Plataforma genérica para desarrollos basados en agentes móviles

Guillermo Glez. de Rivera, Ricardo Ribalda, José Colás, Javier Garrido
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
28049 Madrid
guillermo.gdrivera@uam.es, ricardo.ribalda@uam.es, jose.colas@uam.es, javier.garrido@uam.es

Resumen

En este trabajo se presenta una plataforma de desarrollo genérica, fácilmente modificable, ampliable y abierta, para la prueba y evaluación de sistemas basados en agentes móviles. Los agentes se pueden interconectar mediante gran cantidad de medios físicos como GSM, *Wireless*, sistemas de radio a 433 o 868 MHz, *BlueTooth,* etc, y pueden ser controlados y programados desde cualquier sistema operativo o lenguaje de programación. Su control se puede realizar desde la *web*. Los agentes móviles se pueden adaptar a cualquier entorno, añadiéndose de forma sencilla los nuevos periféricos requeridos. [1], [2] y [3]

1. Introducción

La Inteligencia Ambiental se caracteriza ser un área de trabajo especialmente interdisciplinario, en la que se pueden distinguir dos grandes campos de trabajo. El primero de ellos estaría involucrado en la creación de los elementos hardware necesarios para poder estar frente a un "entorno inteligente", tales como sistemas de comunicación, sistemas biométricos, sistemas de localización... etc. El segundo de ellos se encargaría del desarrollo de los algoritmos y aplicaciones software necesarios para dar funcionalidad al hardware anteriormente mencionado.

Esta amplia interdisciplinaridad necesaria para crear sistemas que ayuden a realizar las tareas diarias a los usuarios, acarrea graves problemas a la hora de desarrollarlos. Por un lado los equipos de investigación no suelen estar especializados en todas las áreas de conocimiento necesarias, lo que hace que las investigaciones acaben en meros algoritmos funcionando sobre un simulador, en hardware inservible por si mismo o, en el peor de los casos, en meras especulaciones. Por otra parte, si el equipo se decide por realizar el trabajo completo para el desarrollo del sistema, se enfrenta a muchos problemas en múltiples campos, lo que supone una gran infraestructura y tiempo de desarrollo.

El presente trabajo describe una plataforma funcional sobre la que poder trabajar sobre entornos inteligentes centrándose en problemas concretos de software, hardware o comunicaciones y olvidándose de los demás, pues ya están implementados y funcionando en la plataforma.

Entre las funcionalidades que ofrece esta plataforma están:

- Acceso al Hardware mediante simples rutinas
- Agentes Móviles (Robots) baratos, extensibles y abiertos
- Adición de nuevo Hardware mediante una sencilla interfaz (Servidor Sensorial)
- Intercomunicación de los agentes desde prácticamente cualquier medio físico
- Acceso al sistema desde cualquier sistema operativo/ lenguaje de programación
- Fácil integración con la *web*

Desde un punto de vista más amplio podemos ver el sistema como un punto de encuentro entre los distintos equipos de investigación, pues permite compartir desarrollos de una forma sencilla, para así crecer entre todos. A todo esto se suma que la plataforma en su totalidad está basada en estándares y herramientas libres. De esta manera todos los desarrollos realizados para esta plataforma se podrían portar a futuras plataformas abiertas de forma sencilla.

Por último, el sistema se puede ver como un gran *Middelware* de agentes/sensores al servicio del usuario [4].

Como ejemplo de uso de la plataforma que se describe en este trabajo, los robots pueden ser utilizados para vigilancia de edificios. Varios robots recorren cada planta, comprobando eventos tales como intrusos, fuego, humedad, fugas de gas, etc. En caso de alerta pueden actuar de la forma programada o bien enviar un aviso a la persona responsable. En cualquier momento (vía web, wap, sms, etc) se puede monitorizar el estado de cada uno de ellos pudiendo incluso ser controlado de forma remota. Para ello se puede utilizar un PC, un teléfono móvil, una PDA, etc.

