La arquitectura Acromovi: Una arquitectura para tareas cooperativas de robots móviles

Patricio Nebot y Enric Cervera

Departamento de Ingeniería y Ciencia de los Computadores
Universidad Jaume I, Castellón, España,
{pnebot, ecervera}@uji.es,
http://www.robot.uji.es/lab/plone/

Resumen Los avances en robótica móvil, poder de computación y comunicaciones inalámbricas han hecho posible el desarrollo de comunidades de robots autónomos. En los últimos años, hay un creciente interés en sistemas de múltiples robots autónomos capaces de llevar a cabo tareas cooperativas. Este proyecto tiene ésto en cuenta y presenta una arquitectura basada en agentes embebidos para el desarrollo de aplicaciones colaborativas para un equipo heterogéneo de robots móviles. La implementación de la arquitectura tiene la capacidad de permitir al equipo de robots el cumplimiento de tales tareas, y también tiene herramientas y características que permiten a los programadores desarrollar aplicaciones complejas en un tiempo razonable. Una plataforma de desarrollo basada en agentes ha sido usada para este propósito, debido a que integra las capacidades necesarias para el desarrollo de aplicaciones distribuidas, y maneja las comunicaciones entre todos los componentes.

Palabras clave: Sistemas multiagente, arquitecturas de control, coordinación de equipos de robots, cooperación, robots móviles

1. Introducción

Este artículo describe la implementación y desarrollo de una arquitectura distribuida para la programación y control de un equipo coordinado de robots móviles heterogéneos, los cuales son capaces de colaborar entre ellos y con personas para el cumplimiento de tareas de servicios en entornos cotidianos.

La presente arquitectura basada en agentes, Acromovi, nació a partir de la idea de que el trabajo en equipo es una capacidad fundamental para un grupo de múltiples robots móviles[1][2].

En los últimos años, hay un interés creciente en el desarrollo de sistemas de múltiples robots autónomos que puedan mostrar un comportamiento colectivo. Este interés es debido al hecho de que tener un único robot con múltiples capacidades puede ser una pérdida de recursos. Diferentes robots, cada uno con su propia configuración, forman un sistema más flexible, robusto y barato. Además,

las tareas a realizar pueden ser demasiado complejas para un único robot, mientras que múltiples robots pueden realizar estas tareas de manera más efectiva[3].

La arquitectura Acromovi es un framework para el desarrollo de aplicaciones por medio de agentes embebidos y agentes interfaz con código nativo de bajo nivel. La arquitectura además implementa la compartición de los recursos de un robot entre todo el grupo. También facilita la cooperación entre los robots de manera que se puedan realizar tareas de modo coordinado. Además, Acromovi es una arquitectura distribuida que trabaja como middleware de otra arquitectura global para la programación de los robots.

Aunque la programación de robots se ha hecho mayoritariamente en C o C++, se ha elegido un sistema de desarrollo de sistemas multiagente basado en Java para el desarrollo de la arquitectura de nuestro equipo de robots[4]. Cualquier sistema de desarollo de sistemas multiagente común podría haber sido elegido, pero se ha optado por implementar la arquitectura por medio de JADE, una herramienta para el desarrollo de sistemas multiagente, implementada en Java, que cumple las especificaciones FIPA.

En la sección 2, se muestra el estado del arte relacionado con las arquitectura de control y sistemas de múltiples robots trabajando cooperativamente. En la sección 3, se describe la arquitectura Acromovi. Primero se verá el diseño lógico, y a continuación se mostraran los diseños e implementación de las dos capas que forman la arquitectura. Finalmente, la sección 5, muestra las conclusiones más relevantes obtenidas de nuestro trabajo.

2. Estado del arte

La investigación en el campo de la robótica móvil cooperativa ha crecido considerablemente en los últimos años[5][6]. Alguno ejemplos de estos trabajos son presentados a continuación.

