SciVerse ScienceDirect
Disponible en www.sciencedirect.com
Revista Iberoamericana de Automática e Informática industrial 10 (2013) 85–95
www.elsevier.es/RIAI

Arquitectura de integración basada en Servicios Web para sistemas heterogéneos y distribuidos: aplicación a robots móviles interactivos.

Alberto Valero-Gómez*, Paloma de la Puente, Diego Rodriguez-Losada, Miguel Hernando, Pablo San Segundo

Centro de Automática y Robótica CAR (UPM-CSIC), C/ José Gutiérrez Abascal, nº2, 28006, Madrid, España.

Resumen

En el desarrollo de sistemas robóticos complejos es común usar herramientas y arquitecturas software que facilitan la tarea de integración y reutilización del código. Ejemplos son el Robot Operating System (ROS) de WillowGarage, Player/Stage, etc. No obstante, la robótica móvil es cada vez más multidisciplinar, involucrando equipos de investigación de diferentes áreas incluyendo las tecnologías del habla, visión, aprendizaje automático, etc. Por ello es altamente improbable que dichos equipos adopten como elemento de desarrollo una plataforma intrínsecamente ligada a la robótica. A veces incluso, las restricciones de lenguajes de desarrollo o sistemas operativos, lo hacen impracticable y perjudicaría tanto a la velocidad de desarrollo como a la eficiencia del producto final. Este artículo presenta una novedosa arquitectura basada totalmente en los estándares de los Servicios Web. La arquitectura permite el desarrollo de agentes software de modo independiente de una forma totalmente desacoplada, manteniendo una intrusividad mínima a la par que una gran sencillez de uso. Usando posteriormente un intermediador o bróker totalmente dinámico se conectan dichos agentes en tiempo de ejecución. Esta arquitectura, que no sustituye a los sistemas ya existentes sino que los complementa, ha sido usada en un complejo demostrador multi-robot con una gran heterogenia de sistemas y agentes, en un proceso de instalación y puesta en funcionamiento de un robot guía interactivo en un nuevo ambiente.

Palabras Clave: Arquitectura Software, Integración de sistemas, Control de Robots.

1. Introducción

La robótica de servicio, y en concreto los robots móviles autónomos, están experimentando en los últimos años un desarrollo e implantación excepcionales. Además de utilizarse en el ámbito más científico y experimental –como por ejemplo los famosos *rovers* Spirit y Opportunity de exploración planetaria- se comienzan a implantar soluciones comerciales u orientadas al público en general. No es raro encontrar ofertas y artículos de robots que desarrollan tareas cotidianas específicas, como el caso de los robots aspiradores de IRobot Roomba®.

Entre los sistemas robóticos de servicio cabría destacar aquellos que reciben el apellido de *sociales*. Estos robots afrontan una serie de dificultades que no están presentes en otros sistemas móviles que no trabajan en entornos con personas: los robots sociales necesitan tener la habilidad de interactuar con las personas que los rodean. La investigación en robots sociales que deben moverse en entornos estructurados y dinámicos cubre varios campos que en la actualidad concentran la mayoría de la actividad investigadora en el área de la robótica. Algunas de los temas afectados son, entre otros, los factores humanos (Human Factors), la interacción multimodal (MMI), el lenguaje natural, el reconocimiento del habla, la robótica emocional, localización y mapeado simultáneo (SLAM), etc.

Los sistemas robóticos diseñados para realizar la guía de los visitantes de un museo o una exposición son posiblemente una de las aplicaciones más investigadas por los centros de investigación en robótica móvil del mundo entero. Desde los ya famosos primeros desarrollos de Rhino y Minerva (Thrun *et al*., 1999) (Burgard *et al.,* 1999), a lo largo de estos últimos años se han desarrollado sistemas similares como Sage (Nourbakhsh *et al,*1999), Mobot, Albert, Maggie (Alonso Martín *et al,* 2010) con esta orientación o con un propósito semejante como Nursebot (Montemerlo *et al,* 2002), Shopbot (Gross *et al,* 2008). De igual forma en la Universidad Politécnica de Madrid se diseño y desarrolló el robot *Urbano* (Rodríguez-Losada *et al,* 2008), el cual, además de servir de plataforma para la investigación, ha sido utilizado con éxito en museos y exhibiciones abiertas al público.

