistemas, sino que asegura que independientemente de las opciones tomadas a la hora de ser implementados van a ofrecer un comportamiento similar asegurando una total interoperatividad.

- Referencias a Aplicaciones (*Informative Specification*). Esta segunda parte (informativa) presenta algunas aplicaciones reales que motivan el uso de estos sistemas. Muestra cómo se puede utilizar en aplicaciones industriales la tecnología que se está presentando en la primera parte del estándar. Pretende dar unas pistas de cómo los sistemas multiagente pueden introducirse a nivel comercial.

FIPA define el término "acción" como cualquier actividad capaz de ejecutar un agente. Una clase especial de acciones son los actos de comunicación (*communicative act* o *speech act*). Todas las comunicaciones entre agentes se realizan a través del envío y recepción de mensajes utilizando el lenguaje de comunicación de agentes o ACL (*Agent Communication Language*). Cada mensaje define una acción que tiene que ser soportada/ejecutada por el destinatario.

FIPA proporciona la normativa del entorno donde los agentes se crean y operan. Establece el modelo lógico de referencia para la gestión de agentes (creación, registro, localización, comunicación, migración y terminación de los agentes). El modelo presenta un conjunto de capacidades lógicas y no implica ninguna configuración física, sino que deja los detalles de implementación a elección del equipo de desarrollo.

El modelo de referencia para la gestión de agentes (*agent management*) está formado por los siguientes componentes lógicos (fig.33):

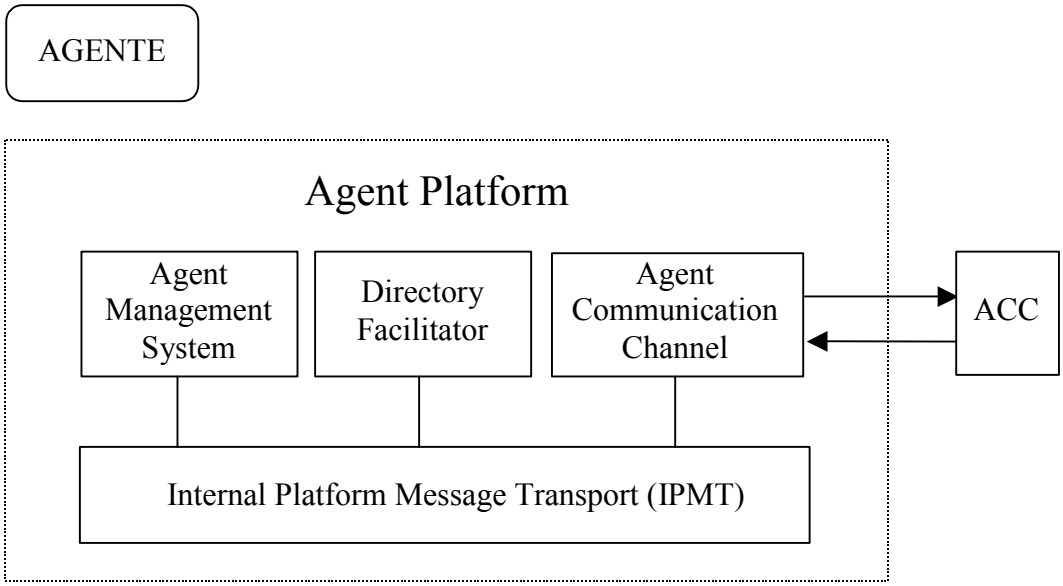


Figura 33: *Modelo Multiagente: arquitectura del estándar FIPA*

- Agente: es el componente básico y principal del modelo. Combina una o más capacidades de servicio dentro de un entorno de ejecución integrado y unificado que proporciona servicios de comunicación, acceso a software externo y acceso a los usuarios. Un agente tiene que tener uno o más dueños. Además debe disponer de una identidad propia proporcionada por un identificador global y único GUID (*Globally Unique Identifier*) denominado nombre del agente. Un agente puede registrarse con un número de direcciones en las cuales puede ser contactado.

- Agent Platform (AP): proporciona la infraestructura física y lógica necesaria en la cual los agentes pueden ejecutarse. Una plataforma de agentes está constituida por el hardware (puede haber varios computadores), el sistema operativo, software de comunicaciones y software de agentes.

- Directory Facilitator (DF): componente que siempre tiene que aparecer en cualquier plataforma de agentes FIPA. Es un agente que proporciona un servicio de "páginas amarillas" a los demás agentes. Un agente puede utilizar DF para registrar sus servicios o para encontrar los servicios ofrecidos por otros agentes.

- Agent Management System (AMS): componente que siempre tiene que aparecer en cualquier plataforma de agentes FIPA, uno por plataforma. Es un agente de gestión que controla el estado y el acceso a la plataforma. También proporciona un servicio de "páginas" que permite la localización de agentes a partir de sus nombres. El AMS es responsable de gestionar operaciones como la creación de agentes, borrado de agentes, decidir si un agente puede registrarse en la plataforma y supervisar la migración de los agentes a/desde la plataforma. El AMS es la autoridad en toda la plataforma.

Puede solicitar a un agente que finalice su ejecución e incluso obligarle a finalizar.

- Agent Communication Channel (ACC): todos los agentes tiene acceso al menos a un ACC. El ACC es el canal de comunicación por defecto entre agentes de diferentes plataformas. Tiene que soportar el protocolo de comunicación para interoperatividad IIOP.

- Internal Platform Message Transport (IPMT): método de intercambio de mensajes dentro de la misma plataforma. Depende de la implementación.

3.7.2.1. ACL

En una sociedad multiagente, los agentes deben cooperar para realizar sus tareas y alcanzar sus objetivos. Los mecanismos de comunicación tradicionales permiten que dos agentes se transfieran información, pero eso es insuficiente cuando el objetivo es conseguir un comportamiento social. Es decir, son necesarios otros mecanismos que permitan dotar a los mensajes intercambiados de un contenido semántico.

Mediante sus acciones, un agente puede contribuir a cambiar el estado de su entorno y así satisfacer sus objetivos. Un agente influye en el conocimiento y acciones de otros agentes mediante un "acto de comunicación" (*speech act*), que se realiza mediante el envío de un mensaje desde el origen al destino. Ambos extremos de la comunicación deben entender el mensaje, y éste debe ser algo más que un intercambio de datos, debe tener asociado un contenido semántico accesible por ambas partes.

La solución adoptada por FIPA se basa en la utilización de mensajes ACL que tienen que ser comprendidos por cualquier agente del sistema. El uso de mensajes ACL permite el envío de información junto a su contenido semántico.

El lenguaje ACL (*Agent Communication Language*) desarrollado por FIPA es una evolución del lenguaje KQML (*Knowledge Query Manipulation Language*) [KQML]. La definición semántica formal de los mensajes ACL ha sido uno de los esfuerzos mayores dentro del estándar FIPA, y otorga a este lenguaje de una gran aceptación como estándar dentro del mundo de los agentes.

Desde este punto de vista, hay que entender un agente como una entidad que entiende el lenguaje ACL y es capaz de intercambiar conocimiento con otros agentes mediante el uso del mismo. Ya no se habla de que la interfaz de un agente (objeto) ofrezca una serie de servicios que se puedan invocar o instanciar, sino que un agente va a proporcionar ciertos servicios que se le pueden solicitar mediante un mensaje ACL que incluya dicha solicitud. Se pasa de un modelo de objetos a un modelo de agentes inteligentes basados en actos de comunicación. Un mensaje ACL está formado por:

- Un identificador del tipo de comunicación. Define el significado principal del mensaje, es decir, lo que el agente origen pretende con dicho mensaje. Por ejemplo: *inform* (un agente transmite información a otro), *request* (un agente solicita a otro que realice una acción), *agree* (un agente acepta la petición realizada), *query* (un agente solicita información a otro).

- Una secuencia de parámetros del mensaje. Es un conjunto de parejas clave-valor que permiten asociar a cada acto de comunicación concreto toda la información necesaria.

3.7.3. Integración de FIPA y MAF

Pueden encontrarse algunas características comunes entre una plataforma MAF y una FIPA relativas a la funcionalidad especificada:

- El FIPA *Agent Management System* (AMS) puede compararse con el sistema de agentes MAF representado por la interfaz MAFAgentSystem. Ambos tienen la responsabilidad de la gestión de los agentes (creación, terminación, suspensión, autentificación, migración, etc.).

- El FIPA *Directory Facilitator* (DF) es similar al componente MAF para los registros representado por la interfaz MAFFinder. La tarea de estas entidades es mantener la información registrada de los agentes en un entorno distribuido.

- El equivalente del FIPA *Agent Communication Channel* (ACC) es el *Object Request Broker* (ORB) en el contexto MAF. Estas entidades se encargan de la transferencia de mensajes en un entorno distribuido de agentes.

- FIPA y MAF proporcionan sus especificaciones independientes de la implementación.

Sin embargo, también existen diferencias asociadas al diseño de cada una de las especificaciones:

- El estándar FIPA aborda toda la funcionalidad necesaria para la ejecución y soporte de agentes móviles mediante un lenguaje (ACL) de descripción de contenidos y acciones de alto nivel. ACL permite la especificación de operaciones y la utilización de protocolos de comunicación de alto nivel.

- El estándar MAF no constituye una base completa para el desarrollo de nuevos sistemas sino que sólo aborda una mínima funcionalidad para así poder adaptarse a las plataformas de agentes ya existentes. La funcionalidad de MAF se establece mediante interfaces IDL y no se utiliza ningún lenguaje de alto nivel sobre los métodos IDL, sino que directamente se corresponden con métodos en la implementación de los objetos.

Los dos estándares pueden ser combinados para unificarse en un único marco de agentes móviles (fig.34). Una solución posible consiste en la integración en la especificación IDL de MAF de operaciones IDL correspondientes a las definidas en FIPA mediante la transferencia de mensajes ACL. Para realizar una plataforma de agentes compatible FIPA y MAF, existen varias posibilidades:

- Las interfaces existentes en MAF (MAFAgentSystem y MAFFinder) tienen que incorporar nuevas operaciones que permitan el acceso a la plataforma FIPA.

- Modificar las operaciones de las interfaces de MAF adaptándolas a los requerimientos de las especificaciones FIPA.

- Añadir nuevas interfaces.

Las dos primeras aproximaciones requieren una modificación del estándar MAF existente, la tercera aproximación es una extensión que no requiere cambios en la especificación.

La especificación FIPA también podría añadir algunos métodos definidos en el estándar MAF. Tendrían que ser métodos que tuvieran una estructura de parámetros simple y que pudieran representarse sin usar un lenguaje de contenidos de alto nivel.

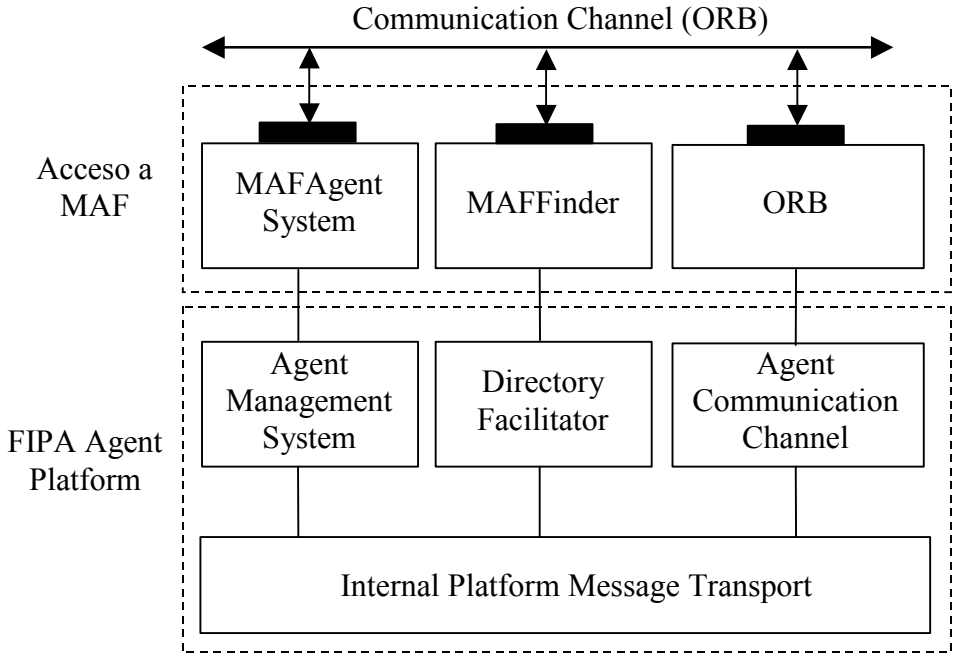


Figura 34: *Modelo Multiagente: integración de MAF y FIPA*

3.8. Implementación de Sistemas de Agentes Móviles

La mayoría de los sistemas de agentes móviles existentes en la actualidad, han partido desde cero para su implementación y han aportado todas o algunas de las características de los estándares descritos mediante soluciones diferentes e independientes.

Los diferentes sistemas disponibles (Telescript, Tacoma, Agent Tcl, Aglets, Voyager, Concordia, Ajanta) [Karnit, 1998], ofrecen una plataforma *software* con un conjunto de primitivas que el programador puede utilizar en el desarrollo de sus agentes para permitirles la migración entre los nodos del sistema.

La implementación de los agentes tiene que realizarse en el lenguaje de programación soportado por el sistema utilizado. Casi todos los sistemas existentes se basan en el lenguaje de programación Java. Esto es debido a que Java, al ser un lenguaje interpretado y ejecutado sobre una máquina virtual independiente de la plataforma, es idóneo [Wong, 1999] para facilitar la transmisión de código y datos. Además, mientras que el uso de otros lenguajes específicos del sistema supondrían al programador la necesidad de aprender un nuevo lenguaje, Java es conocido por la mayoría de los programadores por ser uno de los lenguajes más utilizados en la implementación de las nuevas tecnologías de la información basadas en Internet.

La migración de los agentes se realiza indicando el lugar de destino. Para identificar a los agentes y a los nodos del sistema suele utilizarse, en la mayoría de los sistemas, un servicio de nombres basado en DNS. En la migración de un agente, únicamente los sistemas que permiten capturar el estado de ejecución de dicho agente a nivel de hilo (contexto y pila) ofrecen la posibilidad de continuar con su ejecución a partir de donde se detuvo. El lenguaje Java actualmente no permite realizar dicha captura, por lo que los agentes deben comenzar de nuevo su ejecución una vez concluida la migración.

Las comunicaciones entre los agentes suelen realizarse a través del envío de mensajes o mediante invocación de métodos y, en la mayoría de los sistemas, cuando los agentes implicados se encuentran en el mismo nodo. Aunque en algunos casos también se permite la comunicación remota.

La seguridad en los sistemas suele proporcionarse mediante servicios de autentificación de los agentes y asignación controlada de recursos. En la transmisión de los agentes suele emplearse técnicas de encriptación.

3.9. Conclusiones

En este capítulo se han descrito los fundamentos de los agentes inteligentes. En la programación orientada a objetos o a componentes, el nivel de abstracción consiste

básicamente en encapsular una serie de recursos y datos en un objeto facilitando el acceso a ellos mediante los métodos de dicho objeto. Desde este punto de vista, los agentes, además de poder encapsular recursos y datos, también encapsulan los comportamientos, de forma que para acceder a éstos no se invocan métodos sino que se solicita la realización de acciones. En este sentido, la autonomía del agente puede verse como una encapsulación de comportamientos, obteniéndose de esta forma un nuevo nivel de abstracción.

También se ha realizado un estudio del estado del arte de los sistemas multiagente centrándose en aquellos con características móviles, necesarios para la especificación e implementación de la arquitectura de control de robots que se propone en esta tesis. En este sentido, se han descrito de forma general los estándares existentes en la actualidad conocidos como MAF y FIPA. La idea de los estándares es ofrecer una especificación común para todos los sistemas, siendo el estándar FIPA quien ha recibido el mayor grado de aceptación.

Como conclusión general puede afirmarse la idoneidad de los agentes móviles en la definición e implementación de arquitecturas híbridas para el control de robots móviles de acuerdo a los objetivos planteados en la tesis y requerimientos deducidos en las conclusiones del capítulo uno. En el capítulo siguiente se presenta la arquitectura de control que se propone en esta tesis cuyos componentes son agentes *software* que interactúan a través de una interfaz de comunicaciones que permite, además del acceso a todos los datos del sistema caracterizándolos temporalmente, la movilidad e integración dinámica de los agentes.

Capítulo 4

ARQUITECTURA SC-AGENT

En este capítulo se propone una arquitectura denominada "SC-Agent" para el control de robots móviles inteligentes basada en el paradigma híbrido reactivo/deliberativo y en la delegación de código. Esta arquitectura es multinivel y distribuida, sus principales componentes son agentes software móviles que interactúan con el medio a través de un sistema de comunicaciones basado en una estructura de objetos tipo pizarra distribuida. Dichos agentes pueden ejecutarse de forma local en el propio robot o de forma remota en computadores ajenos basándose en los requerimientos temporales de los mismos.

4.1. Introducción

En las arquitecturas híbridas, las aplicaciones complejas sobre robots móviles se definen mediante la combinación de dos tipos de actividades computacionales: operaciones deliberativas de planificación de objetivos y operaciones reactivas dirigidas por los sensores. Los programas agente descomponen las tareas complejas en comportamientos independientes, los cuales asocian valores de los sensores con acciones a realizar. En primer lugar, el robot elabora la mejor forma de descomponer una tarea en subtareas (proceso de planificación) y, después, selecciona los comportamientos más adecuados para llevar a cabo cada subtarea. La información sensorial tiene que proporcionarse a todos los comportamientos que la necesiten. Además, dicha información también tiene que estar disponible en el proceso de planificación para la construcción de un modelo del entorno.

Un comportamiento se define como una correspondencia entre los valores de un sensor y un patrón de acciones motor, las cuales se aplican para realizar una tarea.

Esta correspondencia es el componente de inteligencia fundamental en la mayoría de los sistemas de robots. Desde principios del siglo XX, en psicología se ha usado la teoría de Esquemas que recientemente se está aplicando en robótica [Arbib, 1986; Arkin, 1998]. Según esta teoría, un patrón de comportamiento está compuesto de al menos un esquema motor, al menos un esquema de percepción, y el conocimiento de cómo coordinar múltiples componentes esquema. Esta separación entre percepción y acción lleva a la composición de un comportamiento mediante la combinación de esquemas de percepción y motores, lo cual sugiere un método de implementación.

Los comportamientos son inherentemente paralelos y distribuidos, al igual que las actividades deliberativas. El diseño del sistema tiene que combinar actividades con restricciones temporales reactivas y actividades con restricciones temporales deliberativas.

4.2. Descripción de la Arquitectura

La mayoría de las arquitecturas de robots móviles están organizadas en niveles. La distribución de los niveles depende del tipo de información que es usada. La información se organiza desde datos sensoriales hasta datos simbólicos que representan mapas de entorno y estados internos del robot.

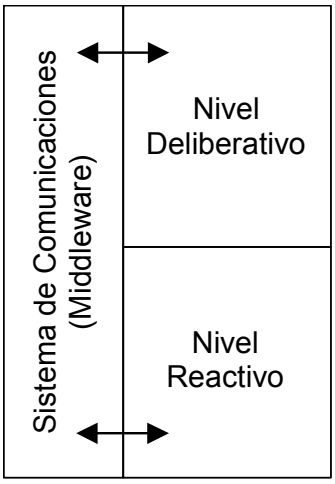


Figura 35: *Arquitectura multinivel propuesta: nivel deliberativo, nivel reactivo y sistema de comunicaciones*

En esta tesis se propone una arquitectura híbrida multinivel y distribuida denominada "SC-Agent". La arquitectura está formada por tres bloques [Posadas, 2002]: un nivel deliberativo, un nivel reactivo y un sistema de comunicaciones interfaz entre ambos niveles (fig.35).

- El nivel deliberativo es un sistema de tiempo real no estricto que representa un nivel de conocimiento superior y ejecuta procesos de planificación basándose en el estado de un modelo simbólico interno del entorno. Como

resultado de la planificación, los objetivos se dividen en patrones de comportamientos que son enviados al nivel reactivo. En las arquitecturas híbridas este proceso se denomina "planificador de misiones" y es la interfaz básica entre el nivel deliberativo y el reactivo. Las técnicas empleadas en este nivel quedan fuera del alcance de esta tesis.

- El nivel reactivo es un sistema de tiempo real estricto conectado con los sensores y los actuadores, tiene que reaccionar en un tiempo acotado ante los diferentes valores de los sensores (estímulos). El nivel reactivo recibe del nivel deliberativo los patrones de comportamientos, los cuales son procesos que se ejecutan concurrentemente y que usan la información sensorial para el cálculo de las acciones que llevarán a cabo los actuadores. Basándose en la teoría de esquemas, los patrones de comportamientos están formados por:

    - Procesos esquemas de percepción: generan información que podrán utilizar otros procesos esquema. Por ejemplo, pueden acceder a los valores de los sensores y llevar a cabo tareas de fusión sensorial.

    - Procesos esquemas motores: generan posibles acciones para los actuadores. Las acciones de un esquema motor generadas en base a determinados esquemas de percepción son el resultado de lo que se denomina comportamiento básico. Por ejemplo, el comportamiento básico de evitar obstáculos podría estar formado por un esquema de percepción que en función del valor de los sensores de infrarrojos detectara la presencia o no de obstáculos y un esquema motor que, detectado un obstáculo, calculara la velocidad a aplicar a los motores de las ruedas para así esquivar dicho obstáculo.

    - Procesos compositores: combinan distintos procesos esquemas y obtienen las acciones finales para los actuadores. Cuando varios esquemas motores calculan diferentes acciones para el mismo actuador, es decir, cuando existen distintos comportamientos básicos sobre el mismo actuador, son los procesos compositores quienes deciden en cada instante las acciones definitivas. Para ello, existen dos posibilidades [Simó, 1997b]:

        - Composición de comportamientos básicos (comportamientos emergentes). Se realiza una suma ponderada de todas las acciones para un mismo actuador obteniendo como resultado la acción definitiva (por ejemplo, la suma de un comportamiento básico de navegar hacia un objetivo más un comportamiento básico de evitar obstáculos que resultara, en un robot, el comportamiento emergente de navegar hacia el objetivo evitando al mismo tiempo los obstáculos encontrados en el camino).

- Conmutación o secuenciación de comportamientos básicos. Existen situaciones donde no se consideran al mismo tiempo todos los comportamientos básicos para el cálculo de las acciones finales (por ejemplo, si un robot tiene que coger un objeto, primero tendrá que moverse hacia él, en ese caso los esquemas motores que generan las acciones necesarias para coger el objeto sólo tendrán que ser considerados después de que el robot se haya movido y haya alcanzado dicho objeto).

En ambos casos, cada comportamiento básico (esquema motor con esquemas de percepción asociados) tiene un peso o contribución en las acciones finales llevadas a cabo (fig.36). Esta contribución se determina por un valor o coeficiente de motivación (entre 0 y 1) [Simó, 1997c; Posadas, 2001] que otro proceso asociado a cada motor esquema tiene que calcular y actualizar de forma continua. Los procesos de motivación utilizan la información sensorial para calcular dichos coeficientes. Cuando un esquema motor tiene una motivación igual a 0 su contribución será nula en la acción final (la conmutación de comportamientos puede realizarse de esta forma), en este caso, podría existir un proceso planificador que parara la ejecución de los comportamientos básicos sin ninguna contribución.

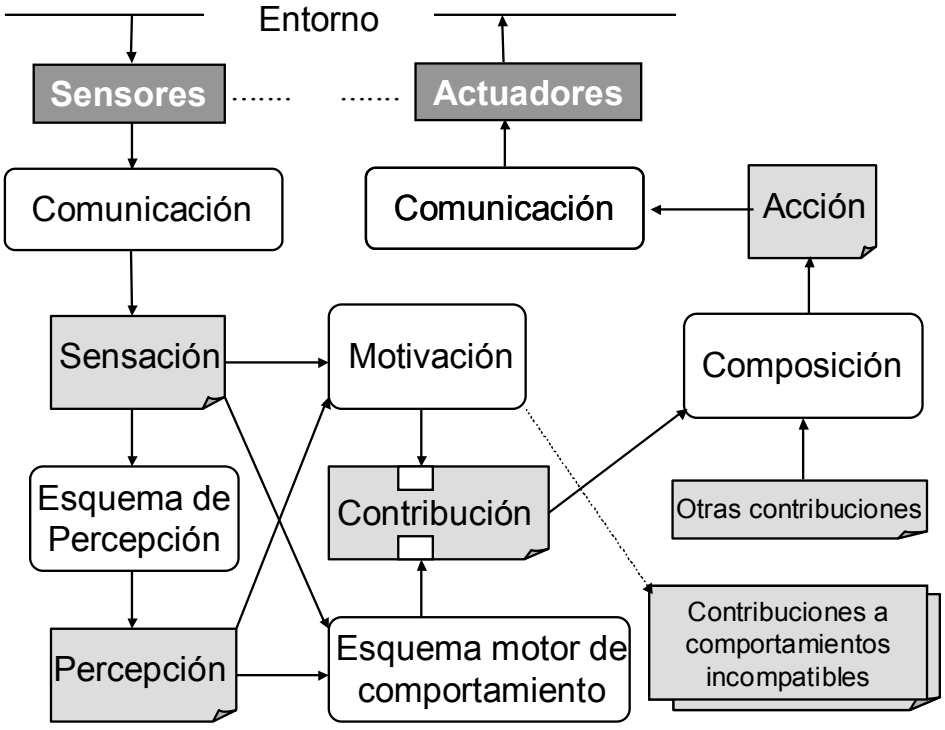


Figura 36: *Nivel reactivo: composición de comportamientos basada en motivaciones*

- El sistema de comunicaciones proporciona la infraestructura *hardware* y *software* necesaria para el acceso a los sensores y actuadores y para la interacción entre los distintos componentes de la arquitectura. Constituye una interfaz entre los niveles reactivo y deliberativo mediante la cual, el nivel deliberativo envía los patrones de comportamientos al nivel reactivo y accede a la información sensorial para incorporarla en el modelo interno simbólico del entorno. El diseño del sistema de comunicaciones contempla los diferentes requerimientos temporales de los niveles reactivo y deliberativo.

4.2.1. Acoplamiento entre los Niveles Deliberativo y Reactivo

El objetivo principal de esta arquitectura es permitir el desarrollo rápido de sistemas de control de robots donde el nivel deliberativo interacciona con el nivel reactivo enviándole las tareas a realizar. Para asegurar el control del robot ante los cambios producidos en el entorno, el nivel reactivo debe poseer cierta autonomía, la cual está directamente relacionada con la capacidad de dicho nivel de reacción en tiempo real.

El grado de autonomía del nivel reactivo está determinado por el tipo de acoplamiento existente con el nivel deliberativo. Dicho acoplamiento, se caracteriza por la forma en la que el nivel deliberativo interacciona con el nivel reactivo. Atendiendo al grado de autonomía, existen varias formas de acoplamiento:

- Envío de referencias a los controladores locales. Es el caso con menor autonomía. El nivel reactivo no tiene ninguna autonomía porque recibe del nivel deliberativo todas las acciones de control correspondientes al nivel más básico que tienen que ser ejecutadas.

- Conmutación de diferentes modos de funcionamiento del robot. En este caso el nivel reactivo tiene preprogramados diferentes modos de operación y el proceso de control consiste en la activación de los mismos cuando son necesarios.

- Planificación en diferido de las acciones a tomar para realizar una misión. Las acciones de alto nivel son generadas por el nivel deliberativo y entonces enviadas al nivel reactivo. El nivel reactivo tiene que poseer la infraestructura necesaria para poder interpretar y ejecutar dichas acciones.

- Influencia o control mediante “Emociones”. Es el caso que exige mayor autonomía al nivel reactivo. El proceso de control por parte del nivel deliberativo se reduce al envío al nivel reactivo de nuevas “emociones” para ser consideradas en el ciclo de vida de dicho sistema. El sistema autónomo tiene que producir y decidir sus propias acciones, lo cual no es una tarea fácil. La arquitectura propuesta puede considerarse una aproximación a esta forma de control donde las “emociones” podrían modelarse mediante los

patrones de comportamientos (procesos esquemas y de motivación) enviados al nivel reactivo. El nivel reactivo genera sus propios comportamientos emergentes como resultado de la composición de los comportamientos básicos.

En la práctica, cualquier robot inteligente es controlado mediante alguna forma de teleoperación (la teleoperación se produce cuando un operador humano controla un robot desde la distancia). El grado de teleoperación de un sistema está directamente relacionado con el grado de interacción entre el usuario y el nivel deliberativo. En última instancia, el operador especificaría únicamente cuál ha de ser la misión o, por el contrario, el nivel deliberativo podría reducirse a la interfaz que permita al usuario el envío de las acciones a ejecutar por los actuadores. En este sentido, la autonomía no evita la teleoperación, sino que ésta puede clasificarse igualmente atendiendo al grado de autonomía del sistema.

4.2.2. Marco del Control Distribuido

El diseño de una arquitectura de control distribuido tiene que tener presente las siguientes características (fig.37):

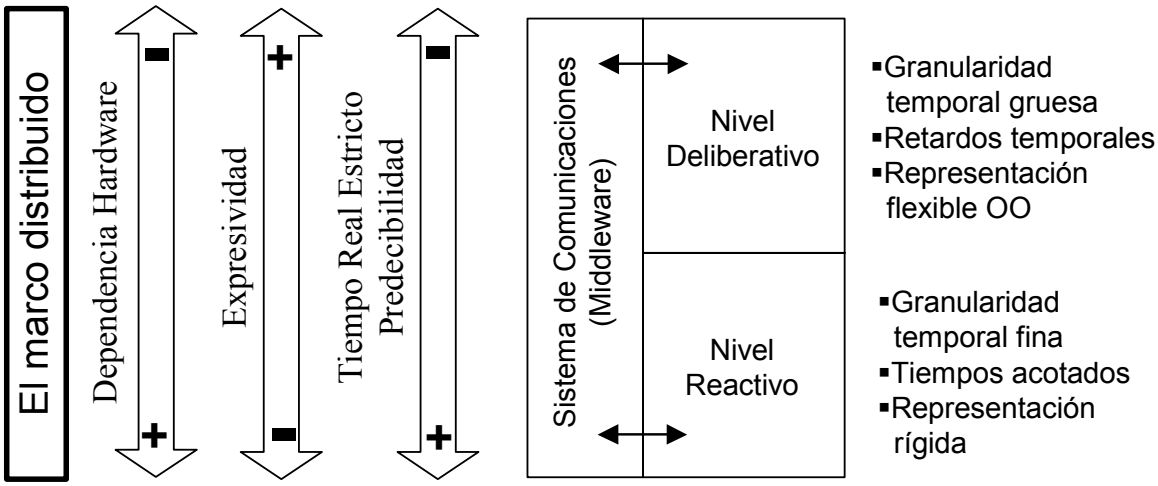


Figura 37: *Marco del control distribuido: restricciones temporales, expresividad vs. predecibilidad*

- Restricciones temporales. El sistema tiene que garantizar las restricciones de tiempo real estricto de las tareas. Esto puede ser difícil debido a los siguientes problemas:

    - Dinámica de los controladores locales. Los controladores locales de sensores y actuadores están sujetos a la dinámica de los transductores utilizados, lo cual introduce retrasos en las medidas de los sensores y en la aplicación de señales de control. Además, en el caso en que el

controlador local tenga capacidad computacional, el tiempo empleado en la misma también introducirá retrasos.

- Retardos de computación. Una sobrecarga de procesamiento o una falta de recursos del Sistema Operativo pueden producir retardos en la ejecución.

- Retardos de transmisión. Algunos canales de comunicación ocasionan retardos excesivos en la transmisión. Por otro lado, el tiempo necesario de transmisión en algunas redes (por ejemplo en Internet) no puede ser acotado y muchas veces las comunicaciones ni siquiera son posibles debido a las condiciones del entorno (por ejemplo en las redes inalámbricas). En este sentido, es difícil controlar los requerimientos de tiempo real desde un nodo remoto si el sistema no tiene ninguna autonomía.

- Diferentes dominios: expresividad vs. predecibilidad. Expresividad y predecibilidad son dos características deseables en cualquier especificación de sistema de control en general. La expresividad es fundamental en la programación de agentes deliberativos y puede proporcionarse mediante un dominio de representación orientada a objetos flexible, lo cual suele implicar granularidad temporal gruesa y retardos temporales. La predecibilidad en la ejecución del código es fundamental para poder analizar el código y asegurar que se ejecutará bajo las restricciones de tiempo real impuestas. Este escenario es el que ofrece el sistema reactivo. La predecibilidad requiere de un dominio con granularidad temporal fina y tiempos acotados, lo cual suele implicar una representación rígida. Los sistemas híbridos son una solución posible donde el nivel reactivo proporciona la predecibilidad y el nivel deliberativo proporciona la expresividad. El sistema de comunicaciones tendrá que proporcionar una interfaz entre ambos niveles.

4.2.3. Movilidad de los Componentes

Los componentes *software* (procesos) de los niveles reactivo y deliberativo son agentes móviles que pueden moverse entre los diferentes nodos de la arquitectura distribuida. El objetivo principal de esta movilidad es reducir la sobrecarga y requerimientos de comunicación del sistema. La movilidad es necesaria de acuerdo con los siguientes criterios:

- La sobrecarga y requerimientos de comunicación del sistema dependen de la ubicación de los agentes. Tanto el nivel reactivo como el deliberativo son distribuidos. El nivel deliberativo envía al nivel reactivo agentes que necesitan diferentes datos sensoriales, la localización o ubicación de estos agentes determinará la sobrecarga y requerimientos de comunicación necesarios para enrutar a sus destinos dicha información sensorial.

- Un agente podrá moverse allí donde las condiciones del entorno sean más favorables (mejor tiempo de ejecución, disponibilidad de recursos, menor necesidad de comunicaciones, etc.).

- Las condiciones del entorno pueden cambiar dinámicamente (recursos del Sistema Operativo, disponibilidad *hardware*, etc.) y puede ser necesario el movimiento de los agentes para que el sistema se adapte a las nuevas condiciones.

La movilidad introduce más expresividad en el sistema. El hecho de que los agentes sean móviles para poder viajar entre los nodos les ofrece la posibilidad de encontrar dinámicamente su ubicación óptima. Esta idea introduce un nuevo nivel semántico en la ejecución de código donde se indica qué hay que ejecutar pero no dónde. La ubicación física de los componentes a ejecutar se realizará en el sistema de forma dinámica e inteligente adaptándola en tiempo real a las necesidades y cambios del mismo. Todo ello implica la necesidad de un índice de medida que indique la adecuación del agente en el nodo donde está en ejecución o si, por el contrario, requiere moverse a otro nodo. Para la especificación y el cálculo de este índice, como se introduce en las siguientes secciones, será fundamental la incorporación al sistema de un control temporal en la ejecución de los agentes y en el acceso a los datos.

4.2.4. Procesamiento vs. Comunicaciones: Control por Delegación de Código

La arquitectura propuesta, basada en agentes móviles, reduce los problemas relacionados con las restricciones temporales al separar el tiempo de comunicaciones del tiempo de procesamiento.

Las comunicaciones pueden clasificarse en dos tipos:

- Comunicaciones fuera de línea: son las comunicaciones necesarias para la configuración del agente antes de su ejecución (por ejemplo, podría requerirse el movimiento de dicho agente mediante la delegación de su código). Este tipo de comunicaciones pueden realizarse antes de la ejecución en tiempo real o utilizando ranuras temporales libres del bus de comunicación.

- Comunicaciones en línea: son las comunicaciones necesarias para obtener los datos en tiempo de ejecución. Sus características temporales (retardos de transmisión) dependen de la ubicación de los datos. En este sentido hay dos posibilidades:

    - Comunicaciones en línea locales. Los datos se producen y/o consumen en el mismo nodo donde el agente es ejecutado.

- Comunicaciones en línea remotas. Los datos se producen y consumen en otros nodos remotos al nodo donde el agente es ejecutado.

La arquitectura propuesta reduce los problemas relacionados con las restricciones temporales reduciendo la sobrecarga de las comunicaciones en línea [Posadas, 2003]. Para ello, se emplean dos fases. En primer lugar, la arquitectura permite realizar todas las comunicaciones fuera de línea con el objetivo de delegar al robot (moviendo y clonando agentes) el código de ejecución necesario. En segundo lugar, la arquitectura procesa el código usando la infraestructura de tiempo real. La primera fase no está acotada temporalmente, el tiempo necesario depende de las condiciones del entorno. En la segunda fase, una vez han finalizado las comunicaciones fuera de línea y el robot dispone de sus agentes, los agentes pueden ejecutarse localmente con un mínimo de comunicaciones en línea. En el caso ideal, si los agentes están ubicados de forma que disponen localmente de toda la información necesaria, los agentes pueden ejecutarse sin comunicaciones externas que interrumpan el procesamiento. De esta forma, al contrario que la primera fase, la segunda fase puede ser acotada temporalmente debido a que el tiempo necesario depende principalmente del código ejecutado localmente y el sistema local es más predecible. Las restricciones de tiempo real estricto en el procesamiento local pueden garantizarse.