En un intento de usar técnicas tradicionales de IA, la arquitectura *GOPHER* fue pensada para resolver problemas de una manera distribuida mediante multirobots en entornos internos[7]. Un sistema central de planificación de tareas (CTPS) comunica con todos los robots y dispone de una visón global de las tareas que han sido hechas o de la disponibilidad de los robots para realizar el resto de tareas.

Otro proyecto interesante, *MICRobES*, es un experimento de robótica colectiva que trata de estudiar la adaptación a largo plazo de una micro-sociedad de robots autónomos en un entorno con humanos. Los robots deben sobrevivir en este entorno así como cohabitar con las personas[8].

CEBOT (CEllular roBOTics System) es una arquitectura jerárquica descentralizada inspirada por la organización celular de las entidades biológicas. Es capaz de reconfigurarse dinámicamente para adaptarse a las variaciones del entorno[9]. Está compuesta por çeldas", y en la jerarquía hay çeldas maestras"que coordina las subtareas y se comunican entre ellas.

Con el fin de reutilizar componentes de software nativo, los agentes pueden ser embebidos en un nivel interfaz o middleware. *ThinkingCap-II*[10] es una

arquitectura desarrollada en un proyecto de plataforma distribuida basada en agentes para robots móviles. Incluye agentes híbridos inteligentes, sistema de planificación basado en tracking visual, componentes integrados de visión, y varias técnicas de navegación. Además, ha sido desarrollada sobre una maquina virtual en tiempo real (RT-Java), implementando un conjunto de comportamientos reactivos.

Por otro lado, *SWARM* es un sistema distribuido hecho de un gran número de robots autónomos. A partir de este proyecto, se ha acuñado el término "inteligencia SWARM"para definir la propiedad de los sistemas de múltiples robots no inteligentes que exhiben un comportamiento colectivo inteligente. Es una arquitectura homogénea en la cual la interacción está limitada a los vecinos más próximos[11].

Finalmente, *Miro* es un middleware para crear aplicaciones para robots móviles autónomos. Está basado en la construcción y uso de un middleware orientado a objetos para hacer el desarrollo de aplicaciones de robot móviles más fácil y rápida, y para fomentar la portabilidad y el mantenimiento del software del robot[12]. Este software además proporciona servicios abstractos genéricos, los cuales pueden ser aplicados en diferentes plataformas robóticas sin realizar modificaciones.

Los trabajos anteriores implementan arquitecturas y sistemas para que equipos de robots puedan realizar tareas cooperativas. Nuestro trabajo consiste en la implementación de una arquitectura de este tipo, resaltando la reusabilidad del software, permitiendo al programador la integración de componentes nativos de software (librerías de visión, módulos de navegación y localización). De este modo, los agentes embebidos son la clave para que el rápido desarrollo de potentes aplicaciones distribuidas.

También en este trabajo es importante resaltar otra característica de la arquitectura, La compartición de recursos de un robot entre todo el equipo, y el fácil acceso a los elementos físicos de los robots por parte de las aplicaciones.

3. Arquitectura ACROMOVI

Establecer mecanismos de cooperación entre robots implica considerar un problema de diseño del comportamiento cooperativo. Es decir, teniendo un grupo de robots, un entorno y una tarea a realizar, ¿cómo debe llevarse a cabo la cooperación? Este problema implica muchos retos, pero uno de los más importantes es la definición de la arquitectura que dará soporte a esta cooperación. Los sistemas multiagente son el entorno natural para tales grupos de robots, haciendo posible la rápida implementación de potentes arquitecturas para la especificación y ejecución de tareas.

Diseño de la arquitectura. Los robots que forman el equipo ya disponen de una arquitectura de programación con librerías nativas en C/C++ para todos sus accesorios. Esta arquitectura está formada por dos capas, la capa inferior, ARIA, se encarga de servir las peticiones de los programas a los distintos componentes

físicos de los robots. La capa superior, Saphira, proporciona servicios para la interpretación de sensores de rango, construcción de mapas, y navegación.