1.1. Motivación

La utilización de un robot en un escenario específico requiere de una serie de tareas preliminares complejas antes de que el sistema pueda funcionar normalmente. Este proceso se denomina

* Autor en correspondencia.
Correos electrónicos:
alberto.valero.gomez@gmail.com (Alberto Valero)

http://dx.doi.org/10.1016/j.riai.2012.11.008

con el término inglés *System Deployment*. Para el caso concreto de los robots orientados al guiado en museos y exhibiciones, la configuración inicial requiere de la construcción del mapa y la identificación, localización y descripción de los distintos lugares a los que los robots deben saber dirigirse. Se puede considerar que esto es equivalente a la formación que un guía humano recibiría en su primer día de trabajo. Sin embargo en un sistema robótico esta operación es en la actualidad compleja y requiere de la asistencia de personal experto en el sistema. Algunos desarrollos, como el robot guía para personas mayores Guido® (Lacey y Rodríguez-Losada, 2008), utilizan un sistema de configuración semiautomática, que requiere de la asistencia de una persona con una capacidad técnica media y cierto nivel de entrenamiento.

Tras trabajar durante varios años con *Urbano*, consideramos que uno de los elementos más críticos a la hora de lograr que estos robots puedan ser ofertados en el mercado es lograr un método de configuración natural, directo y sencillo que pueda ser llevado a cabo por personal no técnico.

Por otra parte, el entorno altamente dinámico de investigación origina típicamente un contexto muy heterogéneo a todos los niveles: diferentes plataformas robóticas con diferentes capacidades sensoriales, grupos de trabajo de diferentes áreas (robótica, inteligencia artificial, tecnologías del habla, etc.), y recursos humanos de niveles técnicos radicalmente diferentes, desde principiantes a expertos. Por otra parte, un *framework* que ahora puede ser extensamente usado, en poco tiempo puede ser sustituido por otro nuevo con nuevas funcionalidades, lo que conlleva una dificultad de integración del software desarrollado en periodos de tiempo diferentes. Este es otro de los problemas que se pretendió afrontar con el desarrollo de la arquitectura que aquí se presenta.

1.2. Frameworks de desarrollo en robótica

En la actualidad existen diversos *frameworks* en la comunidad robótica que permiten el desarrollo desacoplado, la reutilización de código, y la integración rápida gracias a su estructura basada en componentes. Especialmente relevantes son ROS (Robot Operating System) desarrollado por WillowGarage, ORCA (Makarenko, Brooks y Kaupp, 2007), OROCOS (Bruyninckx, 2001), Microsoft Robotics Studio y OpenRDK (Calisi *et al,* 2008). Todos estos paquetes software se basan en la máxima *"divide y vencerás"*. A través del desarrollo de módulos basados en plantillas de código, o jerarquías de clases polimórficas, introducen una metodología de codificación estricta que permite la integración de estos módulos entre sí de un modo rápido y eficaz. Este modo de desarrollo beneficia en muchos casos la coordinación de un equipo de investigadores que trabajan en un mismo proyecto gracias a las reglas de codificación. Sin embargo, lo que se presenta como una ventaja en estos *frameworks*, puede constituir también un inconveniente. La rigidez en la escritura y organización del código puede resultar incómoda para ciertos programadores acostumbrados a un cierto modo de trabajar. Por otro lado, no es extraño que el sistema operativo esté impuesto por el *framework*, lo que obliga a cambiar los hábitos del programador. Esto puede resultar un impedimento para sistemas *ad hoc* como *Urbano*, en el que el sistema de interacción por voz ha sido realizado independientemente por un grupo de investigación que nada tiene que ver con la robótica y tienen sus propios protocolos de programación que siguen desde hace muchos años. Además, algunos de estos *frameworks* no funcionan en los ordenadores que llevan a bordo algunas plataformas robóticas, por lo que todo el control se debe hacer remotamente.