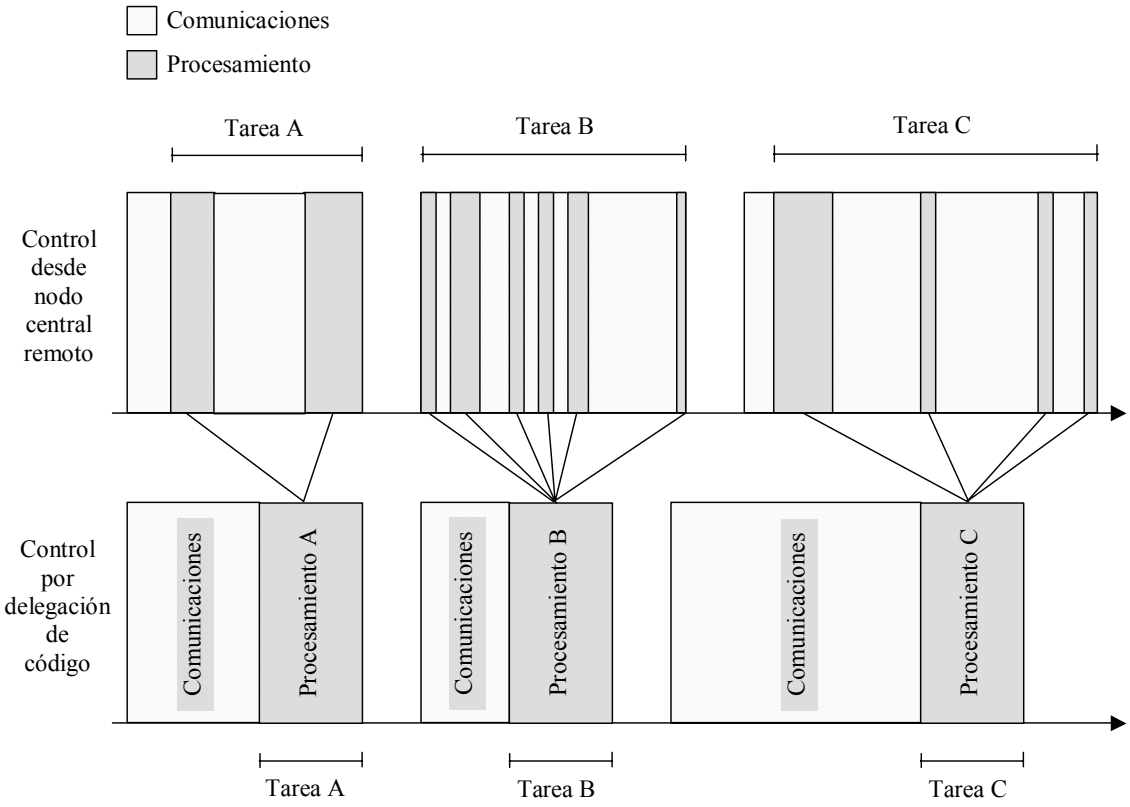


Figura 38: *Comparativa del control de robots desde un nodo central remoto y mediante delegación de código: separación del tiempo de las comunicaciones del tiempo de procesamiento*

En la figura 38 puede observarse la diferencia entre el control de un robot que necesita comunicarse continuamente con un nodo central remoto (esta situación se encuentra en la mayoría de los sistemas de teleoperación donde se comunican referencias para los controladores que se ejecutan localmente), y el control de un robot mediante la arquitectura propuesta. En el primer caso, el robot no dispone localmente del código necesario para realizar sus tareas, por lo que dicho robot tiene que estar continuamente comunicándose con el nodo remoto, donde dicho código está ejecutándose, para recibir las acciones a realizar. Por el contrario, en el segundo caso ideal, el código necesario se ejecuta localmente sin retardos de comunicación.

4.2.5. Requerimientos del Sistema de Comunicaciones

El sistema de comunicaciones ha de constituir la base para la especificación e implementación de los niveles reactivo y deliberativo distribuidos.

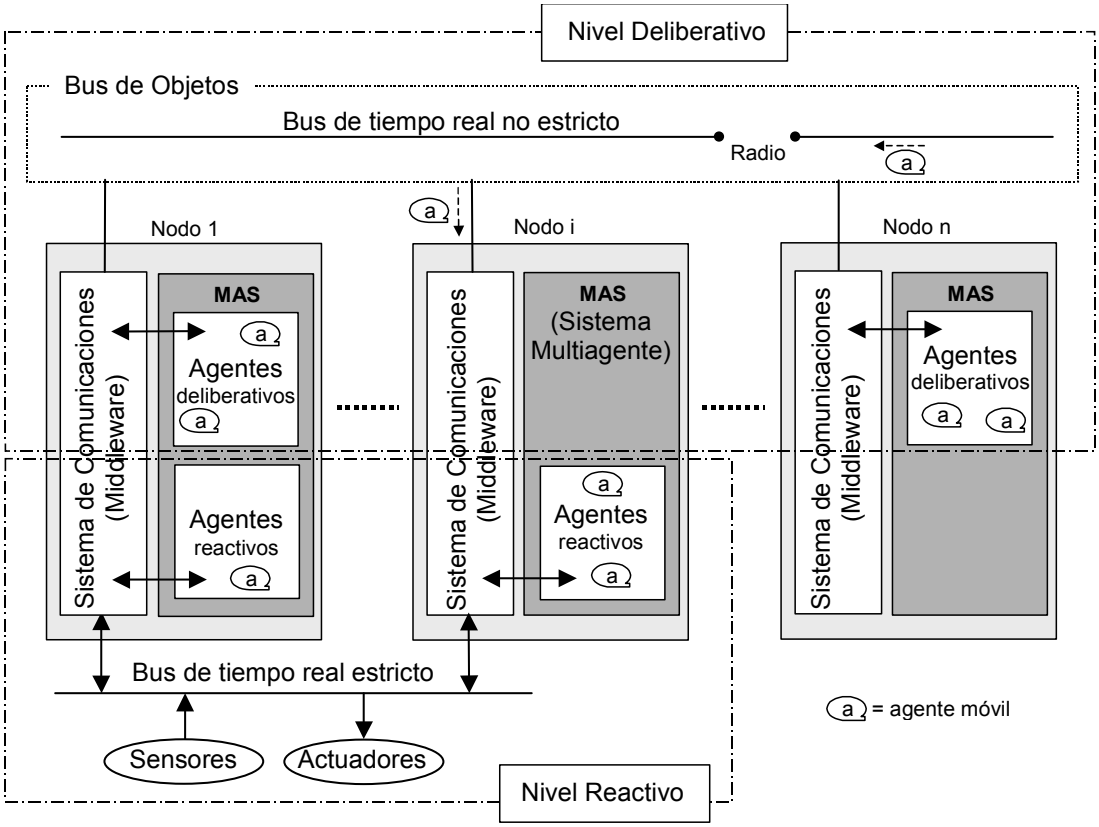


Figura 39: *Estructura y conexiones de la arquitectura distribuida propuesta. Buses de tiempo real estricto y no estricto requeridos por el sistema de comunicaciones*

Para permitir las comunicaciones en el nivel reactivo, los nodos reactivos tendrán que estar interconectados mediante un bus de tiempo real estricto. Por otro

lado, para permitir las comunicaciones en el nivel deliberativo, los nodos deliberativos tendrán que estar interconectados mediante un bus de tiempo real no estricto.

Ambos buses son necesarios debido a las diferencias existentes entre los requerimientos temporales de los niveles reactivo y deliberativo [Posadas, 2000]. El nivel reactivo necesita un bus de comunicaciones que garantice el tiempo de acceso y la transmisión de las tramas de datos para responder en un tiempo acotado a los cambios del entorno. Por su parte, el nivel deliberativo puede usar un bus sin estas restricciones que por el contrario permita comunicar estructuras de datos más complejas (como supondría la propia transmisión de los agentes).

La ejecución en tiempo real estricto de los agentes reactivos solo será posible en aquellos nodos que dispongan de la infraestructura de acceso al bus de tiempo real.

El sistema de comunicaciones o *Middleware* tendrá que estar conectado con ambos buses para así proporcionar una interfaz entre el nivel reactivo y deliberativo que permita transmitir información entre ambos niveles.

En la figura 39 puede observarse la estructura y conexión de los nodos que implementan los distintos niveles de la arquitectura distribuida.

4.2.6. Necesidad del Control Temporal

El control temporal puede aplicarse a dos niveles: control temporal de la ejecución de los procesos y control temporal del acceso a los datos.

El control temporal de la ejecución de los procesos implica el control del tiempo que los procesos o agentes del sistema requieren y emplean para su ejecución. En este nivel pueden incluirse los estudios sobre la planificabilidad de los sistemas que permiten asegurar los tiempos periódicos de ejecución de los procesos. En un sistema donde no se conoce a priori ni el número ni la ubicación de los procesos o agentes que tienen que ejecutarse, características que dependen de la dinámica del sistema, no pueden calcularse los tiempos correspondientes a los "peores casos" de ejecución. Sin embargo, sí pueden medirse los tiempos de ejecución reales, lo cual permite la adaptación dinámica de los agentes en función de sus restricciones temporales y tiempos reales de ejecución (por ejemplo, si un agente no consigue ejecutarse en un tiempo inferior o igual al requerido en su especificación, podría moverse a otro nodo del sistema distribuido donde se cumplieran dichas características temporales).

El control temporal del acceso a los datos implica el control del tiempo requerido y empleado en el acceso a los datos del sistema. En un bus de tiempo real estricto puede estimarse el tiempo de transmisión y acceso a los datos correspondiente al "peor caso" mediante un estudio previo de la planificabilidad del acceso al bus. Sin embargo, en un bus de tiempo real no estricto no puede calcularse dicha estimación, aunque sí pueden obtenerse los "retrasos reales" en el acceso a los

datos. El "retraso real" de un dato proporciona su antigüedad. Utilizar "retrasos reales" implica, al no disponer a priori del tiempo correspondiente al "peor caso", no poder planificar el momento de recepción de los datos. Sin embargo, utilizar "retrasos reales" permite al agente, en el momento de la recepción de los datos, validar los datos considerando su antigüedad y adaptarse dinámicamente a los tiempos de recepción reales (por ejemplo, moviéndose el agente a otro nodo donde los tiempos de recepción sean menores).

Las necesidades de control temporal de la arquitectura propuesta son las siguientes:

- Necesidad del control temporal de la ejecución de los agentes. Debe asegurarse la ejecución de aquellos agentes reactivos cuya omisión pondría en peligro al sistema. Para ello, cada agente del sistema debe poseer como restricción temporal un periodo de ejecución que ha de cumplirse con mayor o menor precisión dependiendo de la importancia del agente (reactivo o deliberativo). Por ejemplo, si el sistema no pudiera cumplir los periodos de ejecución de los agentes necesarios para que el robot vaya a una determinada velocidad esquivando los obstáculos que encuentre en su camino, podría optar por reducir la velocidad (con la consiguiente degradación de prestaciones pero asegurando la integridad del sistema mediante el uso de periodos de ejecución mayores) o podría optar por delegar a otros nodos la ejecución de agentes no críticos (en caso de existir) para así asegurar la ejecución de los estrictamente necesarios en ese momento.

- Necesidad del control temporal del acceso a los datos. La arquitectura requiere un estudio de la planificabilidad del bus de tiempo real estricto y una caracterización temporal de los datos.

    - Estudio de la planificabilidad del bus de tiempo real. Los agentes reactivos de la arquitectura propuesta tienen que responder en un tiempo acotado a los cambios del entorno, para ello es necesario realizar un estudio previo de la planificabilidad del acceso al bus de tiempo real. Este estudio asegura la transmisión periódica de todos los tipos de datos (valores de los sensores y los actuadores) para así poder estimar los tiempos de acceso en el "peor caso".

    - Caracterización temporal de los datos. Es necesario que se proporcione la antigüedad de los datos disponibles en el sistema para así permitir las siguientes tareas:

        - Ejecución en nodos sin infraestructura de bus de tiempo real. Los agentes deliberativos o agentes reactivos con restricciones temporales no estrictas pueden ejecutarse en nodos sin infraestructura de bus de tiempo real. En estos casos el acceso a los datos se realiza a través del bus de tiempo real no estricto, sobre el cual, a diferencia del bus de tiempo real

estricto, no pueden estimarse los tiempos de acceso en el "peor caso". Por ello, el sistema de comunicaciones de la arquitectura propuesta necesita proporcionar los parámetros temporales mediante la medición del "retraso real" en el acceso a los datos.

- Procesos de fusión espacio-tiempo. Generalmente, los procesos deliberativos están relacionados con la integración sensorial, fusión de datos y construcción de mapas de entorno. En este caso, donde la fusión sensorial tanto en el espacio como en el tiempo es necesaria, tiene que asociarse información temporal a los datos de los sensores y a las acciones de control. El problema de realizar una fusión sensorial en el dominio del espacio-tiempo reside en que, por el principio de observabilidad, cuando una aplicación recibe una medida (instante t_{uso}) ya ha transcurrido un determinado tiempo desde su observación (instante t_{obs}), del tiempo que haya transcurrido depende que la medida siga siendo válida o haya que realizar una estimación del valor actual en función de la evolución del sistema. Por lo tanto, es prerrequisito importante poder conocer el intervalo de tiempo dado por t_{uso}-t_{obs} (antigüedad de la información) para poder determinar la validez de una medida e integrarla en la fusión espacio-tiempo.

- Establecer un criterio de movilidad para los agentes. Debido a que el modelo de arquitectura propuesta no es rígido y sus características pueden variar de forma dinámica (carga en cada computador, restricciones temporales de los procesos, número de computadores con y sin acceso al bus de tiempo real), los agentes deberán poder determinar dinámicamente el grado de adecuación al nodo en el que residen mediante el cálculo de algún índice de medida. Para dicho cálculo, es necesario el control temporal tanto de la ejecución de los agentes como del acceso a los datos caracterizándolos con su antigüedad. El índice de medida empleado se ha denominado "Índice de Comodidad" y su formulación se describe en la sección siguiente.

Puede concluirse que la arquitectura propuesta requiere del control temporal de los datos mediante la caracterización de su antigüedad. Para ello, el bus de tiempo real estricto permite estimar los tiempos de acceso en el "pero caso" pero el bus de tiempo real no estricto requiere del uso de otras técnicas basadas en la medición de los "retrasos reales".

Para calcular en una arquitectura distribuida la antigüedad real (no peores tiempos de acceso) de los datos, es preciso disponer, bien de un reloj global y

sincronizado en todos los nodos, bien de un sistema de estimación de los tiempos de transmisión y procesamiento en el acceso a los datos [Kopetz, 1997].

Disponer de un reloj global que permita medir tiempos absolutos en el sistema requiere la sincronización de todos los relojes locales (cada nodo dispone de un reloj local que hay que sincronizar con el resto) lo cual requiere resolver principalmente los siguientes problemas:

- Corrección del desplazamiento (offset). Cada reloj local presenta un desplazamiento de su frecuencia con respecto al resto. El objetivo de la sincronización es obtener un reloj global con una única frecuencia. Para ello tiene que corregirse el desplazamiento de cada uno de los relojes locales tomando como referencia aquel que se considere como global.

- Corrección de la deriva (drift). La frecuencia de cada reloj local, además de presentar un "offset" con respecto al resto, presenta una deriva que también hay que corregir en la sincronización.

Sin embargo, un sistema de estimación de tiempos puede basarse en el uso de medidas relativas, es decir, locales a cada nodo, por lo que se evitan los problemas derivados de la sincronización de relojes. Con este sistema no es necesaria la sincronización de relojes porque lo que se calcula no es el tiempo absoluto de un dato con respecto a un reloj global, sino los retardos del dato que pueden obtenerse con medidas relativas a los relojes de cada nodo. Para ello, por un lado se calculan los retardos de permanencia del dato en cada uno de los nodos (no es necesario un reloj global, basta con el reloj local) y por otro lado se calculan los retardos de transmisión del dato entre nodos (en este caso también existen protocolos de estimación de tiempos de transmisión que no requieren de ningún reloj global). La suma de todos los retardos proporciona la antigüedad requerida del dato sin necesidad de sincronizar relojes.

La arquitectura propuesta está basada en un mecanismo básico de estimación proporcionado por el sistema de comunicaciones [Poza, 2001; Posadas, 2002]. Según este modelo, cada dato del sistema está caracterizado con un campo que acumula sus retardos de transmisión y tiempos de permanencia en los nodos. De esta forma, a partir del valor de dicho campo se proporciona la antigüedad actual del dato.

Ninguno de los estándares de comunicaciones con más relevancia en la actualidad ni, por consiguiente, ninguno de los estándares para sistemas de agentes móviles, ofrece un modelo para el control temporal de la información que se transmite. Es por ello que se hace necesario el obtener un sistema paralelo (que no incompatible) a los estándares actuales que posibilite a las aplicaciones o agentes el disponer de forma transparente de un sistema global de tiempos.

4.2.7. Índice de Comodidad

Se ha denominado "Índice de Comodidad" a la medida en la que se basan los agentes para decidir si el nodo donde residen es adecuado para su ejecución o por el contrario deben moverse a otro nodo distinto.

La ubicación de cada agente en un computador u otro depende de varios factores:

- En función de las restricciones de tiempo real de su tarea. Si el sistema no puede cumplir el periodo de ejecución del agente, éste podría moverse a otro nodo donde sí se cumpliera su periodo de ejecución. A diferencia de los agentes deliberativos, el no cumplimiento del periodo de ejecución de un agente reactivo podría ser crítico en la integridad del sistema.

- En función de la antigüedad de los datos que reciba. Un agente ejecutándose en un computador con acceso directo al bus de tiempo real estricto recibirá los valores transmitidos mucho antes que un agente ejecutándose en otro computador conectado al sistema a través del bus de tiempo real no estricto. Por ello, un agente que forme parte del nivel reactivo probablemente deberá ejecutarse en un nodo con acceso directo al bus de tiempo real estricto, para así minimizar los tiempos tanto de recepción de la información como de actuación y poder realizar un control temporal más preciso evitando las estimaciones realizadas en las transmisiones por buses de tiempo no real. Sin embargo, un agente deliberativo perfectamente podrá ejecutarse en un nodo donde la información temporal de los valores que reciba no sea tan precisa como antes aunque sí con una antigüedad estimada adecuada para la realización de su tarea. Todo ello implica que para cada agente habrá que especificar, como parte de sus restricciones temporales, la máxima antigüedad admisible para cada uno de los datos que utilice en su procesamiento.

La formulación del "Índice de Comodidad" (I_C) viene dada por la siguiente expresión:

$$I_C = Min(I_{Act}, I_{Dat})$$

donde I_{Act} denota un "Índice de Actividad" que es igual a:

$$I_{Act} = \frac{t_{MAXComp} - (t_{Esp} + t_{Proc} + t_{Com})}{t_{MAXComp}}$$

siendo:

- $t_{MAXComp}$: restricción temporal del agente que indica el máximo tiempo de computación o ejecución permitido para el procesamiento de la tarea asignada. Este valor se corresponderá con el periodo de ejecución del agente.

- t_{Esp}: tiempo de espera real correspondiente al retraso producido en la ejecución del agente debido a interrupciones del sistema o expulsiones por tareas más prioritarias del sistema.

- t_{Proc}: tiempo real correspondiente al procesamiento de la tarea asignada al agente.

- t_{Com}: tiempo real correspondiente al retraso producido en la ejecución del agente debido a las tareas de comunicación (por ejemplo, en su caso, para el acceso a los datos).

e I_{Dat} denota un "Índice de Acceso a los Datos" que es igual a:

$$I_{Dat} = Min\left(\frac{t_{MAXAnt}(d) - t_{Ant}(d)}{t_{MAXAnt}(d)}\right) \forall d \in L$$

$$L = \{D_1, D_2, ..., D_n\}$$

siendo:

- n: número de datos que el agente requiere en su ejecución.

- L: lista con el conjunto de datos (D_1, D_2, ..., D_n) que el agente requiere en su ejecución.

- $t_{MAXAnt}(d)$: tiempo máximo de antigüedad del dato d para que sea válido en el procesamiento del agente.

- $t_{Ant}(d)$: tiempo real de antigüedad del dato d.

El "Índice de Actividad" de un agente indica si se están cumpliendo las restricciones de máximo tiempo de cómputo para dicho agente. Para ello, se tiene que contabilizar el tiempo real de ejecución del agente mediante la suma de su tiempo de procesamiento más los retrasos correspondientes a los tiempos de espera por expulsiones e interrupciones y los tiempos de espera por las comunicaciones. Cuando dicha suma supera el valor máximo permitido el valor del "Índice de Actividad" es negativo indicando el incumplimiento de las restricciones temporales establecidas. Cuanto menor sea el índice mayores serán los retrasos.

El "Índice de Acceso a los Datos" de un agente indica si se están recibiendo los datos con una antigüedad adecuada para el procesamiento del agente. Para ello tiene que calcularse, para cada dato, la diferencia entre la máxima antigüedad permitida y

la antigüedad real en ese momento. El valor que establece el "Índice de Acceso a los Datos" viene dado por la menor de las diferencias en proporción calculadas. Un "Índice de Acceso a los Datos" negativo indica que existe al menos un dato que no es válido para el procesamiento del agente. Cuanto menor sea el índice mayor es la antigüedad del dato.

El "Índice de Comodidad" se obtiene tomando el menor de los dos índices ("Índice de Actividad" o "Índice de Acceso a los datos").

Cuanto mayor es el valor del "Índice de Comodidad" ($I_{Comodidad}$), mayor es la adecuación del agente al nodo donde está ejecutándose. El valor máximo para $I_{Comodidad}$ es 1, y se corresponde con el caso ideal donde el coste temporal de ejecución del agente y acceso a los datos es despreciable y, por tanto, igual a 0. El rango de valores entre 0 y 1 indican un $I_{Comodidad}$ adecuado, lo cual significa que el agente cumple con sus restricciones temporales (periodo de ejecución y máxima antigüedad admisible para los datos) en el nodo donde está ejecutándose. Cuanto más cerca de 1 se encuentra el valor del $I_{Comodidad}$, más holgura existe entre los valores reales de los tiempos y los máximos admisibles. Un $I_{Comodidad}$ con valor negativo significa que el agente no está cumpliendo con sus restricciones temporales en el nodo de ejecución. En este caso el agente podría decidir moverse a otro nodo. Cuanto menor sea el valor de $I_{Comodidad}$, mayores serán los retrasos en el cumplimiento del periodo de ejecución del agente o en la antigüedad de los datos recibidos.

Cada agente, en función del valor de su "Índice de Comodidad", podrá decidir si quedarse o no en el nodo donde reside (por ejemplo: si $I_{Comodidad} > 0 \rightarrow$ permanecer en el nodo, si $I_{Comodidad} < 0 \rightarrow$ moverse a otro nodo). En este caso es el propio agente quien decide qué hacer, sin embargo, el sistema también podría realizar tareas de reajuste de ejecución de agentes en cada nodo en función de los mismos "Índices de Comodidad". Podría existir un agente deliberativo que se encargara de esta función.

4.2.8. Conclusiones

La conclusión de esta descripción general es que el diseño de una arquitectura, adecuada a los sistemas híbridos para el control de robots móviles, tiene que contemplar la naturaleza distribuida de la información sensorial y de los comportamientos. La idea principal es que el sistema de comunicaciones constituye la base para la construcción de los esquemas motores y de percepción, los cuales deben extenderse al nivel de computación deliberativo. Por ello, en la sección siguiente se realiza una descripción detallada del sistema de comunicaciones para así, en el capítulo cinco, poder desarrollar los niveles reactivo y deliberativo basados en agentes móviles que emplean dicho sistema.

4.3. Sistema de Comunicaciones

La evolución de los modelos clásicos de comunicaciones distribuidas, caracterizada por los modelos de código móvil y sistemas de agentes, tiende a desacoplar la estructura clásica de cliente/servidor, donde el cliente inicia una conexión con un servidor para solicitar la ejecución de un determinado método del mismo y posteriormente obtener el resultado, por un modelo donde no hay una clara distinción entre clientes y servidores, sino que los componentes *software* asumen un papel u otro indistintamente, y además no es necesario el conocimiento a priori de ningún tipo de información sobre los componentes, como clásicamente lo eran los *stubs* y los esqueletos, sino que de una forma dinámica puede saberse qué ofrece cualquier componente *software* al resto, en este punto tiene especial importancia la reciente aparición de los modelos de programación con capacidades de reflexión, que permiten el análisis por parte de los componentes del código de los mismos. Todo ello conlleva la especificación de un lenguaje de comunicación común e independiente del lenguaje de programación que permita la comunicación entre las aplicaciones existentes e incluso la integración al sistema de futuras aplicaciones. Este lenguaje de comunicación debe basarse en el uso de mensajes caracterizados semánticamente, es decir, la transmisión tanto de los contenidos como del significado de los mismos. Además, dichos contenidos podrán ser tanto datos como código a ejecutar en otros nodos.

Los estándares actuales tienden a reunir todas estas características en sistemas donde el protocolo IIOP basado en la transmisión de mensajes sobre TCP/IP se ha consolidado como el protocolo estándar para asegurar la interoperatividad entre las aplicaciones.

En cuanto a las necesidades de un sistema distribuido para el control de robots móviles, la diferencia entre las restricciones temporales de las tareas reactivas y deliberativas tiene que reflejarse en el diseño del sistema de comunicaciones. Las comunicaciones de tiempo real estricto requieren una infraestructura caracterizada por la predicción en los procesos de computación y un estricto marco semántico de comunicación, características adecuadas para las actividades de comportamientos. Por otro lado, las comunicaciones de propósito general (sin restricciones de tiempo real estricto) requieren una infraestructura caracterizada por un alto y flexible nivel semántico de comunicación, característica necesaria en el cómputo deliberativo y la transmisión de agentes *software*. La información sensorial también debe estar disponible en los procesos de planificación para la construcción de un mapa o modelo interno del entorno. Dicha construcción requiere información temporal asociada a los datos u objetos distribuidos para así permitir algún tipo de fusión de datos espacial y temporal. Por lo tanto, la información temporal de los objetos distribuidos tiene que estar disponible en todas las infraestructuras de comunicación a pesar de las diferencias existentes en predicción y granularidad temporal.

Ninguno de los estándares actuales para comunicaciones en sistemas distribuidos ofrece una interfaz para la caracterización temporal de la información. Por ello, es necesario la especificación de un sistema de comunicaciones que permita el acceso a los datos proporcionando la antigüedad de los mismos de una forma transparente al usuario. Es decir, independientemente desde donde se acceda al dato, distintos niveles o distintos nodos de la arquitectura, el sistema de comunicaciones tendrá que ofrecer, mediante una interfaz definida, el tiempo transcurrido desde que se generó el dato, calculando para ello tanto los retardos de transmisión entre los nodos de la arquitectura como la permanencia del dato en dichos nodos. Al final, las aplicaciones que utilicen la información podrán conocer en cualquier instante la antigüedad de la misma.

El diseño del sistema de comunicaciones no tiene por qué ser incompatible con los estándares actuales sino que puede estar basado en los mismos ofreciendo nuevas funcionalidades orientadas al control temporal de la información. La interoperatividad entre distintas implementaciones puede asegurarse mediante el uso de protocolo IIOP en la transmisión de los mensajes.

El sistema de comunicaciones desarrollado recibe el nombre de sistema SC [Posadas, 1997] y está basado en el uso de una pizarra [Nii, 1989] de objetos distribuida que permite la interoperatividad entre aplicaciones independientemente de su ubicación física y lenguaje de implementación. Las aplicaciones se comunican escribiendo y leyendo sobre los objetos de la pizarra empleando una interfaz común denominada FSA [Pérez, 2001 y 2002]. Cada objeto de la pizarra distribuida puede verse como un canal lógico de transmisión de información (por ejemplo, el objeto correspondiente al valor de un sensor de infrarrojos o el objeto correspondiente al valor de la velocidad aplicada a un motor).

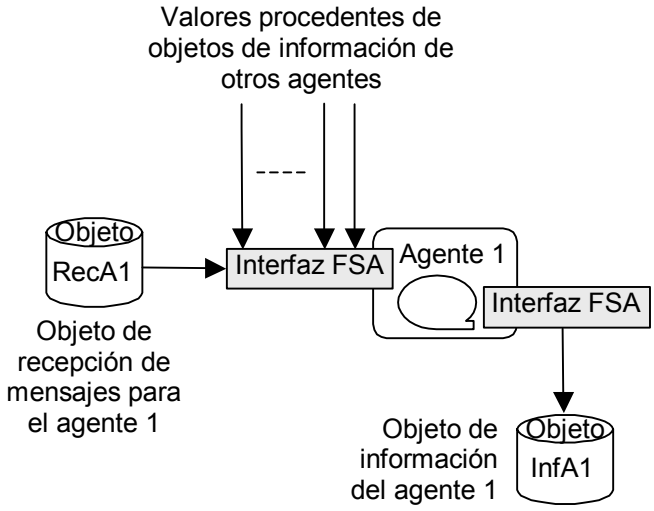


Figura 40: *Sistema de Comunicaciones SC. Objetos de la pizarra distribuida asociados a cada agente*

Existen dos formas de interacción o comunicación entre componentes *software* o agentes a través de la pizarra. Ambas pueden coexistir requiriéndose la existencia por cada componente de al menos dos objetos asociados (fig.40):

1. Un objeto para la información del componente. La información del componente se escribe en el objeto y quien desea acceder a ella sólo tiene que leer el objeto.

2. Otro objeto para enviar mensajes o datos al componente. Cuando alguien desea enviar un mensaje o datos a un componente escribe en el objeto, el componente recibe el mensaje o dato mediante la lectura del objeto.

Para las lecturas de los objetos el sistema ofrece un medio de notificación automática basado en eventos.

En líneas generales, el sistema SC permite la recepción automática de los datos, los caracteriza temporalmente proporcionando su antigüedad y además ofrece una interfaz para la transmisión y localización de los agentes *software*.

4.3.1. Interfaz FSA

Los procesos o agentes interactúan con el sistema SC mediante el uso de una interfaz o estructura de programación denominada modelo marco-sensor-adaptador o FSA (*Frame-Sensor-Adapter*).

El modelo FSA [Pérez, 2002] ha sido desarrollado para definir una manera común de acceder a cualquier recurso de comunicaciones. Está basado en la definición de tres interfaces (fig.41): *IFrame*, *ISensor* e *IAdapter*.

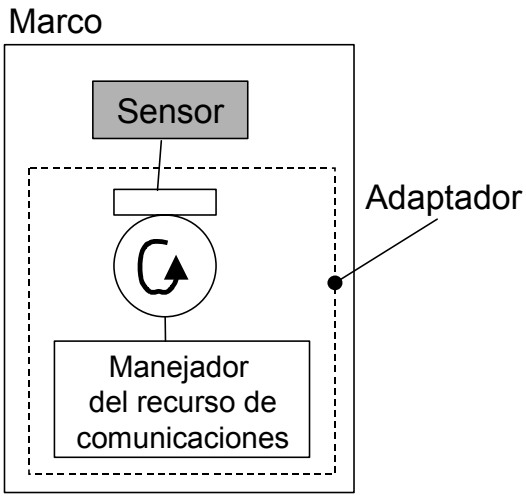


Figura 41: *Sistema de Comunicaciones SC. Interfaz de acceso común: modelo FSA Marco-Sensor-Adaptador*

- Interfaz *IFrame*: define un objeto marco para la instanciación de los objetos adaptadores y sensores. Este objeto relaciona los adaptadores con los sensores y permite una coordinación más sencilla desde la aplicación o el agente que los utilice.

- Interfaz *ISensor*: define un objeto sensor que permite la obtención automática de datos procedentes de los recursos de comunicaciones. Para ello los objetos sensor tienen que utilizar los correspondientes objetos adaptadores. Un objeto sensor puede verse como un receptor y contenedor de datos. El sensor abstrae el dato del sistema de comunicaciones empleado.

- Interfaz *IAdapter*: define un objeto adaptador que permite el acceso local a un determinado recurso de comunicaciones mediante el uso de un conjunto de funciones específicas. Para cada recurso de comunicaciones tiene que implementarse un objeto adaptador con esta interfaz. De esta forma, los procesos conectarán y accederán con una misma interfaz a cualquier recurso de comunicaciones utilizando el adaptador correspondiente. Las aplicaciones podrán recibir de forma automática los valores de los datos mediante el registro de los objetos sensor en los objetos adaptador. Por ejemplo, si una aplicación está interesada en los valores de los sensores de infrarrojos (los cuales podrían ser enviados a través de un bus de campo CAN), tendrá que implementar un objeto sensor y registrarlo en el adaptador correspondiente (adaptador del bus CAN). De este modo, cuando el adaptador reciba nuevos valores de los sensores de infrarrojos, el adaptador actualizará el objeto sensor (que ha sido implementado para contener dichos valores) y avisará a la aplicación de ello.

4.3.1.1. Especificación IDL

Para la especificación formal de la interfaz FSA se ha utilizado el lenguaje estándar de definición de interfaces o IDL. En primer lugar, se especifican las estructuras de datos (fig.42) empleadas en la definición de las interfaces. A continuación, se presentan las especificaciones IDL de las interfaces *IAdapter* (fig.43) e *Isensor* (fig.44). En secciones posteriores, se describe la metodología empleada para el uso de las funciones definidas.

```
/*Tipos de datos*/

typedef string XML; /*secuencia de caracteres*/
typedef sequence<octet> buffer; /*secuencia de bytes*/
typedef unsigned long numBytes; /*cantidad o número de bytes*/
typedef string dirIP; /*dirección IP de un nodo*/
typedef long long time_mcs; /*tiempo en microsegundos*/
typedef long long count; /*cuenta local del reloj de un nodo*/
```

```
/*Definición de los objetos de la pizarra del sistema SC*/

struct objeto_SC
 {
   XML       nombre; /*nombre lógico del objeto*/
   buffer    valor; /*secuencia de bytes correspondientes al
                      valor o mensaje del objeto*/
   numBytes  tam; /*Tamaño del valor o mensaje*/
   dirIP     localizacion; /*procedencia del valor o mensaje*/
   time_mcs  delay; /*retardo-antigüedad del valor o mensaje*/
   count     ticks; /*cuenta local correspondiente a la última
                      actualización de la antigüedad*/
};

typedef sequence<objeto_SC> lista_objetos; /*pizarra*/
typedef sequence<XML> lista_nombres_objetos;

/*Excepciones*/

exception OperacionImposible {};
exception ObjetoDesconocido {};
exception ArgumentoInvalido {};
```

Figura 42: *Modelo FSA. Especificación IDL de las estructuras de datos generales*

```
/*Interfaces*/

interface IAdapter
 {
   void Start() raises (OperacionImposible);
   /*Start: Función que inicializa el adaptador del
     recurso de comunicaciones*/

   void Stop() raises (OperacionImposible);
   /*Stop: Función que detiene el adaptador del
     recurso de comunicaciones*/

   void RegisterSensor(in ISensor sensor, in XML nombreObjeto)
                      raises (ObjetoDesconocido,
                              ArgumentoInvalido,
                              OperacionImposible);
   /*RegisterSensor: Función que permite el registro de
     un sensor en el adaptador de comunicaciones asociándolo
     a un objeto determinado. De esta forma, el sensor
     registrado recibirá los valores o mensajes enviados
     al objeto*/

   void Write(in XML nombreObjeto,
              in buffer valor, in numBytes tam,
              in time_mcs delay, in count ticks)
             raises (ObjetoDesconocido,
                     ArgumentoInvalido,
                     OperacionImposible);
```

```
   /*Write: Función que permite la escritura de un
     mensaje o valor sobre un objeto determinado. Al
     realizar la escritura se proporciona la antigüedad del
     mensaje o valor en el instante correspondiente a la
     cuenta local "ticks"*/

   numBytes Read(in XML nombreObjeto,
                 out buffer valor,
                 out time_mcs delay, out count ticks)
                raises (ObjetoDesconocido,
                        ArgumentoInvalido,
                        OperacionImposible);
   /*Read: Función que permite la lectura del valor
     de un objeto determinado. Con la lectura se obtiene
     la antigüedad del valor en el instante correspondiente
     a la cuenta local "ticks"*/

   count TicksNow()raises (OperacionImposible);
   /*TicksNow: Función que retorna el valor de la cuenta
    local en el instante actual*/

   time TimeFromTicks(in count ticks, out count newTicks)
                     raises (ArgumentoInvalido,
                             OperacionImposible);
   /*TimeFromTicks: Función que retorna el tiempo
    transcurrido desde la cuenta local "ticks" hasta el
    instante actual definido por la cuenta "newTicks"*/

 };
```

Figura 43: *Modelo FSA. Especificación IDL de la interfaz IAdapter*

```
interface ISensor
 {
   time_mcs GetTimes(in time_mcs delay, in count ticks,
                     in IAdapter adapter)
                    raises (ArgumentoInvalido,
                            OperacionImposible);
   /*GetTimes: Función que retorna la antigüedad de un valor
     o mensaje hasta el instante actual de la invocación.
     Con los parámetros se indica el retardo acumulado del
     valor hasta el instante correspondiente a la cuenta
     local "ticks"*/

   /*Un mismo sensor puede estar registrado en varios
     adaptadores (correspondientes a diferentes recursos de
     comunicaciones).Los siguientes métodos son ejecutados por
     los adaptadores donde se encuentre registrado el sensor*/

   void OnStart(in IAdapter adapter)
               raises (ArgumentoInvalido,
                       OperacionImposible);
```

```
   /*OnStart: Función ejecutada cuando se inicializa un
     adaptador en el que está registrado el sensor. Con el
     parámetro se indica el adaptador desde donde se ejecuta la
     función*/

   void OnStop(in IAdapter adapter)
              raises (ArgumentoInvalido,
                      OperacionImposible);
   /*OnStop: Función ejecutada cuando finaliza un adaptador en
     el que está registrado el sensor. Con el parámetro se
     indica el adaptador desde donde se ejecuta la función*/

   void OnMessage(in XML nombreObjeto,
                  in buffer valor, in numBytes tam,
                  in time_mcs delay, in count ticks,
                  in IAdapter adapter)
                raises (ObjetoDesconocido,
                        ArgumentoInvalido,
                        OperacionImposible);
   /*OnMessage: Función ejecutada cuando un adaptador en el que
     está registrado el sensor recibe un nuevo valor (buffer de
     bytes) para un objeto al que está conectado el sensor. Con
     los parámetros se indican el nombre del objeto, el buffer
     de bytes, su tamaño, el retardo del valor en el instante
     correspondiente a la cuenta ticks,
     y el adaptador desde donde se invoca la función*/

 };
```

Figura 44: *Modelo FSA. Especificación IDL de la interfaz ISensor*

4.3.1.1.1 Estructura IDL de los agentes

La estructura de datos de todos los agentes se basa en la especificación del sistema de comunicaciones.

```
interface IAgente
 {
   string        nombre; //Nombre del agente
   dirIP         localizacion; //Ubicación del agente
   buffer        codigo; //Código serializado del agente
   numBytes      tam_codigo; //Tamaño del código serializado
   unsigned long periodoEjecucion; //Se corresponde con el
                                      máximo tiempo de ejecución
                                      del código

   unsigned long num_objetos_recepcion_inf;
   lista_nombres_objetos objetos_recepcion_inf;
   /*Número y nombre de los objetos de la pizarra a los que
    el agente deberá acceder para obtener los datos que
    necesita en su ejecución*/
```

```
  sequence<unsigned long> lista_Maxima_Antiguedad;
  /*Máxima antigüedad admisible para cada uno de los tipos
   de datos*/

  unsigned long num_objetos_envio_inf;
  lista_nombres_objetos objetos_envio_inf;
  /*Número y nombre de los objetos de la pizarra a los que
   el agente enviará la información que genere*/

  long indice_comodidad;
};
```

Figura 45: *Estructura IDL de los agentes*

Cada agente deberá disponer de los nombres de los objetos de la pizarra distribuida con los que debe conectar para recibir o enviar información. Además, de acuerdo a las restricciones temporales necesarias, cada agente deberá estar caracterizado por un periodo de ejecución de su código y unos tiempos máximos de antigüedad de los datos que debe procesar, características necesarias para el cálculo del "Índice de Comodidad". Por otro lado, el propio agente deberá disponer de su código para poder moverse entre los nodos de la arquitectura.