Sin embargo, el código nativo existente está orientado al procesamiento desde un único controlador, sin definir mecanismos de colaboración entre robots o aplicaciones. La arquitectura Acromovi trata de superar este problema mediante la introducción de una nueva capa sobre la arquitectura nativa que permita una fácil colaboración y cooperación.

Además, la arquitectura Acromovi es capaz de subsumir cualquier otro software extra, como ACTS o VisLib (librerías de visión). La subsunción de librerías C/C++ es un rápido y poderoso método para el desarrollo de código Java y por tanto agentes embebidos con toda la funcionalidad de las librerías nativas.

Como se puede ver en la figura 1, la arquitectura Acromovi añade una capa middleware entre la arquitectura del robot y las aplicaciones, que permite la colaboración y cooperación de los robots del equipo. Este middleware está implementado siguiendo un enfoque basado en agentes.

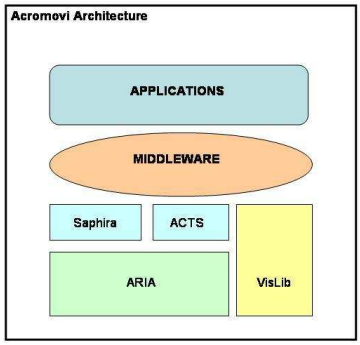
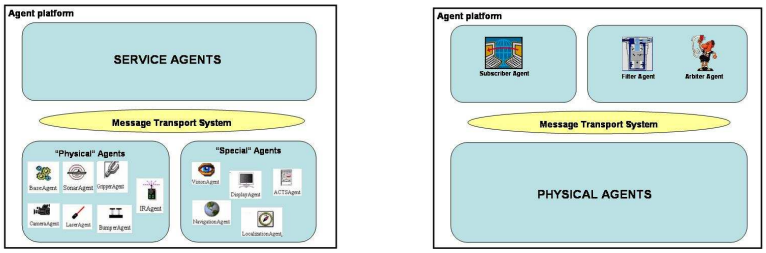


Figura 1. Diagrama general de la arquitectura

Como puede verse en la figura 2, la capa middleware ha sido dividida en dos subcapas. En la capa inferior, hay un conjunto de componentes que se encargan del manejo de los elementos físicos del robot, como el sonar, láser, base, ... El resto son componentes especiales que ofrecen servicios a la capa superior, como visión, navegación o localización.

Estos componentes solamente realizan dos tipos de tareas o funciones, el procesamiento de las peticiones que reciben y el envío de los resultados producidos. Así, éstos podrían haber sido implementados como componentes software, pero debido a razones de eficiencia, facilidad de implementación e integración con la estructura global, se decidió implementarlos como agentes que encapsulan la funcionalidad requerida.

La capa superior de la arquitectura comprende una gran variedad de agentes embebidos y de supervisión, como por ejemplo agentes para monitorizar el estado de los agentes de la capa inferior, agentes que proporcionan servicios de



(a) Capa inferior (b) Capa superior

Figura 2. La capa middleware

subscripción a un determinado componente, agentes que arbitran o controlan el acceso a un componente, y cualquier otro tipo de agente útil para la arquitectura o para una determinada aplicación. Además, estos agentes actúan como enlace entre las aplicaciones y el acceso a los componentes físicos del robot.