1.3. Contribución

Este artículo presenta como contribución específica una arquitectura software que permite la integración de distintos agentes software. Estos agentes pueden ser desarrollados en distintos lenguajes de programación y para diferentes plataformas y sistemas operativos. Esta arquitectura no sustituye a ninguno de los frameworks robóticos existentes, sino que los complementa, permitiendo la integración y comunicación ente ellos, aportando una gran versatilidad a sistemas robóticos distribuidos heterogéneos desarrollados con distintas tecnologías software.

Esta nueva arquitectura se caracteriza por el uso de los estándares actuales de los Servicios Web (WS) y una fuerte componente de funcionamiento dinámico que permite conectar en tiempo de ejecución los diferentes agentes, permitiendo un desarrollo totalmente desacoplado de los mismos y permitiendo su integración.

Como prueba de concepto e implementación, esta arquitectura ha sido utilizada en el desarrollo del demostrador Robonauta, un sistema de instalación y puesta en marcha de un robot guía interactivo mediante el uso de diferentes robots. Robonauta integra múltiples tecnologías y desarrollos de diferentes equipos y sistemas operativos.

El artículo se estructura de la forma siguiente: en las secciones segunda y tercera se describen brevemente los robots del sistema así como el sistema de navegación usado en los mismos. La sección 4 expone la arquitectura propuesta y la sección 5 ilustra la aplicación de la misma en diferentes experimentos con los robots. Finalmente las conclusiones se resumen en la sección 6.

2. Robots

Urbano es un robot social diseñado para realizar tareas de navegación autónoma y a la vez gozar de capacidades de interacción hombre-robot. Está construido sobre una plataforma B21r de iRobot, la cual está equipada con un sistema de locomoción de cuatro ruedas de tipo *synchrodrive*. Incluye una cara mecatrónica y dos brazos robóticos que permiten a *Urbano* expresar emociones tales como alegría, tristeza, sorpresa o enfado (Figura 1). Posee dos anillos de sensores ultrasónicos y uno de infrarrojos para detectar obstáculos cercanos a diferentes alturas. Además en la parte alta del robot (1.5m de altura) se ha incluido un escáner laser SICK LMS200 situado horizontalmente, utilizado fundamentalmente para la navegación, localización y construcción de mapas 2D. Asimismo, también dispone de entrada y salida amplificada de audio (para la voz). Cabe destacar en este punto que en este robot intervienen hasta 5 grupos de trabajo diferentes (brazos, cara, voz, navegación, emociones), 4 de ellos alejados de los frameworks de desarrollo típicos en robots móviles como ROS. Una descripción del robot y sus capacidades se puede encontrar en (Rodríguez-Losada *et al,* 2008).

Doris es también un robot social, similar a *Urbano* tanto en su función como en el diseño (Figura 1 dcha.). La cara mecatrónica es una versión mejorada de la realizada para *Urbano,* de igual forma que lo serán los brazos robóticos actualmente en fase de desarrollo. La plataforma robótica móvil en que se basa es PatrolBot de MobileRobots.Inc, la cual es de tipo diferencial. Otra diferencia importante existente entre ambos robots es que el escáner laser de *Doris* se sitúa a escasos centímetros del suelo, lo

que dificulta el seguimiento de personas al detectar las piernas en vez de el tronco, dificulta la localización dado que generalmente hay más objetos a nivel bajo, pero por otra parte es algo más seguro para navegar, precisamente por poder detectarlos.

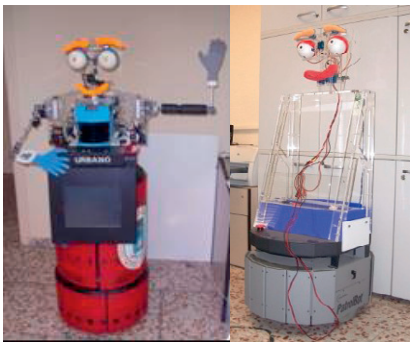

Figura 1: *Urbano* (izquierda) y Doris (derecha).

Figura 2: El robot de exploración Nemo.

Finalmente *Nemo* (Figura 2) se diseñó bajo el concepto de robot de exploración. Está realizado sobre la plataforma Pioneer 3AT de MobileRobots.Inc. El robot puede obtener información tridimensional del entorno dado que cuenta con un laser escáner montado sobre una unidad *pan-tilt*. Además cuenta con una cámara HD orientable, por lo que el robot además de obtener los datos de distancia puede capturar las imágenes correspondientes con una alta resolución.