2. Arquitectura Hardware

En la figura 1 se muestra un esquema del conjunto completo. Básicamente consiste en un conjunto de agentes o robots que actúan como servidores, pudiéndose comunicar entre ellos, y un conjunto de clientes que utilizarán a los primeros. La comunicación servidor-servidor y servidor-cliente se puede realizar por cualquier tipo de red, incluido Internet. [5] [6]

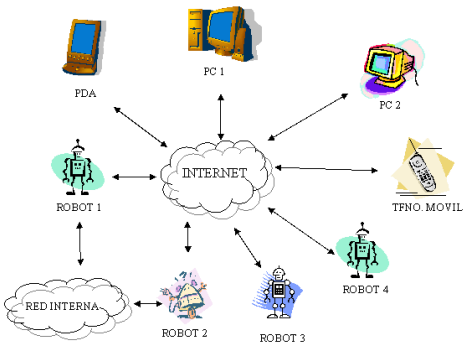

Figura 1. Arquitectura del Sistema

En este capitulo se discuten los elementos hardware que componen los agentes.

2.1. Hardware

El sistema está montado sobre una rígida estructura de aluminio. La flexibilidad que permite el montaje facilita ajustarse de una manera muy eficaz a cada aplicación.

Como mecanismo de tracción se ha elegido un sistema de tres ruedas, dos de ellas motrices. Cada rueda motriz está controlada con un potente motor paso a paso, lo que permite un posicionamiento muy fino del robot. Para obtener el máximo rendimiento de los motores y liberar al sistema central del control de las ruedas se ha utilizado un microprocesador especializado, modelo M3410 CP Pilot Motion [7], para el que se ha desarrollado una librería especializada para ser utilizada desde Linux. El acceso a dicho microcontrolador se realiza a través de un puerto USB.

El control central está basado en la arquitectura del PC. Se utiliza una placa base VIA EPIA M10000, con un microprocesador Via C3/EDEN [8].

Para la comunicación entre robots se han implementado una gran cantidad de elementos de red, tales como Ethernet, Wireless, puertos serie y paralelo, USB, Bluetooth, GSM/GPRS y sistemas de radio en la banda de 433 ó 868MHz. El modo principal utilizado es Wireless, ya que es el que mejor se ajusta a la mayoría de las necesidades. El resto se han incluido tanto para hacer el sistema más genérico como para poder añadir cierta redundancia.

Figura 2. Fotografía del Agente

Para la conexión interna de los diferentes elementos, como sensores o actuadores, se utiliza exclusivamente el puerto USB, debido principalmente a que es un estándar soportado por la mayoría de los periféricos, bastante extendidos,

ser de bajo coste y que en su mayoría no necesitan de una fuente de alimentación externa.

También se ha incluido una cámara de vídeo USB, montada sobre un sistema que permite moverla tanto en el plano vertical como en el horizontal, controlado por dos servomotores. La cámara se conecta a un puerto USB de la placa base y los servomotores están controlados por un Servidor Sensorial, descrito a continuación. En la figura 2 se muestra un agente.

2.2. Servidor Sensorial

En algunos casos un sensor determinado puede necesitar mucha atención para llevar a cabo su tarea, o simplemente el sistema original no contempla un tipo de sensor en concreto. Para estos casos se ha incluido un pequeño sistema de control autónomo, basado en microcontrolador, que actúa como interfaz entre el sistema de control central y el sensor, de manera que a través de una sencilla petición puede gobernar y controlar las demandas del sensor sin mantener ocupado al sistema central. Estos pequeños sistemas de control se denominan Servidores Sensoriales (SS) y habrá tantos como sean necesarios. La conexión al sistema central se realiza a través de USB. En el sistema descrito en este trabajo los SS están basado en la plataforma de desarrollo GP_Bot[9]. Una visión global del SS se muestra en la figura 3.

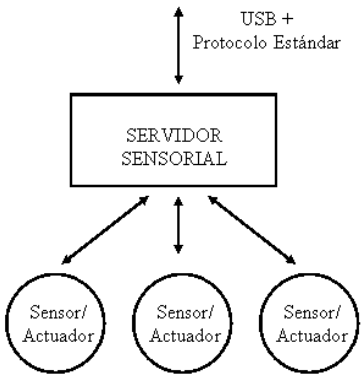

Figura 3. Servidor Sensorial

Para estandarizar la forma de acceso al Servidor Sensorial se ha definido un protocolo de comunicaciones. En cada uno de estos servidores se ejecuta una aplicación que atiende cualquier demanda. El servidor toma el control de la operación y devuelve la lectura correspondiente cuando esté lista.