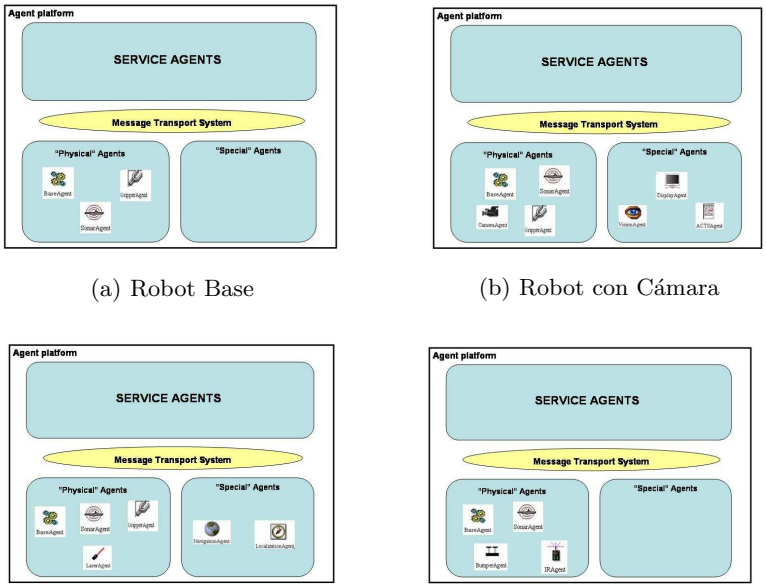
Cabe destacar que debido al carácter heterogéneo del equipo de robots y dependiendo de la configuración de cada robot en particular, pueden haber diferentes capas middleware para los robots. Estas capas middleware son generadas en tiempo de ejecución de acuerdo con los elementos que el robot tenga activos en el momento de su ejecución o puesta en funcionamiento. Estas diferentes configuraciones pueden verse en la figura 3.

Finalmente, sobre la capa middleware está la capa de aplicaciones, que también pueden ser implementadas como agentes. Estas aplicaciones, para acceder a los componentes físicos de los robots, deben comunicar con los agentes del middleware, los cuales luego acceden a la capa nativa que es la que controla a nivel físico el robot.

Una característica muy importante de la arquitectura creada es la escalabilidad de la arquitectura. Es decir, que la arquitectura puede crecer de una manera rápida y sencilla. Una vez una aplicación ha sido testada, si es útil para la arquitectura en general, ésta puede ser fácilmente convertida en un nuevo agente de la capa superior del middleware. A partir de ese momento, este nuevo agente se encargaría de ofrecer servicios, que puede ser de gran ayuda, a otras aplicaciones que se puedan crear. De este modo, cada aplicación que creemos puede incrementar nuestra arquitectura para poder crear aplicaciones cada vez más complejas y más interesantes, siguiendo un diseño "bottom-up".

Implementación de la arquitectura. Dado que el middleware es simplemente un conjunto de agentes embebidos, una herramienta para la programación de sistemas multiagente ha sido seleccionada para su implementación.

La herramienta elegida ha sido JADE (Java Agent DEvelopment Framework). Es un framework para el desarrollo de aplicaciones basadas en agentes de acuerdo con las especificaciones FIPA. Está totalmente implementado en Java y



(a) Robot Base (b) Robot con Cámara

(c) Robot con Láser (d) Powerbot Base

Figura 3. Diferentes capas middleware.

simplifica la implementación de sistemas multiagente por medio de un middleware y un conjunto de herramientas gráficas que soportan las fases de desarrollo y depurado. El middleware de JADE implementa una plataforma para la ejecución de agentes y un framework para desarrollo de las aplicaciones. Además, proporciona facilidades para el agente como manejo del ciclo de vida, servicio de nombres y de páginas amarillas, transporte de mensajes y una librería de protocolos de interacción FIPA listos para ser usados[13].

La arquitectura Acromovi se ha implementado siguiendo las especificaciones de JADE. Así, cada robot involucrado en la arquitectura es un contenedor principal. Cada uno de estos contenedores funciona como host de una red distribuida. En cada uno de los contenedores hay un grupo de agentes. Estos agentes son creados en tiempo de ejecución dependiendo de la configuración del robot, como ya se dijo anteriormente. Cada uno de estos agentes representa uno de los elementos que el robot tiene activos en ese momento. Esto puedo verse en la figura 4.

Como se ha dicho, los elementos del robot están representados y manejados por agentes. Una breve explicación de los agentes que forman la arquitectura, tanto de la capa inferior como de la capa superior, se muestra a continuación.