De esta forma se dispone de dos tipos diferenciados de robots: *Urbano* y *Doris* concentran sus capacidades en la capacidad interactiva y social, mientras que *Nemo* tiene como principal característica la habilidad para obtener una representación tridimensional del espacio que recorre.

En un principio se pensó en combinar en un mismo robot estas capacidades. Sin embargo incluir en *Urbano* o *Doris* el sistema de escaneado 3D no es conveniente por varias razones. En primer lugar, la altura de estos robots y el movimiento y posicionamiento de los brazos dificulta el montaje de este sistema en la estructura. En segundo lugar, el tamaño y lo aparatoso del funcionamiento del escáner 3D dañarían la interactividad con el hombre, que es para lo que se han diseñado de forma específica. Cabe no obstante destacar que el avance en el escaneado 3D que han supuesto los sensores como Microsoft Kinect, sí que permitiría actualmente esta posibilidad, tecnología que no estaba disponible durante el desarrollo de este trabajo.

3. Sistema de navegación

A continuación se describen brevemente los diferentes algoritmos y funcionalidades utilizados en el sistema de navegación autónoma de los robots móviles *Urbano, Doris,* y *Nemo*. Dichos algoritmos son implementados e integrados como módulos del *framework* robótico OpenRDK (Open Robot Development Kit), que funciona en Linux y Mac OS/X. Esta herramienta se hace cargo de aspectos como la concurrencia, la sincronización y la consistencia de los datos de los distintos componentes involucrados en estas operaciones, además de incluir una serie de componentes básicos que facilitan enormemente el desarrollo de los algoritmos de bajo nivel. Los algoritmos de navegación, localización y mapeado se integran y ejecutan en un agente diferente para cada uno de los robots.

Se ha utilizado este sistema frente a otros como ROS por dos razones. La primera, y quizás más importante, es el histórico de colaboración de nuestro grupo de investigación con el grupo liderado por el profesor D. Nardi en la universidad "La Sapienza" de Roma, lugar de origen del OpenRDK, *framework* que ya habíamos utilizado anteriormente (Valero *et al.* 2009) y en el que ya teníamos desarrollados diversos módulos funcionales. Por otra parte, este *framework* está orientado a procesos ligeros (o hilos) compartiendo un espacio de memoria común, lo que es altamente eficiente al compartir grandes cantidades de datos, como los mapas del entorno. De esta forma se permite desarrollar de forma separada algoritmos de localización o mapeado, de planificación y control, y luego que ejecuten simultáneamente trabajando sobre el mismo mapa compartido de forma transparente para los programadores y con mayor eficiencia que en un sistema orientado a procesos con paso de mensajes como ROS.

3.1. Modelado y localización 2D

A pesar de los avances en SLAM desarrollados para *Urbano* en los últimos años (Rodríguez-Losada *et al*., 2007) (Pedraza *et al*., 2009), estos trabajos siguen un paradigma de representación mediante entidades geométricas, lo que los hace parcialmente incompletos de cara a una navegación segura y por lo que tradicionalmente requerían intervención humana de cara a definir manualmente los recorridos seguros del robot. Para poder implementar una puesta en marcha totalmente automática, se prefirió usar un modelo de mapa de celdillas, utilizando una implementación clásica conocida como GMapping (Grisetti *et al,* 2005), el cual propone un algoritmo basado en el filtro de partículas (Doucet *et al*. 2001), ampliamente conocido y utilizado por su robustez y eficiencia. La etapa de construcción del mapa se asume como hipótesis que se realiza mediante guiado humano y en un proceso atendido, de tal forma que los peligros no visibles por la percepción bidimensional de un robot no suponen una amenaza para la seguridad.

Aunque el reciente auge de los sistemas sensoriales está suponiendo un gran avance en técnicas de modelado 3D, aun no están suficientemente desarrollados los sistemas de planificación y control en dichos modelos, motivo por el que no se optó por usar directamente un modelo 3D.