Este protocolo permite conectar cualquier elemento inteligente, basado en cualquier microprocesador o microcontrolador. Debido al uso de USB se podrán conectar tantos SS como sean necesarios.

3. Arquitectura Software

Mediante un conjunto de agentes como los descritos en el punto anterior se puede construir un sistema que permite llevar a cabo tres tipos de acciones diferentes:

- Supervivencia: El robot debe ser una entidad autónoma con la posibilidad de tomar decisiones por él mismo, por ejemplo evitar obstáculos.
- Control Remoto: Debe ser posible controlar al robot de manera remota, por ejemplo moverlo siguiendo un determinado camino, o mostrarnos las lecturas que esté haciendo del entorno, como transmitir las imágenes que esté captando.
- Cooperación: cada robot debe ser capaz de interactuar con cualquier otro para llevar a cabo una única tarea, por ejemplo trazar un plano de un recinto.

Para resolver esto se utiliza un único servidor por agente que distribuye las tareas a cada elemento de control. No tendrá que hacer frente a los problemas de competición por el hardware ni se duplicarán esfuerzos en el desarrollo del mismo.

El servidor podrá funcionar como un CGI mantenido por un servidor *web* (como *Apache*) o como un servidor en sí mismo.

3.1. Procesos Remotos

Como primera opción hay que definir la forma de comunicación con el servidor. Se necesita un sistema simple, pues el objetivo final del trabajo es ofrecer una plataforma sencilla de utilizar y de ampliar. Debe soportar diferentes lenguajes de programación estándar para no forzar al desarrollador a aprenderse uno nuevo, sino que podrá seguir utilizando el que use habitualmente, así como estar abierto a otros nuevos. Finalmente

debe soportar diferentes sistemas operativos de una forma transparente.
Las alternativas más maduras que existen en la actualidad son: Corba, DCOM, .NET Remoting, Soap, RMI, XML-RPC.

- Corba: Es un lenguaje popular para escribir software distribuido orientado a objetos, está muy soportado y posee una muy buena IDL (*Interface Definition Language*), pero es muy complejo, requiere clientes muy sofisticados y es muy difícil implementarlo.
- DCOM: Es la respuesta de Microsoft a Corba, es más fácil de usar, sin embargo, sólo funciona en MS-Windows.
- .NET Remoting: Al igual que DCOM, es una alternativa de Microsoft a la realización de procedimientos remotos. Sin embargo, su uso está prácticamente restringido a su plataforma .NET.
- Soap: Se basa en XML+HTTP, pero su especificación no es tan buena como debiera, además posee elementos innecesarios.
- RMI: Es el sistema de distribución de software de Java. Es muy potente y muy fácil de usar, sin embargo sólo puede ser usado desde Java, por lo que no es una buena opción para nuestro sistema, que pretende todo lo contrario, ser lo mayormente soportado posible.

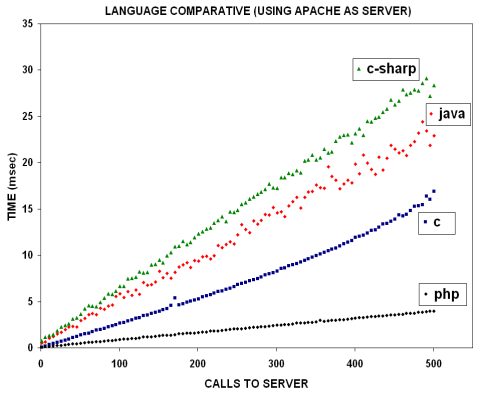

Figura 4. Comparativa de diferentes lenguajes

La elección final ha sido el uso de XML-RPC [2], basado en el protocolo HTTP para el transporte y en XML para el código. Es, por tanto, un sistema estándar y fácil de utilizar. Así mismo, tiene implementaciones para la mayoría de los lenguajes de programación. En el caso de utilizar un lenguaje que no soporte XML-RPC se puede implementar desde cero de una manera simple, pues las especificaciones del estándar están perfectamente documentadas.