Los agentes que forman la capa inferior del middleware han sido conceptualmente divididos en 3 grupos, dependiendo de la parte en la que el agente realiza su tarea. Así, hay "body agents"(base, gripper, sonar, bumper y IR agents),

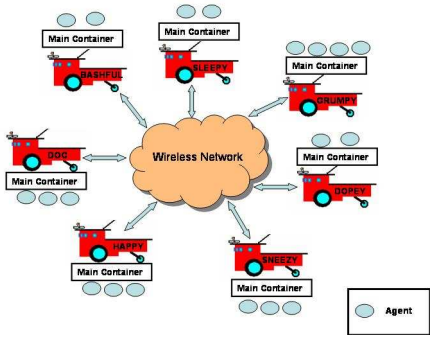


Figura 4. Estructura de la implementación

"laser agents"(laser, localization y navigation agents) y "vision agents"(camera, ACTS, vision y display agents).

Los "body agents"son los agentes necesarios o básicos para el funcionamiento del robot y corresponden con los principales componentes físicos de los robots del equipo. **Base Agent**, se encarga de todo lo relacionado con el movimiento de los motores y el estado interno del robot. **Gripper Agent**, agente encargado del buen funcionamiento de la pinza y de permitir a otros agentes la manipulación de ésta. **Sonar Agent**, encargado de que el sónar funcione correctamente y de devolver los valores apropiados cuando éstos son requeridos por otros agentes o aplicaciones. **Bumper Agent**, agente que maneja los bumpers del robot, y devuelve el valor de éstos cuando es requerido. **IR Agent**, agente dedicado al manejo de los 4 sensores de infrarrojos equipados en uno de los robots. Estos sensores tienen la función de prevenir al robot de escaleras o agujeros. El agente se dedica a avisar del estado de estos sensores cuando es necesario.

Los "laser agentsçomunican con el dispositivo láser y permiten el uso y manejo de dos módulos muy importantes para un robot móvil, la localización y la navegación. **Laser Agent** permite múltiples operaciones con el láser y se encarga de su correcto funcionamiento. **Localization Agent** se encarga de localizar al robot en un entorno conocido. **Navigation Agent** calcula el mejor camino a seguir entre dos puntos y mueve el robot por ese camino.

Los "vision agents"se encargan de capturar imágenes a través de la cámara del robot y realizar operaciones con esas imágenes. **Camera Agent** mueve físicamente la cámara e informa sobre su estado. **ACTS Agent** se usa para el tracking por color de objetos. **Vision Agent** captura las imágenes, y puede visualizarlas o modificarlas, mediante el uso de la librería Vislib. **Display Agent** modifica y procesa operaciones sobre las imágenes capturadas por el anterior agente, mediante el uso de las librerías Java 2D.

A continuación se describen los agentes de la capa superior. Estos agentes mantienen un carácter genérico, es decir, pueden tomar diferentes formas dependiendo del agente de la capa inferior al que sirven. Así, podría existir más de un agente del mismo tipo pero sirviendo a diferentes agentes de la capa inferior.

Estos agentes también han sido conceptualmente divididos en 2 grupos. Por un lado, hay un agente genérico encargado de controlar aquellos agentes de la capa inferior que pueden servir información o datos de una manera continua. El agente genérico encargado de ésto se llama **Subscriber agent**. Por otro lado, hay dos agentes encargados del control de los agentes de la capa inferior que puedan tener problemas de concurrencia en el acceso. Los agentes de la capa inferior candidatos a usar agentes de este tipo son aquellos que pueden mover físicamente alguna parte del robot, como puede ser la base, la pinza o la cámara. Los dos agentes encargados de su control son **Filter agent** y **Arbiter agent**. El funcinamiento específico de estos agentes se ve en profundidad en la siguiente sección.

4. Protocolos de interacción

Como muchos otros sistemas multiagente, Acromovi está compuesto por múltiples agentes interactivos. Estos agentes necesitan un modo de comunicarse entre ellos con el fín de alcanzar sus objetivos o aquellos de la sociedad en la que interactúan. Las herramientas que posibilitan esta comunicación entre los diferentes agentes de un sistema son los protocolos.