3.2. Mapa de navegabilidad

Una vez que se dispone de un mapa bidimensional, un robot con capacidades de percepción en 3D puede enriquecer dicho mapa con información de traversabilidad. Si además se dispone de la trayectoria seguida por el robot que ha construido dicho mapa 2D, y que podemos asumir como segura porque ya ha sido ejecutada por el otro robot, este proceso se puede automatizar.

En el sistema (de desarrollo propio), se usa el robot *Nemo* que comienza a ejecutar la trayectoria recibida de forma completamente autónoma y por medio de un sencillo controlador

proporcional, localizándose en el entorno mediante el mapa 2D y el algoritmo de localización de GMapping. Nótese que la diferente altura de los escáneres laser puede llegar a generar discrepancias entre el modelo construido previamente y la observación real no debidas a errores en la posición del robot, que no obstante son bien absorbidas por el algoritmo de localización. Cuando se ha recorrido una cierta distancia (típicamente 1-2 m), el robot se para y realiza un escaneado tridimensional del entorno. Este conjunto de medidas son transformados posteriormente de forma que se trasladan a un formato de matriz de nube de puntos 3D referidos al sistema de referencia del robot.

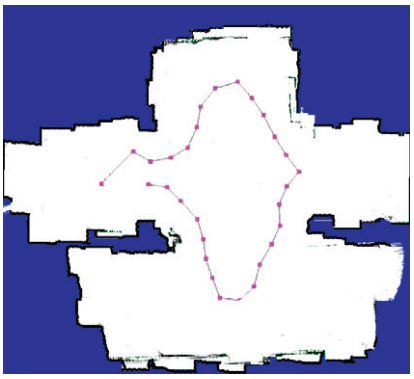

Figura 3. Mapa 2D con la trayectoria realizada, antes de comenzar la adquisición tridimensional del entorno.

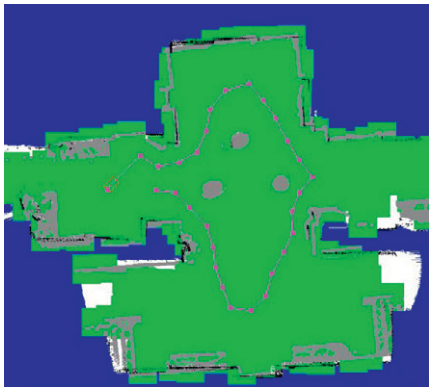

Figura 4. Mapa de Traversabilidad generado combinando la información del mapa 2D con los datos del escáner 3D. Nótese 3 grandes obstáculos con forma de círculo en el centro de la figura que no eran visibles en el mapa 2D.

Se ha implementado un sencillo algoritmo que a pesar de su simplicidad ha demostrado ser eficaz para el tipo de entornos en los que se mueven nuestros robots. Aquellos puntos cuya coordenada z en valor absoluto, está dentro de un cierto umbral (típicamente 5-10 cm) harán que se marque como navegable (suelo visible) la celdilla correspondiente en el mapa, mientras que si dicha cota está en el intervalo correspondiente a la altura del robot, se marcará como no navegable, con prioridad además sobre las navegables (aunque se vea el suelo, si se ve también un obstáculo por encima del suelo, es no navegable). Las celdillas de las que se carece de información se marcarán como zona desconocida y por tanto no navegable. Se asume una posición conocida, suministrada en este caso por el localizador de GMapping. La figura 3 muestra el mapa, la trayectoria obtenida por *Urbano* y los puntos en los que *Nemo* parará para obtener la información 3D. En la figura 4 se muestra el mapa de navegabilidad o traversabilidad generado por *Nemo*.

3.3. Control de movimiento

El planificador de trayectorias utilizado es el descrito por (Calisi *et al,* 2005), que se reutilizó con modificaciones menores por estar ya implementado como módulo de OpenRDK. Dicho planificador calcula un camino al punto de destino como sucesión de segmentos rectilíneos que no colisionan con obstáculos. Dichos obstáculos son extraídos tanto del mapa 2D como de la información 3D adquirida previamente. La trayectoria se calcula sobre un grafo de conectividad del entorno que se va calculando incrementalmente según de construye el mapa 2D. Dicha trayectoria es pasada al módulo de control de movimiento.