En la figura 45 se presenta, mediante el lenguaje IDL, la estructura general de los agentes del sistema.

4.3.1.2. Espacio de Nombres Lógicos

La base de la comunicación e interacción entre los distintos componentes de la arquitectura propuesta consiste en el acceso a los objetos definidos en la pizarra distribuida para su posterior escritura y lectura. Para facilitar la interoperatividad entre diferentes plataformas y diferentes lenguajes de programación, se requiere la especificación de un espacio de nombres que permita referenciar dichos objetos de forma independiente a la implementación. Para modelar y configurar dicho espacio de nombres, se utiliza el lenguaje estándar XML que permite la definición de un fichero de texto donde se especifican, mediante una estructura de árbol, todos los nombres con los que pueden referenciarse cada uno de los objetos de la pizarra. A partir de este fichero, el sistema de comunicaciones SC creará dichos objetos y permitirá que las aplicaciones accedan a ellos a través de sus nombres lógicos.

En el fichero XML se distinguen dos partes donde se realizan diferentes definiciones:

- Definición de nombres primitivos. Primera parte del fichero XML donde se define cada objeto de la pizarra mediante un nombre único en el sistema que lo diferencia del resto de objetos. A estos nombres se les conoce como nombres primitivos y serán los utilizados internamente por el sistema SC para acceder a los objetos de forma unívoca. Junto a la definición del nombre primitivo de un objeto también se especifican los nodos (mediante

direcciones IP) donde el sistema SC creará dicho objeto y por tanto replicará sus valores. La configuración estándar consiste en tener cada objeto en todos los nodos del sistema. Sin embargo, dependiendo de dónde se requiera la información, pueden realizarse otras configuraciones con el objetivo de reducir el tráfico de red ocasionado por las replicaciones.

- Definición de nombres lógicos. Segunda parte del fichero XML donde se define un espacio de nombres con una estructura de árbol que permite referenciar a los objetos de la pizarra con nombres lógicos (especificados en XML de forma jerárquica como una ruta en un árbol) en lugar de emplear sus nombres primitivos. Las aplicaciones utilizarán los nombres lógicos para acceder a los objetos de la pizarra. El uso de este espacio de nombres facilita la gestión de los objetos permitiendo, por ejemplo, referenciar un mismo objeto (nombre primitivo) con diferentes nombres lógicos, y referenciar a un conjunto de objetos mediante un solo nombre lógico o mediante el uso de caracteres comodín como *. Además, dinámicamente puede modificarse o ampliarse el espacio de nombres con nuevas referencias sin modificar los nombres primitivos, es decir, sin necesidad de reconfigurar el sistema SC.

La utilización de este espacio de nombres lógicos por parte de las aplicaciones implica la necesidad, en la implementación de la interfaz FSA, de un módulo o subsistema que en base a las definiciones del fichero XML realice la conversión de los nombres lógicos en los nombres primitivos utilizados por el sistema SC y viceversa.

A continuación se presenta un ejemplo de fichero XML (fig.46) donde se realiza una sencilla definición de nombres primitivos y el acceso a los mismos mediante un árbol de nombres lógicos.

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

<!-- Ejemplo de especificación de un documento XML -->

<Sistema_SC name="DocumentoXML" version="0.0">

<!-- Definición de la tabla de datos primitivos. Cada dato
primitivo tiene asociada la información referente a los nodos
TCP/IP donde se replicará -->

<PrimitiveData>
    <Data name="Dato_00" description="Dato sin preproceso
asociado al sensor de ultrasonidos" size="12"
format="%2d%3d%s">
          <AllHosts/>
    </Data>
    <Data name="Dato_01" description="Dato preprocesado asociado
al sensor de ultrasonidos" size="12" format="%Preprocesado">
          <AllHosts/>
    </Data>
```

```
    <Data name="Dato_02" description="Dato asociado al sensor de
infrarrojos A" size="12" format="%Preprocesado">
          <AllHosts/>
    </Data>
    <Data name="Dato_03" description=" Dato asociado al sensor
de infrarrojos B" size="" format="%Preprocesado">
          <Host name="" IP="158.42.0.0"/>
          <Host name="" IP="158.42.0.1"/>
    </Data>

</PrimitiveData>

<!-- Definición de los enlaces a los datos primitivos. Se
establece una estructura de árbol para el espacio de nombres.
Cada nodo de la estructura lógica puede tener asociado un
enlace a un dato primitivo -->

<NameSpaceTree>
    <Node name="Robot_A" link="">
          <Node name="Nivel_Reactivo" link="">
                <Node name="SensorUltrasonidos" link="Dato_01">
                      <Node name="SinPreproceso" link="Dato_00">
                      </Node>
                </Node>
                <Node name="SensoresInfrarrojos">
                      <Node name="InfrarrojosA" link="Dato_02">
                      </Node>
                      <Node name="InfrarrojosB" link="Dato_03">
                      </Node>
                </Node>
          </Node>
          <Node name="DatosPreprocesados" link="">
                <Node name="Infrarrojos" link="">
                      <Node name="A" link="Dato_02">
                      </Node>
                      <Node name="B" link="Dato_03">
                      </Node>
                </Node>
                <Node name="Ultrasonidos" link="Dato_01">
                </Node>