Un protocolo es simplemente un conjunto de reglas usadas para hacer posible una comunicación. En el dominio de los sistemas multiagente, un protocolo es un conjunto de reglas que guian la interacción que tiene lugar entre varios agentes y que gobiernan como la información es entregada. Estas reglas definen que mensajes son posibles para cada estado particular de interacción. Esa interacción tiene que ser realizada de tal forma que los agentes intercambien información. La interacción hace posible que los agentes puedan cooperar y coordinarse para conseguir realizar sus tareas.

Existen dos tipos diferentes de protocolos para ser definidos en un sistema multiagente, los protocolos de comunicación y los protocolos de interacción. Los protocolos de comunicación habilitan a los agentes para poder intercambiar y comprender los mensajes. Los protocolos de interacción se definen como series de actos comunicativos, formando conversaciones, para poder conseguir alguna forma de coordinación entre los agentes.

Como la arquitectura Acromovi ha sido implementada por medio de JADE, los protocolos de comunicación que se usan son los que proporciona, tales como TCP/IP, HTTP, RMI, ... La ventaja que ofrece JADE es que permite un uso transparente de estos protocolos. Además de que incorpora un mecanismo de transporte que se comporta como un camaleón, adaptando a cada situación el mejor protocolo de los disponibles, todo de forma transparente para el programador.

Por el otro lado, están los protocolos de interacción. Se utilizan los propuestos por FIPA, que además están implementados en las propias librerias de JADE. Algunos de estos protocolos son los protocolos Çontract Net", .$^{\text{En}}$glish Auction.$^{\text{o}}$ "Dutch Auction". Pero estos protocolos resultan demasiado generales y no tienen en cuenta todas las características de la arquietctura Acromovi. Por tanto, se

han implementado dos nuevos protocolos, que añaden nuevas funcionalidades al sistema.

4.1. Protocolos de interacción

Los protocolos de interacción son series de actos de comunicación para conseguir alguna forma de coordinación entre los agentes. FIPA propone un conjunto de protocolos de interacción para sistemas de multiagente. JADE define una biblioteca con éllos, listos para usarse. Pero estos protocolos son generales y no tratan con algunas de las características del sistema. Por tanto, se han implementado dos nuevos protocolos que añaden nuevas funcionalidades al sistema.

El primer protocolo implica una modificación del protocolo de suscripción. En el protocolo de interacción FIPA para suscripción, un agente solicita tener notificaciones regulares de la información requerida. En el nuevo protocolo implementado, este enfoque cambia.

En este protocolo hay implicados dos tipos de agentes. Por una parte, los *suscriptores* son esos agentes que quieren tener un flujo regular de información de un elemento del robot, tales como los valores de las lecturas del sonar y el láser o un flujo continuo de imagenes provenientes del sistema de visión. Por otro lado, los *proveedores* son aquellos agentes que sirven los flujos de información. Estos agentes están estrictamente unidos a los agentes que manejan los elementos del robot.

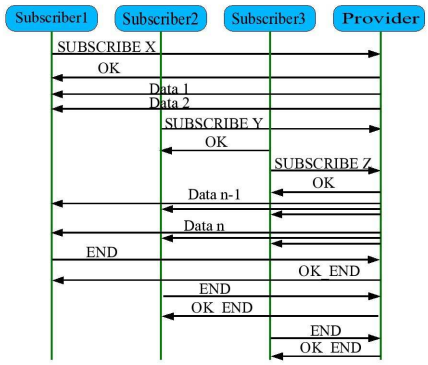


Figura 5. Protocolo de suscripción

Los agentes proveedores solicitan continuamente información a los agentes a los que están unidos. Así siempre tienen los valores actuales de estos agentes. La frecuenacia a la cual los agentes proveedores solicitan la información a los agentes controladores de los elementos viene dada por la frecuencia real a la que los elementos pueden servir la información.