El módulo de control de movimiento, también disponible como modulo de OpenRDK, funciona de un modo reactivo. Establece como punto de destino el punto final del segmento correspondiente de la trayectoria. Para alcanzar este punto calcula una nueva trayectoria en la que se tienen en cuenta las características cinemáticas y dinámicas del robot. Esta nueva subtrayectoria se calcula teniendo en cuenta tanto el mapa (2D y 3D) como la información obtenida por el sensor laser 2D. De este modo el algoritmo es capaz de evitar obstáculos en movimiento o que no estaban presentes en el mapa inicial. Esta trayectoria es curva y compatible con la cinemática del robot, por lo que el movimiento es suave. Para seguir esa trayectoria se aplica un simple control proporcional.

No se ha llegado a analizar el funcionamiento de este sistema de control de movimiento frente a otras soluciones, pero en la práctica, ha demostrado funcionar muy bien siempre y cuando la información de localización fuera suficientemente buena.

3.4. Detección y seguimiento de personas

Para la detección y seguimiento (*tracking*) de personas, se utiliza un algoritmo propio con tres niveles: en el primer nivel se realiza una segmentación de los datos brutos del laser, eliminando las paredes rectas y segmentando mediante el gradiente de profundidad de rango de medidas los posibles grupos de puntos candidatos a ser una persona. En el segundo nivel se realiza un seguimiento de las hipótesis definidas en el nivel anterior, requiriendo continuidad espacial y movimiento durante varios instantes de tiempo para confirmar dicha hipótesis. Una vez que la hipótesis ha sido confirmada, se le asigna un identificador numérico creciente y se empieza a realizar el seguimiento con un EKF (un filtro independiente para cada persona) cuyo estado es la posición y la velocidad de dicha persona (Figura 5). Según dicho filtro, las personas que no son detectadas en un instante de tiempo van aumentando su covarianza, de tal forma que si la covarianza supera un umbral, se elimina a dicha persona del seguimiento.

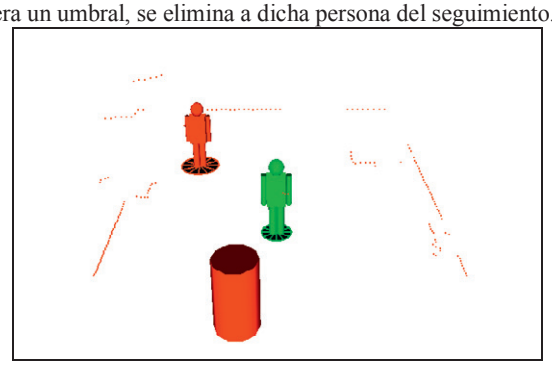

Figura 5. Segmentación y seguimiento de personas a partir de datos del laser.

El mismo algoritmo también provee la información necesaria para integrarse con el resto de subsistemas del robot: notifica cuando ha aparecido una persona nueva en su rango, de tal forma que se le pueda saludar o comenzar a seguir, calcula posiciones a una cierta distancia (típicamente 1m) de la persona a la que está siguiendo, y también notifica cuando ha perdido a la persona a la que seguía. Se puede ver un video del sistema en funcionamiento en (Valero *et al*., 2012), procesando un set de datos real.

3.5. Modelado 3D

Con el recorrido de *Nemo,* además de crearse el mapa de navegabilidad, se procesa la nube de puntos 3D para generar modelos basados en primitivas de los elementos del entorno (Figura 6). Estos modelos se enriquecen con la información de la cámara de forma que se incluyen las texturas sobre su superficie dando al modelo tridimensional gran realismo, todo ello en un desarrollo propio. A diferencia de nuestro trabajo previo (de la Puente *et al.*, 2009), en este caso el procesamiento se realiza off-line, de forma que los requisitos de eficiencia ya no son tan críticos, y no está integrado aun en el sistema de navegación de los robots. No obstante, este modelo constituye nuestra principal línea de investigación actual en navegación autónoma, tal y como se resume en las conclusiones.