          </Node>

    </Node>

</NameSpaceTree>

</Sistema_SC>
```

Figura 46: *Especificación XML del espacio de nombres para referenciar los objetos del sistema de comunicaciones SC*

En la figura 47 puede observarse la estructura de árbol correspondiente al espacio de nombres del ejemplo planteado en la figura 46.

Las operaciones de "escribir" y "leer" sobre un objeto de la pizarra tienen que realizarse sobre un nombre lógico (nodo del árbol del espacio de nombres) que tenga asociado un enlace (nombre primitivo).

En el ejemplo, la orden de escritura o lectura sobre "/Robot_A/Nivel_Reactivo/ SensorUltrasonidos", se traduciría en una escritura o lectura sobre el objeto denotado con el nombre primitivo "Dato_01". Por el contrario, la orden de escritura o lectura sobre "/Robot_A/Nivel_Reactivo/SensorUltrasonidos/SinPreproceso", se traduciría en una escritura o lectura sobre el objeto denotado con el nombre primitivo "Dato_00".

Se puede registrar un mismo objeto sensor (interfaz FSA) con varios objetos de la pizarra mediante una única referencia a un nodo del árbol o nombre lógico y el comodín *.

En el ejemplo, al registrar un objeto sensor con el nodo del árbol "/Robot_A/DatosPreprocesados/*", se registraría el sensor con todos los objetos de la pizarra cuyos nombres primitivos queden bajo "DatosPreprocesados", en este caso los objetos correspondientes a los nombres primitivos "Dato_01", "Dato_02" y "Dato_03".

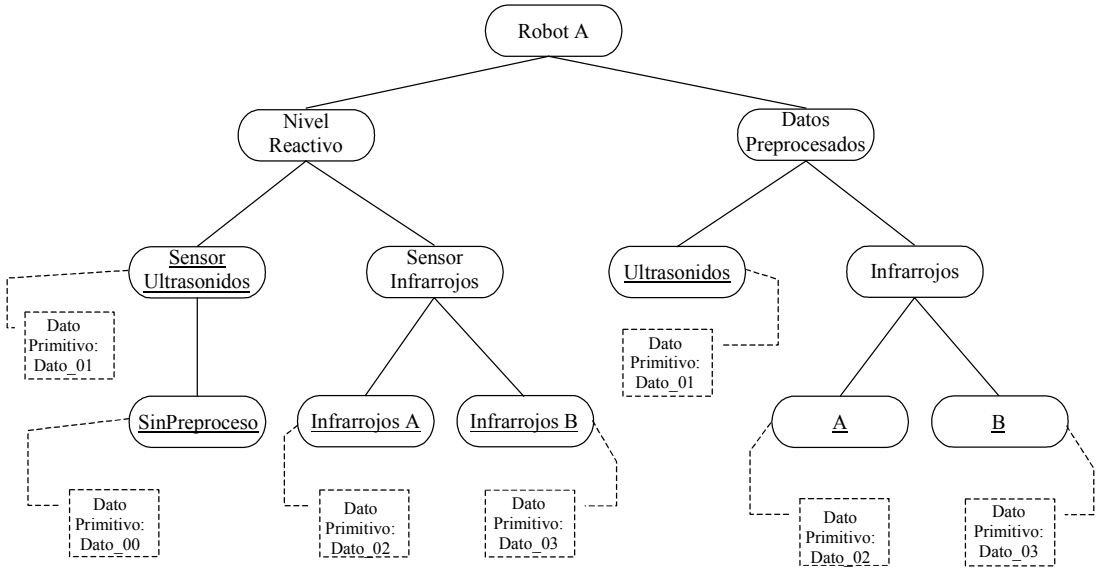


Figura 47: *Árbol del espacio de nombres*

4.3.1.3. Metodología

La utilización del modelo de programación o interfaz FSA por parte de una aplicación para acceder a los recursos de comunicaciones consta de varios pasos (fig.48).

Primero tiene que crearse un objeto marco. Segundo, se instancian en dicho marco los objetos adaptador necesarios y los sensores implementados. Tercero, se

registran los sensores en los adaptadores (función "RegisterSensor" del adaptador) para configurar la recepción automática de los datos. Y cuarto, finalmente se ejecuta la función "Start" de los adaptadores (lo cual implica que los adaptadores ejecutan automáticamente la función "OnStart" de los sensores registrados).

A partir de este momento, cuando un adaptador reciba datos para un sensor registrado, ejecutará automáticamente la función "OnMessage" de dicho sensor. La aplicación recibe una copia de los nuevos datos a través de los parámetros de la función "OnMessage".

Para finalizar las comunicaciones tiene que ejecutarse la función "Stop" de los adaptadores (lo cual implica que los adaptadores ejecutan automáticamente la función "OnStop" de los sensores registrados).

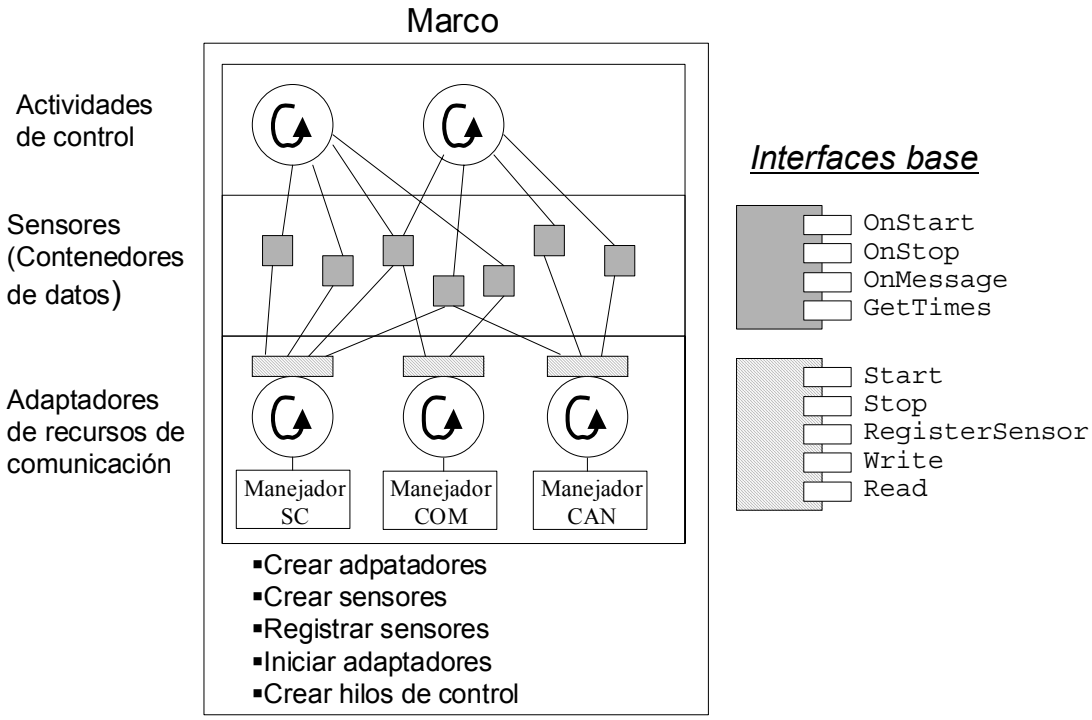


Figura 48: *Metodología de programación con el modelo FSA*

El acceso al sistema SC, mediante la interfaz FSA y metodología descritas, se basa en el desarrollo del correspondiente objeto adaptador (adaptador SC) que deberá estar disponible para todos los procesos. Si existieran diferentes sistemas de agentes basados en diferentes lenguajes de implementación, habría que desarrollar un adaptador SC para cada uno de los sistemas o lenguajes de programación.

Las comunicaciones necesarias entre el adaptador SC y el sistema SC dependen de la implementación realizada y no de la interfaz FSA, siendo transparentes a las aplicaciones o agentes.

Los pasos por parte de los agentes para acceder al sistema SC son los siguientes:

1. Crear objetos sensor según la interfaz *ISensor* mediante la implementación de las funciones: "OnStart", "OnMessage" y "OnStop". Este es el único requerimiento o tarea que tiene que asumirse desde el agente para poder acceder al sistema SC y comunicarse con el resto de agentes. Dependiendo del lenguaje de implementación del agente, posteriormente utilizará el adaptador SC correspondiente para registrar sus sensores.

2. Registrar los objetos sensor mediante la función "RegisterSensor" del objeto adaptador SC. Con esta función los agentes asocian un objeto sensor (parámetro "sensor") con un objeto (parámetro nombreObjeto) de la pizarra distribuida. El parámetro "nombreObjeto" es el nombre lógico del objeto de la pizarra (objeto SC).

   Una vez hecho esto, cuando el valor del objeto SC cambia en la pizarra distribuida, el objeto adaptador SC ejecuta la función "OnMessage" del objeto sensor. De esta forma, el agente que registró el sensor recibe automáticamente desde el sistema SC el nuevo valor a través del parámetro "valor" de la función "OnMessage".

3. Para enviar valores o mensajes a través del sistema SC, el agente únicamente tiene que ejecutar la función local "Write" del adaptador SC. Con esta función se cambia el valor (parámetro "valor") de un objeto (parámetro "nombreObjeto") de la pizarra distribuida.

El sistema SC también tiene que proporcionar acceso al bus de tiempo real estricto. Dicho acceso debe proporcionarse a través de un componente software que actúe de pasarela realizando las comunicaciones necesarias entre el sistema SC y el bus de tiempo real estricto.

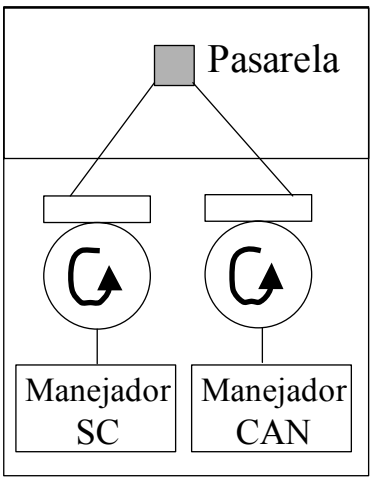


Figura 49: *Modelo de pasarela basado en FSA. Permite la conexión de distintos recursos de comunicaciones*

El desarrollo de pasarelas que conecten distintos recursos de comunicaciones con el sistema SC es sencillo mediante el uso del modelo FSA (fig.49). Al disponer el bus de tiempo real estricto de su propio adaptador, el objeto o componente pasarela únicamente tendrá que enlazar ambos adaptadores (adaptador SC y adaptador del bus de tiempo real estricto).

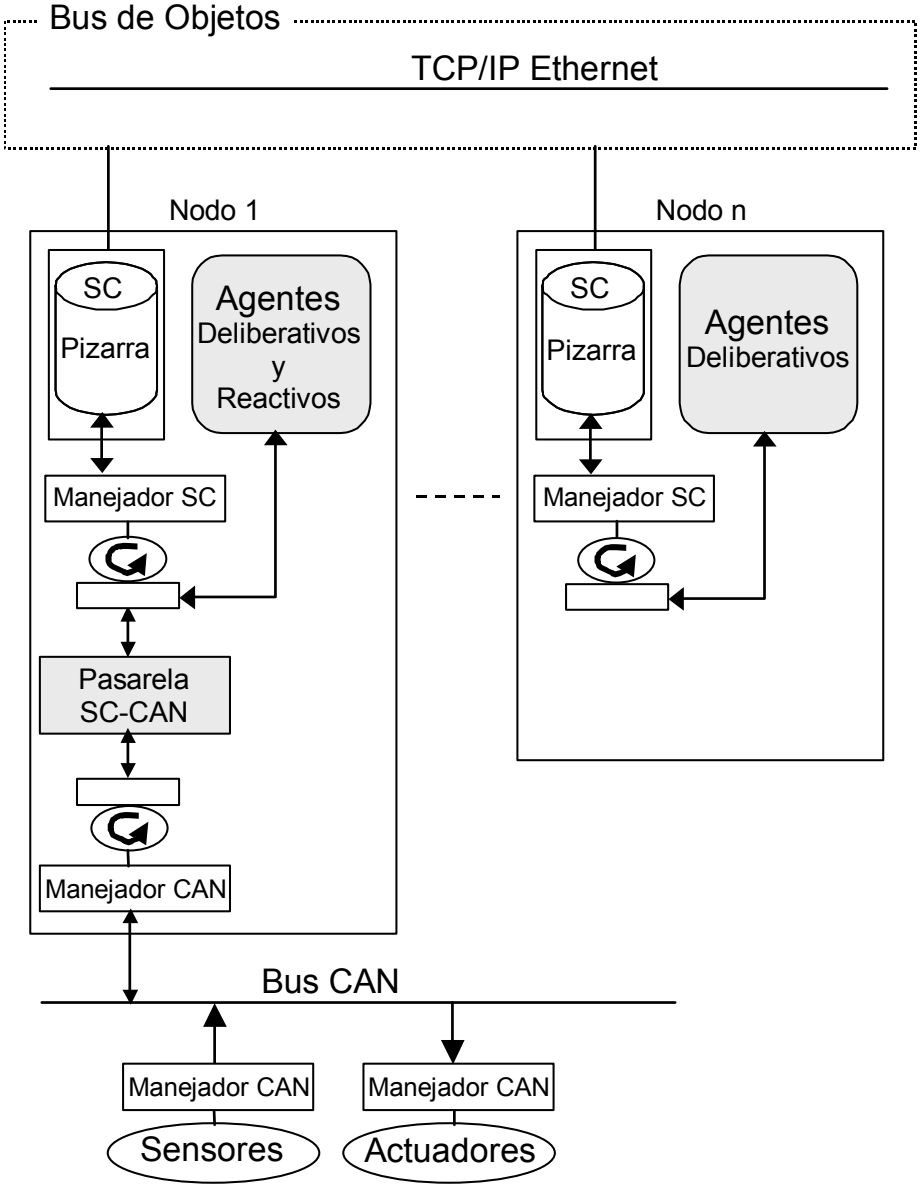


Figura 50: *Conexión de los componentes de la arquitectura con el Sistema de Comunicaciones SC a través del modelo FSA y uso de pasarelas*

Desde este punto de vista, por ejemplo, podría desarrollarse un objeto adaptador para el bus CAN (adaptador CAN) y un objeto pasarela entre los adaptadores SC y

CAN (pasarela SC-CAN). Se ha elegido el bus CAN por sus características de tiempo real estricto [Bosch, 1991; Tindell, 1994a y b].

La pizarra distribuida del sistema SC debe extenderse a los datos del bus de tiempo real estricto, en el caso del ejemplo, la red CAN. Para ello, la pasarela SC-CAN debe realizar las traducciones necesarias entre el protocolo CAN y los datos SC (fig.50). Esta pasarela tiene que soportar el formato de los mensajes CAN y realizar un proceso de mapeado estableciendo una correspondencia bidireccional entre los objetos SC y los identificadores CAN. Este mapeado permitirá a cualquier proceso, desde cualquier nodo de la red, el acceso a la información procedente del bus CAN mediante el sistema SC y el esquema de notificación automática definido (basado en el modelo FSA).

La implementación de la pasarela que permite tanto la recepción como el envío de datos a través del bus de tiempo real estricto mediante el acceso a los objetos de la pizarra distribuida es independiente de la interfaz FSA y transparente a los agentes. Éstos sólo leen o escriben en los objetos correspondientes de la pizarra para recibir o enviar datos o mensajes a través del bus de tiempo real estricto.

En la figura 50 se presenta un esquema general de conexión de los componentes de la arquitectura (agentes reactivos y deliberativos) tanto al bus de tiempo real estricto como al bus de tiempo real no estricto mediante el uso del sistema SC basado en dos adaptadores y una pasarela entre ambos buses.

Los agentes reactivos y deliberativos situados en el nodo sin infraestructura de bus de tiempo real estricto, también podrán acceder a los datos del mismo mediante el acceso a los objetos correspondientes de la pizarra. El sistema SC se encarga, a través del bus de tiempo real no estricto, de hacer las réplicas en todos los nodos de la información procedente del bus de tiempo real estricto.

4.3.2. Control Temporal

El sistema SC incorpora un sistema de estimación de tiempos que permite a cualquier aplicación, independientemente de su ubicación, conocer en tiempo real la antigüedad de toda la información que recibe.

Este sistema de estimación está basado en el uso de los relojes locales de los nodos para contabilizar el tiempo que cada valor permanece en un nodo, y en la acumulación de los retardos de transmisión cuando los valores se mueven entre los nodos [Poza, 2001; Posadas, 2002].

Para ello, todos los valores de los objetos de la pizarra del sistema SC presentan un campo con información temporal acerca de su antigüedad. En concreto, el campo dispone de la siguiente información:

- Tiempo en microsegundos transcurrido desde que se generó el valor.

- Tiempo local en el que se actualizó por última vez el tiempo transcurrido.

La información disponible indica la antigüedad del valor hasta una determinada cuenta del reloj correspondiente al nodo donde se encuentra dicho valor. A partir de esta información, el sistema SC siempre puede proporcionar de forma actualizada la antigüedad del valor con sólo sumar el tiempo transcurrido desde la cuenta de reloj local referenciada.

Cuando un valor o mensaje de un objeto SC abandona un nodo a través del sistema SC, su campo temporal se actualiza sumándole el tiempo que dicho valor ha permanecido en el nodo. Este campo, además, acumula todos los retardos de transmisión al moverse el valor de un nodo a otro.

Para ello, los valores sólo deben viajar de una máquina a otra a través del sistema SC, de esta forma el SC estima el tiempo de transmisión por la red y actualiza los tiempos del valor en la máquina receptora. Este requerimiento no es una limitación, sino que forma parte de la transparencia proporcionada por el sistema SC en la transmisión de la información a través de los nodos del sistema, incorporando, además, un deseado control temporal en el acceso a la información que permitirá la ejecución de tareas bajo restricciones de tiempo real estricto y no estricto.

La forma en la que un proceso obtiene la antigüedad del valor de un objeto de la pizarra es a través de la interfaz FSA. Cualquier proceso puede obtener en cualquier instante la antigüedad de un valor mediante la función "GetTimes" de la interfaz ISensor. La función "GetTimes" calcula automáticamente la antigüedad del valor del objeto SC teniendo en cuenta el tiempo transcurrido desde que llegó al nodo donde se encuentra.

En la figura 51 puede observarse un ejemplo de cómo el sistema de estimación de tiempos actualiza el campo temporal de los valores y cómo calcula la antigüedad de los mismos mediante la información disponible.

Cuando el valor proviene del bus de tiempo real estricto, éste tiene predefinido un retardo de transmisión máximo resultado del estudio de la planificabilidad del bus. El sistema SC recibe este valor y asigna como antigüedad del mismo ($t_{retardo}$) el tiempo correspondiente a su retardo máximo de transmisión. El sistema SC también incorpora en el campo de tiempo la cuenta del reloj (*tick count*) local correspondiente al instante en que llegó el valor ($t_{llegada}$). Si el valor recibido abandonara el nodo, antes de ello el sistema SC actualizaría la antigüedad del valor ($t_{retardo}$) sumándole el tiempo que hubiera permanecido en dicho nodo.

Cuando el sistema SC recibe un valor que proviene de otro nodo, debe actualizar la antigüedad del mismo con el tiempo estimado en la transmisión del valor entre los

nodos y asignar igualmente la cuenta de reloj del nuevo nodo correspondiente al instante de la recepción.

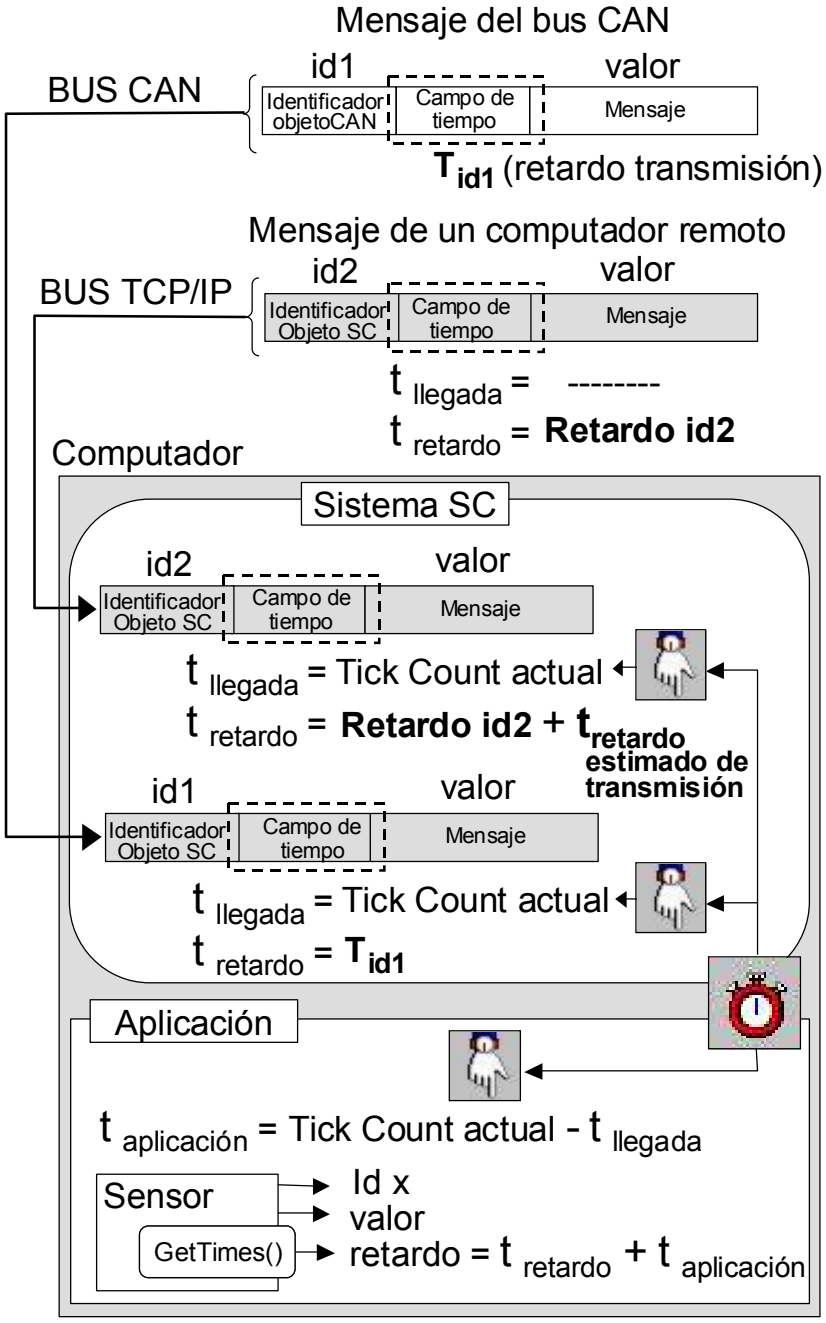


Figura 51: *Caracterización temporal de la información: cálculo de la antigüedad de los valores de los objetos del sistema SC*

4.3.3. Comparación

El diseño del sistema SC presentado, que constituye el sistema de comunicaciones de la arquitectura híbrida para el control de robots móviles propuesta en esta tesis, reúne las características necesarias para la especificación de los niveles reactivo y deliberativo de dicha arquitectura. Los componentes de estos niveles son agentes *software* cuya interacción se produce a través de la escritura y lectura de los objetos de la pizarra distribuida del sistema de comunicaciones SC.

El objetivo fundamental del sistema de comunicaciones es proporcionar acceso a toda la información del sistema independientemente del nivel desde donde se acceda. Además, también tiene que proporcionarse la antigüedad de dicha información para permitir a los agentes tomar decisiones sobre su validez y utilización.

El sistema de comunicaciones propuesto incorpora características no contempladas explícitamente en los estándares actuales de comunicaciones distribuidas como CORBA. El punto de vista de tratamiento del tiempo real ofrecido en los estándares difiere de las necesidades de una arquitectura donde el entorno de aplicación es dinámico y no pueden preestablecerse los requerimientos de procesamiento. En estos casos, los sistemas tienen que adaptarse dinámicamente a sus objetivos para lo cual, la posibilidad de incorporación de nuevos componentes, la movilidad de los mismos, y la obtención de la antigüedad de los datos, son características deseables que tiene que ofrecer el sistema de comunicaciones.

El sistema SC (fig.52) puede situarse al mismo nivel del sistema CORBA y aunque puede utilizarse sin necesidad de CORBA no tiene por qué ser incompatible. Podría desarrollarse un núcleo CORBA que implementara su infraestructura de comunicaciones mediante el sistema SC. De esta forma el sistema SC podría proporcionar a CORBA la caracterización temporal de los datos. Por otro lado, en lo que se refiere a interoperatividad entre distintas plataformas, el sistema SC puede encapsular los valores de los objetos SC en mensajes compatibles con el estándar IIOP y así transmitirlos entre las diferentes plataformas.

El sistema SC, mediante la interfaz común FSA, permite la conexión de aplicaciones o agentes desarrollados con diferentes lenguajes de programación (C++, java, .NET, etc.). Otra de las características necesarias en un sistema de comunicaciones para una arquitectura de control de robots móviles es el acceso a diferentes tipos de recursos de comunicación como son los buses de tiempo real estricto (bus CAN) y los buses de tiempo real no estricto (bus Ethernet). A diferencia de los estándares actuales, el sistema SC sí ofrece este requerimiento mediante el uso de las denominadas pasarelas entre los distintos recursos o buses.

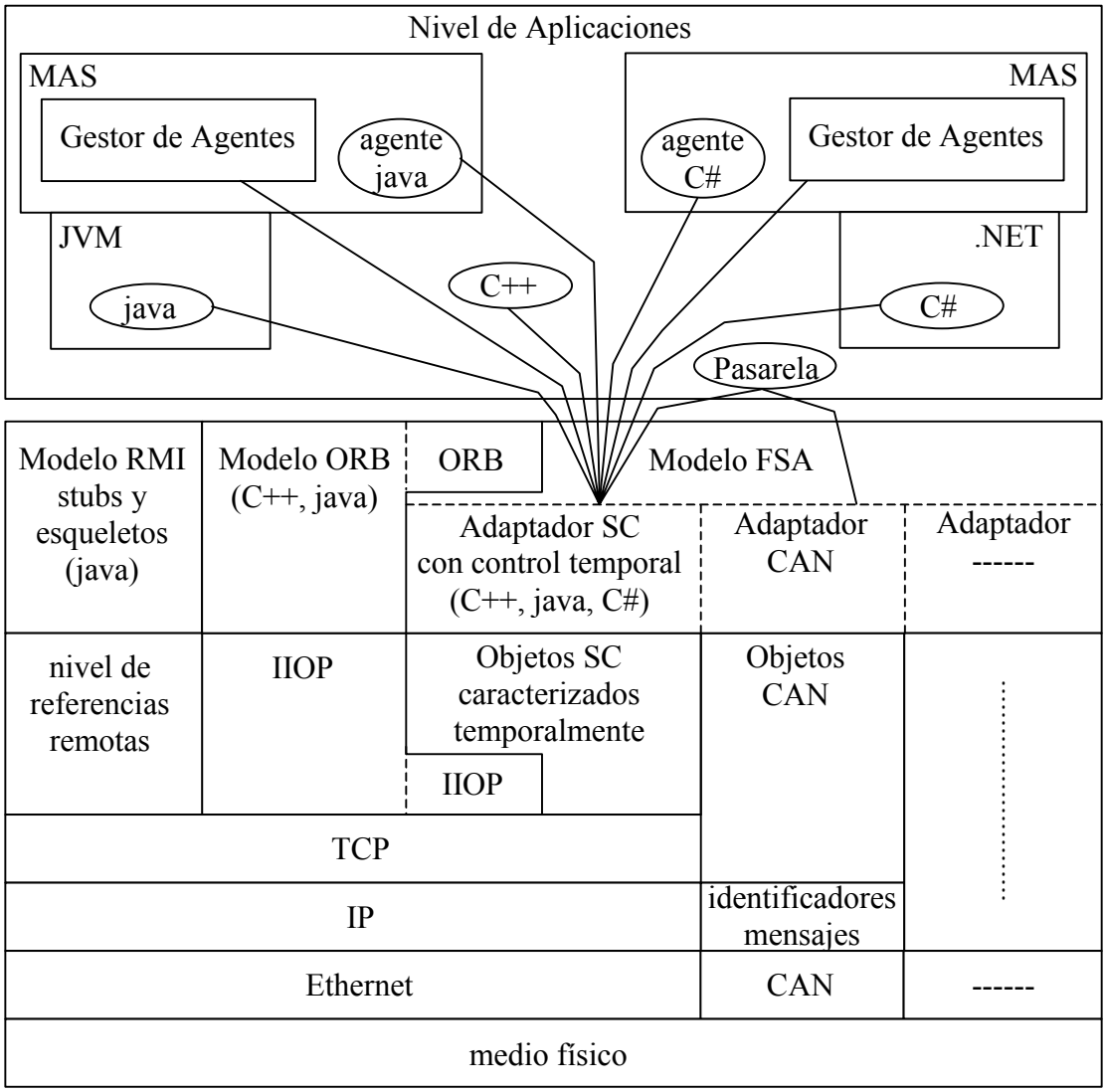


Figura 52: *Nivel del Sistema de Comunicaciones SC*

	Invocación Remota	Delegación de Código	Acceso Multi-lenguaje	Información Temporal	Múltiples Infraestructuras Comunicación
CORBA	✓	✗	✓	✗	✗
RMI	✓	✓	✗	✗	✗
SC	✓	✓	✓	✓	✓

Tabla 5: *Comparativa CORBA-RMI-SC*

En la tabla 5 se presenta un resumen comparando las características que proporciona el sistema SC con las características ofrecidas por el modelo CORBA y el modelo RMI.

4.4. Conclusiones

En este capítulo se ha descrito de forma general la arquitectura híbrida para el control de robots móviles basada en la delegación de código que se propone en esta tesis. La arquitectura, denominada SC-Agent, es distribuida y formada por un nivel reactivo y un nivel deliberativo cuyos componentes *software* son agentes móviles que interaccionan a través de un sistema de comunicaciones conocido como sistema SC. Este sistema de comunicaciones proporciona una interfaz entre los niveles reactivo y deliberativo y constituye la base sobre la que se sustenta el diseño de la arquitectura. El sistema SC proporciona el acceso transparente a toda la información disponible en la arquitectura caracterizándola temporalmente mediante el control de su antigüedad.

La especificación del sistema de comunicaciones resulta esencial para el diseño de la arquitectura de control propuesta.

Dicha especificación se ha realizado mediante el lenguaje estándar de definición de interfaces IDL. Como resultado, se ha presentado un sistema de comunicaciones basado en el uso de una pizarra de objetos distribuida que permite la comunicación entre los agentes mediante la escritura y lectura de los objetos de la pizarra. Cada componente o agente de la arquitectura tiene asociado uno o varios objetos de la pizarra que permiten la interacción entre agentes, bien para enviar mensajes u órdenes, bien para comunicar información (por ejemplo, un agente escribe en un objeto de la pizarra la información sensorial que genera y los agentes que lo desean leen dicho objeto para recibirla). De esta forma, la incorporación de nuevos componentes en el sistema se reduce a la especificación de nuevos objetos en la pizarra. Lo cual hace a la arquitectura ideal para la incorporación dinámica de agentes, característica requerida en los entornos de control de robots móviles.

El diseño del sistema de comunicaciones está orientado a la consecución de los requerimientos de comunicación planteados en el capítulo primero para los sistemas de control de robots móviles. Su fundamento teórico está basado en el estudio previo de los estándares actuales de comunicaciones distribuidas realizado en el capítulo segundo, así como en las tendencias más recientes de comunicación basadas en la delegación de código y más concretamente en los agentes móviles estudiados en el capítulo tercero.

El diseño de cualquier arquitectura requiere la especificación de sus componentes y la forma de interconexión de los mismos. En este capítulo se ha establecido la base para el diseño de la arquitectura propuesta.

En el capítulo siguiente, tomando como base el sistema de comunicaciones desarrollado, se describen los niveles reactivo y deliberativo de la arquitectura SC-Agent así como los agentes que los forman. La relación e interacción entre los distintos componentes de ambos niveles dependerá de las conexiones existentes con los diferentes objetos de la pizarra del sistema de comunicaciones. A través de estos objetos, también podrá realizarse la transmisión de código y movilidad de los agentes.

Capítulo 5

ARQUITECTURA SC-AGENT: AGENTES

En este capítulo se diseñan los niveles reactivo y deliberativo de la arquitectura SC-Agent. Con este objetivo, se describen los distintos tipos de agentes que componen dichos niveles y las relaciones existentes entre los mismos. El sistema de comunicaciones de la arquitectura, proporcionado por el sistema SC, constituye la base sobre la que se sustenta el diseño de los niveles reactivo y deliberativo permitiendo la interacción entre los distintos agentes que los forman así como el movimiento de los mismos entre los nodos de la arquitectura distribuida.

5.1. Introducción

En el capítulo anterior se propuso el diseño de una arquitectura híbrida para el control de robots móviles denominada SC-Agent. El objetivo principal consiste en obtener una arquitectura de control que se adapte de forma rápida a distintas implementaciones de robots móviles. Para ello, el diseño de la arquitectura se fundamenta en el sistema de comunicaciones SC y la interfaz común de acceso FSA que se adaptan a cualquier recurso de comunicación o buses que interconecten los distintos nodos de la arquitectura, y que permiten la incorporación dinámica de sus componentes.

En este capítulo se aborda el estudio de los diferentes componentes *software* que son necesarios en los distintos niveles de la arquitectura SC-Agent. Fundamentalmente, los componentes principales son los agentes móviles que constituyen los niveles reactivo y deliberativo. En cuanto al sistema de comunicaciones SC, también deben especificarse los componentes que se encargan de mantener actualizada la pizarra distribuida.

La relación e interacción entre los agentes queda definida por medio de las conexiones existentes con los objetos de la pizarra. Las comunicaciones entre los agentes se realizan mediante la escritura y lectura de dichos objetos.

El objetivo es obtener un sistema multiagente o MAS que permita la comunicación entre los agentes mediante el uso de un lenguaje estándar de comunicación entre agentes como es el ACL. De esta forma, no será necesaria la definición de las interfaces de los objetos o agentes que permitan realizar las comunicaciones mediante el acceso a métodos, sino que los agentes se comunicarán enviando directamente las acciones o mensajes necesarios. El envío de dichos mensajes podrá realizarse a través de los objetos SC.

El sistema multiagente también debe permitir la movilidad de los agentes, para lo cual es necesario la definición de al menos un componente o agente que implemente las características requeridas en estos sistemas que se describieron en el capítulo tres.

A continuación, en primer lugar, se presentan de forma general los distintos tipos de componentes y agentes existentes en la arquitectura, así como las interfaces necesarias para interaccionar con el sistema. Seguidamente, se describe con detalle el diseño y estructura del nivel reactivo basado en agentes móviles y, para finalizar, se realiza la descripción del nivel deliberativo también basado en agentes.

5.2. Agentes

Los principales componentes de la arquitectura SC-Agent son agentes móviles, los cuales forman los niveles reactivo y deliberativo de la arquitectura [Posadas, 2003]. Las características de estos agentes son distintas dependiendo del nivel al que pertenezcan. Por un lado, los agentes correspondientes al nivel reactivo, tendrán que implementar los esquemas de percepción y motores necesarios así como los procesos de motivación asociados y procesos compositores, por otro lado, los agentes correspondientes al nivel deliberativo, tendrán que implementar principalmente las tareas de razonamiento y generación de modelos internos del entorno.

Para poder definir los componentes y agentes que forman los distintos niveles de la arquitectura, primero se realiza una clasificación de los tipos de objetos posibles. Después, en base a la clasificación realizada, se presentan los distintos componentes necesarios en la arquitectura. Finalmente, se concluye con las interfaces necesarias que permitan la interacción entre los componentes o agentes.

5.2.1. Tipos de Objetos

La especificación de una arquitectura distribuida requiere la especificación de los componentes que la forman y la forma de interconexión o comunicación entre los mismos.

Los distintos componentes *software* que pueden formar parte de un sistema distribuido pueden clasificarse inicialmente en objetos pasivos y objetos activos.

- Objetos pasivos. Los objetos pasivos son aquellos que no presentan hilos de ejecución, por lo tanto, son los objetos que constituyen los datos del sistema distribuido. A su vez, pueden clasificarse en objetos pasivos locales y objetos pasivos distribuidos.
    - Locales. Un objeto pasivo local es un dato ubicado en un nodo en concreto, es decir, es un dato local a un nodo. Para el acceso a dicho dato se requiere el acceso al nodo donde se encuentra.
    - Distribuidos. Un objeto pasivo distribuido es un dato ubicado en varios nodos del sistema distribuido, es decir, es un dato distribuido entre varios nodos. El dato puede estar distribuido bien mediante réplicas, bien de forma parcial.
        - Replicado. Un dato se denomina objeto pasivo distribuido replicado cuando toda la información que proporciona el dato se encuentra reproducida íntegramente en distintos nodos. El acceso a dicha información puede realizarse accediendo a cualquiera de los nodos donde el dato está replicado. El acceso a un solo nodo implica el acceso a toda la información.
        - Parcial. Un dato se denomina objeto pasivo distribuido parcial cuando toda la información que aporta el dato se encuentra repartida entre diferentes nodos. El acceso a toda la información requiere el acceso a todos los nodos donde se encuentran las distintas copias parciales del dato. El acceso a un solo nodo sólo permite el acceso a una parte de la información del dato.
- Objetos activos. Los objetos activos son aquellos compuestos por uno (monohilo) o varios (multihilo) hilos de ejecución, por lo tanto, son los objetos que constituyen los procesos del sistema distribuido. Dichos procesos, en su ejecución, acceden y generan la información o valores de los objetos pasivos o datos. Los objetos activos también pueden clasificarse en objetos activos locales y objetos activos distribuidos.
    - Locales. Un objeto activo local es un proceso cuyos hilos de ejecución se encuentran ubicados en un solo nodo. A su vez, se clasifican en

estáticos, cuando dicha ubicación es invariante en el tiempo, y móviles, cuando dicha ubicación puede cambiar en el tiempo, es decir, el proceso tiene la capacidad de moverse (con todos sus hilos) de un nodo a otro.

- Distribuidos. Un objeto activo distribuido es un proceso cuyos hilos de ejecución se encuentran repartidos entre diferentes nodos. A su vez, se clasifican en estáticos, cuando dicha distribución es invariante en el tiempo, y móviles, cuando dicha distribución puede cambiar en el tiempo, es decir, cada hilo del proceso puede moverse entre los nodos.

5.2.2. Componentes de la Arquitectura

Los niveles y sistema de comunicaciones de la arquitectura propuesta están formados por varios objetos pasivos y activos (fig.53).

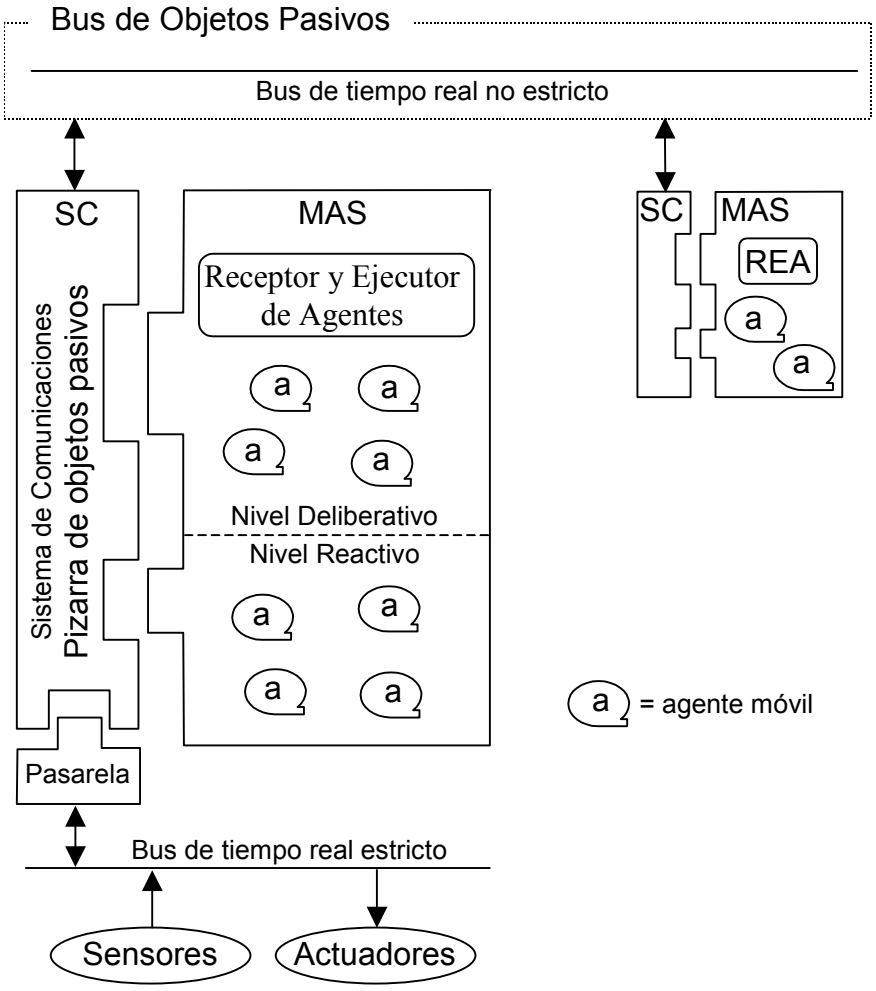


Figura 53: *Componentes de la arquitectura SC-Agent propuesta*

El sistema de comunicaciones SC, está constituido por los siguientes elementos:

- Pizarra distribuida. Estructura de datos formada por objetos pasivos distribuidos replicados. Son los objetos del sistema distribuido que contienen los datos y código necesarios para la ejecución de los objetos activos.

- Componente SC. Es un objeto activo distribuido estático. El componente SC requiere su ejecución distribuida en cada uno de los nodos que forman parte de la arquitectura. Se encarga de la actualización de la pizarra distribuida mediante el establecimiento de las comunicaciones necesarias.

- Componentes pasarelas. Son objetos activos locales estáticos que conectan el componente SC con distintos recursos de comunicación. Cada objeto pasarela está asociado a un determinado recurso de comunicación y reside en el nodo con acceso a dicho recurso. Los objetos pasarela realizan la conversión entre los distintos formatos de los mensajes empleados en los diferentes canales de comunicación.

Los niveles reactivo y deliberativo están formados principalmente por agentes *software* que pueden clasificarse como objetos activos locales móviles. En el nivel reactivo se encuentran los agentes correspondientes a los esquemas de percepción, los esquemas motores, las motivaciones y los compositores. En el nivel deliberativo se encuentran los agentes correspondientes a los procesos de planificación, generación de mapas, monitorización e interfaz con el usuario. En este nivel también se encuentra el objeto "Receptor y Ejecutor de Agentes" denominado REA que es un objeto activo distribuido estático. El componente REA, al igual que el componente SC, también requiere su ejecución distribuida en cada uno de los nodos que forman parte de la arquitectura. La misión de REA es, en primer lugar, la recepción a través del sistema SC de los agentes móviles deliberativos y reactivos y, en segundo lugar, la ejecución de los mismos en el nodo receptor. En este sentido, el objeto REA tiene la función de ejecutar los agentes reactivos proporcionados por los agentes del nivel deliberativo.

El conjunto de todos los agentes constituye el denominado sistema multiagente o MAS.

El sistema de comunicaciones SC proporciona el medio físico y lógico de conexión entre los agentes. La conexión física se realiza de una forma transparente a los agentes a través de los buses de tiempo real estricto y no estricto. La conexión lógica se realiza a través de los objetos pasivos que constituyen la pizarra. Los agentes se comunican mediante el envío y recepción tanto de datos como de código a través de la escritura y lectura de los objetos de la pizarra distribuida utilizando, para ello, el modelo de interfaz común FSA descrito en el capítulo cuarto. El modelo FSA junto al modelo de pizarra de objetos distribuida encapsula las comunicaciones necesarias haciéndolas transparentes a las aplicaciones o agentes.

El movimiento de los agentes puede realizarse a través del sistema SC mediante el envío de mensajes al objeto REA del nivel deliberativo. Dichos mensajes proporcionarán el código necesario para la ejecución del agente.

5.2.3. Interfaces

La especificación de los distintos componentes de la arquitectura SC-Agent propuesta: el componente SC y pasarelas, que constituyen el sistema de comunicaciones, y los agentes, que constituyen los niveles reactivo y deliberativo, requiere la definición de un conjunto de interfaces que permitan la interacción y comunicación entre estos componentes de forma independiente a su implementación. Dicha especificación, además, tendrá que considerar la posibilidad de interoperatividad entre componentes implementados con diferentes lenguajes de programación.

En la arquitectura propuesta, la comunicación entre los distintos objetos activos o agentes se realiza mediante el envío y recepción de mensajes a través de los objetos pasivos del sistema que constituyen la pizarra distribuida.

El modelo de comunicación basado en pizarra proporciona una conexión permanente de los objetos activos independientemente de su ubicación física. Cuando un objeto activo quiere comunicarse con otro objeto activo sólo tiene que conectarse al objeto pasivo correspondiente, enviar allí los mensajes, y el objeto activo destino los recibirá automáticamente independientemente de dónde se encuentre y sin necesidad de un conocimiento previo de esta ubicación por parte de los objetos.

En este modelo de comunicación solamente es necesaria la definición de una interfaz común que permita el establecimiento y el acceso a los objetos pasivos de la pizarra. Mediante el uso de esta interfaz, cualquier objeto activo podrá comunicarse con el resto de objetos. Los agentes recibirán a través de los objetos de la pizarra mensajes procedentes de otros agentes. Estos mensajes podrían estar basados en el uso de un lenguaje de comunicación estándar como el ACL, de forma que cada mensaje podría llevar información semántica de cómo interpretar su valor, etc.

La interfaz común utilizada (fig.54), que permite la comunicación entre los componentes de la arquitectura y el acceso a toda la información del sistema, se corresponde con el modelo FSA descrito en el capítulo cuarto.

Tal como se vio en el capítulo anterior, la interfaz FSA proporciona el acceso al sistema SC el cual permite la conexión de los diferentes nodos de la arquitectura distribuida mediante el uso de distintos recursos de comunicación (buses de tiempo real no estricto como el bus Ethernet y buses de tiempo real estricto como el bus CAN). Por lo tanto, la interfaz FSA facilita a las aplicaciones o agentes el acceso de forma transparente a cualquiera de los recursos de comunicación mediante la

escritura y lectura de los objetos de la pizarra distribuida. El sistema SC, de forma transparente a las aplicaciones, se encarga de actualizar en todos los nodos del sistema los valores de dichos objetos empleando para ello los distintos canales de comunicación que interconectan a los nodos.

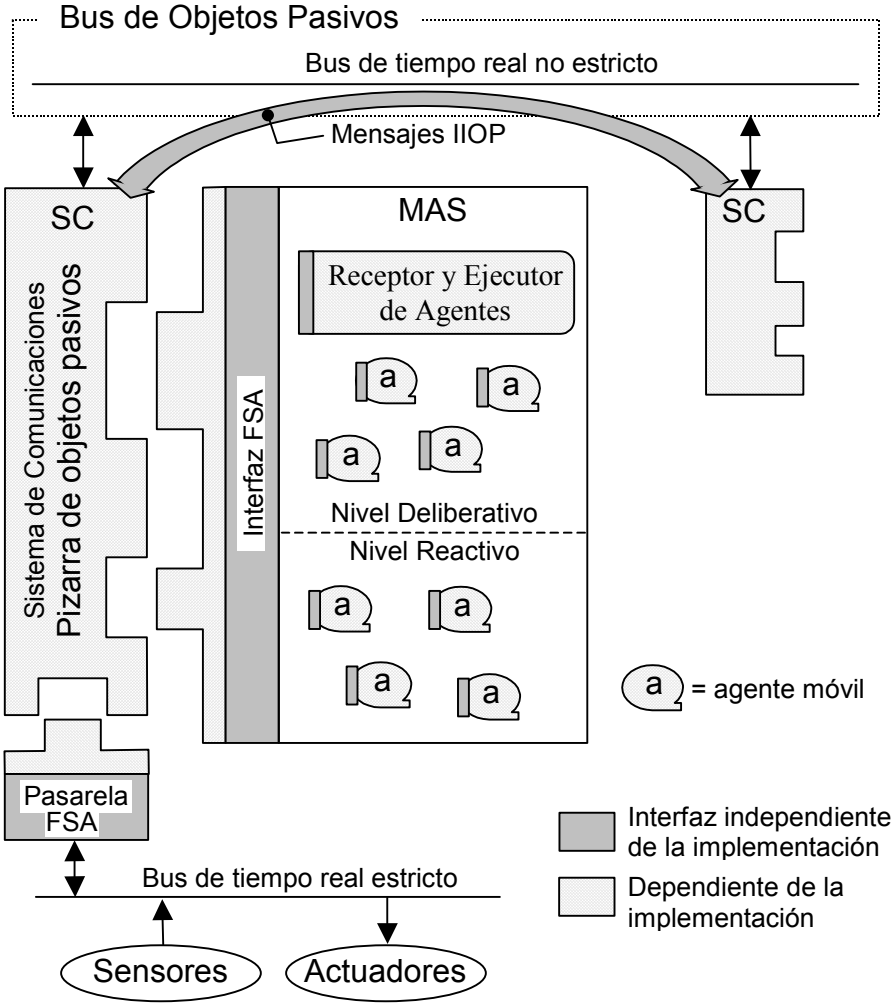


Figura 54: *Interfaces de los componentes de la arquitectura propuesta*

La interoperatividad se asegura a varios niveles:

- El uso de la interfaz FSA permite la interoperatividad entre los agentes independientemente del lenguaje de programación en el que se implementen y de su ubicación física en el sistema. Por ejemplo, podrían comunicarse agentes programados en código manejado java o CLR (.NET) con agentes en código nativo programados en C++.

- La definición de los objetos pasivos o datos de la pizarra se realiza mediante la especificación del espacio de nombres lógicos (basado en XML) descrito en el capítulo cuarto. Lo cual permite a las aplicaciones referenciar mediante

nombres lógicos los objetos a los que se desea acceder independientemente del lenguaje de programación empleado. Los métodos de la interfaz FSA admiten como parámetros estos nombres lógicos.

- La interoperatividad entre sistemas SC implementados sobre diferentes plataformas (Windows, Linux, etc.) y con diferentes lenguajes de programación se permite mediante el empleo del protocolo IIOP en la transmisión entre los componentes SC de los mensajes de actualización de los objetos de la pizarra distribuida. Esta posibilidad implica la extensión del estándar IIOP mediante la definición de nuevos tipos de mensajes específicos para la transmisión de los valores de los objetos y la antigüedad de los mismos.

5.3. Nivel Reactivo

Los niveles reactivo y deliberativo disponen de unos patrones de comportamientos y motivaciones asociadas cuya composición determina el comportamiento emergente del sistema. El nivel reactivo es el encargado de la composición de los comportamientos básicos.

5.3.1. Componentes y Agentes

El nivel reactivo (fig.55) está compuesto por agentes *software* móviles (objetos activos) que interactúan con los sensores y actuadores a través de la lectura y escritura de objetos definidos en la pizarra distribuida (objetos pasivos) del sistema SC.

En concreto, los componentes del nivel reactivo son los siguientes:

- Sensores y actuadores. Están interconectados mediante el bus de tiempo real estricto.
- Objetos activos (procesos que modelan a los sensores y actuatores):
  - Patrones de comportamientos:
    - Esquemas de percepción. Son agentes que acceden a los valores de los sensores y producen información sensorial.
    - Esquemas motores. Son agentes que acceden a la información de los esquemas de percepción (información sensorial) y producen posibles acciones (contribuciones).
  - Motivaciones. Cada esquema motor tiene asociada una motivación. Las motivaciones son agentes que acceden a la información de los

esquemas de percepción (información sensorial) y ponderan las contribuciones de los esquemas motores.

- Compositores. Son agentes que computan las acciones definitivas para ser enviadas a los actuadores. Acceden a la información de los esquemas motores (contribuciones) y a la información de las motivaciones asociadas (peso de las contribuciones).

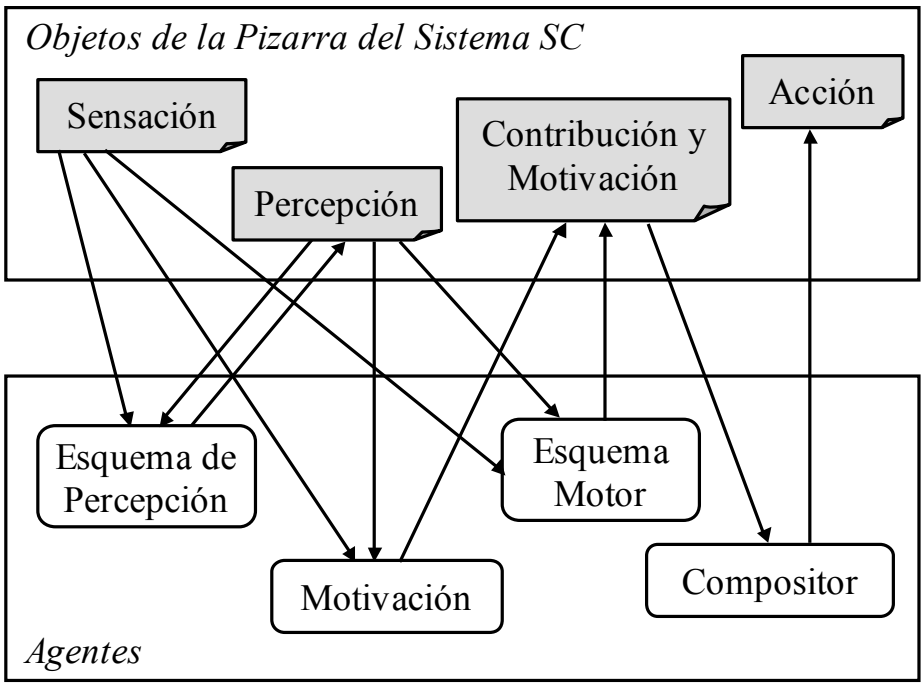


Figura 55: *Relación entre objetos pasivos y activos del nivel reactivo*

- Objetos pasivos (constituyen la información soportada por la pizarra). De forma general, pueden distinguirse varios tipos:

  - Sensación: objeto que representa el valor o conjunto de valores directamente medidos por un sensor.

  - Percepción: objeto que representa el resultado de la elaboración y fusión de las sensaciones. Las percepciones almacenan los resultados de la ejecución de los agentes esquemas de percepción.

  - Contribución: objeto que representa el valor o resultado de un comportamiento básico. Las contribuciones almacenan los resultados de la ejecución de los agentes esquemas motores.

  - Motivación: objeto que representa el coeficiente de motivación asociado a una contribución. Las motivaciones almacenan los resultados de la ejecución de los agentes de motivación.

- Acción: objeto que actualiza con su valor los actuadores. Una acción obtiene su valor mediante la ponderación de todas sus contribuciones. Las acciones almacenan los resultados de la ejecución de los agentes compositores.

Todos los agentes descritos son procesos concurrentes y distribuidos que acceden localmente a la información del SC y que generan nueva información independientemente de lo que los otros agentes estén haciendo. Los agentes, al obtener los valores de los objetos de la pizarra, también obtienen la información temporal (antigüedad) de los mismos.

5.3.2. Conexión entre los agentes

La conexión necesaria entre los agentes del nivel reactivo [Posadas, 2003] se realiza a través del sistema SC.

Símbolo	Descripción	
S_i	Sensor i	$\forall i \in [1,\ n]$
TS_i	Periodo de transmisión de los valores del Sensor i	
OS_i	Objeto SC asociado al Sensor i	
EP_k	Esquema de Percepción k	$\forall k \in [1,\ m]$
OEP_k	Objeto SC asociado al Esquema de Percepción k	
EM_q	Esquema Motor q	$\forall q \in [1,\ p]$
OEM_q	Objeto SC asociado al Esquema Motor q	
m_q	motivación q	
Om_q	Objeto SC asociado a la motivación q	
C_s	Compositor s	$\forall s \in [1,\ r]$
OA_s	Objeto SC asociado al Actuador s	
TA_s	Periodo de transmisión de los valores del Actuador s	
A_s	Actuador s	

Siendo: *n* el número de sensores, *m* el número de esquemas de percepción, *p* el número de esquemas motores y motivaciones asociadas, y *r* el número de actuadores y compositores correspondientes.

Tabla 6: *Nomenclatura en el nivel reactivo*

Cada componente del nivel reactivo de la arquitectura (sensor, actuador, esquema de percepción, esquema motor o motivación), a excepción de los compositores, tiene asociado en el SC un objeto cuyo valor refleja el valor de dicho componente (tabla 6) (fig.56). Así, existe un objeto por cada sensor, actuador, esquema de percepción, esquema motor o motivación. Los compositores no tienen asociados objetos en el SC, sino que se encargan de proporcionar los valores a los objetos asociados a los actuadores.

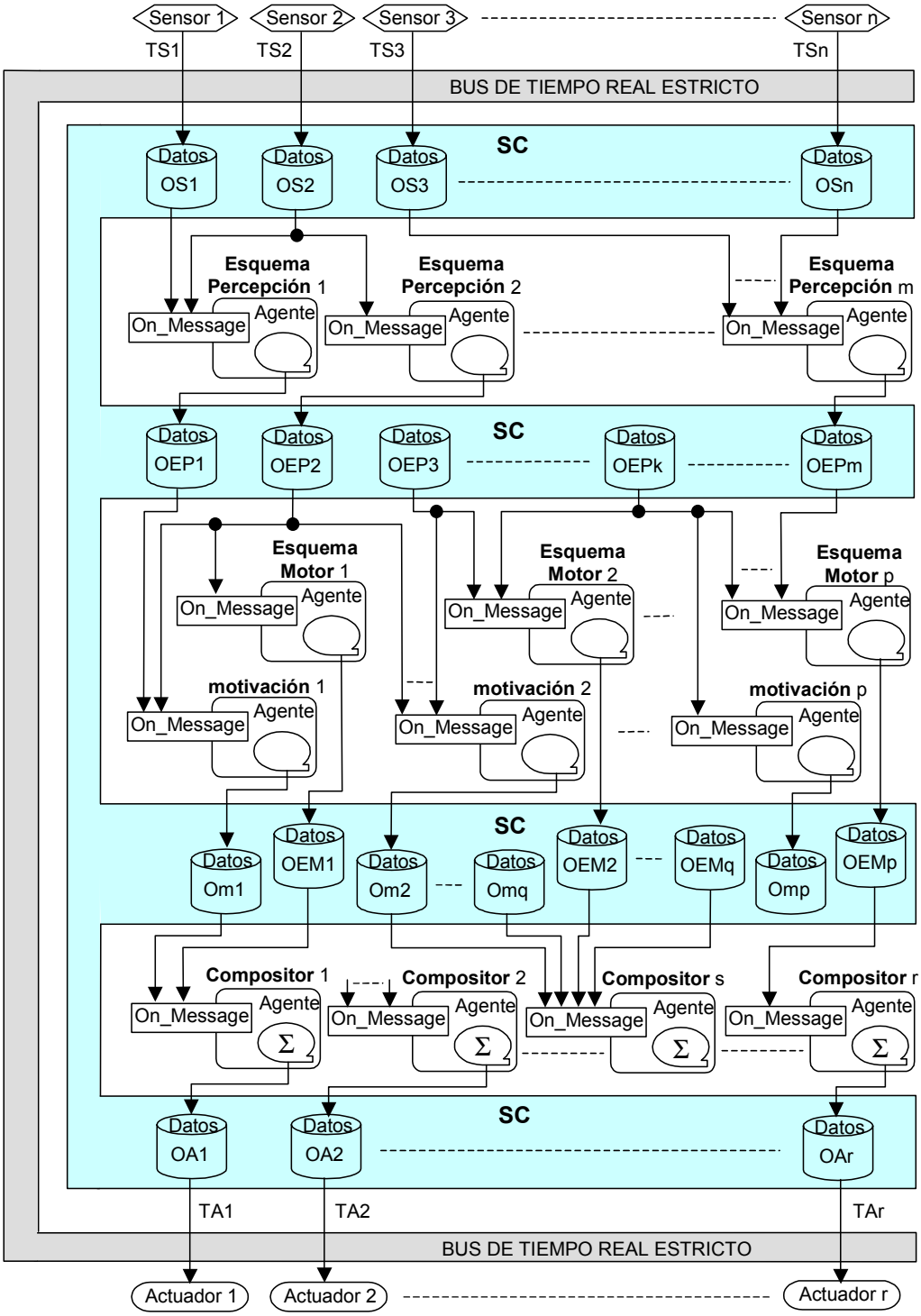


Figura 56: *Nivel reactivo: conexión y funcionamiento de los agentes a través del sistema SC*

Cada sensor (S_i) del robot está controlado por un módulo sensor de procesamiento (procesador empotrado) con acceso al bus de tiempo real estricto, de forma que este módulo transmite con cierto periodo (TS_i) el valor del sensor asociado enviando un mensaje al bus de tiempo real estricto.

La pasarela entre el sistema SC y el bus de tiempo real estricto recibe todos los mensajes procedentes de todos los módulos sensores y los envía a los objetos (OS_i) correspondientes del SC. Como resultado, todos los valores de todos los sensores se encuentran accesibles a través del sistema SC para cualquier componente software de cualquier nivel de la arquitectura.

Cada esquema de percepción se conecta a los objetos (OS_i) del SC cuyos valores necesita para generar su resultado. Por ejemplo, un esquema de percepción (EP_k) podría requerir los valores de los sensores de infrarrojos para determinar la presencia o no de un obstáculo. Seguidamente, el esquema de percepción envía la información sensorial generada a su objeto (OEP_k) asociado.

Por otro lado, cada esquema motor obtiene la información que necesita de los esquemas de percepción conectándose de igual modo a los objetos del SC (OEP_k) correspondientes. A partir de la información sensorial que obtiene un esquema motor, éste genera una respuesta en forma de acción. Por ejemplo, un esquema motor encargado de evitar obstáculos, al recibir de un esquema de percepción la existencia de un obstáculo, generaría las órdenes oportunas para esquivarlo. Este esquema motor podría funcionar indicando la velocidad para los motores de las ruedas, por lo que al ser informado de la presencia de un obstáculo, podría indicar las velocidades adecuadas a cada rueda para girar y no chocar. Cada esquema motor (EM_q) envía las acciones que genera a su objeto (OEM_q) correspondiente del SC.

Cada esquema motor tiene asociada una motivación. Las motivaciones son componentes software que se ejecutan de forma independiente y determinan el grado de importancia de las acciones generadas por los esquemas motores. Por ejemplo, la importancia de las acciones generadas por dos esquemas motores, uno indicando cómo ir a un objetivo y otro indicando cómo ir a recargar baterías, no es la misma dependiendo de si el robot dispone o no de la energía suficiente para alcanzar dicho objetivo. Cada motivación (m_q) se conecta a los objetos (OEP_k) del SC que necesita para determinar la importancia de su esquema motor (EM_q) asociado. El resultado lo envía al objeto SC (Om_q) correspondiente.

El último componente software del nivel reactivo son los compositores. El resultado de los compositores serán las acciones que el robot, en cada momento, llevará definitivamente a cabo. Para ello, los compositores proporcionan los valores a los objetos asociados a los actuadores del robot.

Cada actuador tiene un objeto SC asociado, los valores de este objeto son las acciones que el actuador realiza. Por ejemplo, el actuador (A_s) correspondiente a la velocidad de los motores de las ruedas, ejecuta la velocidad indicada en su objeto (OA_s) asociado del SC. Para ello, al igual que los sensores, cada actuador está controlado por un módulo de procesamiento (procesador empotrado) con conexión al bus de tiempo real estricto. En este caso, la pasarela entre el sistema SC y el bus de tiempo real estricto se encarga de enviar, también con cierto periodo para cada actuador (TA_s), los valores de los objetos de los actuadores mediante mensajes a

través del bus de tiempo real estricto. Cada módulo de procesamiento recibe los mensajes que le corresponden y realiza las acciones indicadas.

Cada actuador es controlado únicamente por un solo compositor, aunque este compositor puede cambiar dinámicamente.

Un compositor (C_s) determina la acción para su actuador asociado (A_s) utilizando los valores (contribuciones) generados por uno o varios esquemas motores y motivaciones asociadas. Un compositor se denomina básico o primitivo cuando utiliza únicamente el valor de un esquema motor para proporcionar la acción definitiva al actuador. Por otro lado, el compositor se denomina abstracto si utiliza varios valores (contribuciones) de distintos esquemas motores para generar la acción definitiva.

Un compositor abstracto es un conjunto de varios patrones de comportamientos básicos, donde cada comportamiento tiene un determinado peso en la acción definitiva. La forma en que estos comportamientos básicos intervienen en las acciones finales (peso o grado de influencia), determina el mecanismo mediante el que el compositor abstracto gobierna al actuador. Como ya se comentó en el capítulo anterior, existen dos posibilidades:

- Secuenciación o conmutación de comportamientos básicos.
- Composición de comportamientos básicos (comportamiento emergente).

En ambos casos, el empleo de los valores de las motivaciones será imprescindible.

Independientemente del tipo de compositor (básico o abstracto), éste se conectará a los objetos del SC (esquemas motor OEM_q y motivaciones Om_q) que necesite y enviará las acciones generadas al objeto (OA_s) del actuador correspondiente.

Todos los agentes reactivos descritos son proporcionados por el nivel deliberativo a través del bus de tiempo real no estricto. Los agentes son móviles y pueden viajar a través del sistema distribuido. Sin embargo, la ejecución de los agentes reactivos con restricciones de tiempo real estricto solamente será posible en los nodos con acceso a la infraestructura de bus de tiempo real.

Esta arquitectura se adapta de forma rápida a cualquier implementación independientemente del número de sensores o adaptadores disponibles. Los componentes *software*: esquemas de percepción y motores, motivaciones y compositores pueden incorporarse dinámicamente al sistema. De esta forma, la implementación de la arquitectura para un sistema determinado consiste en la instanciación de los agentes correspondientes, los cuales, dada la modularidad de la arquitectura, pueden desarrollarse de manera independiente e incorporarse al sistema rápidamente mediante el uso del sistema SC.

5.4. Nivel Deliberativo

El nivel deliberativo se encarga de la planificación de las tareas para conseguir los objetivos asignados al robot. Para ello, divide dichos objetivos en subtareas y envía los componentes *software* necesarios (esquemas de percepción, esquemas motores, motivaciones y compositores) al nivel reactivo.

5.4.1. Agentes y Conexión con el Sistema

El nivel deliberativo es distribuido y está compuesto por los siguientes agentes software móviles (fig.57):

- Agente Interfaz con el Usuario. Interactúa con el usuario para recibir los objetivos o tareas que el robot tiene que realizar. Independientemente del grado de inteligencia de un robot, siempre existirá un nivel superior correspondiente a la interfaz con el usuario quien le ordena (teleoperación) de una forma más o menos directa las tareas a realizar. El nivel de detalle con el que se indican dichas tareas está directamente relacionado con el grado de autonomía del robot.

- Agente Planificador. Es el componente principal del nivel deliberativo. Planifica cómo alcanzar los objetivos asignados al robot en base a un modelo interno del entorno. Se encarga de la descomposición y secuenciación de dichos objetivos y de la inyección de agentes en el nivel reactivo.

- Agente Generador de Mapas. Se encarga de la construcción de un modelo interno del entorno.

- Agente Monitor. Se encarga de supervisar el rendimiento del sistema y los posibles problemas que surjan.

Todos los agentes pueden acceder a toda la información disponible en la arquitectura mediante el uso de la interfaz común FSA del sistema de comunicaciones SC. De esta forma, los agentes del nivel deliberativo se conectan rápidamente al resto de la arquitectura pudiendo obtener del nivel reactivo la información sensorial que requieran. El sistema SC, tal como se describió, proporciona dicha información a cualquier nodo de la arquitectura (con o sin infraestructura de bus de tiempo real estricto) caracterizándola con su antigüedad.

5.4.2. Sistema Multiagente

El nivel deliberativo debe proporcionar al nivel reactivo sus agentes (agentes esquemas de percepción, agentes motores, agentes de motivación y agentes compositores), lo cual implica la necesidad de un sistema que permita la delegación del código necesario.

Además, todos los agentes móviles (deliberativos y reactivos) tienen que poder moverse a través del bus de tiempo real no estricto entre los distintos nodos de la arquitectura, para así permitir su ejecución allí donde las condiciones del entorno sean más favorables.

Todo ello implica que la arquitectura SC-Agent debe incorporar un sistema de agentes móviles que permita dichos requerimientos, el sistema proporcionado se basa en los estándares existentes.

Los estándares de sistemas multiagente móviles se componen principalmente de los siguientes elementos: componente infraestructura de comunicaciones, componente control de agentes y componente directorio de agentes. En la arquitectura SC-Agent propuesta estos componentes se identifican con el sistema SC (componente infraestructura de comunicaciones) y con el objeto denominado Receptor y Ejecutor de Agentes o REA (que engloba a los componentes para el control y directorio de agentes).

El componente REA gestiona las distintas operaciones de control sobre los agentes y hace posible el movimiento de los agentes entre los distintos nodos.

Tal como se describió en la sección 5.2.2., REA es un objeto activo distribuido estático, lo cual significa que se requiere una instancia del mismo en ejecución en cada nodo de la arquitectura distribuida, al igual que sucede con el componente SC.

El objeto REA tiene asociado un objeto (OREA) en la pizarra distribuida a través del cual recibe los mensajes de los distintos agentes indicando sus acciones o intenciones, por ejemplo, la intención de moverse a un determinado nodo. En la sección siguiente se describe con detalle el mecanismo y formato de los mensajes empleados.

En la figura 57 pueden observarse los distintos agentes del nivel deliberativo y la conexión con los agentes del nivel reactivo a través del sistema SC. Todos los agentes se sitúan sobre un sistema multiagente que dispone del componente REA para la gestión y movimiento de los mismos.

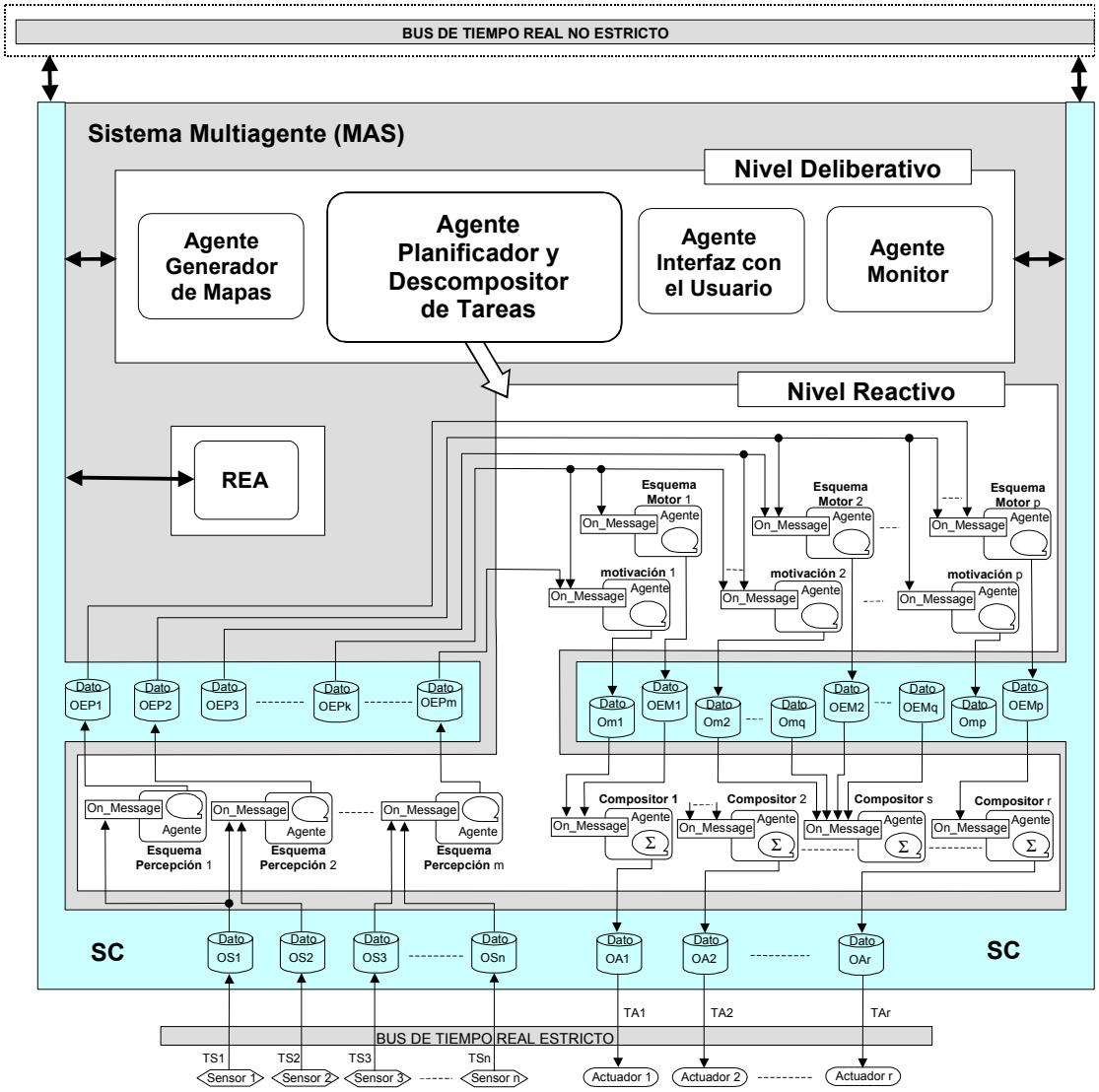


Figura 57: *Agentes del nivel deliberativo y sistema multiagente: movilidad y conexión con el nivel reactivo a través del sistema SC*

5.4.2.1. Interoperatividad

Un aspecto importante en la arquitectura SC-Agent es proporcionar la interoperatividad entre agentes independientemente de su ubicación física e independientemente del lenguaje de programación empleado para su implementación. En este sentido, la comunicación mediante mensajes a través de los objetos de la pizarra distribuida hace transparente la ubicación de los agentes, y el empleo de la interfaz FSA hace transparente el lenguaje de implementación del agente. Sin embargo, para que la comunicación sea efectiva y exista un entendimiento entre los agentes, además debe utilizarse un formato común para los mensajes transmitidos.

Todo esto requiere de la especificación de un lenguaje de comunicación entre los agentes, para ello se ha utilizado el lenguaje estándar de comunicación entre agentes conocido como ACL.

El sistema SC se encargará, a través de los objetos correspondientes de la pizarra, de la transmisión de los mensajes ACL entre los distintos nodos del sistema distribuido. Los valores de los objetos de la pizarra se corresponderán con mensajes ACL. Cuando un agente se conecta a un objeto de la pizarra mediante la interfaz FSA del sistema SC, al final lo que recibirá serán mensajes ACL que deberá interpretar con la información incorporada en los mismos.

El movimiento de los agentes entre los nodos también se realiza mediante el envío de un mensaje ACL al objeto REA correspondiente. Cuando el objeto REA de un nodo recibe un mensaje ACL indicando el movimiento de un agente a dicho nodo, el objeto REA obtiene el código serializado de dicho agente a través del contenido del mensaje y lo ejecuta en el sistema multiagente o MAS. Para ello, es necesario que tanto el objeto REA como el código recibido estén implementados en el mismo lenguaje de programación.

Sin embargo, como la implementación de los agentes es independiente del lenguaje de comunicación empleado, la interacción entre agentes de distintos lenguajes de programación también es posible. En el caso del movimiento de código y agentes, podría haber en un mismo nodo, conectados al sistema SC, una implementación del objeto REA para la ejecución de agentes java y otra implementación del objeto REA para la ejecución de agentes .NET. Ambos objetos REA recibirían todos los mensajes ACL procedentes de otros agentes independientemente del lenguaje de programación. Sin embargo, en el caso de mensajes indicando un movimiento de código, lógicamente, sólo el objeto REA basado en el lenguaje de programación de dicho código tendría la capacidad de ejecutarlo. El mensaje ACL tiene que proporcionar toda la información sobre el tipo de mensaje, la procedencia y destino del mismo, lenguaje de programación empleado, etc.

En la figura 58 puede observarse el diseño correspondiente a la composición de mensajes para su transmisión a través del sistema SC. Desde el nivel de aplicación, los agentes o procesos se comunicarán mediante el envío y recepción de mensajes ACL. Estos mensajes ACL serán los valores que las aplicaciones escribirán y leerán a través de los objetos de la pizarra. Para ello, las aplicaciones utilizarán, tal como se describió, la interfaz FSA del sistema SC. Antes de ser transmitidos estos valores o mensajes, el sistema SC les añade una cabecera con la información correspondiente al nombre lógico (basado en XML) del objeto de la pizarra asociado y a la antigüedad del valor o mensaje. De esta forma, cuando el mensaje es recibido por el componente SC de los distintos nodos del sistema distribuido, el componente SC puede conocer con qué objeto de la pizarra va asociado el mensaje y puede actualizar la información temporal del mismo relativa a su antigüedad. Para asegurar la interoperatividad entre varias implementaciones del sistema SC sobre diferentes plataformas y distintos lenguajes de programación, los mensajes SC pueden

encapsularse en un mensaje IIOP codificándose según el estándar CDR de representación común de datos.

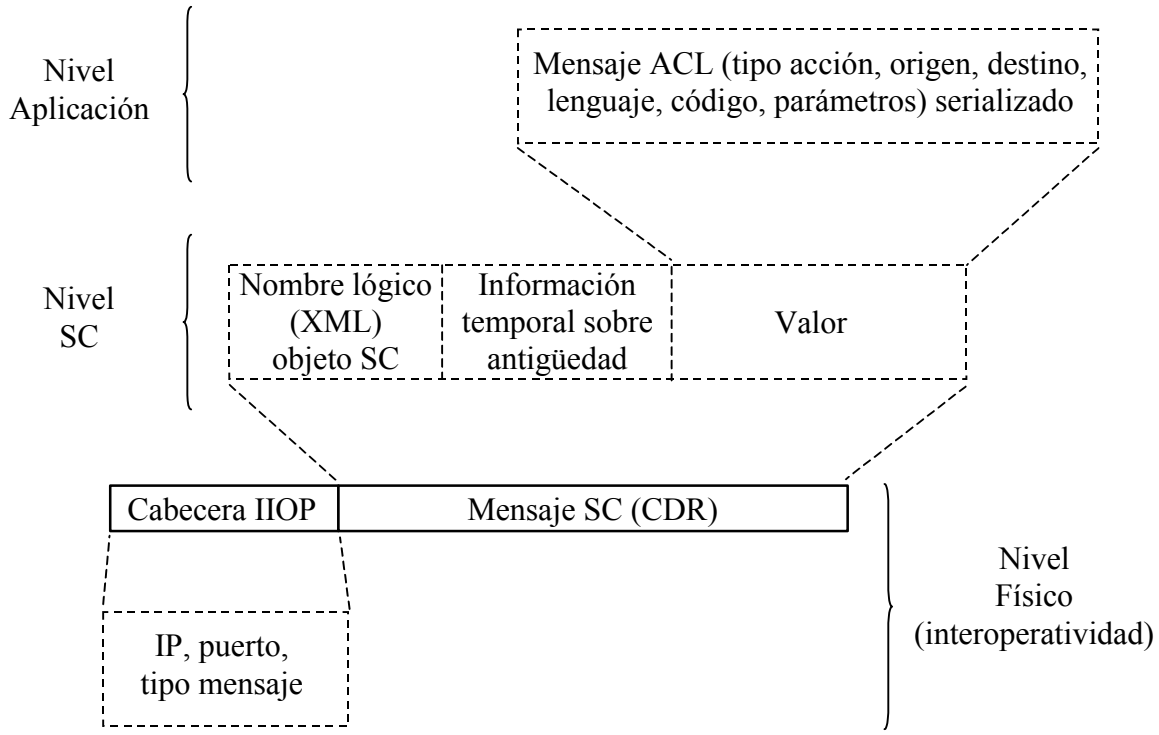


Figura 58: *Formato de los mensajes para las comunicaciones entre agentes a través del sistema SC: uso del protocolo IIOP, nombres lógicos (XML) y lenguaje de comunicación ACL*

El diseño presentado requiere la especificación de los distintos tipos de mensajes ACL que pueden transmitirse entre los agentes. En una primera clasificación, pueden diferenciarse:

- Mensajes orientados a la comunicación general de la información del sistema (valores de los sensores, resultados de los esquemas de percepción, de los esquemas motores, motivaciones, etc.). Este tipo de mensajes no requieren respuesta, sino que con ellos únicamente se hace disponible cierto tipo de información a todo el sistema. Un agente interesado en dicha información (por ejemplo, un esquema de percepción interesado en los valores de los sensores de infrarrojos) tendrá que estar conectado (a través de la interfaz FSA del sistema SC) al objeto u objetos correspondientes de la pizarra.

- Mensajes orientados a la comunicación de una acción o intención por parte de un agente (decisión de moverse a otro nodo, decisión por parte del agente planificador de ejecutar determinados agentes esquemas, conexión entre dos agentes para entablar una conversación, etc.). Estos mensajes requieren una respuesta, bien por parte del sistema, bien por parte de otro agente. Por

ejemplo, en el caso de un mensaje correspondiente a la ejecución en un determinado nodo de un agente, el objeto REA de dicho nodo tendrá que recibir el mensaje e iniciar la ejecución del agente indicado.

En ambos casos, el agente que envía el mensaje tiene que conocer el nombre del objeto de la pizarra donde escribirlo. En el primer caso, únicamente hay que utilizar el objeto que tiene asociado el propio agente. Serán el resto de agentes, que deseen conocer la información que éste proporciona, los que tendrán que conocer el nombre del objeto que tiene asociado el agente para así poder obtener su información. En el segundo caso, cuando un agente quiere comunicar con otro agente, éste tiene que conocer el objeto de la pizarra asociado al agente con el que desea comunicar. Sin embargo, si la acción a realizar es una acción de control (creación, movimiento, clonación, destrucción de agentes, etc.) o localización de agentes, sólo es necesario conocer el nombre del objeto de la pizarra asociado al componente REA de los nodos distribuidos. Por defecto, en todos los nodos se recibirá el mensaje y solamente el objeto REA del nodo donde va dirigido el mensaje (información proporcionada con los parámetros del mensaje ACL) realizará la acción indicada.

La utilización de los nombres de los objetos de la pizarra para distribuir información en todo el sistema y para comunicar entre sí los distintos componentes (agentes) de la arquitectura, permite una sencilla adaptación de la arquitectura a las necesidades del sistema en cada caso. Es decir, la incorporación de nuevos agentes en el sistema que puedan acceder a toda la información del mismo se reduce a proporcionar a dichos agentes los nombres correspondientes de los objetos de la pizarra. Es independiente quién genera y dónde se genera esa información, son características que pueden cambiar incluso dinámicamente. Por otro lado, la incorporación de nuevos agentes en el sistema que puedan proporcionar nueva información y comunicarse con cualquier otro agente, se reduce a la definición de nuevos objetos en la pizarra asociados a los nuevos agentes.

En la ejecución de un agente sólo hay que indicar los objetos de la pizarra a los que desea conectarse para obtener información y, en su caso, los nombres de los objetos que hay que crear en la pizarra para que este nuevo agente pueda proporcionar nueva información o para que pueda recibir mensajes de otros agentes.

La relación entre los distintos agentes se configura mediante el establecimiento de conexiones entre los objetos de la pizarra. Esta característica hace que los componentes de la arquitectura sean totalmente independientes entre sí. Un componente depende únicamente de la información que necesita independientemente de quién, cómo o dónde se proporciona, características que pueden cambiar sin que afecte a la ejecución del componente.

En cuanto al envío de los mensajes SC, encapsulados mediante el protocolo IIOP, es preciso extender el estándar IIOP con la definición de un nuevo tipo de mensaje que se utilizará para crear una cabecera IIOP específica del sistema SC.

Los tipos clásicos de los mensajes IIOP están orientados a la interacción de aplicaciones cliente/servidor y a la invocación remota de métodos. En el caso del sistema SC, puede crearse un nuevo tipo de mensaje orientado al envío de información entre el componente SC de los distintos nodos de la arquitectura distribuida para la actualización de los valores de los objetos de la pizarra. Este nuevo tipo de mensaje IIOP recibe el nombre de: SCObjectValue.

5.4.2.2. Mensajes ACL

Los agentes se comunican a través del sistema SC mediante el empleo de la interfaz FSA. Esta interfaz permite la escritura y lectura de los objetos de la pizarra distribuida cuyos valores se corresponden con los mensajes ACL. Cuando un agente escribe sobre un objeto de la pizarra, el valor proporcionado debe ser un mensaje ACL. Por otra parte, cuando un agente se conecta con un objeto de la pizarra para recibir automáticamente sus valores, éstos serán, igualmente, mensajes ACL.

En la implementación de la interfaz FSA se dispone del método que permite a un agente el envío de valores que previamente ha compuesto formando un mensaje ACL. Sin embargo, también podría disponerse de métodos que a partir de sus parámetros compusieran automáticamente el mensaje ACL para ser transmitido.

Los mensajes ACL orientados a la comunicación general de la información del sistema tienen la siguiente estructura:

```
(informSC
    :sender A.B.C.D
    :receiver allHosts
    :language format
    :content expression
)
```

Figura 59: *Mensaje ACL para comunicar información del sistema*

Donde *A.B.C.D* hace referencia a la dirección IP de donde procede la información, "allHosts" indica que el mensaje no va dirigido a un destino en concreto y *format* hace referencia a un nombre que identifica el formato del contenido del mensaje. Dicho contenido o valor se proporciona a continuación del parámetro "content" mediante *expression* que hace referencia a un valor formateado según *format*.

Por otro lado, los mensajes ACL orientados a la comunicación de una acción o intención por parte de un agente tienen la siguiente estructura:

```
(msgAgent
    :sender param1
    :receiver param2
    :language param3
```

```
    :content param4
    ..............
)
```

Figura 60: *Mensaje ACL para comunicar acciones*

Donde *msg*Agent indica una orden para la realización por parte del objeto REA de alguna de las siguientes acciones:

- "ExecuteAgent": indica la ejecución de un agente, bien debido a la creación del mismo o a un movimiento entre nodos.

```
(ExecuteAgent
    :sender agent
    :receiver A.B.C.D
    :language programmingLanguage
    :content code
)
```

Figura 61: *Mensaje ACL correspondiente a la acción de ejecutar un agente*

Donde *agent* hace referencia al nombre del agente que envía la acción, *A.B.C.D.* hace referencia a la dirección IP del nodo donde debe ejecutarse el nuevo agente (dicha acción la realizará el componente REA de dicho nodo), *code* es el contenido del mensaje correspondiente al nombre y código serializado del agente que debe ejecutarse, y *programmingLanguage* hace referencia al lenguaje de implementación de dicho agente (podría tener los valores "C++", "java", ".NET", etc.). A través del valor del parámetro "language" el componente REA puede determinar si es capaz o no de ejecutar el código recibido. En un mismo nodo podría haber varios componentes REA para distintos lenguajes de programación.

- TerminateAgent": indica la terminación de un agente.

```
(TerminateAgent
    :sender agent
    :receiver A.B.C.D
    :language programmingLanguage
    :content name
)
```

Figura 62: *Mensaje ACL correspondiente a la acción de terminar un agente*

Donde *agent* hace referencia al nombre del agente que envía la acción, *A.B.C.D.* hace referencia a la dirección IP del nodo donde se encuentra el agente que hay que finalizar (dicha acción la realizará el componente REA

de dicho nodo), *name* es el contenido del mensaje correspondiente al nombre del agente que debe terminarse, y *programmingLanguage* hace referencia al lenguaje de implementación de dicho agente.

Por otro lado, el objeto REA también tiene la función de registrar los nombres de todos los agentes que están en ejecución. De esta forma, el componente REA de cada nodo gestiona los agentes que ejecuta para poder facilitar una lista con los nombres de todos los agentes que hay en ejecución en su nodo. Cualquier agente puede solicitar dicha información enviando el mensaje ACL correspondiente al objeto REA. Dicho mensaje presenta la siguiente estructura:

```
(requestREA
    :sender agent
    :receiver A.B.C.D
    :language programmingLanguage
    :content SCObject
    :reply-with id
)
```

Figura 63: *Mensaje ACL para solicitar información sobre los agentes en ejecución al objeto REA*

Donde *agent* hace referencia al nombre del agente que envía la acción, *A.B.C.D.* hace referencia a la dirección IP de donde se solicita la lista de agentes en ejecución (dicha lista la proporcionará el componente REA de dicho nodo), *programmingLanguage* hace referencia al lenguaje de implementación de dichos agentes, *SCObject* es el contenido del mensaje que se corresponde con el nombre del objeto SC sobre el que se debe responder, e *id* hace referencia a un identificador que utilizará el objeto REA para responder al mensaje.

El componente REA, del nodo donde se recibe el mensaje, responderá a esta petición mediante el envío de un nuevo mensaje a través de su escritura sobre el objeto de la pizarra distribuida indicado en el contenido del mensaje de solicitud. La estructura de este nuevo mensaje es:

```
(informREA
    :sender A.B.C.D
    :receiver agent
    :language programmingLanguage
    :content list
    :in-reply-to id
)
```

Figura 64: *Mensaje ACL correspondiente a la respuesta del objeto REA*

Donde *A.B.C.D.* hace referencia a la dirección IP del nodo desde donde se está respondiendo con la lista de agentes en ejecución en dicho nodo, *agent* hace

referencia al nombre del agente que envió la petición, *programmingLanguage* hace referencia al lenguaje de implementación de los agentes de la lista, *list* es el contenido del mensaje correspondiente a la lista con los nombres de los agentes que se encuentran en ejecución en el nodo con la dirección *A.B.C.D.*, e *id* hace referencia al identificador que se envió en el mensaje de solicitud. Este identificador sirve para emparejar el mensaje de solicitud con el de respuesta.

Para facilitar el envío de los mensajes ACL correspondientes a la gestión del control de los agentes reactivos, es decir, las operaciones de ejecución y detención de los esquemas de percepción, los esquemas motores, las motivaciones y los compositores, que lleva a cabo el componente REA de cada nodo del sistema tras la recepción del mensaje adecuado, podría proporcionarse un conjunto de métodos que convirtieran automáticamente sus parámetros en los mensajes ACL correctos. El uso de estos métodos no es obligatorio, pues lo importante es componer el mensaje ACL y enviarlo al objeto correspondiente de la pizarra. Sin embargo, mediante estos métodos se facilitaría el trabajo al realizarse de forma automática la composición del mensaje ACL y el envío del mismo al objeto de la pizarra asociado al componente REA.

5.5. Conclusiones

En este capítulo se han descrito los niveles reactivo y deliberativo de la arquitectura SC-Agent mediante la definición e interacción de sus componentes, los cuales son principalmente agentes *software* móviles.

Los distintos componentes de la arquitectura se sitúan entorno al sistema de comunicaciones SC, el cual permite la interconexión de los mismos. De la misma forma, el sistema de agentes necesario para permitir la gestión y movilidad de los agentes, también se proporciona mediante la conexión al sistema SC de un componente denominado REA, cuya funcionalidad está basada en los estándares de sistemas de agentes descritos en el capítulo tres.

Como conclusión principal, se destaca que el diseño de la arquitectura SC-Agent queda caracterizado por la modularidad, independencia y flexibilidad de sus componentes. Al basarse el diseño en un sistema de comunicaciones general orientado a los requerimientos comunicacionales de las arquitecturas de control híbridas, la arquitectura SC-Agent propuesta permite la rápida conexión y comunicación entre todos sus componentes, los cuales pueden incorporase y cambiarse de forma dinámica en el sistema en función de las necesidades de control e implementación que en cada momento o caso particular se requieran. De esta forma, la arquitectura se adapta de forma rápida a los recursos *hardware* disponibles (sensores, actuadores, buses, etc.).

Los agentes pueden comunicarse independientemente de su ubicación en la arquitectura mediante el envío de mensajes a través de los objetos de la pizarra

distribuida. Al utilizar la interfaz FSA para el acceso a la pizarra, los agentes también podrán comunicarse independientemente del lenguaje de programación que haya sido empleado en su implementación. Es necesario utilizar un formato común para los mensajes (como el definido en el estándar ACL) de forma que los agentes puedan entenderse entre ellos.

En el capítulo siguiente se exponen las distintas implementaciones realizadas de la arquitectura SC-Agent y los resultados obtenidos. En concreto, se describen los prototipos desarrollados para validar el sistema de comunicaciones SC, las técnicas de delegación de código, el control temporal de la información del sistema y el movimiento de los agentes en función del índice de comodidad. Asimismo, se presenta la implementación realizada de la arquitectura para su aplicación al control de un determinado robot físico real.

Capítulo 6

ARQUITECTURA SC-AGENT: IMPLEMENTACIÓN

En este capítulo se describe la implementación de la arquitectura SC-Agent. En primer lugar, se presenta el sistema de comunicaciones SC desarrollado, así como su aplicación a nivel industrial en entornos distribuidos. Entornos que tienen en común la necesidad de comunicación entre programas heterogéneos y donde la modularidad software es un requisito fundamental. A continuación, se describe un prototipo software basado en el mismo sistema SC que realiza el control de robots simulados móviles mediante la arquitectura SC-Agent. Finalmente, partiendo de este prototipo, la arquitectura SC-Agent se contextualiza aplicándola al control de un robot real denominado YAIR.

6.1. Introducción

El objetivo de la implementación de la arquitectura SC-Agent, es obtener un sistema de control modular que pueda adaptarse de forma rápida a distintos ámbitos de aplicación, siendo uno de ellos el control de robots móviles. Por ello, dicha implementación está basada en el desarrollo de un componente común, necesario en todas las implementaciones, que permite la adaptación e incorporación rápida del resto de componentes que dependen de cada caso particular.

Este componente común consiste en el sistema de comunicaciones SC, el cual permite la comunicación y conexión entre el resto de componentes o agentes de la arquitectura.

En este sentido, la implementación de la arquitectura SC-Agent puede estructurarse en las siguientes partes:

1. Implementación del sistema de comunicaciones SC. Cuyas características principales son:

   - Proporcionar la pizarra distribuida de objetos.
   - Asegurar la consistencia de la pizarra en todos los nodos del sistema actualizando los valores de los objetos.
   - Establecimiento y mantenimiento automatizados de las conexiones entre los nodos de la arquitectura a través del bus de tiempo real no estricto, necesarias para la actualización de la pizarra.
   - Actuar de pasarela entre el bus de tiempo real estricto y el bus de tiempo real no estricto permitiendo el acceso transparente a los mismos desde cualquier nodo y cualquier nivel de la arquitectura.
   - Proporcionar a los agentes la interfaz de acceso basada en el modelo marco-sensor-adaptador o FSA.

2. Implementación del sistema multiagente a través de los objetos REA (receptor y ejecutor de agentes) necesarios en función de los lenguajes de programación empleados en el desarrollo de los agentes móviles.

3. Implementación de los agentes de control requeridos en el ámbito concreto de aplicación (principalmente los agentes deliberativos y agentes esquemas del nivel reactivo).

En la sección siguiente se expone la implementación realizada del sistema SC, la cual constituye la parte de la arquitectura SC-Agent que es independiente del ámbito de aplicación. A continuación, se describen las implementaciones completas de la arquitectura para su aplicación en el control de robots móviles. Una primera sección se dedica a la descripción de los prototipos software desarrollados para la evaluación y validación de las distintas características de la arquitectura. Y una segunda sección se centra en las pruebas realizadas con robots reales.

6.2. Sistema de Comunicaciones SC

El acceso a la información distribuida tiene que proporcionarlo el sistema de comunicaciones. Con este objetivo, se ha desarrollado el sistema de comunicaciones SC descrito en los capítulos cuatro y cinco.

El sistema SC implementado ( ) ofrece las siguientes características:

- Oculta los detalles de comunicación mediante la interfaz común FSA.

- Mantiene una representación interna de los datos mediante el uso de una estructura de objetos tipo pizarra distribuida. Dicha estructura se actualiza continuamente con los nuevos valores de los objetos e información temporal asociada. SC necesita la ejecución de un programa servicio en cada uno de los nodos del sistema. Estos servicios establecen las comunicaciones necesarias entre los nodos del sistema a través del bus de tiempo real no estricto para asegurar la consistencia de las copias de la pizarra de objetos distribuida.

- Todos los valores de los objetos de la pizarra distribuida están caracterizados con información temporal acerca de su antigüedad. De esta forma, todos los procesos o agentes, independientemente de su lenguaje de implementación y de su ubicación física, pueden obtener en cualquier instante y de forma actualizada la antigüedad de la información a la que acceden.

- Los procesos o agentes únicamente necesitan realizar accesos locales para contactar con todo el sistema, minimizándose de esta forma los retardos de comunicación. Por ejemplo, cuando un esquema de percepción necesita obtener el valor de un sensor, solamente tendrá que acceder al objeto asociado a dicho sensor, el cual se encuentra definido en el servicio SC local (que está ejecutándose en el mismo computador donde se encuentra el esquema de percepción).

- Sistema orientado a eventos. Implementa un sistema de notificación que permite asociar la ejecución de determinado código con eventos específicos (por ejemplo, un cambio en el valor de un objeto).

- Recepción ordenada de datos sin pérdida de información. El sistema SC asegura que todos los procesos interesados en los valores de un objeto de la pizarra reciben dichos valores en el mismo orden en el que se introdujeron en el sistema sin producirse la pérdida de ninguno de ellos.

- Permite la conexión y comunicación entre los diferentes componentes de la arquitectura distribuida, bien dentro de un mismo nivel o incluso entre niveles.

Actualmente, el sistema de comunicaciones SC desarrollado puede ejecutarse sobre plataformas basadas en tecnología NT de Microsoft. Los distintos componentes del sistema SC, descritos en el capítulo cinco, han sido implementados mediante los siguientes módulos *software* (fig.65 y 66):

- El módulo "núcleo del componente SC". Implementa la pizarra, recibe el nombre abreviado de "SCore", está desarrollado mediante el lenguaje de programación C++, y debe ejecutarse en cada computador que forme parte del sistema. Dichos computadores o nodos tienen que estar conectados a

través de una red Ethernet (bus de tiempo real no estricto) que permita a las instancias locales "SCore" la actualización de la información de la pizarra. Para ello, el sistema de comunicaciones SC emplea el protocolo TCP/IP.

- El módulo "interfaz SCore". Recibe el nombre de "SCoreDLL" y debe ser utilizado por cualquier componente software (procesos o agentes desarrollados en C++, código interpretado java o .NET, etc.) para conectar con el sistema SC y comunicarse con cualquier otro componente. "SCoreDLL" es una librería de acceso dinámico que proporciona a los procesos que enlazan con ella la interfaz FSA (marco-sensor-adaptador) (fig.65) necesaria para el acceso a la pizarra de objetos distribuidos del "núcleo del componente SC". Existen diferentes implementaciones de la librería que permiten la interoperatividad entre distintos componentes desarrollados mediante distintos lenguajes de programación. Actualmente existen versiones de la librería que enlazan con procesos C++, procesos java y procesos .NET, todos ellos utilizan el mismo sistema de comunicaciones SC (mismos objetos de la pizarra distribuida) para acceder a toda la información del sistema. Cualquier versión de la librería "SCoreDLL" proporciona a los agentes o procesos, según el modelo de interfaz FSA, el objeto "adaptador" asociado al sistema SC que se ha denominado "AdaptSCore". Las comunicaciones entre el módulo "interfaz SCore" y el módulo "núcleo del componente SC" se realizan de forma transparente a los procesos o agentes mediante el intercambio dinámico de datos o DDE (Dynamic Data Exchange).

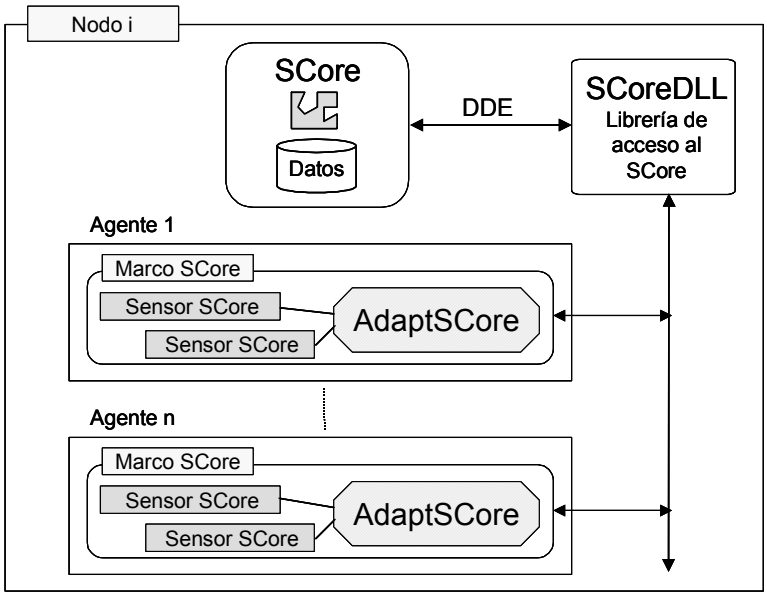


Figura 65: *Implementación del sistema de comunicaciones SC: núcleo del SC o SCore, e interfaz SCore o librería SCoreDLL que proporciona a los agentes el objeto adaptador AdaptSCore según el modelo de interfaz FSA*

- El módulo "Pasarela entre el bus de tiempo real estricto y "SCore". Proporciona el acceso a los sensores y actuadores de la arquitectura. No todos los computadores que forman parte del sistema tienen acceso al bus de tiempo real estricto, por lo que dicho módulo únicamente deberá ejecutarse en aquellos ordenadores que sí tengan el acceso. Sin embargo, el sistema "SC" distribuye toda la información a todos los computadores, por lo que al final todos los procesos o agentes tendrán acceso transparente al bus. Actualmente, hay desarrollada una versión de la pasarela que permite el acceso a un bus CAN de tiempo real, se ha denominado pasarela "SC-CAN" y realiza las conversiones necesarias entre los mensajes CAN y los objetos correspondientes de la pizarra distribuida [Posadas, 2000; Pérez, 2000]. La selección del bus CAN se debe a sus características de tiempo real estricto [Bosch, 1991; Tindell, 1994a y b].

El acceso al sistema SC también puede realizarse desde nodos no compatibles con la plataforma NT (denominados nodos heterogéneos), como pueden ser nodos con el sistema Unix o con el sistema AS-400. Para ello, se ha extendido el uso de las pasarelas mediante la implementación de un componente pasarela (fig.66) que comunica el sistema SC con cualquier aplicación externa mediante el establecimiento de conexiones TCP/IP estándar.

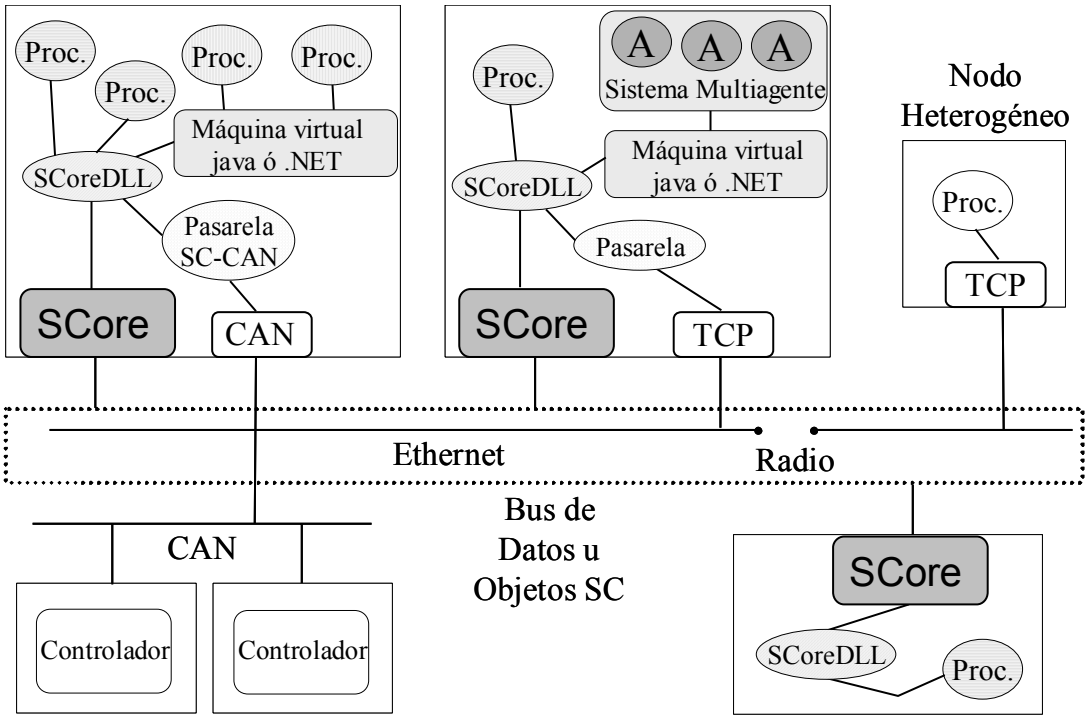


Figura 66: *Implementación del sistema de comunicaciones SC: empleo de pasarelas para el acceso al bus CAN de tiempo real estricto y la interoperatividad con nodos heterogéneos*

6.2.1. Aplicación del Sistema de Comunicaciones SC

El sistema de comunicaciones SC implementado ha sido testado en proyectos en el ámbito de los sistemas distribuidos y la robótica móvil. El sistema ha sido expuesto a condiciones extremas de testeo, comprobándose su conformidad y su alto rendimiento.

Se han realizado varios prototipos para validar su funcionamiento. Entre ellos destaca la modificación del sistema experto CLIPS de manera que los hechos pueden ser introducidos mediante el sistema SC. Esto permite la programación sencilla de sistemas distribuidos expertos. También existen implementaciones a nivel de INTERNET donde un servidor WEB conectado al sistema SC ofrece la información que se desee monitorizar. De este modo, mediante el acceso remoto desde un navegador, puede también interaccionarse con el sistema.

Con el sistema SC se ha conseguido abstraer de forma práctica el problema del desarrollo de aplicaciones distribuidas. En este sentido, el sistema SC se ha implantado industrialmente demostrando sus características de fácil adaptación y sencillez a la hora de configurarlo en un entorno de trabajo real. Entorno donde se requieren comunicar varias aplicaciones situadas en diferentes computadores garantizando, además, la robustez de las conexiones y del sistema.

En las siguientes secciones se describen, en primer lugar, dos aplicaciones del sistema SC a nivel industrial, para a continuación, exponer su aplicación en la implementación de la arquitectura de control propuesta en esta tesis.

6.2.1.1. Planta de Producción de Objetos Decorativos de Porcelana

El sistema SC se ha implantado en una planta de producción de objetos decorativos de porcelana perteneciente a la empresa internacional “Lladró” demostrando sus capacidades de prototipado rápido y reconfiguración en sistemas de adquisición de datos y razonamiento distribuido (fig.67) [Posadas, 1997].

En la planta existen diez puestos de trabajo donde los operarios manipulan los moldes de las figuras de porcelana. Existen seis puestos que pueden actuar de llenado o de vaciado y cuatro puestos que actúan de desmoldeo.

Los moldes tienen que pasar primero por un puesto de llenado, seguidamente pasar al almacén para formar la capa, después por un puesto de vaciado y finalmente por uno de desmoldeo. Además, los moldes deben secarse en un horno situado en la misma planta para aumentar su productividad.

La información correspondiente a cada una de las figuras de porcelana (número y tipo de moldes, tiempos de secado, etc.) se encuentra en la base de datos de un ordenador llamado central.

La planta dispone de un gran almacén robotizado donde se encuentran los moldes vacíos y donde los moldes llenos de pasta de porcelana esperan durante el proceso de formación de capa.

El sistema asigna trabajo llevando de forma robotizada los correspondientes moldes a los diferentes puestos y una vez realizado dicho trabajo los operarios ordenan la retirada de los moldes. Para ello, cada puesto dispone de un ordenador en el que se ejecuta una aplicación correspondiente con la tarea que se desempeña en dicho puesto. Todos los ordenadores se encuentran conectados a través de una red local con el ordenador central y todos ellos, incluido el central, tienen una instancia del Sistema de Comunicaciones SC.

El ordenador central está conectado, además, a un puesto de mando de los robots.

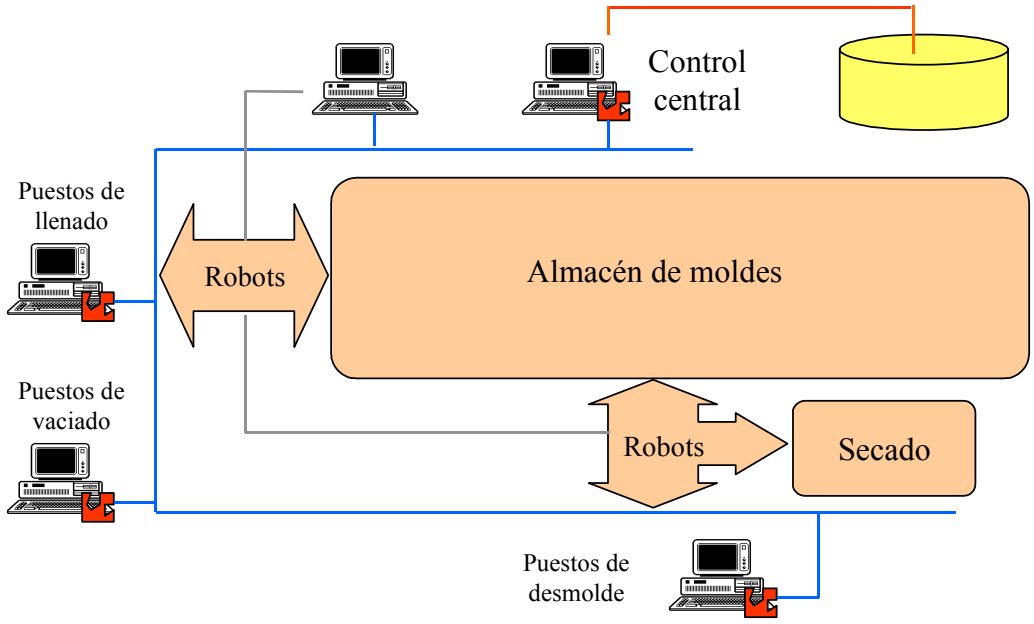


Figura 67: *Implantación del sistema SC en una planta de producción de objetos de porcelana*

El SC es el medio por el que fluyen los mensajes entre los robots, los puestos de los operarios y el ordenador central.

El SC se encarga de llevar los mensajes de los operarios al ordenador central para que así éste pueda ordenar nuevas tareas a los robots. De la misma manera el SC envía las respuestas a los operarios indicando en cada puesto la situación actual.

Las aplicaciones que se ejecutan en los diferentes puestos y en el ordenador central sólo tienen que conectarse, a través de la interfaz ofrecida por el sistema SC, con la instancia SC local y con los objetos correspondientes. El sistema SC es quien se encarga de las comunicaciones y de que toda la información esté actualizada en

cada uno de los puestos. Cualquier problema de fallo en las conexiones entre los ordenadores es resuelto por el sistema SC garantizando así la robustez del sistema.

6.2.1.2. Terminal Marítima de Contenedores

El sistema SC está actualmente funcionando en la Terminal Marítima de Contenedores "Príncipe Felipe" del Puerto de Valencia [Poza, 2002 y 2003].

Una Terminal Marítima de Contenedores es la conexión entre el transporte marítimo y terrestre de contenedores. Para llevar a cabo este proceso es necesario almacenar temporalmente los contenedores por medio de máquinas especializadas hasta que sean enviados a sus destinos (fig.68). Estas terminales normalmente son grandes y, debido al impacto medioambiental de los grandes puertos y sus grandes costes, es necesario optimizar la utilización de este espacio y los recursos implicados en la gestión de los contenedores. Con este objetivo, se ha desarrollado un sistema denominado GAMA (Gestión Automática Marítima) donde el sistema SC constituye la base para la integración y soporte de diferentes tecnologías en lo relativo a las comunicaciones, gestión de datos, control e interfaz humana.

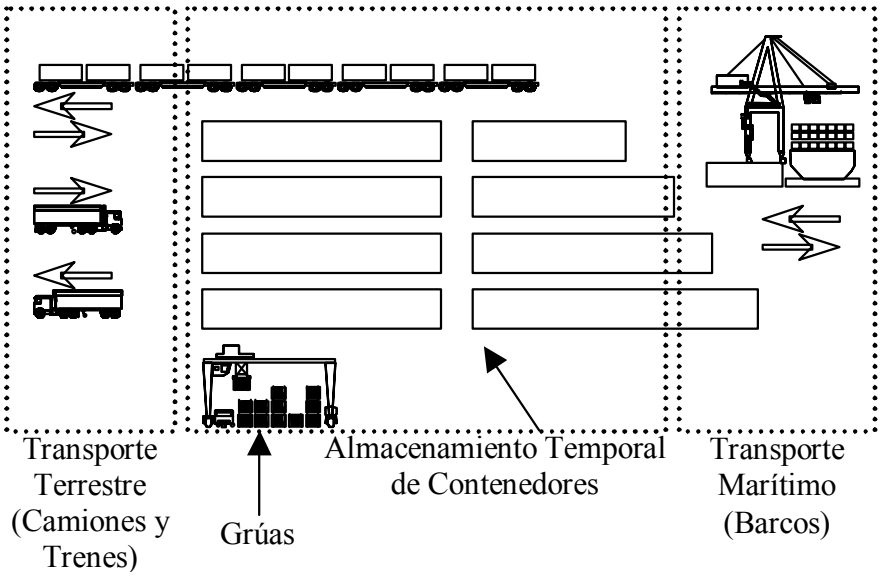


Figura 68: *Terminal Marítima de Contenedores*

Los contenedores son ampliamente empleados en el transporte de productos debido al servicio punto a punto que ofrecen. Además, por medio de grúas especializadas, las operaciones sobre el transporte marítimo y terrestre pueden ser realizadas rápidamente. Con el fin de optimizar los costes de las operaciones, toda la gestión de la terminal de contenedores está automatizada. El sistema de comunicaciones SC ha permitido la integración en el sistema de componentes heterogéneos inteligentes. Por medio de este sistema, "agentes de operación" pueden ser desarrollados a un alto nivel y de forma transparente.

Para el correcto funcionamiento de la terminal desde el punto de vista de la calidad de servicio y la optimización de recursos las aplicaciones propuestas deben tener funciones muy especializadas (fig.69):

- Gestor de planta: aplicación que determina la distribución óptima de los contenedores en la terminal en base a unos criterios concretos o políticas de ubicación.

- Gestor de grúas: aplicación que optimiza los movimientos de las máquinas a lo largo de la terminal en las operaciones con los contenedores.

- Visor de planta: aplicaciones que monitorizan en tiempo real el estado de la planta y de las máquinas implicadas en el trabajo con los contenedores.

- Sistema de comunicaciones: orientado a datos y a eventos y compuesto por el sistema SC junto a un conjunto de pasarelas que permiten interconectar diferentes plataformas heterogéneas.

Las aplicaciones dentro del sistema están organizadas de manera distribuida, de forma que cada proceso puede funcionar en cualquier lugar de la red con tan sólo conectarse al sistema SC (fig.69).

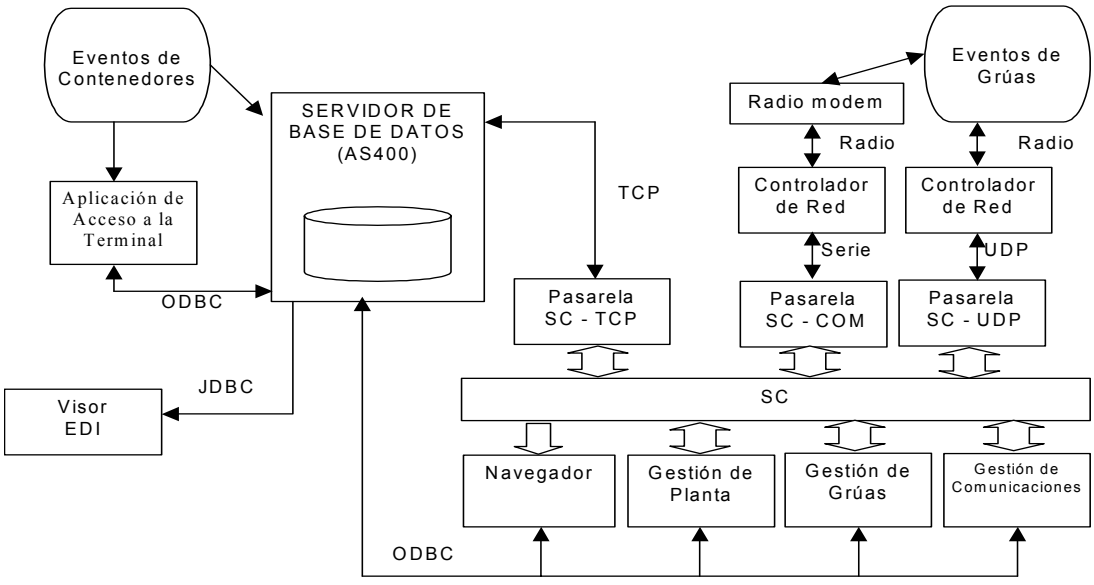


Figura 69: *Implantación del Sistema SC en la Terminal Marítima "Príncipe Felipe" de Valencia*

El sistema SC permite, dentro de la estructura de aplicaciones distribuida, comunicar los eventos producidos por los cambios habituales de contenedores y máquinas, y el acceso a los datos sobre el estado de la terminal necesarios para la toma de decisiones en la distribución de contenedores y en la gestión de las máquinas.

Cuando se genera un evento físico en la terminal de contenedores, como puede ser la llegada de un nuevo contenedor o el movimiento de una grúa, se genera un evento lógico que desencadena una serie de mensajes. Estos mensajes viajan a través del sistema SC para llegar finalmente a una o más aplicaciones que los procesan; por ejemplo, el gestor de contenedores que deberá decidir la ubicación del nuevo contenedor o el visor de planta que deberá visualizar el cambio de posición de una grúa. El soporte a esta comunicación orientada a eventos lo proporciona el canal SC y las pasarelas correspondientes.

El sistema también puede simular y testear diferentes políticas de distribución de contenedores y de gestión de grúas. Esta disposición permite comprobar la respuesta del sistema en el funcionamiento real y la respuesta del sistema ante variaciones importantes del entorno.

La propia arquitectura de aplicaciones y de comunicaciones, hace que el sistema sea adaptable, escalable y flexible a cambios del entorno industrial o del entorno de funcionamiento.

6.2.1.3. Arquitectura de Control para Robots Móviles

El sistema SC constituye la base para el diseño y especificación de la arquitectura híbrida de control de robots móviles propuesta en esta tesis.

Con este objetivo, el sistema SC se ha utilizado en el desarrollo de un prototipo software que implementa y valida la arquitectura propuesta así como en el control de robots físicos reales también basados en dicha arquitectura.

En las secciones siguientes se describen con detalle las implementaciones realizadas de la arquitectura multinivel y distribuida propuesta donde el sistema SC constituye el sistema de comunicaciones que permite la interacción entre los niveles reactivo y deliberativo para el control de robots móviles.

6.3. Prototipo Desarrollado

En esta sección se describe un prototipo software [Posadas, 2003] basado en agentes cuyos objetivos son:

- Simular robots móviles en un entorno específico.
- Implementar la arquitectura de control propuesta.
- Validar las técnicas de delegación de código.
- Validar la caracterización temporal de los datos

- Validar el control de robots mediante el uso de agentes móviles que pueden decidir su ubicación en función de la antigüedad de los datos que reciben.

- Facilitar la aplicación de diferentes técnicas de control y finalmente reemplazar los robots y entorno simulados por robots reales en un entorno real.

El prototipo permite la simulación de robots moviéndose en un entorno mediante la composición de comportamientos básicos. Los comportamientos emergentes de los robots simulados resultan de la composición de los esquemas motores realizada mediante la suma de sus contribuciones.

El prototipo presenta las siguientes restricciones:

- Sigue el modelo más puro de comportamientos emergentes [Simó, 1997b], considerando todos los coeficientes de motivación constantes e iguales a 1.

- Los esquemas motores tienen que ser compatibles para que no se realicen composiciones conceptualmente absurdas.

Ha sido implementado en el lenguaje de programación Java y utiliza el modelo FSA para acceder al sistema de comunicaciones SC implementado en C++.

Su estructura está formada principalmente por los siguientes componentes *software* (fig.70): "entorno simulado", "robot simulado", "nivel deliberativo", "nivel reactivo" y "agentes reactivos móviles". Los componentes de los niveles deliberativo y reactivo acceden a los valores de los sensores a través del sistema de comunicaciones SC. El sistema SC proporciona modularidad e independencia a los componentes del prototipo, los cuales interaccionan a través de los objetos de la pizarra distribuida.

Para la implementación de los componentes simulados se ha utilizado el sistema de invocación de métodos remotos RMI proporcionado por el lenguaje de programación Java.

Se han realizado varias versiones del prototipo orientadas a implementar distintas características de la arquitectura SC-Agent. En la primera versión, se valida la técnica de control basada en la delegación de código, para ello, los robots reciben por delegación el código correspondiente a los comportamientos que tienen que ejecutar. Así, desde una consola principal, pueden enviarse nuevos comportamientos a los robots en funcionamiento para que sean considerados en el proceso de composición realizado. En la segunda versión, el objetivo es dotar a los agentes reactivos de movilidad y permitir que se muevan entre los nodos en función de la antigüedad de los datos requeridos que reciban. En esta segunda versión, también se valida la necesidad de caracterizar temporalmente a los datos para poder determinar su utilidad en los procesos de fusión.

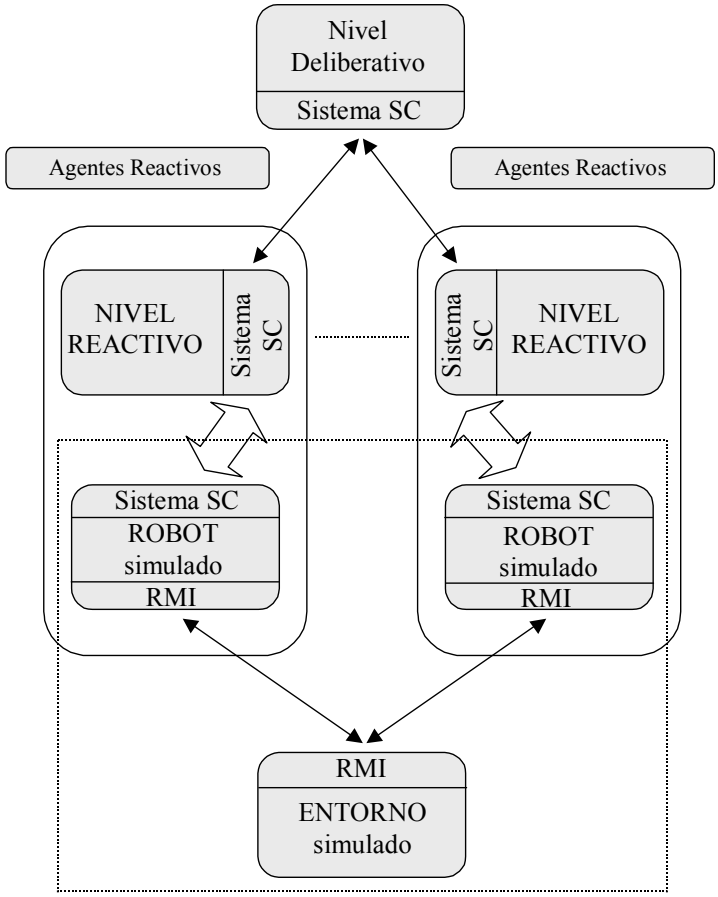


Figura 70: *Componentes software del prototipo desarrollado*

El funcionamiento general del prototipo es el siguiente: el nivel deliberativo, en función del objetivo a alcanzar, envía al nivel reactivo los componentes *software* necesarios: esquemas de percepción, esquemas motores y compositores. El nivel reactivo ejecuta dichos componentes que determinan los movimientos del robot. El resultado de la ejecución de un esquema motor (su contribución) es el valor correspondiente a la velocidad a aplicar a los motores de las ruedas izquierda y derecha del robot. Los esquemas motores utilizan, para el cálculo de su contribución, los valores (posición estimada del robot, detección de obstáculo) correspondientes a los resultados de la ejecución de los esquemas de percepción. Los procesos compositores se encargan de realizar la composición (suma ponderada) de las distintas contribuciones de los esquemas motor obteniendo los valores de velocidad definitivos. La frecuencia de ejecución de todos estos procesos determina la máxima velocidad del robot, a mayor frecuencia de ejecución mayor velocidad del robot.

A continuación se describen los distintos componentes del prototipo. Finalmente, se presentan los resultados obtenidos en las simulaciones realizadas para cada una de las versiones.

6.3.1. Componente Robot Simulado

Los objetivos de este componente son:

- Simular el funcionamiento de un robot con unas determinadas características.
- Proporcionar al resto de componentes la misma información que se obtendría del robot real equivalente.
- Poder realizar el control del robot simulado de la misma forma que se haría con el robot real. Para los agentes que realizan el control no debería existir ninguna diferencia.
- El objetivo último es, una vez validado el sistema de control, sustituir el robot simulado por el equivalente real manteniendo los agentes de control.

El componente *software* denominado "robot" simula el funcionamiento de un robot con las siguientes características:

- Robot circular de dimensiones configurables con dos ruedas con sendos motores controlados de forma independiente.
- Contador de pulsos para cada rueda que permite determinar la posición del robot suponiendo ausencia de rozamientos.
- Ocho sensores de infrarrojos (seis en la parte delantera y dos en la trasera) con rango de alcance configurable.

Los robots simulados son una versión *software* del minirobot *Khepera*.

Pueden existir varios robots funcionando al mismo tiempo, es decir, varias instancias del componente "robot". Cada una de ellas puede simularse en el mismo o en diferentes computadores. Independientemente de ello, todos los robots operan sobre un mismo escenario definido por el componente *software* "entorno".

El control de cada robot simulado se realiza a través de la ejecución de los diferentes componentes reactivos (esquemas de percepción, esquemas motores y compositores) que almacenan sus resultados en los objetos correspondientes de la pizarra distribuida del sistema SC. En un sistema real, el componente pasarela obtendría los valores del SC y los enviaría a través del bus de tiempo real a los controladores de los actuadores del robot. En el caso del prototipo, al no existir el bus de tiempo real no se utiliza la pasarela, sino que es el propio robot simulado quien accede al sistema SC para obtener los valores de velocidad que debe aplicar a las motores controladores de las ruedas. De esta forma, si se deseara sustituir el robot simulado por el robot real de las mismas características, solamente habría que incorporar al sistema el componente pasarela adecuado entre el sistema SC y el bus

de tiempo real empleado. El control del robot realizado por los agentes correspondientes a los esquemas de percepción, esquemas motores y compositores podría ser el mismo.

La funciones que realiza cada instancia del componente "robot" son:

- Recibe a través del sistema SC los valores para la velocidad de los motores.
- Actualiza periódicamente el valor de los contadores de pulsos de acuerdo con la velocidad de los motores.
- Proporciona al resto de componentes los valores de los contadores de pulsos enviándolos al sistema SC.
- Proporciona al resto de componentes los valores de los sensores de infrarrojos enviándolos al sistema SC.

El "robot" calcula los valores de los sensores de infrarrojos en base a la ecuación característica del sensor. Para dicho cálculo necesita las distancias de los sensores a los obstáculos más próximos.

Para obtener las distancias de los sensores a los obstáculos más próximos, el "robot" interactúa directamente con el componente *software* "entorno". Éste le proporciona, dependiendo de la posición del "robot", dichas distancias. El "entorno" puede estar simulándose en un computador diferente al de la simulación del "robot". Por ello, el acceso siempre se realiza por invocación remota de un método concreto del "entorno".

El "robot" también actualiza constantemente su posición dentro del "entorno" simulado, aunque únicamente con la finalidad de acceder al lugar concreto donde se encuentra para solicitar las distancias que le separan de los obstáculos. Es importante destacar que el "robot" simulado no proporciona su posición al resto de componentes, al igual que no lo haría el robot real, sino que tendrá que existir un esquema de percepción adecuado que realice los cálculos necesarios para estimar dicha posición a partir del valor de los contadores de pulsos.

6.3.2. Componente Entorno Simulado

El componente *software* denominado "entorno" simula el escenario común donde operan los robots. Es decir, realiza una representación del mundo real y permite que los robots simulados accedan a ella para determinar los valores de sus sensores.

El entorno simulado se basa en la especificación de las posiciones de un conjunto de obstáculos con formas rectangulares y circulares. Dicha especificación

se realiza a través de una interfaz gráfica (fig.71) con el usuario y puede ser almacenada en un fichero de texto.

Únicamente existe una instancia del componente "entorno" en todo el sistema. La instancia puede ejecutarse en cualquier computador y todos los "robots" simulados accederán a ella para obtener las distancias de sus sensores a los obstáculos.

Las función que proporciona la única instancia del componente "entorno" consiste en proporcionar a los robots las distancias de sus sensores a los obstáculos más próximos. Para ello dispone de un método que los robots invocan de forma remota. Dicho método debe invocarse indicándose la posición del robot en el entorno, tras lo cual devuelve las distancias correspondientes.

6.3.3. Componente Nivel Deliberativo

El componente *software* denominado "nivel deliberativo" se encarga de la selección de los componentes reactivos que tienen que ejecutarse en el robot para la consecución de los objetivos asignados por el usuario. Está formado por los siguientes agentes:

- Agente receptor y ejecutor de agentes (REA). Objeto activo que tiene que ejecutarse, al igual que el objeto activo SCore del sistema SC, en cada uno de los nodos del sistema distribuido. Se encarga de la recepción y ejecución del código correspondiente a los agentes reactivos.

- Consola central. Objeto activo que ofrece una interfaz al usuario en forma de consola (fig.71) que le permite tanto la asignación de los objetivos para cada robot (a modo de destinos a alcanzar) como la monitorización de los movimientos efectuados por los mismos. En dicha consola, el usuario también puede observar los obstáculos detectados por los robots.

Puede haber varias instancias de la consola central en diferentes computadores. Cada consola accede al sistema SC para obtener la información que necesita para su funcionalidad. Las funciones que proporciona cada instancia de la consola son:

- Permite al usuario la selección de los destinos que tienen que alcanzar los robots.

- Permite la ejecución de un agente deliberativo que, en función de los objetivos asignados por el usuario, selecciona los agentes reactivos (esquemas de percepción, esquemas motores y compositores) que tienen que ejecutarse en cada uno de los robots y los lanza a ejecución mediante su envío al sistema SC. Los objetos REA de los nodos del sistema serán los encargados de la recepción de dichos agentes y de iniciar la ejecución de los mismos.

- Visualiza en pantalla los robots simulados situándolos en las posiciones que ocupan dentro del entorno. Para ello, mediante el acceso al sistema SC, la consola obtiene las posiciones de los robots estimadas por los esquemas de percepción correspondientes.

- Visualiza en pantalla los obstáculos detectados por los robots situándolos en las posiciones que ocupan dentro del entorno. Para ello, mediante el acceso al sistema SC, la consola obtiene las posiciones estimadas de los obstáculos detectados por los esquemas de percepción correspondientes.

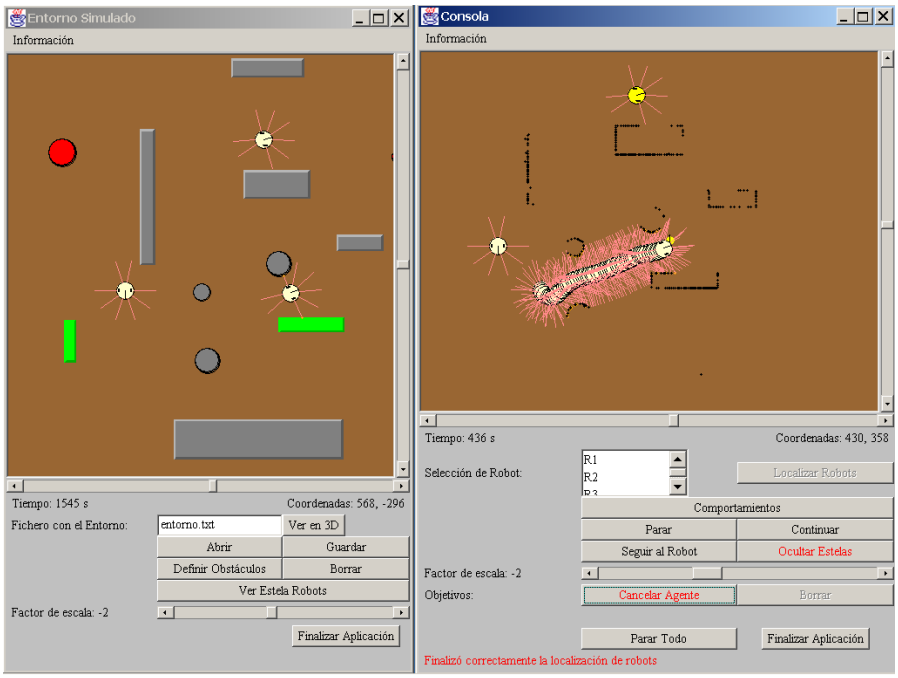


Figura 71: *Prototipo desarrollado: entorno y robots simulados, y consola del nivel deliberativo*

6.3.4. Componente Nivel Reactivo