El funcionamiento del protocolo puede verse en la figura 5. Los agentes suscriptores en el momento de la suscripción indican con qué frecuencia quieren

recibir los datos del agente proveedor ("SUBSCRIBE X"). El agente proveedor les sirve los datos a cada agente suscriptor en el momento adecuado ("DATA n"). Así, si un suscriptor quiere obtener los datos solamente cada dos veces que el sonar genera datos, el agente proveedor le manda al agente suscriptor los datos cada dos lecturas del sonar. El agente proveedor mantiene una lista activa en la que se indica el agente que ha hecho cada petición y a que frecuencia quiere obtener los datos.

El segundo protocolo implementado es completamente nuevo. Está pensado para controlar el acceso a los elementos problemáticos. ¿Cuáles son estos elementos problemáticos? Son todos esos elementos que pueden tener un funcionamiento erróneo si uno o más agentes pueden acceder y ejecutar operaciones al mismo tiempo en ellos. Estos elementos son la base, la pinza, la cámara.. Así, con este protocolo se intentan resolver los problemas de concurrencia. Otra característica importante de este protocolo es que previene que los agentes realicen operaciones con los elementos problemáticos que puedan poner en peligro al robot. Por ejemplo, si un agente quiere mover el robot dos metros, y hay una pared a un metro de distancia.

El funcionamiento conceptual del protocolo se puede ver en la figura 6. En este protocolo hay cuatro agentes implicados: el solicitante, el árbitro, el filtro, y el agente que maneja el elemento problemático. El agente solicitante es el agente que quiere utilizar el elemento problemático. Este agente debe solicitar permiso al agente árbitro para poder utilizar el elemento (REQUEST"). Así, cada agente que quiera utilizar un elemento problemático debe solicitar permiso al agente árbitro correspondiente.

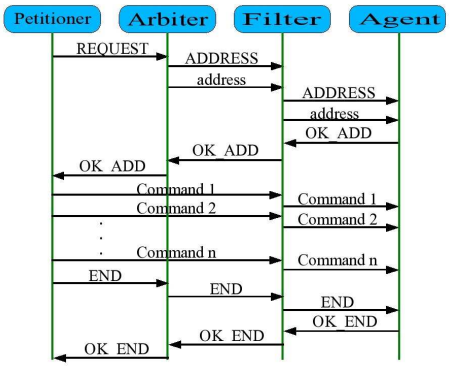


Figura 6. Protocolo de permiso

El agente árbitro mantiene una cola con las peticiones que han hecho los agentes solicitantes y va dando permiso siguiendo una estrategia FIFO. Cuándo el permiso es dado, el agente solicitante puede utilizar el elemento hasta que termine su tarea.

Cuándo el agente árbitro acepta la petición de un determinado agente, manda un mensaje con la dirección del agente solicitante al agente filtro indicando que agente que tiene el permiso (.ªDDRESS",.ªddress"). El agente filtro, cuando recibe ese mensaje, manda un mensaje igual al agente que se encarga del funcionamiento del elemento, para informarle acerca de que agente tiene el control del elemento y a cuál debe mandar las respuestas de las órdenes ejecutadas.

Cuándo el agente que maneja el elemento recibe la dirección del agente solicitante, manda un mensaje al agente filtro confirmando que está listo para recibir las órdenes (.ºK_ADD"). El agente filtro le manda otro mensaje al agente árbitro y finalmente, el agente árbitro le manda un mensaje al agente solicitante para confirmarle que tiene el permiso para utilizar el elemento.

Desde este momento, el agente solicitante puede mandar tantos mensajes como quiera hasta terminar su tarea (çommand n"). Los mensajes deben ser dirigidos al agente filtro, y éste los manda al agente del elemento si no son peligrosos para el robot. Así, el agente filtro se encarga de mantener la integridad de los robots. Si la orden mandada al agente filtro es conflictiva, el agente filtro no la manda al agente del elemento, y manda una notificación del error al agente solicitante.

Cuándo el agente solicitante termina su tarea, manda un mensaje al agente árbitro indicandole que ha terminado de usar el elemento (.ENDª). El agente árbitro manda un nuevo mensaje al agente filtro para informarle que el agente ha terminado y que debe esperar una nueva petición. El agente filtro manda un mensaje al agente del elemento para confirmarle que el agente solicitante ha terminado su trabajo. El agente del elemento manda un mensaje nuevo que confirma que ha recibido el mensaje al agente filtro (.ENDª_OK"). Este le manda un mensaje al árbitro como confirmación. Y el árbitro le manda un mensaje al agente solicitante para confirmarle que ya no tiene el permiso.