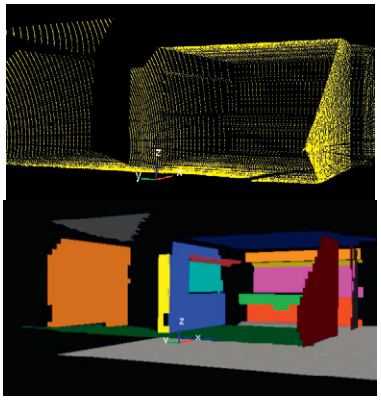

Figura 6. Segmentación de los datos 3D del laser para segmentación y modelado del entorno basado en características geométricas.

4. Arquitectura del sistema

El trabajo desarrollado en cada uno de los subsistemas resulta en los siguientes agentes:

- Agentes de síntesis y reconocimiento de voz. Uno por cada robot con audio (*Urbano* y *Doris*). Desarrollado por el Grupo de Tecnologías del Habla de la UPM, funcionando obligatoriamente en Windows.
- Agentes de navegación, uno por cada robot. Desarrollados usando OpenRDK (y por tanto corriendo obligatoriamente en Linux o Mac OS/X), en colaboración con la Universidad de Roma La Sapienza.
- Agentes de control de la cara, desarrollados por alumnos de 3er curso de Ingeniería Técnica, en Windows (*Urbano* y *Doris*).
- Agente de control de los brazos (*Urbano*). Multiplataforma.
- Agente de gestión. Uno por cada robot. Multiplataforma. Se encarga de gestionar a alto nivel las tareas encomendadas a cada robot particular.
- Agente coordinador. Uno general, multiplataforma. Se encarga de coordinar el funcionamiento de los diferentes robots.

Resulta obvio apreciar que los diferentes agentes de un mismo robot tienen que comunicarse para realizar la tarea asignada; por ejemplo, si el sistema de detección de personas detecta una persona en su camino, puede informar al modulo de voz y éste solicitar a dicha persona que se retire del camino. Pero igualmente importante resulta en este caso la comunicación entre los diferentes robots, por ejemplo para compartir el mapa del entorno.

La arquitectura desarrollada parte de una serie de requisitos de diseño importantes, que se enuncian a continuación:

- Lenguaje C++: dado que es el lenguaje más extendido en el desarrollo de software para robots y que todo nuestro software previo estaba implementado en dicho lenguaje.
- Multiplataforma: al menos deberá funcionar sobre Windows y Linux.
- Sencillez de uso y de aprendizaje: aunque poco cuantificable, muchas de las decisiones de diseño quedan condicionadas por este aspecto, dado que la plataforma es a menudo utilizada por estudiantes y programadores nóveles.
- Ligera: debe poder ser ejecutado y compilado en sistemas embebidos.
- No intrusiva. El agente debe de ser una aplicación normal de usuario, al que se le añade la parte de comunicaciones, no a la inversa como sucede con otros *frameworks*.
- Cada agente deberá contar con un mecanismo estándar de publicación de su interfaz de servicios para permitir tanto la consulta como la invocación de forma dinámica.
- El protocolo de la interfaz debe ser legible directamente por el ser humano y en formato XML tanto por razón del desarrollo como de la posible depuración de un sistema específico.
- Utilización de protocolos y formatos estándar y extendidos siempre que sea posible.

No obstante, la característica más deseada era la capacidad de desarrollar dichos agentes sin ninguna interfaz en común, sin necesidad de compartir ningún código, fichero o especificación de protocolo en tiempo de desarrollo, de tal forma que los desarrolladores de dichos agentes pudieran realizar su tarea de forma totalmente independiente.

4.1. Jabon: SOAP para C++

La proliferación de la Web e Internet, así como la mejora de las comunicaciones y computadores, han propiciado la extensión de sistemas distribuidos ligados a las mismas, frente a la caída de otros sistemas distribuidos como CORBA o RPCs, motivo por el que se decidió seguir esta tendencia.