El componente nivel reactivo lo constituyen los distintos tipos de agentes reactivos que pueden ser ejecutados por el nivel deliberativo a través del sistema SC. En concreto, los diferentes tipos de agentes reactivos son: agentes esquemas de percepción, agentes esquemas motores y agentes compositores.

La originalidad del prototipo desarrollado consiste en que el código correspondiente a cada agente reactivo es instanciado en tiempo de ejecución. Es decir, el sistema distribuido puede no disponer del código de los agentes en tiempo de compilación, sino que, una vez en funcionamiento, recibe dicho código para ser ejecutado cuantas veces sea necesario. De esta forma, mientras se está realizando la simulación de un robot, pueden implementarse y enviarse dinámicamente nuevos agentes reactivos que serán incluidos y considerados en el proceso de composición.

Además, en la segunda versión del prototipo, los agentes pueden, por iniciativa propia, moverse a otro nodo con el objetivo de obtener de forma más rápida (con menor antigüedad) los datos que necesitan.

De forma predefinida, se han desarrollado los siguientes agentes por considerarse básicos en cualquier control de robots: agente esquema de percepción de posición, agente esquema de percepción detector de obstáculos, agente esquema motor para evitar obstáculos, agente esquema motor para ir a destino, agente esquema motor explorador y agente compositor por suma. A continuación se explica el funcionamiento de cada uno de estos agentes, los cuales interaccionan a través del sistema SC para obtener y proporcionar la información requerida en cada caso.

6.3.4.1. Agente Esquema de Percepción de Posición

Accede al sistema SC para obtener los valores de los contadores de pulsos. En función de ellos, estima la posición actual del robot.

Es importante destacar que, cuanto menores sean los desplazamientos del robot entre dos muestras consecutivas de los valores de los pulsos, mejores serán las estimaciones efectuadas. De esta forma, una frecuencia de muestreo pequeña obliga al robot a moverse lentamente.

La información de la posición estimada del robot se envía al sistema SC para que esté disponible al resto de componentes.

6.3.4.2. Agente Esquema de Percepción Detector de Obstáculos

Accede al sistema SC para obtener los valores actuales de los sensores y la posición estimada del robot. En función de dicha información, determina si hay o no un obstáculo en el entorno y estima su posición en caso de haberlo. Para estimar la posición de un obstáculo se utiliza la ecuación característica de los sensores obteniendo la distancia a dicho obstáculo a partir de la posición del sensor. Esta posición se obtiene acorde a la posición estimada del robot y a las características de ubicación de los sensores en el mismo.

Cuando se detecta un obstáculo se envía la información al sistema SC. De esta forma, cualquier componente interesado en dicha información podrá recibirla automáticamente.

Existen varias versiones de este agente en base a la utilización o no de la antigüedad de los valores recibidos para determinar la ubicación de los obstáculos detectados. Con los valores de los sensores de infrarrojos se puede conocer la existencia de un obstáculo, pero para decidir su ubicación en el entorno, es necesario saber la posición donde se encontraba el robot en el instante en que se obtuvieron dichos valores. En este sentido, el uso de la antigüedad de los datos, proporcionada por el sistema SC, será fundamental en el cálculo de la ubicación de los obstáculos y

así poder generar mapas del entorno. En secciones posteriores se describen los resultados obtenidos.

6.3.4.3. Agente Esquema Motor para Evitar Obstáculos

Accede al sistema SC para obtener los valores de los sensores. En función de esta información, calcula la velocidad a aplicar a los motores de control de las ruedas del robot que evite la colisión con los posibles obstáculos que hubiera alrededor.

Este agente se basa en el algoritmo de Braitenberg [Braitenberg, 1984], donde la actuación sobre los motores es una suma ponderada de las lecturas de los sensores. El agente utiliza una matriz predefinida acorde a las características del robot con los coeficientes de Braitenberg a aplicar en la suma ponderada y así calcular el valor de las velocidades.

El resultado de las velocidades obtenidas se envía al sistema SC para que el agente compositor pueda utilizarlas en los cálculos de las velocidades finales a aplicar.

6.3.4.4. Agente Esquema Motor para Ir a Destino

Accede al sistema SC para obtener la posición actual estimada del robot. En función de esta información y acorde con la posición destino a alcanzar, calcula la velocidad que habría que aplicarse a los motores de control de las ruedas del robot.

Este agente determina las velocidades para cada rueda necesarias para moverse hacia unas coordenadas concretas del entorno.

El resultado de las velocidades obtenidas se envía al sistema SC para que el agente compositor pueda utilizarlas en los cálculos de las velocidades finales a aplicar.

6.3.4.5. Agente Esquema Motor Explorador

El objetivo de este agente es mover al robot por todo el entorno para permitir una exploración del mismo, pudiéndose obtener así una visión del entorno basada en los obstáculos detectados.

Su funcionamiento es equivalente al "agente esquema motor para ir a destino" pero en este caso, cuando el robot alcanza un destino, selecciona automáticamente uno nuevo.

6.3.4.6. Agente Compositor por Suma

Accede al sistema SC para obtener los distintos valores de velocidad (contribuciones) proporcionados por los diferentes agentes motores. Utilizando esta

información, determina los movimientos del robot como resultado de un proceso de composición.

La composición se realiza mediante el cálculo de la media aritmética (al ser el valor de todas las motivaciones igual a uno) de los valores proporcionados por cada uno de los esquemas motores del robot. Obteniéndose, como resultado, los valores definitivos para las velocidades de las ruedas. De esta forma, cada uno de los esquemas motores del robot proporciona una contribución al comportamiento emergente.

Este agente envía al sistema SC las velocidades a aplicar en cada instante. El robot simulado accede a esta información y en base a ella actualiza periódicamente los valores de los contadores de pulsos.

Es importante destacar que los valores de las velocidades aplicadas deben ser tales que aseguren que, entre dos actualizaciones consecutivas de dichos valores, el robot no va a desplazarse una distancia superior a la que se encuentra el obstáculo más próximo detectado o, en el caso de no haber obstáculos, una distancia superior al alcance de los sensores de infrarrojos. De esta manera se asegura que el robot no colisionará con ningún obstáculo, dando tiempo a detectarlos, rectificar las velocidades y esquivarlos. Cuanto mayor sea la frecuencia de actualización mayor será la velocidad a la que se podrá mover el robot y, lógicamente, una frecuencia de actualización pequeña obligará al robot a desplazarse muy lentamente.

Por lo tanto, los distintos periodos de ejecución de los agentes (periodos de actualización de la información en el sistema SC) determinarán la máxima velocidad a la que podrá moverse el robot.

6.3.5. Pruebas Realizadas y Resultados

Con la primera versión del prototipo, se han realizado varias pruebas para validar la delegación del código de los esquemas de percepción y esquemas motores y comprobar los resultados de la composición de los mismos.

En la figura 72 se presenta el resultado de enviar a un robot únicamente un esquema motor para seguir un objetivo. Se observa que el robot se mueve hacia el objetivo de forma directa y atravesando todos los obstáculos sin esquivarlos.

Por otro lado, en la figura 73, se presenta el resultado de la composición de las contribuciones de dos esquemas motores enviados al robot: uno para seguir un objetivo y otro para evitar obstáculos. Se observa que el robot se dirige hacia su objetivo evitando los obstáculos que encuentra en su trayectoria. La composición de los esquemas motores funciona correctamente obteniéndose el comportamiento emergente requerido.

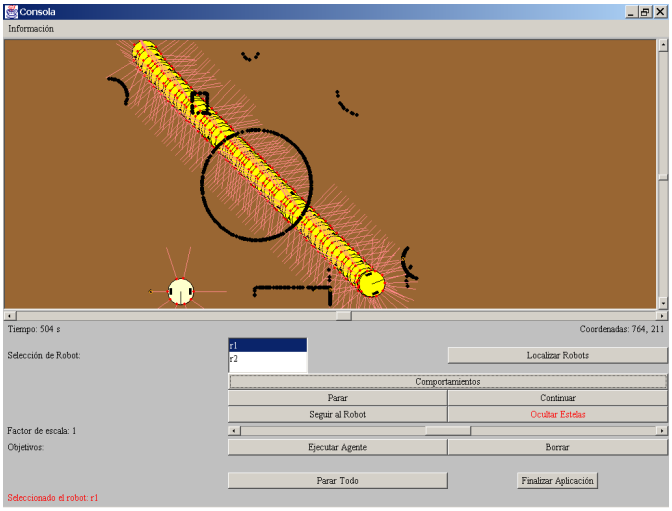


Figura 72: *Resultado sólo esquema motor para ir a destino*

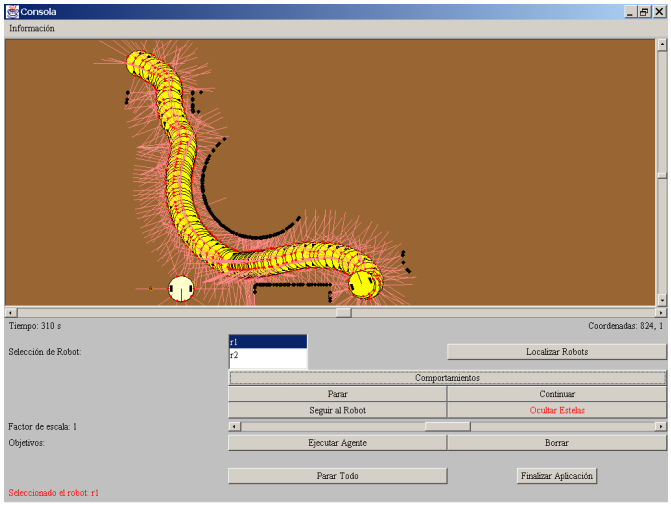


Figura 73: *Resultado composición esquemas motores para evitar obstáculos e ir a destino*

Con la segunda versión del prototipo, los objetivos han sido: comprobar la necesidad de la caracterización temporal de los datos propuesta en la arquitectura SC-Agent, y permitir la movilidad de los agentes por iniciativa propia.

En cuanto al primer objetivo, las pruebas han consistido en comparar los resultados obtenidos para la ubicación en el entorno de los obstáculos detectados por distintas versiones del "agente de percepción detector de obstáculos". Para ello, la consola realiza una representación visual del escenario captado por dicho agente. A partir de esta información, el nivel deliberativo podría construir un mapa interno simbólico del entorno necesario para la planificación óptima de las tareas y movimientos a realizar.

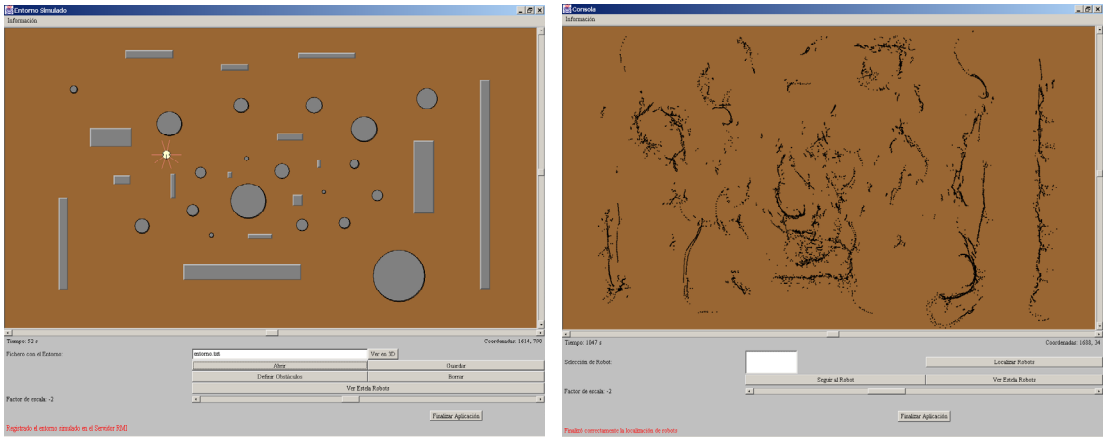
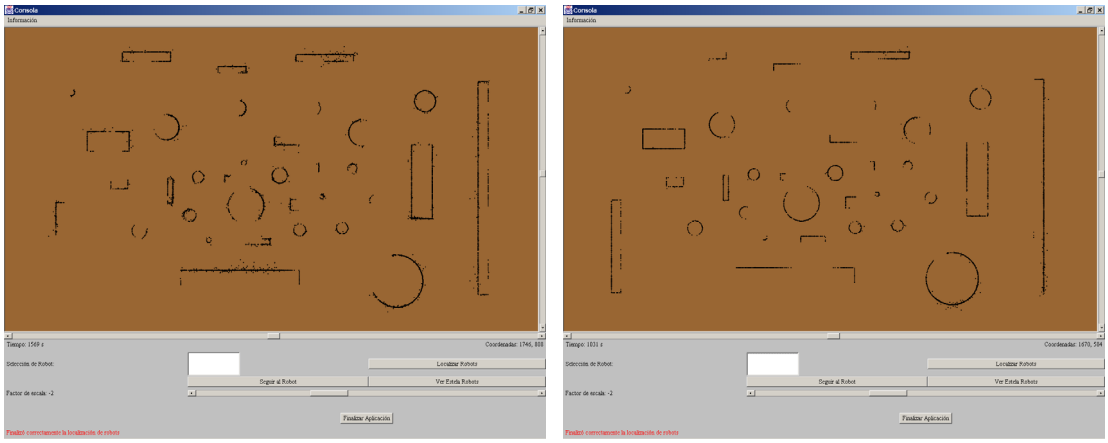


Figura 74: *Entorno real recorrido por el robot y mapa de obstáculos detectados y ubicados sin considerar la antigüedad de los datos empleados en el cálculo*



Precisión: 50 milisegundos Precisión: 25 milisegundos

Figura 75: *Mapas de obstáculos obtenidos al considerar la antigüedad de los datos y utilizando precisiones de 50 y 25 milisegundos*

La primera versión del "agente detector de obstáculos" no utiliza la antigüedad de los datos para determinar la ubicación de los obstáculos detectados. De esta forma, cuando los valores de los infrarrojos indican la existencia de un obstáculo, el agente detector asigna la ubicación del mismo en la última posición del robot estimada que recibió. El emparejamiento de los datos, realizado de esta forma, debe suponer que tanto los valores de los sensores de infrarrojos, como los valores de los contadores de pulsos, como la información procedente de otros agentes, se reciben al mismo tiempo y con una periodicidad idéntica. Sin embargo, en un sistema real, esto no es así, pues los distintos datos se generan en lugares diferentes y son procesados por módulos distintos y además independientes.

En la figura 74, se presenta el resultado obtenido al realizar el robot la exploración de un determinado entorno. Puede observarse que el mapa visualizado en la consola, correspondiente a la ubicación de los obstáculos detectados que realiza el "agente detector de obstáculos", no se parece al entorno real recorrido por el robot.

Para solucionar el problema, la solución consiste en considerar en el emparejamiento de los datos su antigüedad. Así, únicamente podrán emparejarse los valores de los sensores de infrarrojos con valores de posición estimada del robot que tengan una antigüedad similar. Lógicamente, habrá valores que habrá que rechazar por no poderse emparejar. El número de parejas, es decir, de obstáculos detectados, y la exactitud del mapa que se obtenga, dependen de los periodos de ejecución de los agentes (que determinan la periodicidad con la que se reciben los distintos datos), de los retardos empleados en la transmisión de la información a sus destinos, y de la precisión del emparejamiento, es decir, de la máxima diferencia de tiempos que se permite para validar una pareja.

En la figura 75, se observan los resultados obtenidos con una segunda versión del "agente detector de obstáculos" que sí tiene en cuenta la antigüedad de la información que recibe para determinar la ubicación de los obstáculos. Se presentan los mapas obtenidos con precisiones de 50 y 25 milisegundos. Cuanto mayor es la precisión, es decir, cuanto menor es la máxima diferencia de tiempos que se permite para validar una pareja, los obstáculos se detectan con mayor dificultad pero la información generada es más exacta.

En cuanto al objetivo de permitir la movilidad de los agentes por iniciativa propia, se han realizado pruebas donde el agente, cuando recibe los datos que requiere con una antigüedad superior a la máxima especificada en sus requerimientos, se mueve por cuenta propia al lugar de donde proceden los datos que necesita.

En concreto, se ha situado al agente móvil correspondiente al "esquema motor de evitar obstáculos", basado en el algoritmo de Braitenberg, en un nodo diferente al nodo donde se generan los valores de los sensores de infrarrojos. La ejecución del "esquema motor de evitar obstáculos" es fundamental en el sistema para que el robot no choque con los obstáculos, por ello, cuando este agente detecta que la antigüedad de los valores de infrarrojos que recibe es superior a la máxima permitida, decide moverse al lugar de procedencia de los mismos para así obtenerlos con menor retardo y poder cumplir sus requerimientos.

El movimiento del agente se realiza a través del sistema SC y del agente REA que se encarga de recibirlo y lanzarlo a ejecución en el nodo destino. Durante el instante de movimiento del agente, el robot no debe chocar con ningún obstáculo. Por ello, si dicho movimiento durara más que el periodo de ejecución del agente, el robot sería detenido por el "agente compositor" tras detectar la falta de envío de información por parte del "agente de evitar obstáculos".

Una vez reestablecido el agente en su nuevo destino, éste continúa enviando la información necesaria para evitar obstáculos. En caso de haberse tenido que detener el robot, el "agente compositor" reanuda el envío de velocidades.

Para forzar que el "agente motor de evitar obstáculos" reciba los valores de infrarrojos con más antigüedad de la máxima permitida, puede forzarse el retardo de los mismos por la red para llegar al nodo donde se encuentra el agente, o bien pueden saturarse los recursos del Sistema Operativo de dicho nodo de modo que el retardo se produzca en los periodos de ejecución de los procesos y por lo tanto, a la hora de utilizar la información recibida, ya haya transcurrido un tiempo excesivo.

En la tabla 7 se especifican los periodos de ejecución de los agentes empleados en las pruebas descritas.

NODO 1	
Componente o Agente	Periodo
Actualización del robot simulado	50 milisegundos
Envío de los valores de los sensores de infrarrojos	150 milisegundos
Envío de los pulsos de los contadores	50 milisegundos
Esquema Percepción Posición Estimada	50 milisegundos
Esquema Motor Explorador	100 milisegundos
Compositor Velocidad	50 milisegundos

NODO 2	
Componente o Agente	Periodo
Esquema Percepción Detectar Obstáculos	200 milisegundos
Esquema Motor Braitenberg para evitar obstáculos, inicialmente se sitúa en el nodo 2 pero cuando la antigüedad de los datos supera los 50 milisegundos, entonces se mueve por iniciativa propia al nodo 1	50 milisegundos
Consola	Sin periodo, por eventos

Tabla 7: *Periodos de ejecución de los agentes empleados en las pruebas descritas*

Con el prototipo implementado y pruebas realizadas se han validado las técnicas de delegación de código, composición de comportamientos, empleo de la información temporal de los datos y el movimiento de los agentes. Los resultados obtenidos son satisfactorios y constituyen la base para poder desarrollar y aplicar la arquitectura SC-Agent en sistemas reales.

6.4. Robot YAIR

La arquitectura SC-Agent también se ha implementado sobre un prototipo de robot móvil [Simó, 1997a y c] denominado YAIR (*Yet Another Intelligent Robot*) (fig.76).

Figura 76: *Robot YAIR*

El robot YAIR dispone principalmente de tres nodos transductores y de un nodo denominado principal con capacidad de procesamiento multitarea (fig.77).

El nodo principal dispone del sistema de comunicaciones SC y es el encargado de la ejecución de los distintos agentes reactivos de la arquitectura. Estos agentes tienen que acceder a la información que proporcionan los nodos transductores y tienen que enviar a los mismos las acciones a realizar. Para ello, estos nodos y agentes, que constituyen el nivel reactivo de la arquitectura, tienen que disponer de una computación y comunicación en tiempo real, lo cual se proporciona mediante la conexión de dichos nodos a través de una red CAN. Este bus de campo, que fue desarrollado para la industria del automóvil, se utiliza actualmente en numerosas áreas tecnológicas, entre ellas, la robótica móvil, gracias a su fiabilidad y versatilidad. Su mecanismo de acceso al medio, su capacidad multimaestro y su cobertura en la detección de errores lo hacen idóneo para su utilización en sistemas

distribuidos de tiempo real. Los nodos reactivos utilizan este medio para la distribución de la información sensorial. En el nodo principal, es el sistema SC el encargado de realizar el mapeado entre los mensajes CAN y los objetos asociados de la pizarra de forma que el acceso a dicho bus por parte de los agentes reactivos sea transparente [Pérez, 2000].

La conexión del nivel deliberativo con el nivel reactivo se realiza a través del sistema de comunicaciones SC.

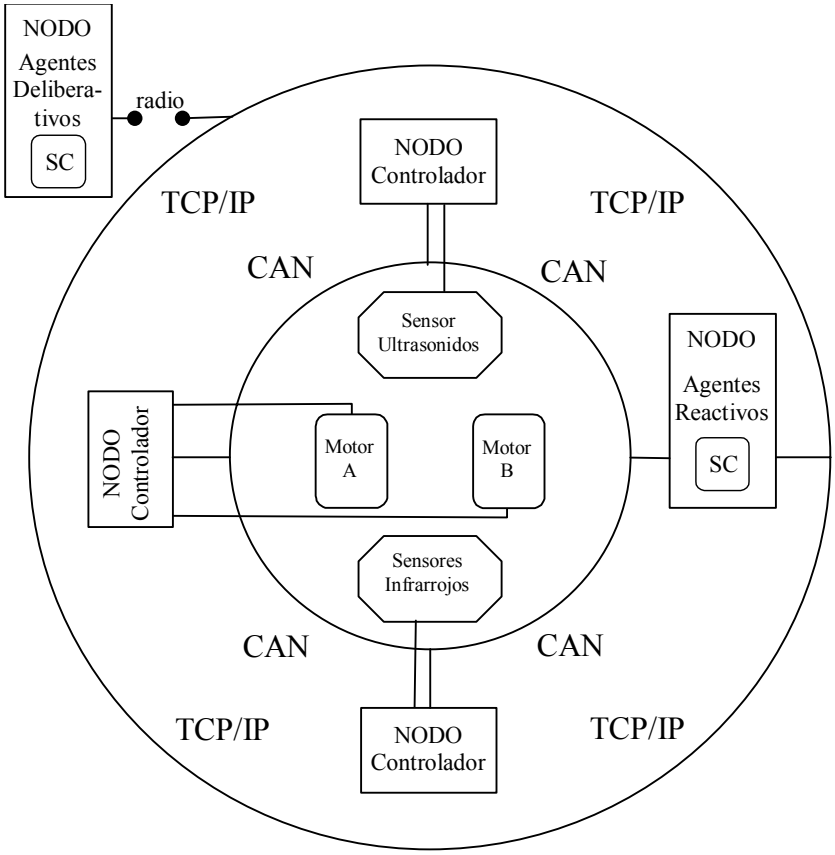


Figura 77: *Conexión de los nodos del robot YAIR*

Si el robot YAIR dispusiera de otro nodo con capacidad de procesamiento multitarea (fig.78), conectado al nodo principal a través de una red Ethernet, podría encargarse de la ejecución de los agentes deliberativos. Estos agentes, a través del sistema SC también instanciado en dicho nodo, accederían a toda la información de la arquitectura y enviarían al nodo principal los agentes reactivos necesarios. El nivel deliberativo, aún no teniendo restricciones temporales duras, podría asegurar un tiempo de respuesta promedio bueno. Sin embargo, al no disponerse de otro nodo local al robot YAIR, este procesamiento tiene que realizarse también en el nodo principal o bien de forma remota (fig.77). Para ello, el nodo principal permite la conexión, a través del sistema SC, con otros nodos externos al robot mediante una red radio ethernet. En estos nodos remotos pueden ejecutarse los procesos o agentes deliberativos con restricciones temporales no estrictas, liberando de esta forma la

carga de procesamiento en el nodo principal disponible en YAIR. En este sentido, se han realizado varias pruebas con el objetivo de validar el sistema de comunicaciones y arquitectura implementada. Dichas pruebas y resultados se exponen en la sección siguiente.

En la figura 78 se presenta el diseño general de la implementación de los niveles de la arquitectura propuesta sobre el robot YAIR independientemente del número de sensores, actuadores y nodos disponibles. La arquitectura propuesta permite la rápida incorporación de nuevos nodos en el sistema distribuido mediante el empleo del sistema de comunicaciones SC.

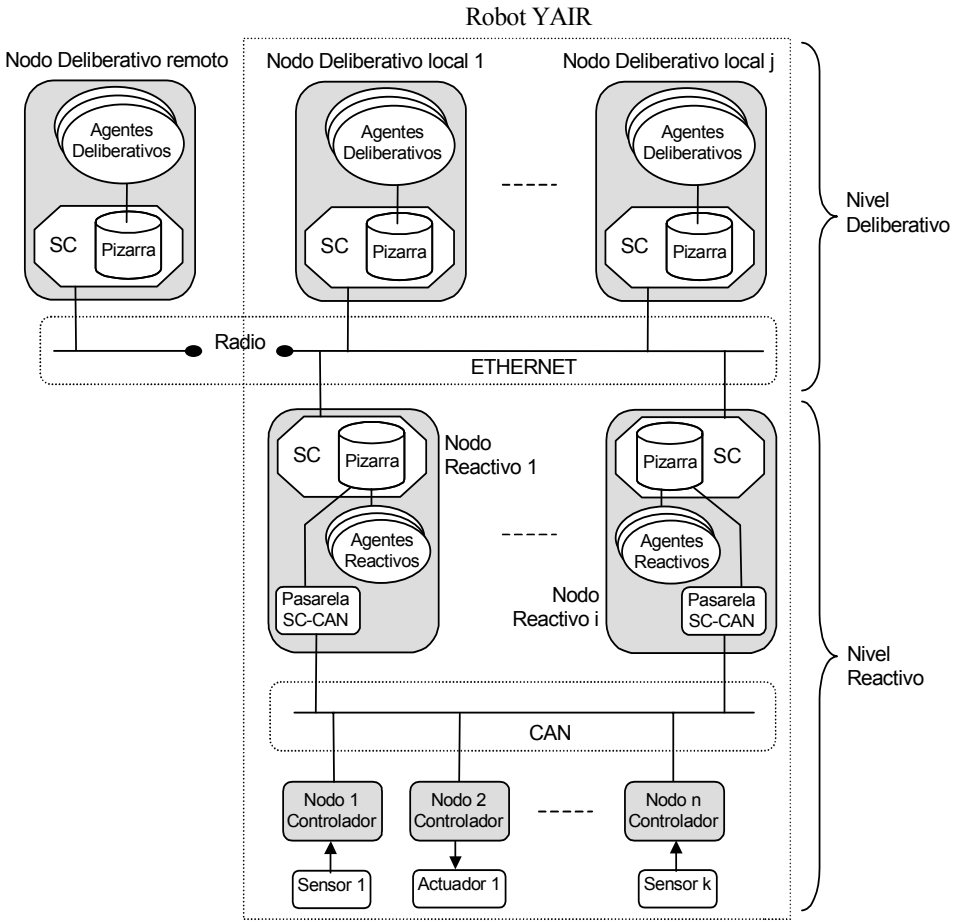


Figura 78: *Implementación de los niveles de la arquitectura en el robot YAIR*

Los nodos controladores del nivel reactivo envían la información de los sensores y reciben las acciones para los actuadores mediante el envío y la recepción de mensajes a través del bus CAN [Pérez, 2003a y b]. Las características del bus CAN permiten la especificación de un sistema de objetos mediante el uso de variables distribuidas con identificadores únicos. La estructura de objetos que forma la pizarra del sistema SC se actualiza de forma constante por la pasarela SC-CAN reflejando los valores de los objetos CAN. Mediante el acceso a la pizarra, cualquier agente

accede de forma transparente al sistema de objetos distribuidos CAN. Cualquier cambio en los objetos CAN se refleja en la pizarra y viceversa. Únicamente es necesario que uno de los computadores con el SC disponga de acceso físico al bus CAN, el resto actualizarán su copia de la pizarra a través de éste.

De este modo, cuando un agente necesita los valores de ciertos sensores para evaluar un determinado comportamiento básico, por ejemplo, evitar obstáculos, únicamente necesita realizar el acceso al objeto de la pizarra donde se hallan los valores de estos sensores. Previamente, de forma independiente, otro agente debe almacenar la información sensorial en dicho objeto.

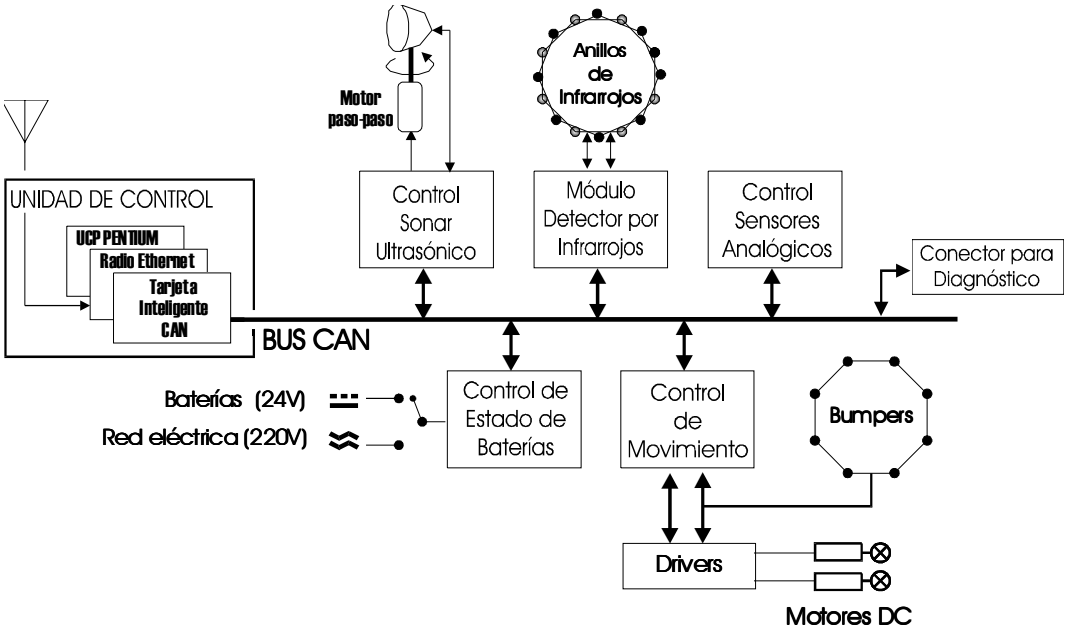


Figura 79: *Arquitectura hardware del robot YAIR*

A continuación se describen las características generales de cada uno de los nodos disponibles en el robot YAIR (fig.79) [Simó, 1999]:

- Nodo reactivo de control de movimiento. Este nodo consta de tres procesadores, un procesador de supervisión que se encarga de las comunicaciones CAN y dos procesadores que controlan de manera síncrona el movimiento de las dos ruedas motrices del robot.

- Nodo reactivo de control de infrarrojos. Este nodo proporciona los valores de un anillo de sensores de infrarrojos disponible en el robot para la localización de objetos.

- Nodo reactivo de control del sonar ultrasónico. Este nodo proporciona información sobre los valores de un sensor de ultrasonidos también para la localización de objetos aunque a una distancia mayor.

- Nodos reactivos auxiliares. Entre los que podemos destacar un monitor de estado de batería, cámara digital, micrófonos estéreo para tracking sonoro, sensores de temperatura, etc.

- Nodo reactivo principal. Se trata de un PC empotrado que gestiona las comunicaciones con el exterior y ejecuta los agentes reactivos proporcionados por el nivel deliberativo a través del sistema SC.

6.4.1. Pruebas Realizadas y Resultados

Se han realizado varias pruebas con el robot YAIR. Debido a la estructura híbrida de la arquitectura SC-Agent, comunicaciones de tiempo real estricto en el nivel reactivo y comunicaciones de tiempo real no estricto en el deliberativo, se han realizado diferentes análisis para cada nivel.

Nodo	Tipo de mensaje	Bytes del mensaje	Periodo (ms)	Latencia (ms) peor caso
Central	Solicitud infrarrojos	2	8	0,203
	Solicitud odometría	2	10	0,276
	Solicitud ultrasonidos	2	300	0,349
Controlador odometría	Respuesta odometría	4	10	0,479
Controlador infrarrojos	Respuesta infrarrojos	8	8	0,571
Controlador ultrasonidos	Respuesta ultrasonidos	200	300	4,823

Tabla 8: *Resultados del estudio de la planificabilidad del bus CAN: peores casos de latencia*

El primer análisis, relacionado con el bus de tiempo real estricto, tiene el objetivo de asegurar que el bus CAN puede soportar la carga de comunicaciones necesaria para el envío de la información sensorial requerida por las distintas tareas de control del sistema distribuido, asegurando el cumplimiento de las cotas temporales asignadas. Para ello, se ha realizado el estudio de la planificabilidad del bus [Posadas, 2000 y 2002] suponiendo que existe un nodo central (procesador multitarea) que requiere la información sensorial procedente del resto de nodos conectados al bus CAN (controladores de los sensores de infrarrojos, de ultrasonidos y de odometría) para así obtener un mapa local de obstáculos y poder controlar el robot mediante el envío de las velocidades necesarias que eviten los obstáculos. El análisis de la latencia del bus CAN requiere el uso de periodos fijos de transmisión de mensajes, por ello, se asignan periodos de transmisión para los diferentes tipos de mensajes enviados por los nodos y, tras el estudio realizado, se obtienen los peores casos de latencia. En la tabla 8 se observa que los resultados obtenidos aseguran que la información puede ser transmitida por el bus con cotas temporales máximas,

permitiendo a los procesos situados en los nodos usar el bus CAN bajo restricciones de tiempo real estricto.

El siguiente paso en los análisis, consiste en la evaluación de la conexión de los procesos deliberativos mediante el sistema SC. Se han realizado varias pruebas con el objetivo de validar el acceso al bus CAN a través del sistema SC y pasarelas correspondientes. Dicho acceso debe poder realizarse tanto por procesos ejecutados en los nodos locales al robot como por procesos ejecutados en nodos remotos. Además, también podrían existir nodos heterogéneos que tuvieran que acceder al sistema.

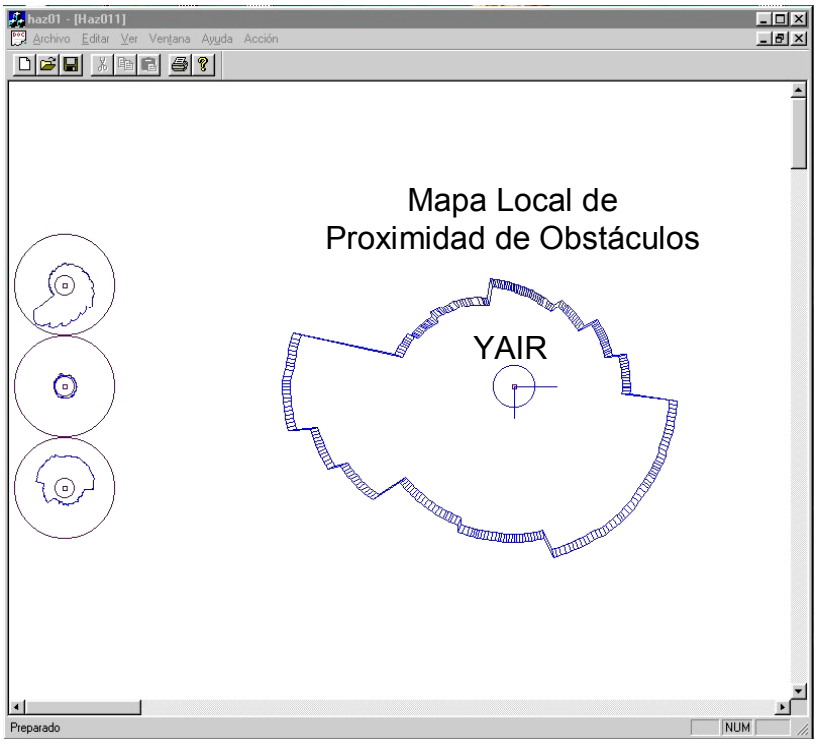


Figura 80: *Módulo deliberativo para obtención de un mapa de obstáculos local al robot*

En primer lugar, el estudio se ha centrado en los nodos homogéneos (nodos con instancias del sistema SC) [Posadas, 2000 y 2002]. Para ello, se ha utilizado un aplicación *software* (en la figura 80 se presenta su interfaz gráfica) que obtiene desde un nodo remoto la información de los sensores de ultrasonidos e infrarrojos para formar un anillo cerrado de múltiples arcos con sus correspondientes radios que representan la distancia del robot a los obstáculos. Este mapa local se actualiza cada 300 ms mientras el robot se mueve en un entorno real evitando los obstáculos. El objetivo de las pruebas es estresar el sistema de comunicaciones SC para evaluar su rendimiento. En este sentido, los agentes reactivos para evitar obstáculos también se ejecutan en el módulo descrito utilizando la información del mapa generado.

Como resultado de estas pruebas, en la tabla 9 pueden observarse los periodos de ejecución, correspondientes a las distintas tareas realizadas, que han permitido el control del robot de forma satisfactoria.

Tareas	Periodo (ms)
Obtener información de odometría	100
Enviar acciones de control	100
Obtener información de ultrasonidos	300
Obtener información de infrarrojos	300

Tabla 9: *Resultados del control desde nodos remotos a través del sistema SC y pasarela para acceso al bus CAN*

Para validar las facilidades proporcionadas por el sistema SC para que plataformas no homogéneas puedan acceder al sistema mediante el uso del concepto pasarela, se han realizado una serie de pruebas basadas en la teleoperación [Pérez, 2001 y 2002]. El acceso y control remoto de robots móviles es necesario en determinadas ocasiones. Por ejemplo, cuando se desea cambiar o cancelar su modo de funcionamiento, para monitorizar el estado o valores de los sensores-actuadores, o para realizar un control manual. Lo ideal es proporcionar un medio de acceso sencillo y en tiempo de ejecución. En este sentido está enfocado el trabajo desarrollado.

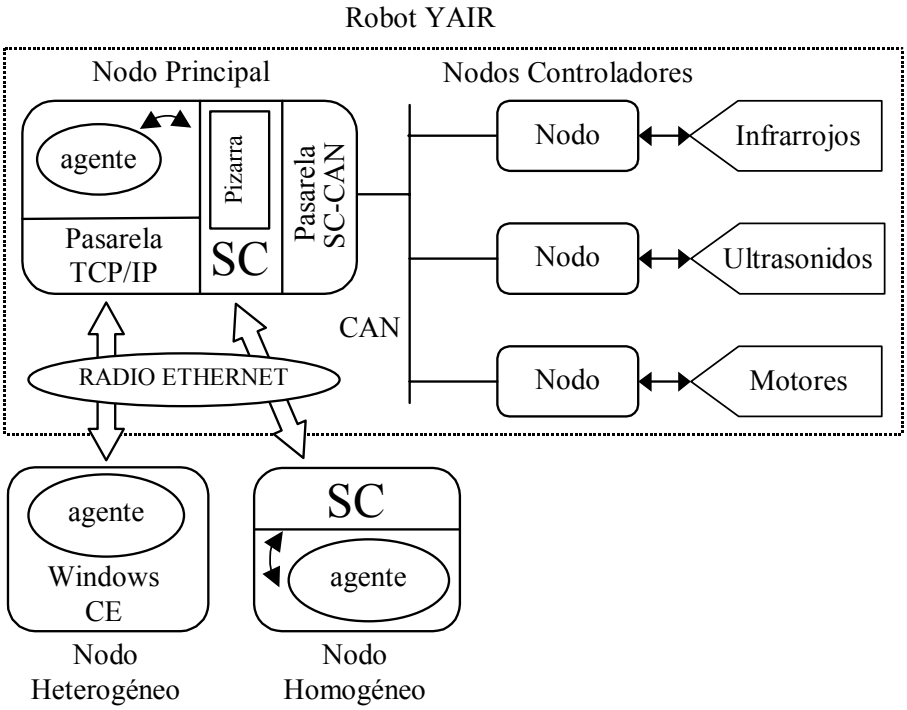


Figura 81: *Acceso al bus CAN a través del sistema SC y pasarelas correspondientes desde nodos remotos tanto homogéneos como heterogéneos*

Para ello, se ha implementado una aplicación sobre Windows CE que permite el envío de velocidades al robot. La aplicación en Windows CE se ejecuta sobre un PC empotrado y establece conexión vía radio ethernet con una pasarela situada en uno de los nodos del robot (fig.81). La pasarela envía al SC local los mensajes procedentes del PC empotrado. El SC los redirige a la pasarela CAN y ésta última se encarga de introducirlos en el bus CAN de forma que los nodos del nivel reactivo puedan recibirlos y ejecutar sus órdenes.

La aplicación sobre Windows CE ofrece una sencilla interfaz gráfica que permite al usuario indicar la dirección a seguir por el robot (fig.82). Dicha dirección es enviada a los nodos controladores a través de un conexión vía radio y el sistema SC con las pasarelas correspondientes.

Figura 82: *Control del robot YAIR desde un nodo heterogéneo. El operario envía direcciones de movimiento sin preocuparse de los obstáculos*

La composición de las ordenes de teleoperación (dirección deseada) con los comportamientos reactivos (evitar obstáculos) determina las órdenes definitivas a ejecutar por los actuadores. De esta forma, el resultado es un sistema de teleoperación amigable y robusto donde el operador no debe preocuparse sobre las posibles colisiones.

Según este modelo, la teleoperación puede considerarse, bien como la ejecución de un conjunto de comportamientos que contribuyen a las acciones de control junto con los comportamientos ejecutados en los nodos reactivos, bien como la ejecución de los procesos de motivación asociados a los comportamientos ejecutados en el nivel reactivo.

6.5. Conclusiones

La implementación de la arquitectura SC-Agent ha permitido validar sus características. Por un lado, el sistema de comunicaciones SC se ha aplicado a nivel industrial demostrando una gran estabilidad y versatilidad. Por otro lado, con el prototipo *software* desarrollado se han hecho pruebas sobre la delegación y composición de comportamientos, el control temporal de los datos y el movimiento de los agentes. Los resultados han sido satisfactorios y además se han visto reforzados por la aplicación de la arquitectura para el control de un robot real.

En resumen, la implementación de la arquitectura SC-Agent para un sistema determinado consiste en la instanciación de los agentes necesarios, característica que permite el desarrollo rápido de sistemas de control.

Como conclusión, el control de robots mediante técnicas de delegación de comportamientos y agentes presenta las siguientes ventajas:

- Cada componente puede desarrollarse de forma independiente del resto de componentes. Una vez puesto en funcionamiento un sistema determinado, pueden desarrollarse nuevos componentes e incorporarse a dicho sistema de una forma rápida y sencilla sin interrupciones.

- El control del robot se realiza con sólo enviarle los patrones de comportamientos (principalmente esquemas de percepción y motores) adecuados. Para cambiar el funcionamiento de un robot sólo hay que enviarle dinámicamente nuevos componentes.

- Las comunicaciones necesarias para el control son mínimas una vez se han delegado los componentes necesarios o bien éstos se han movido al lugar de procedencia de los datos, ya que entonces el procesamiento es local.

CONCLUSIONES

En esta tesis se ha propuesto una arquitectura híbrida multinivel para el control de robots móviles. Los componentes de los niveles reactivo y deliberativo son agentes móviles que interactúan a través de un sistema de comunicaciones interfaz basado en una pizarra de objetos distribuida con soporte temporal. Los agentes se comunican y acceden a la información sensorial mediante la lectura y escritura de los objetos de la pizarra.

El diseño de la arquitectura contempla las diferencias existentes entre los requerimientos de tiempo real de los componentes reactivos y deliberativos, para ello emplea distintos buses de conexión. Además, el sistema de comunicaciones caracteriza temporalmente todos los objetos de la pizarra proporcionando la antigüedad de los valores de los mismos.

Las características principales de la arquitectura propuesta son la modularidad y adaptabilidad que permiten la rápida implementación de sistemas de control independientemente del número de sensores y actuadores. El empleo de agentes móviles para especificar los procesos de planificación y los patrones de comportamientos (esquemas de percepción, esquemas motores, motivaciones y compositores) permite que éstos puedan incorporarse al sistema dinámicamente. De esta forma, la implementación de la arquitectura para un sistema determinado consiste en la instanciación de los agentes correspondientes.

La arquitectura propuesta proporciona un nuevo nivel semántico donde se especifican los procesos pero no su ubicación física. Éstos podrán moverse allí donde las condiciones del entorno sean más favorables y la sobrecarga de las comunicaciones sea menor.

Contribuciones

El principal objetivo, que se planteó en la introducción de esta tesis, fue el obtener una estructura modular y portable que permitiera el desarrollo rápido de

sistemas de control de robots. Este objetivo general se ha conseguido mediante la definición de la arquitectura SC-Agent, la cual constituye la principal contribución de esta tesis.

En concreto, se han realizado las siguientes contribuciones:

- Especificación de las características que ha de poseer una arquitectura híbrida para el control de robots móviles que interactúa en un entorno dinámico. Como característica innovadora, se plantea un nuevo nivel semántico donde la ubicación de los componentes de la arquitectura no esté predefinida, permitiéndose así una mayor adaptabilidad a los cambios del sistema y entorno. La idea, además, es permitir que los componentes independientes puedan incorporase a la arquitectura modular en tiempo de ejecución.

- Visión del tiempo real desde una perspectiva diferente a la clásica. En este sentido, el sistema de tiempo real no se basa en un estudio previo de la planificabilidad de los procesos, los cuales podrían no estar disponibles a priori, sino que el cumplimiento de las restricciones de tiempo real ha de basarse en la adaptabilidad del sistema a los recursos disponibles.

- Especificación de las necesidades comunicacionales que permitan el diseño de una arquitectura con las características planteadas. En resumen, el sistema de comunicaciones debe ser la base del diseño de la arquitectura contemplando las diferentes restricciones temporales existentes. Debe permitir: el acceso a los datos por parte de los componentes, la caracterización temporal de dichos datos mediante su antigüedad, y la movilidad de los componentes entre los diferentes nodos del sistema distribuido.

- Determinación de la idoneidad de los agentes *software* móviles para la especificación de los componentes de las arquitecturas híbridas.

- Especificación, diseño e implementación de un sistema de comunicaciones, denominado sistema SC, adecuado a las arquitecturas de control híbridas. El sistema SC constituye un sistema de comunicaciones de propósito general que reúne las características planteadas como necesarias para el diseño de una arquitectura híbrida. Su aportaciones principales son:

    - Especificación de una interfaz común de acceso transparente a diferentes recursos de comunicación denominada FSA o Marco-Sensor-Adaptador.

    - Acceso a los datos del sistema mediante el uso de una pizarra de objetos distribuida.

- Caracterización temporal de los datos a través de su antigüedad. Medición de los retrasos que sufren los datos hasta alcanzar su destino a través de los distintos nodos de la arquitectura.

- Soporte a distintas restricciones temporales mediante el empleo de dos buses (uno de tiempo real estricto y otro de tiempo real no estricto) para la transmisión de la información entre los nodos del sistema distribuido.

- Especificación y diseño de la arquitectura SC-Agent basándose en el sistema SC y los agentes *software* móviles. Las características principales que definen a la arquitectura SC-Agent son:

  - Multinivel. La arquitectura está formada por tres bloques: un nivel reactivo, un nivel deliberativo y el sistema de comunicaciones SC que actúa de interfaz entre ambos niveles.

  - Distribuida. Todos los niveles de la arquitectura son distribuidos. El sistema de comunicaciones SC conecta a los nodos del nivel reactivo mediante un bus de tiempo real estricto y a los nodos del nivel deliberativo mediante un bus de tiempo real no estricto.

  - Comportamiento emergente. El funcionamiento de la arquitectura está basado en la ejecución de patrones de comportamientos formados por esquemas de percepción y esquemas motores cuya composición da lugar a los comportamientos emergentes o exhibidos por el robot.

  - Tiempo real. En el nivel reactivo se puede asegurar la ejecución en tiempo real de los patrones de comportamientos al acceder a los sensores y actuadores a través del bus de tiempo real estricto.