Ahora, el agente árbitro toma la suiguiente petición de la cola o espera hasta que llegue una nueva.

5. Conclusiones

La principal conclusión que se puede obtener es que se ha logrado que la compartición de recursos de un robot entre el resto de robots del equipo se haga de una manera fácil y sencilla. Otra gran ventaja de la arquitectura es que permite la reutilización, mediante el uso de componentes nativos, por medio de agentes que proporcionan al programador un enorme conjunto de herramientas para robots móviles.

Por otro lado, se puede resaltar la escalabilidad del sistema, ya que una vez una aplicación ha sido testada, puede ser convertida en un nuevo agente e incrementar así la arquitectura.

Además, la arquitectura permite un rápido desarrollo de aplicaciones multirobot. En solo unas pocas semanas, estudiantes con bajos conocimiento de Java han sido capaces de desarrollar y programar agentes de diversas utilidades.

Por último, cabe destacar que diversas aplicaciones han sido implementadas para testar la arquitectura. Algunas de estas aplicaciones son el seguiemiento de robots por medio de visión, la navegación de robots usando la técnica del flujo óptico, y varias aplicaciones menores de depuración del sistema. Hay que remarcar que en todas estas aplicaciones, la arquitectura se ha comportado de la forma esperada, confirmando que es apta para la utilidad que se le planteó inicialmente.

Agradecimientos

Este trabajo ha sido financiado en parte a cargo del proyecto GV05/137 de la Generalitat Valenciana, y en parte a cargo del proyecto DPI2005-08203-C02-01 del Ministerio de Educación y Ciencia.

Referencias

1. Jung, D., Zelinsky, A.: An architecture for distributed cooperative planning in a behaviour-based multi-robot system. Journal of Robots and Autonomous Systems **26** (1999) 149–174
2. Mataric, M.J.: New directions: Robotics: Coordination and learning in multirobot systems. IEEE Intelligent Systems **13** (1998) 6–8
3. Arai, T., Pagello, E., Parker, L.E.: Guest editorial: Advances in multirobot systems. IEEE Transactions on Robotics and Automation **18** (2002) 655–661
4. Nebot, P., Gomez, D., Cervera, E.: Agents for cooperative heterogeneous mobile robotics: a case study. In: Proc. IEEE Int. Conf. on Systems, Man and Cybernetics. (2003)
5. Parker, L.: Distributed autonomous robotic systems 4. In: Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS2000). (2000) 3–12
6. Cao, Y., Fukunaga, A., Kahng, A.: Cooperative mobile robotics: Antecedents and directions. Autonomous Robots **4** (1997) 1–23
7. Caloud, P., et al.: Indoor automation with many mobile robots. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems(IROS). (1990)
8. Picault, S., Drogoul, A.: The microbes project, an experimental approach towards open collective robotics. In: Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS'2000). (2000)
9. Fukuda, T., Iritani, G.: Construction mechanism of group behavior with cooperation. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems(IROS). (1995)
10. Cáceres, D., Martínez, H., Zamora, M., Balibrea, L.: A real-time framework for robotics software. In: Int. Conf. on Computer Integrated Manufacturing (CIM-03). (2003)
11. Johnson, J., Sugisaka, M.: Complexity science for the design of swarm robot control systems. In: 26th Annual Conference of the IEEE Industrial Electronics Society (IECON). (2000)
12. Enderle, S., Utz, H., Sablatng, S., Simon, S., Kraetzschmar, G., Palm, G.: Miro: Middleware for autonomous mobile robots. In: IFAC Conference on Telematics Applications in Automation and Robotics. (2001)
13. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: Jade a white paper. EXP in search of innovation (Special Issue on JADE) **3** (2003) 6–19