Actualmente hay disponibles más de 100 implementaciones oficiales que dan soporte a más de 40 lenguajes de programación [10]. En la figura 4 se puede ver la velocidad de tratamiento de peticiones en función del lenguaje, se han elegido los lenguajes más representativos.

Otra gran ventaja de XML-RPC es que soporta Reflexión, es decir, los servicios que es capaz de ofrecer pueden ser auto-descritos.

Finalmente, destacar la mayor de las ventajas: abrir a los robots a la Web. Hoy en día, XML-RPC es el protocolo que más se utiliza para intercomunicar los sistemas Web.

Como ejemplo de aplicación, el algoritmo 1 muestra un trozo de código C# para acceder a un agente y controlar el movimiento de las ruedas.

```
using CookComputing.XmlRpc;
using System;

[XmlRpcUrl("http://192.168.0.1:8080/RPC2")]
class Proxy : XmlRpcClientProtocol   {

   [XmlRpcMethod("motor.velocity")]
   public int Mover(int motor, int
velocity, int aceleracion, int
deceleracion)
   {
   return (int)Invoke("Mover",
   new Object[]{ motor, velocity,
aceleracion, deceleracion });
   }

   [XmlRpcMethod("motor.stop")]
   public int Stop()
   {
   return (int)Invoke("Stop", new
Object[]{1});
        }
}
```

Algoritmo 1. Código de acceso a los motores del agente

3.2. Transporte

Una vez que se ha definido el tipo de servidor y las comunicaciones con él, se debe decidir la manera de procesar o transportar las peticiones.
La respuesta a esto es simple: TCP/IP.

TCP/IP opera sobre multitud de soportes físicos y puede ser encapsulado de manera sencilla en otros protocolos, como ATM. El hecho de funcionar sobre múltiples soportes físicos es de gran utilidad pues en robótica hay algunos escenarios donde está totalmente descartado el uso de un medio físico en particular y otros en los que las condiciones de trabajo imponen el uso de uno determinado.

En el diseño realizado, se da soporte a los siguientes medios físicos: *Ethernet, Wireless 802.11*, puerto serie, puerto paralelo, módem *GSM, USB, Bluetooth* e *IrDa*. El uso de cualquier otro tipo no implica cambios en el diseño debido a luso de TCP/IP.

3.3. Sistema Operativo

Una vez decidido el protocolo de transporte de alto nivel y el procedimiento para las llamadas remotas, el siguiente paso es la elección del sistema operativo.

Puesto que la interoperabilidad está garantizada, debido a que las comunicaciones se realizan a través de XML-RPC, se de deben analizar otros elementos, tales como la conectividad y la compatibilidad con diferentes arquitecturas hardware. La opción elegida ha sido utilizar Linux [11].

3.4. Especificaciones Finales

Finalmente, las especificaciones software se describen en la tabla siguiente:

Arquitectura	Monoservidor
RPC	XML-RPC
Transporte	TCP/IP
Sistema Operativo	Linux

4. Diseño de la Plataforma

La plataforma está formada por dos elementos: Cliente y Servidor. El cliente realiza las peticiones y supervisa los servidores a través de llamadas XML-RPC. Si un cliente no es capaz de utilizar el medio físico de la red del servidor envía las peticiones a través de un *Bridge*. El servidor está instalado en los robots y el cliente puede estar tanto en los robots como en cualquier otro equipo. Los clientes instalados en los robots se usan para tareas de supervivencia, como evitar choques, y para realizar tareas de cooperación entre ellos. Es decir, el servidor maneja el hardware y el cliente desarrolla el trabajo lógico.

4.1. El cliente

El cliente es un programa que realiza las peticiones al servidor o bien lo supervisa. Está formado por tres partes:

- Aplicación: Lo crea el usuario final en el lenguaje que él decide. Debe ser capaz de ejecutarse en cualquier sistema operativo.
- Librería del robot: Están desarrolladas como parte de la plataforma para algunos lenguajes de programación. Simplemente abstrae al usuario del protocolo XML-RPC.
- Librería XML-RPC: Desarrollada por terceras personas. Implementa el protocolo XML-RPC. Debida a que está basado en estándares abiertos, es posible desarrollarla desde cero, si fuera necesario.