  - Modular. La especificación de los niveles de la arquitectura está basada en componentes. En el nivel reactivo se encuentran los diferentes patrones de comportamientos y en el nivel deliberativo, principalmente, las tareas de planificación, interfaz con el usuario, monitorización y generación de mapas del entorno y fusión sensorial.

  - Independencia. Cada uno de los componentes de la arquitectura se ejecuta de forma independiente al resto. La interacción y conexión entre los mismos se realiza a través del sistema SC e interfaz FSA mediante la lectura y escritura de los objetos de la pizarra distribuida. De esta forma, la comunicación entre los componentes también es independiente de su ubicación física y de su lenguaje de programación.

  - Agentes. Los componentes se desarrollan de forma natural mediante agentes *software*.

  - Movilidad. Los agentes poseen la característica de movilidad que les permite decidir su ubicación física en el sistema utilizando el denominado "índice de comodidad". El cálculo de este índice determina la adecuación de un agente al nodo donde se encuentra en función de los retardos sufridos en su ejecución y de la antigüedad de los datos que recibe.

  - Control por delegación. La movilidad de los agentes también permite realizar un control remoto separando el tiempo empleado en las comunicaciones del tiempo empleado para el procesamiento. En primer lugar, se delega el código de los agentes a los nodos reactivos del robot y, en segundo lugar, se ejecuta localmente bajo restricciones de tiempo real.

  - Adaptabilidad. La arquitectura se adapta rápidamente a cualquier número de nodos, sensores o adaptadores. Los agentes pueden incorporarse al sistema de forma dinámica en función de las necesidades de cada momento.

- Implementación de la arquitectura SC-Agent. Como última contribución, se ha implementado la arquitectura SC-Agent a través de un prototipo *software* que valida las diferentes características ofrecidas. También se ha realizado dicha implementación sobre un robot real demostrando sus capacidades de control.

Trabajos Futuros

De los resultados obtenidos se deduce que para la explotación de la arquitectura propuesta, basada en agentes móviles, es necesario un avance en las capacidades de procesamiento ofrecidas por los nodos manejadores de los distintos sensores y actuadores. Estos nodos, al final, deben permitir el establecimiento de los sistemas de agentes móviles asegurando las restricciones temporales de ejecución. En este sentido, también han de producirse avances en el desarrollo de los agentes móviles.

Se ha establecido una arquitectura de control con un diseño sólido sobre un sistema de comunicaciones de ámbito general. Establecidos los principios de organización, las distintas líneas de trabajo futuras, tanto tecnológicas como científicas, pueden enmarcarse en los siguientes puntos:

Tecnológicos

- Extensión del sistema de comunicaciones SC a otros Sistemas Operativos. Los objetivos en esta línea son:

  - Obtención de una versión del sistema SC para un Sistema Operativo de Tiempo Real. La idea es utilizar el Sistema Operativo RT-Linux.

De esta forma, los agentes podrían moverse a nodos donde pudiera asegurarse el cumplimiento de sus periodos de ejecución.

- Obtención de una versión del sistema SC para Windows CE. Así, un PC empotrado se convertiría en un nodo homogéneo al sistema SC permitiéndole, por tanto, el acceso transparente a toda la información.

- Evaluación de la interoperatividad entre las distintas versiones SC a través del protocolo IIOP. Especificación del formato y tipos de mensajes necesarios. En este ámbito, también puede desarrollarse un núcleo CORBA ORB cuya infraestructura de comunicación sea el sistema SC, el cual proporcionaría a CORBA la caracterización temporal de los datos.

- Extensión del espacio de nombres lógicos (basados en XML) mediante el desarrollo de un módulo para la conversión automática de los nombres lógicos de los datos en los nombres primitivos utilizados por el sistema SC y viceversa. Los agentes, a través de la interfaz FSA, indican nombres lógicos que el sistema internamente debe convertir a la representación utilizada.

- Aplicación de la arquitectura SC-Agent sobre un robot real que disponga de varios nodos locales con capacidad de procesamiento multitarea. El objetivo es evaluar la distribución automática de los agentes entre los nodos locales del robot en función de su "índice de comodidad".

- Aplicación de los procesos de motivación para determinar el peso de las contribuciones en el comportamiento emergente realizadas por los distintos patrones de comportamientos. En esta línea, también se ha de trabajar en la secuenciación de comportamientos necesaria en determinadas situaciones.

Científicos

- Extensión del sistema multiagente desarrollado. Existen varias propuestas:

  - Desarrollo de un sistema multiagente basado en la reciente plataforma de programación .NET. Los objetivos son mejorar la adaptabilidad de la arquitectura explotando las características del mecanismo de reflexión ofrecido, y la evaluación de la interacción a través del sistema SC entre agentes programados en código manejado CLR (.NET), código manejado java e incluso código nativo C++.

  - Especificación completa, basada en el lenguaje ACL, de los mensajes necesarios para la comunicación entre los agentes.

  - Extensión del objeto REA (Receptor y Ejecutor de Agentes) para la incorporación de un sistema de control y directorio de los agentes compatible con el estándar FIPA.

  - Una línea de trabajo interesante consiste en disponer de un sistema de agentes móviles sobre un Sistema Operativo de Tiempo Real. De esta forma, la adaptabilidad del sistema podría explotarse al máximo. Si el sistema no pudiera garantizar los periodos de ejecución de los agentes, podría optar por aumentar los periodos con la consiguiente degradación de prestaciones pero asegurando la integridad del sistema o podría optar por delegar a otros nodos la ejecución de aquellos agentes no críticos.

- En cuanto al nivel deliberativo, hay varios aspectos que han quedado fuera del alcance de este trabajo:

  - Desarrollo de un "agente planificador de misiones" que realice la división de los objetivos del robot en los patrones de comportamientos necesarios para su envío al nivel reactivo.

  - Desarrollo de un "agente generador de mapas" que obtenga mapas globales del entorno para su utilización en los procesos de planificación. En este línea deben realizarse trabajos sobre fusión espacio-temporal de información procedente de distintos tipos de sensores.

- Por último, una línea de trabajo importante es la coordinación de varios robots que implementan la arquitectura SC-Agent. El enfoque realizado se basa en considerar a los robots, no como entidades con autonomía propia, sino como recursos del sistema que los distintos agentes *software*, que sí disponen de autonomía propia, pueden utilizar a conveniencia. Por ejemplo, un mismo agente implicado en una tarea determinada podría disponer de varios robots para llevarla a cabo. Este punto de vista permite abordar la coordinación de los robots mediante la coordinación de los agentes *software* que los utilizan.

Publicaciones Relacionadas

La línea de investigación desarrollada en esta tesis se ha reflejado, a lo largo del proceso de investigación, en un conjunto de publicaciones que avalan tanto el trabajo realizado como los resultados obtenidos. En este sentido, las principales publicaciones se detallan a continuación:

Artículo en Revista

- Posadas, J.L., Pérez, P., Simó, J.E., Benet, G., Blanes, F., (2002). Communications Structure for Sensory Data in Mobile Robots. Engineering Applications of Artificial Intelligence, vol. 15, no. 3, (2002) 341-350.

Artículo en Capítulo de Libro

- Poza, J.L, Posadas, J.L., Simó, J.E., Crespo, A., (2002). Data and event management in a maritime terminal of containers. New Technologies For Computer Control, chapter 12, pp. 269-274. Elsevier Science. I.S.B.N.: 0-08-043700-1.

Artículos en Congresos Internacionales

- Simó, J.E., Crespo, A., Blanes, F., Posadas, J.L., (1997). Vehicle Guidance with Distributed Selection of Emergent Behaviour. Proceedings of IFAC International Symposium on Artificial Intelligence in Real Time Control - AIRTC´97. Kuala Lumpur (Malasia). ISBN 0-08-042927-0.

- Posadas, J.L., Pérez, P., Simó, J.E., Benet, G., Blanes, F., (2000). Communications Structure for Sensor Fusion in Distributed Real Time Systems. Proceedings of 6th International Workshop on Algorithms and Architectures for Real-Time Control - AARTC'2000. Mallorca (España). I.S.B.N.: 0-08-043685-4.

- Posadas, J.L., Poza, J.L., Simó, J., (2001). Communication System for a Multisensor Robot Distributed Architecture. X Simposio de Ingeniería Eléctrica - SIE'2001. Santa Clara (Cuba). I.S.B.N.: 959-250-024-x.

- Pérez, P., Posadas, J.L., Simó, J.E., Benet, G., Blanes, F., (2001). Communication Architecture for Mobile Robots Teleoperation. 1st. IFAC Conference on Telematics Applications in Automation and Robotics - TA'2001. Weingarten (Alemania). I.S.B.N.: 0-08-043856-3.

- Poza, J.L, Posadas, J.L., Simó, J.E., Crespo, A., (2001). Data and Event Management in a Maritime Terminal of Containers. Proceedings of IFAC Conference on New Technologies for Computer Control - NTCC'2001, pp. 269-274. Hong Kong (China).

- Pérez, P., Posadas, J.L., Simó, J.E., Benet, G., Blanes, F., (2002). A Software Framework for Mobile Robot Sensor Fusion and Teleoperation. Proceedings of the 15th IFAC world Congress. Barcelona (España). I.S.B.N.: 008044184X.

- Pérez, P., Posadas, J.L., Benet, G., Blanes, F., Simó, J.E., (2003). An Intelligent Sensor Architecture for Mobile Robots. 11th International Conference on Advanced Robotics - IEEE ICAR'03. University of Coimbra (Portugal).

- Pérez, P., Posadas, J.L., Poza, J.L., Benet, G., Blanes, F., Simó, J.E., (2003). Communications jitter influences on Control loops using Protocols for Distributed Real-Time Systems on CAN bus. 5th IFAC International

Symposium on Intelligent Components and Instruments for Control Applications - SICICA'03. Aveiro (Portugal).

- Poza, J.L, Posadas, J.L., Simó, J.E., Crespo, A., (2003). Communication System Architecture to Manage a Containers Terminal. 9th IEEE International Conference on Emerging Technologies and Factory Automation - ETFA2003. Calouste Gulbenkian Foundation. Lisbon (Portugal).

Artículos en Congresos Nacionales

- Posadas, J.L., Simó, J., Blanes, F., (1997). Un modelo para el desarrollo de aplicaciones distribuidas. Servidor de Comunicaciones. Jornadas Españolas de Automática - JA´97. Gerona (España).

- Simó, J., Benet, G., Blanes, F., Posadas, J.L., Pérez, P., (1999). Mobile Robot Systems Development: Perception Modules. II Jornadas de Tiempo Real. Gandía (España).

- Pérez, P., Posadas, J.L., Simarro, R., Benet, G., Blanes, F., (2000). Comunicaciones para fusión sensorial en sistemas distribuidos de tiempo real sobre CAN. Simposio Español de Informática Distribuida - SEID'2000. Orense (España). I.S.B.N.: 84-8158-163-1.

- Posadas, J.L., Simó, J.E., Poza, J.L., Pérez, P., Benet, G., Blanes, F., (2003). Una arquitectura para el Control de Robots Móviles mediante Delegación de Código y Sistemas Multiagente. IV Workshop de Agentes Físicos - WAF'03. Alicante (España). I.S.B.N.: 84-607-7171-7.

ÍNDICE DE FIGURAS

Figura 1: Relación entre los distintos capítulos de la tesis 29

Figura 2: Filosofía del paradigma deliberativo: Sentir-Planear-Actuar (S-P-A) 33

Figura 3: Filosofía del paradigma reactivo: Sentir-Actuar (S-A) 35

Figura 4: Filosofía del paradigma híbrido: Planear, Sentir-Actuar (P, S-A) 37

Figura 5: Teoría de Esquemas: descomposición de los patrones de comportamientos en esquemas de percepción y esquemas motores 40

Figura 6: Arquitectura híbrida AURA: estructura de dos niveles deliberativo y reactivo constituidos por cinco subsistemas y generación del comportamiento por composición de esquemas 42

Figura 7: Arquitectura híbrida SFX: estructura de pizarra con agentes independientes y generación del comportamiento por inhibición 44

Figura 8: Inhibición de comportamientos en la arquitectura FSX: los comportamientos tácticos inhiben a los comportamientos estratégicos 45

Figura 9: Arquitectura híbrida 3T: estructura de tres niveles según restricciones temporales y generación del comportamiento por activación a través de eventos de los skills o comportamientos primitivos 46

Figura 10: Arquitectura híbrida GLAIR: estructura de tres niveles según grado de representación y generación del comportamiento por inhibición 47

Figura 11: Arquitectura híbrida Saphira: estructura de datos común con componentes independientes y generación del comportamiento por secuenciación 49

Figura 12: Modelo de comunicaciones Cliente-Servidor: asociación de procesos servidores a recursos compartidos, comunicación mediante mensajes punto a punto entre los clientes y servidores 57

Figura 13: Modelo de comunicaciones RPC: comunicación mediante invocación de servicios, ocultación del sistema de mensajes y empaquetado de los datos 59

Figura 14: Modelo de comunicaciones de objetos distribuidos: comunicación mediante objetos remotos, empaquetado de los datos y de las referencias a los objetos 61

Figura 15: Nivel del proxy o interfaz local del objeto remoto en el modelo de comunicaciones de objetos distribuidos 61

Figura 16: CORBA: especificación estándar del modelo de comunicaciones de objetos distribuidos basada en el componente gestor de peticiones ORB 63

Figura 17: Componentes de la arquitectura CORBA 64

Figura 18: Interoperatividad entre ORB's: invocación de operaciones entre cliente/servidor con diferentes ORB's 65

Figura 19: Interoperatividad entre ORB's: uso de puentes (bridges) y del protocolo GIOP 66

Figura 20: Nivel del protocolo IIOP: implementación del protocolo GIOP según TCP/IP 66

Figura 21: Ejemplo de interoperatividad entre ORB's (1) 68

Figura 22: Ejemplo de interoperatividad entre ORB's (2) 68

Figura 23: Ejemplo de interoperatividad entre ORB's (3) 69

Figura 24: Ejemplo de interoperatividad entre ORB's (4) 69

Figura 25: Ejemplo de interoperatividad entre ORB's (5) 69

Figura 26: Arquitectura RT-CORBA: tiempo real en CORBA 70

Figura 27: Modelo de objetos distribuidos en Java: arquitectura de Invocación de Métodos Remotos RMI 73

Figura 28: Java IDL: implementación para java de la arquitectura CORBA 74

Figura 29: Modelo de Código Móvil: migración de objetos (transmisión de datos y código) entre entornos cliente/servidor 76

Figura 30: Evolución de los diferentes modelos de programación hasta el modelo multiagente actual 80

Figura 31: Taxonomía de un agente: componente de conocimiento, componente de tareas y componente de comunicaciones 90

Figura 32: Modelo Multiagente: estándar MAF para sistemas de agentes móviles basado en CORBA 93

Figura 33: Modelo Multiagente: arquitectura del estándar FIPA 95
Figura 34: Modelo Multiagente: integración de MAF y FIPA 98
Figura 35: Arquitectura multinivel propuesta: nivel deliberativo, nivel reactivo y sistema de comunicaciones 102
Figura 36: Nivel reactivo: composición de comportamientos basada en motivaciones..... 104
Figura 37: Marco del control distribuido: restricciones temporales, expresividad vs. predecibilidad 106
Figura 38: Comparativa del control de robots desde un nodo central remoto y mediante delegación de código: separación del tiempo de las comunicaciones del tiempo de procesamiento 109
Figura 39: Estructura y conexiones de la arquitectura distribuida propuesta. Buses de tiempo real estricto y no estricto requeridos por el sistema de comunicaciones 110
Figura 40: Sistema de Comunicaciones SC. Objetos de la pizarra distribuida asociados a cada agente..... 119
Figura 41: Sistema de Comunicaciones SC. Interfaz de acceso común: modelo FSA Marco-Sensor-Adaptador..... 120
Figura 42: Modelo FSA. Especificación IDL de las estructuras de datos generales..... 122
Figura 43: Modelo FSA. Especificación IDL de la interfaz IAdapter..... 123
Figura 44: Modelo FSA. Especificación IDL de la interfaz ISensor..... 124
Figura 45: Estructura IDL de los agentes 125
Figura 46: Especificación XML del espacio de nombres para referenciar los objetos del sistema de comunicaciones SC 127
Figura 47: Árbol del espacio de nombres 128
Figura 48: Metodología de programación con el modelo FSA 129
Figura 49: Modelo de pasarela basado en FSA. Permite la conexión de distintos recursos de comunicaciones 130
Figura 50: Conexión de los componentes de la arquitectura con el Sistema de Comunicaciones SC a través del modelo FSA y uso de pasarelas 131
Figura 51: Caracterización temporal de la información: cálculo de la antigüedad de los valores de los objetos del sistema SC..... 134
Figura 52: Nivel del Sistema de Comunicaciones SC 136
Figura 53: Componentes de la arquitectura SC-Agent propuesta..... 142
Figura 54: Interfaces de los componentes de la arquitectura propuesta 145
Figura 55: Relación entre objetos pasivos y activos del nivel reactivo 147

Figura 56: Nivel reactivo: conexión y funcionamiento de los agentes a través del sistema SC 149

Figura 57: Agentes del nivel deliberativo y sistema multiagente: movilidad y conexión con el nivel reactivo a través del sistema SC 154

Figura 58: Formato de los mensajes para las comunicaciones entre agentes a través del sistema SC: uso del protocolo IIOP, nombres lógicos (XML) y lenguaje de comunicación ACL..... 156

Figura 59: Mensaje ACL para comunicar información del sistema..... 158

Figura 60: Mensaje ACL para comunicar acciones..... 159

Figura 61: Mensaje ACL correspondiente a la acción de ejecutar un agente..... 159

Figura 62: Mensaje ACL correspondiente a la acción de terminar un agente..... 159

Figura 63: Mensaje ACL para solicitar información sobre los agentes en ejecución al objeto REA 160

Figura 64: Mensaje ACL correspondiente a la respuesta del objeto REA 160

Figura 65: Implementación del sistema de comunicaciones SC: núcleo del SC o SCore, e interfaz SCore o librería SCoreDLL que proporciona a los agentes el objeto adaptador AdaptSCore según el modelo de interfaz FSA 166

Figura 66: Implementación del sistema de comunicaciones SC: empleo de pasarelas para el acceso al bus CAN de tiempo real estricto y la interoperatividad con nodos heterogéneos..... 167

Figura 67: Implantación del sistema SC en una planta de producción de objetos de porcelana..... 169

Figura 68: Terminal Marítima de Contenedores 170

Figura 69: Implantación del Sistema SC en la Terminal Marítima “Príncipe Felipe” de Valencia..... 171

Figura 70: Componentes software del prototipo desarrollado..... 174

Figura 71: Prototipo desarrollado: entorno y robots simulados, y consola del nivel deliberativo..... 178

Figura 72: Resultado sólo esquema motor para ir a destino..... 182

Figura 73: Resultado composición esquemas motores para evitar obstáculos e ir a destino 182

Figura 74: Entorno real recorrido por el robot y mapa de obstáculos detectados y ubicados sin considerar la antigüedad de los datos empleados en el cálculo 183

Figura 75: Mapas de obstáculos obtenidos al considerar la antigüedad de los datos y utilizando precisiones de 50 y 25 milisegundos 183

Figura 76: Robot YAIR..... 186

Figura 77: Conexión de los nodos del robot YAIR 187
Figura 78: Implementación de los niveles de la arquitectura en el robot YAIR 188
Figura 79: Arquitectura hardware del robot YAIR 189
Figura 80: Módulo deliberativo para obtención de un mapa de obstáculos local al robot 191
Figura 81: Acceso al bus CAN a través del sistema SC y pasarelas correspondientes desde nodos remotos tanto homogéneos como heterogéneos 192
Figura 82: Control del robot YAIR desde un nodo heterogéneo. El operario envía direcciones de movimiento sin preocuparse de los obstáculos 193

ÍNDICE DE TABLAS

Tabla 1: Definición de las tres funciones elementales o primitivas de un robot en términos de entradas y salidas 32

Tabla 2: Relación de las tres funciones elementales o primitivas de un robot según el paradigma deliberativo 33

Tabla 3: Relación de las tres funciones elementales o primitivas de un robot según el paradigma reactivo 36

Tabla 4: Relación de las tres funciones elementales o primitivas de un robot según el paradigma híbrido 38

Tabla 5: Comparativa CORBA-RMI-SC 136

Tabla 6: Nomenclatura en el nivel reactivo 148

Tabla 7: Periodos de ejecución de los agentes empleados en las pruebas descritas 185

Tabla 8: Resultados del estudio de la planificabilidad del bus CAN: peores casos de latencia 190

Tabla 9: Resultados del control desde nodos remotos a través del sistema SC y pasarela para acceso al bus CAN 192

REFERENCIAS BIBLIOGRÁFICAS

[AGLETS] IBM research. AGLETS software development kit. http://www.trl.ibm.com/aglets/index_e.htm

[Arbib, 1986] Arbib, M., (1986). Schema Theory, The Handbook of Brain Theory and Neural Networks. MIT Press, Cambridge.

[Arkin, 1998] Arkin, R.C., (1998). Behavior-Based Robotics. MIT Press, Cambridge.

[Arkin, 1990] Arkin, R.C., (1990). Integrating Behavioural, Perceptual and World Knowledge in Reactive Navigation. Robots and Autonomous Systems, 6, 105-122.

[Bates, 1992] Bates, E.A., Elman, J.L., (1992). Connectionism and the study of change. Tech. Rep. No. 9202. The University of California at San Diego.

[Becker, 2002] Becker, L.B., Pereira, C.E., (2002). SIMOO-RT An Object-Oriented Framework for the Development of Real-Time Industrial Automation Systems. IEEE Transactions on Robotics and Automation, vol. 18, no. 4, August 2002.

[Bollella, 2000] Bollella, G., Gosling, J., Brosgol, B., (2000). The Real-Time Specification for Java, version 1.0. Addison-Wesley, 2000. http://www.rtj.org

[Bonasso, 1997] Bonasso, R.P., Firby, J., Gat, E., Kortenkamp, D., Miller, D., Slack, M., (1997). A Proven Three-tiered Architecture for Programming Autonomous Robots. Journal of Experimental and Theoretical Artificial Intelligence, vol. 9, no. 2, 1997.

[Bosch, 1991] Bosch, (1991). CAN Specification 2.0A. © Robert Bosch GmbH.

[Braitenberg, 1984] Braitenberg, V., (1984). Vehicles: Experiments in Synthetic Psychology. MIT Press, Cambridge.

[Brennan, 2002] Brennan, R.W., Fletcher, M., Norrie, D.H., (2002). An Agent-Based Approach to Reconfiguration of Real-Time Distributed Control Systems. IEEE Transactions on Robotics and Automation, vol. 18, no. 4, August 2002.

[Brooks, 1986] Brooks, R.A., (1986). A robust layered control architecture for a mobile robot. IEEE Journal of Robotics and Automation, 2, (1), 14-23.

[Brooks, 1991a] Brooks, R.A., (1991). Elephants don't play chess. Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, 3-15 MIT Press, London.

[Brooks, 1991b] Brooks, R.A., (1991). Intelligence without representation. Artificial Intelligence, 47, 139-159.

[Brugali, 2002] Brugali, D., Fayad, M.E., (2002). Distributed Computing in Robotics and Automation. IEEE Transactions on Robotics and Automation, vol. 18, no. 4, August 2002.

[Burmeister, 1998] Burmeister, B., Bussmann, S., Haddadi, A., Sundermeyer, K., (1998). Agent-Oriented Techniques for Traffic and Manufacturing Applications: Progress Report. Agent Technology. Editorial Springer.

[Chapman, 1987] Chapman, D., (1987). Planning for conjunctive goals. Journal of Artificial Intelligence, 32 (3), 333-337.

[Chown, 1999] Chown, E., (1999). Making predictions in an uncertain world: Environmental structure and cognitive maps. Adaptative Behaviour 7(1), 17-33.

[CORBA, 2002] OMG. Corba Specification version 3.0.2 formal/02-12-02, (2002).

[Dorigo, 1998] Dorigo, M., Colombetti, M., (1998). Robot Shaping. An experiment in behavior engineering. A Bradford Book. The MIT Press.

[FIPA] The Foundation for Intelligent Physical Agents. http://www.fipa.org

[Franklin, 1996] Franklin, S., Graesser, A., (1996). Is it an agent, or just a program?. Proceedings Third International Workshop on Agent Theories, Architectures and Languages. Budapest, Hungary, 193-206.

[Guerraoui, 1999] Guerraoui, R., Fayad, M.E., (1999). OO Distributed Programming Is Not Distributed OO Programming. Communications on the ACM, vol. 42, no. 4, 101-104, April 1999.

[Hassan, 2001] Hassan, H., Simó, J., Crespo, A. (2001). Flexible real-time mobile robotic architecture based on behavioural models. Engineering Applications of Artificial Intelligence 14 (2001) 685-702.

[IEEE 1451] NIST National Institute of Standards and Technology, IEEE 1451 website. http://ieee1451.nist.gov

[Jennings, 1995] Jennings, N.R., Corera, J.M., Laresgoiti, I. (1995). Developing industrial multi-agent systems. Proceedings of the first International conference on Multi-agent Systems, ICMAS-95, 423-430.

[Jennings, 1998] Jennings, N.R., Wooldridge, M.J., (1998). Applications of Intelligent Agents. Agent Technology. Springer.

[JSP] JavaServer Pages. Dynamically generated Web content. http://java.sun.com/products/jsp

[Karnit, 1998] Karnit, N.M., Tripathi, A.R., (1998). Design Issues in Mobile-Agent Programming Systems. IEEE Concurrency July-September 1998, pp. 52-61.

[Konolige, 1998] Konolige, K., Myers, K., (1998). The Saphira Architecture for Autonomous Mobile Robots. Artificial Intelligence and Mobile Robots. D. Kortenkamp, R. Bonasson, R. Murphy, editors, MIT Press, 1998.

[Kopetz, 1997] Kopetz, H., Nossal, R., (1997). Temporal Firewalls in Large Distributed Real-Time Systems. Proceedings of 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems. IEEE Press.

[KQML] Knowledge Query and Manipulation Language. http://www.cs.umbc.edu/kqml

[Lammens, 1993] Lammens,J.M., Hexmoor, H., Shapiro, C., (1993). On Elephants and Men. Proceedings of the NATO Advanced Study Institute on the Biology and Technology of Intelligent and Autonomous Agents. Trento. Italy. March 1993.

[Lange, 1999] Lange D.B., Oshima, M., (1999). Seven Good Reasons for Mobile Agents. Communications of the Association of Computing Machinery, v. 42, no. 3.

[Laufmann, 1998] Laufmann, S.C., (1998). Agent Software for Near-Term Success in Distributed Applications. Agent Technology. Ed. Springer.

[Maes, 1991] Maes. P., (1991). Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back. MIT Press, London.

[MAF] Mobile Agent Facility Specification V1.0 January 2000. http://www.omg.org/technology/documents/corbafacilities_spec_catalog.htm

[McGhan, 1998] McGhan, H., O'Connor, M., (1998). PicoJava: A Direct Execution Engine For Java Bytecode. IEEE COMPUTER, 31 (10): 22-30, 1998.

[Mensch, 1993] Mensch, A., Kersual, D., Crespo, A., Carpillet, F., (1993). REAKT Architecture. Workshop on Integration Technology for Real-Time Intelligent Control Systems. IRTICS Workshop, Madrid.

[Murphy, 2000] Murphy, R.R., (2000). Introduction to AI Robotics. MIT Press.

[Musliner, 1993] Musliner, D.J., Durfee, E.H., Shin, K.G., (1993). CIRCA: A Cooperative Intelligent Real-Time Control Architecture. IEEE Transactions on Systems, Man, and Cybernetics, vol. 23, no. 6, 1993.

[Natarajan, 2002] Natarajan, B., Gokhale, A.S., Schmidt, D.C., Wang, N., Gill, C.D., (2002). Towards Dependable Real-time and Embedded CORBA Systems. IEEE Workshop on Dependable Middleware-Based Systems, Washington, D.C., June 2002.

[Nii, 1989] Nii, H. P., (1989). Introduction, in: Blackboard architectures and applications. Edited by V. Jagannathan, Rajendra Dodhiawala and Lawrence S. Baum, (Perspectives in artificial intelligence, volume 3). Academic Press, Boston, pp. xix-xxix.

[Nilsen, 1998] Nilsen, K., (1998). Adding real-time capabilities to Java. Communications of the ACM, volume 41, issue 6, 1998.

[Nilsson, 1980] Nilsson, N.J., (1980). Principles of Artificial Intelligence. Morgan Kaufmann, Ed. Los Altos, CA.

[Nwana, 1996] Nwana, H., Lee, L., Jennings, N., (1996). Coordination in Software Agent Systems. BT Technology Journal, 14(4), 1996.

[O'Brien, 1996] O'Brien, P., Wiegand, M.E., (1996). Agent-based business process management. Int. Journal of Cooperative Information Systems, 5 (2, 3), 105-130.

[OMG] Object Management Group. http://www.omg.org

[Penrose, 1989] Penrose, R., (1989). La Nueva Mente del Emperador. Oxford University Press.

[PERC] NewMonics Inc. Portable Executive for Reliable Control. http://www.newmonics.com

[Pérez, 2000] Pérez, P., Posadas, J.L., Simarro, R., Benet, G., Blanes, F., (2000). Comunicaciones para fusión sensorial en sistemas distribuidos de tiempo real sobre CAN. Simposio Español de Informática Distribuida - SEID'2000. Orense (España). I.S.B.N.: 84-8158-163-1.

[Pérez, 2001] Pérez, P., Posadas, J.L., Simó, J.E., Benet, G., Blanes, F., (2001). Communication Architecture for Mobile Robots Teleoperation. 1st. IFAC Conference on Telematics Applications in Automation and Robotics - TA'2001. Weingarten (Alemania). I.S.B.N.: 0-08-043856-3.

[Pérez, 2002] Pérez, P., Posadas, J.L., Simó, J.E., Benet, G., Blanes, F., (2002). A Software Framework for Mobile Robot Sensor Fusion and Teleoperation. Proceedings of the 15th IFAC world Congress. Barcelona (España). I.S.B.N.: 008044184X.

[Pérez, 2003a] Pérez, P., Posadas, J.L., Benet, G., Blanes, F., Simó, J.E., (2003). An Intelligent Sensor Architecture for Mobile Robots. 11th International Conference on Advanced Robotics - IEEE ICAR'03. University of Coimbra (Portugal).

[Pérez, 2003b] Pérez, P., Posadas, J.L., Poza, J.L., Benet, G., Blanes, F., Simó, J.E., (2003). Communications jitter influences on Control loops using Protocols for Distributed Real-Time Systems on CAN bus. 5th IFAC International Symposium on Intelligent Components and Instruments for Control Applications - SICICA'03. Aveiro (Portugal).

[Pnueli, 1986] Pnueli, A., (1986). Specification and development of reactive systems. Information Processing 86. Elsevier, North-Holland.

[Posadas, 1997] Posadas, J.L., Simó, J., Blanes, F., (1997). Un modelo para el desarrollo de aplicaciones distribuidas. Servidor de Comunicaciones. Jornadas Españolas de Automática - JA´97. Gerona (España).

[Posadas, 2000] Posadas, J.L., Pérez, P., Simó, J.E., Benet, G., Blanes, F., (2000). Communications Structure for Sensor Fusion in Distributed Real Time Systems. Proceedings of 6th Workshop on Algorithms and Architectures for Real-Time Control - AARTC'2000. Mallorca (España). I.S.B.N.: 0-08-043685-4.

[Posadas, 2001] Posadas, J.L., Poza, J.L., Simó, J., (2001). Communication System for a Multisensor Robot Distributed Architecture. X Simposio de Ingeniería Eléctrica - SIE'2001. Santa Clara (Cuba). I.S.B.N.: 959-250-024-x.

[Posadas, 2002] Posadas, J.L., Pérez, P., Simó, J.E., Benet, G., Blanes, F., (2002). Communications Structure for Sensory Data in Mobile Robots. Engineering Applications of Artificial Intelligence, vol. 15, no. 3, (2002) 341-350.

[Posadas, 2003] Posadas, J.L., Simó, J.E., Poza, J.L., Pérez, P., Benet, G., Blanes, F., (2003). Una arquitectura para el Control de Robots Móviles mediante Delegación de Código y Sistemas Multiagente. IV Workshop de Agentes Físicos - WAF'03. Alicante (España). I.S.B.N.: 84-607-7171-7.

[Poza, 2001] Poza, J.L, Posadas, J.L., Simó, J.E., Crespo, A., (2001). Data and Event Management in a Maritime Terminal of Containers. Proceedings of IFAC Conference on New Technologies for Computer Control - NTCC'2001, pp. 269-274. Hong Kong (China).

[Poza, 2002] Poza, J.L, Posadas, J.L., Simó, J.E., Crespo, A., (2002). Data and event management in a maritime terminal of containers. New Technologies For Computer Control, chapter 12, pp. 269-274. Elsevier Science. I.S.B.N.: 0-08-043700-1.

[Poza, 2003] Poza, J.L, Posadas, J.L., Simó, J.E., Crespo, A., (2003). Communication System Architecture to Manage a Containers Terminal. 9th IEEE International Conference on Emerging Technologies and Factory Automation - ETFA2003. Calouste Gulbenkian Foundation. Lisbon (Portugal).

[Rao, 1995] Rao, A.S., Georgeff, M.P., (1995). BDI agents: from theory to practice. Proceedings of the first International conference on Multi-agent Systems, ICMAS-95, 312-129.

[RT-CORBA, 2002] OMG. Real-Time Corba Specification version 1.1 formal/02-08-02, (2002).

[Schmidt, 2000] Schmidt, D.C., Kuhns, F., (2000). An Overview of the Real-Time CORBA Specification. IEEE Computer (Special Issue on OO RT Distributed Computing), vol. 33, pp. 56-63, June 2000.

[Seetharaman, 1998] Seetharaman, K., Siegel, J., Vinoski, S., Schmidt, D.C., Henning, M., Haggerty, P., (1998). The CORBA Connection. Communications on the ACM, vol. 41, no. 10, 34-79, October 1998.

[Shoham, 1993] Shoham, Y., (1993). Agent-oriented programming. Artificial Intelligence, 60(1), 51-92.

[Simó, 1997a] Simó, J., Crespo A., Blanes, J.F., (1997). Behaviour Selection in the YAIR Architecture. Proceedings of IFAC Conference on Algorithms and Architectures for Real Time Control. Vilamoura, Portugal. AARTC'97.

[Simó, 1997b] Simó, J.E., (1997). Una arquitectura basada en motivaciones para el control de robots móviles. PhD Thesis, Polytechnic University of Valencia.

[Simó, 1997c] Simó, J.E., Crespo, A., Blanes, F., Posadas, J.L., (1997). Vehicle Guidance with Distributed Selection of Emergent Behaviour. Proceedings of IFAC International Symposium on Artificial Intelligence in Real Time Control AIRTC´97. Kuala Lumpur (Malasia). ISBN 0-08-042927-0.

[Simó, 1999] Simó, J., Benet, G., Blanes, F., Posadas, J.L., Pérez, P., (1999). Mobile Robot Systems Development: Perception Modules. II Jornadas de Tiempo Real. Gandía (España).

[Singhal, 1998] Singhal, S., Nguyen, B., Tyma, P., Bohrer, K., Johnson, V., Nilson, A., Rubin, B., Nilsen, K., (1998). The Java Factor. Communications on the ACM, vol. 41, no. 6, pp. 34-56, June 1998.

[Sloman, 1998] Sloman, A., (1998). What's an AI Toolkit For?. Proceedings AAAI-98 Workshop on Software Tools for Developing Agents. Madison, July 1998.

[Sloman, 1999] Sloman, A., Logan, B., (1999). Building Cognitively Rich Agents Using the SIM_AGENT_TOOLKIT. Communications of the Association of Computing Machinery v. 42, no. 3.

[SOAP] Simple Object Access Protocol (SOAP) 1.1 (2000). http://www.w3.org/TR/SOAP

[TAO] Distributed Object Computing (DOC) Group. Real-Time CORBA with TAO. http://www.cs.wustl.edu/~schmidt/TAO.html

[Tindell, 1994a] Tindell, K.W., Burns, A., (1994). Guaranteeing Message Latencies on Control Area Network (CAN). Proceedings of 1st International CAN Conference (iCC'94). Can in Automation (CiA) Ed., Mainz.

[Tindell, 1994b] Tindell, K.W., Hansson, H., Wellings, A.J., (1994). Analysing Real-Time Communications: Controller Area Network (CAN). Proceedings of IEEE RealTime Systems Symposium RTSS'94. F. Jahanian and K. Ramaritham, eds., IEEE Computer Society Press, pp. 259-263.

[Winter, 1996] Winter, C.S., Titmuss, R., Crabtree, B., (1996). Intelligent agents, mobility and multimedia information. Proceedings First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, PAAM'96, London.

[Wong, 1999] Wong, D., Paciorek, N., Moore, D., (1999). Java-based Mobile Agents. Communications of the Association of Computing Machinery, ACM v. 42, no. 3, 1999.

[Wooldridge, 1995] Wooldridge, M., Jennings, N.R., (1995). Intelligent agents: theory and practice. The Knowledge Engineering Review, 10(2), 115-152.

[XML] Extensible Markup Language (XML) 1.0 2º edition (2000). http://www.w3.org/TR/REC-XML