4.2. El servidor robótico

El servidor está instalado en el robot. Básicamente está compuesto por los siguientes elementos:

- Interfaces de red: Permiten al robot comunicarse en diferentes redes con distintos medios físicos.
- Servidor WEB: Su función es negociar y procesar las peticiones HTTP, utilizando CGI, trasmite dichas peticiones al Servidor Robótico.
- Servidor Robótico: Se ejecuta a través de CGI y su función es analizar las peticiones del servidor Web. Las procesa y devuelve la respuesta a la página Web desde donde se realizó dicha petición. Arbitra el acceso de diferentes dispositivos para evitar conflictos. Además, distribuye las peticiones para cada driver, los cuales realizan el acceso a bajo nivel de cada dispositivo.

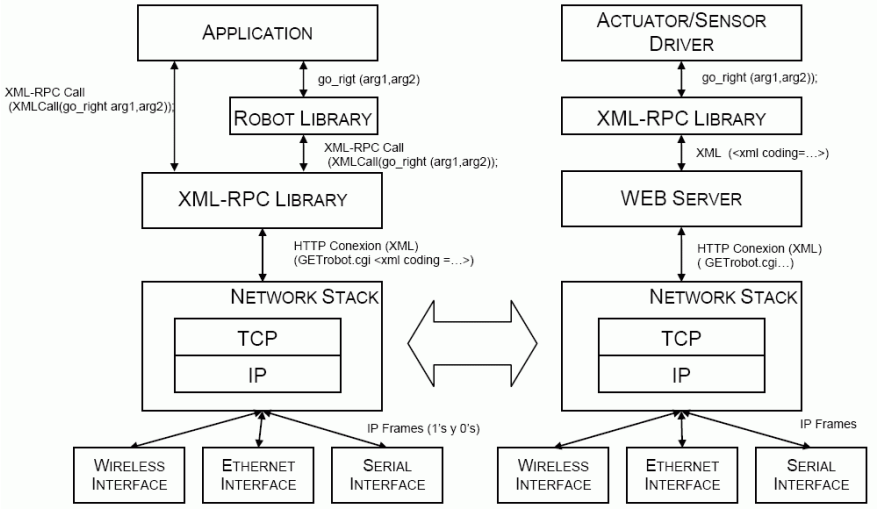

Figura 5. Pila de Comunicación

4.3. Pila de Comunicación

Una vez descrita la arquitectura, se describe a continuación cómo se procesa una petición por los distintos elementos en la plataforma. El camino recorrido puede ser visualizado en la figura 5.

1. Una aplicación decide transmitir una petición, para lo que usa una librería robótica proporcionada por nosotros o, en el caso que no este disponible, la librería XML-RPC. Si la petición es errónea se devuelve un error.
2. La petición es transformada en un documento XML que será transmitido usando en protocolo XML-RPC.
3. La comunicación se divide en tramas TCP/IP que serán transmitidas a través de una de las diferentes redes en función del destinatario del mensaje u otros parámetros. En el caso de pérdidas de tramas TCP retransmitirá la trama para obtener transparencia semántica. En el caso de que fuera necesario, un *bridge* retransmitirá la petición a otro medio físico, sin variar su contenido.
4. Una vez que las tramas hallan llegado a su destino, se reordenarán y se decodificarán de nuevo en una petición HTTP que será transmitida al servidor WEB
5. El servidor Web identificará el mensaje y lanzara el servidor robótico a través de CGI.
6. El servidor robótico obtiene un XML que es analizado a través de la librería XML-RPC.
7. El servidor, a través de los distintos *drivers*, realiza la petición a los distintos elementos hardware.
8. La respuesta obtenida se transformará de nuevo en un documento XML usando la librería XML-RPC.
9. El documento XML es transmitido al servidor Web que lo retransmitirá al cliente siguiendo los pasos anteriormente descritos en orden inverso.

5. Conclusiones y Trabajos Futuros

Uno de los problemas a resolver en el futuro es la transmisión de grandes cantidades de datos entre sistemas, puesto que XML/RPC no es especialmente eficiente en estos casos. Para esto se están estudiando dos soluciones: comunicación a través de RTP o HTTP sin sobrecarga de XML/RPC. La ventaja de RTP es que soporta múltiples destinos pero la implementación en el servidor y en el cliente es muy compleja. Usando HTTP sin sobrecarga la implementación es más sencilla pero hace un uso poco eficiente de los recursos y con sobrecarga de red responde peor que RTP.

Con HTTP sin sobrecarga se han realizado pruebas satisfactorias, figura 6. La prueba realizada fue una cámara en el frente del robot que

permitía el guiado por una habitación. Los datos de la cámara eran transmitidos a través de HTTP, de modo que podían ser visualizados desde cualquier navegador web. El robot era controlado a través de un *Canvas* para que el movimiento fuera lo más intuitivo posible para el usuario. La aplicación de control fue desarrollada en C#.

Así mismo se está estudiando la definición y creación de un sistema de registro global de los elementos que permita la creación de un elemento *Middelware* poderoso en el que se puedan redirigir peticiones al agente/sensor más adecuado en función de su disponibilidad/funcionalidad. Con esto podríamos realizar llamadas a la plataforma del estilo: "obtener la temperatura" o "cerrar la puerta" y el sistema buscaría el elemento más adecuado al que realizar la petición.

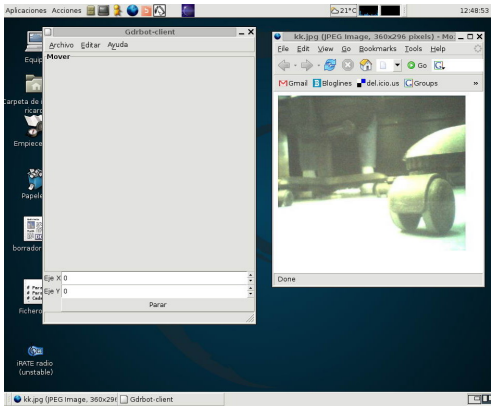

Figura 6. Captura de Ejemplo de Aplicación

Por último se está desarrollando un sistema para evitar obstáculos de forma colaborativa [12]. Cada agente envía a los demás una serie de información (cromosoma) sobre cómo ha resuelto cada obstáculo. Al final, los agentes poseen un conocimiento común que les ayuda a evitar de forma óptima los siguientes obstáculos.

Referencias

[1] D. Wang; X. Ma and X. Dai, "Web-based robotic control system with flexible framework". Proc. ICRA '04. 2004 IEEE International Conference on Robotics and Automation, Volume: 4, pp:3351-3356.

[2] S. Dissanaike; P. Wijkman and M. Wijkman, "Utilizing XML-RPC or SOAP on an embedded system", Proc. 24th International Conference on Distributed Computing Systems Workshops, 2004, pp438–440.

[3] X. Wang; M. Moallem and R.V. Patel." An Internet-based distributed multiple-telerobot system". IEEE Transactions on Systems, Man and Cybernetics, Part A, Vol:33, Issue: 5, Sept. 2003, pp:627 – 634.

[4] Navarro L., et al., 2002. Millibots. The development of a framework and algorithms for a distributed heterogeneus robot team. In IEEE Robotic and Automation Magazine, vol 2, no 4, pp 31-40.

[5] K. Taylor and B. Dalton, "Internet robots: a new robotics niche", IEEE Robotics & Automation Magazine, Vol: 7 , Issue: 1 , March 2000, pp:27-34.

[6] H. Hiraishi, H. Ohwada and F. Mizoguchi. "Web-based Communication and Control for Multiagent Robots". Proc.IEEE/RSJ Int. Conference on Intelligence Robots and Systems. Victoria B.C. , (Canada), 1998, pp 120-125.

[7] PMD Corporation. http://www.pmdcorp.com

[8] VIA Technologies Inc, http://www.viavpsd.com/product/

[9] G. González de Rivera et al. 2002. GP_BOT: Plataforma Hardware para la enseñanza de Robótica en Ingeniería Informática. (TAEE'02). pp 67-70. ULPG.

[10] Web:http://xmlrpc.scripting.com/directory/1568/implementations

[11] Web: http://www.kernel.org

[12] G. Glez. de Rivera, K. Koroutchev, R. Ribalda, J. Garrido, "Occlusion Avoiding using Group Evolution Learning". En Preparación