Muchos de los requisitos enunciados anteriormente se pueden lograr mediante el uso de *Web Services* (WS), en el que el Protocolo de Acceso Simple a Objetos (SOAP – Simple Object Acess Protocol) utiliza mensajes XML sobre la capa de transporte HTTP para realizar invocaciones a operaciones remotas. Más allá, el Lenguaje de Definición de Servicios Web (WSDL) permite especificar y publicar estos mensajes, sus parámetros, las operaciones y las posibles conexiones. Todos estos acrónimos son de hecho recomendaciones de la organización W3C. Se resumen en la siguiente tabla los acrónimos usados en esta sección:

Tabla 1: Acrónimos comunes

Acrónimo	Definición
WS	**Web Service.** Tecnología que sigue un conjunto de protocolos y estándares (típicamente abiertos y basados en la web) para comunicación ente aplicaciones.
RPC	**Remote Procedure Call**. Llamada a procedimiento remoto, invocación de un método en una máquina externa conectada por red mediante un sistema como CORBA, Java RMI, etc.
SOAP	**Simple Object Access Protocol**. Protocolo de acceso simple a objetos, define la estructura de mensajes XML (básicamente, *envelope, header y body*), la capa de mensajería de servicios web.
WSDL	**Web Service Definition Language**. Lenguaje de definición de servicios web. Es un documento XML que especifica los contenidos de los mensajes (lo que puede ir dentro de un "envelope" SOAP) que puede aceptar y recibir un determinado Servicio Web, así como detalles de conexión y codificación de datos
HTTP	**Hyper Text Transfer Protocol**. Protocolo de transferencia de hipertexto, se usa en servicios web como la capa de transporte de mensajes SOAP. Es decir, lo que realmente se envía por la red son mensajes SOAP precedidos de cabeceras estándares HTTP (con métodos GET y para servicios web, sobre todo POST)
W3C	**World Wide Web Consortium**. Comunidad internacional que define estándares en la web e internet, llamados "recomendaciones" (SOAP, WSDL son recomendaciones)
XML	**Extensible Markup Language**. Los servicios web están fuertemente basados en XML; tanto los mensajes enviados como las especificaciones de los protocolos en WS usan XML.
XML-Schema	Lenguaje XML para definir estructuras de información. En su parte 2 incluye una especificación de codificación de datos, tanto simples (int, float) como compuestos (date, time) y de usuario (complexType)

Los desarrollos más extendidos de SOAP en C++ son gSOAP (Robert *et al.,* 2002), posiblemente el estándar de facto, y Apache Axis++. Aunque son bastante completos, estas herramientas no satisfacen los requisitos mencionados anteriormente por lo que en el marco de este trabajo se ha desarrollado la herramienta *Jabon*. *Jabon* es una implementación *open-source* y disponible on-line en C++ de SOAP con características novedosas que se exponen a continuación:

4.1.1. Núcleo Dinámico para Servicios Web

En el centro de la plataforma se encuentra una librería que implementa clases de comunicaciones mediante sockets y HTTP, así como de clases que representan cada uno de los elementos de los diferentes protocolos: existen clases para manejar mensajes SOAP, clases para la especificación de WSDL, clases para tipos de datos mediante XML-Schema, todas ellas pensadas para representar, manejar e incluso crear y eliminar la información pertinente en tiempo de ejecución. Este núcleo dinámico es precisamente la gran diferencia frente a soluciones como gSOAP que básicamente genera todo el código de comunicaciones de forma estática, sin que en tiempo de ejecución se tenga ninguna constancia de las interfaces. Dicho de otra forma, el núcleo dinámico desarrollado por *Jabon* provee de un completo mecanismo software de *reflexión* sobre los mensajes y protocolos que los otros sistemas no implementan. Por otra parte, *Jabon* implementa un modelo de comunicación asíncrono multi-hilo de tal forma que no interfiere en el flujo de ejecución principal de las aplicaciones que lo utilizan.

4.1.2. Generador de código

Jabon incluye un generador de código C++ que en vez de utilizar como entrada un fichero WSDL y para facilitar la tarea del usuario, sólo requiere de un fichero de cabecera ".h" de la aplicación. Lo único que tiene que realizar el programador es etiquetar con un comentario específico los métodos de la clase que desea que sean implementados como Servicio Web. El generador analizará el documento del usuario y creará como consecuencia todo el código fuente y la estructura necesaria para la compilación en la aplicación del usuario. Así por ejemplo si se deseara poner a disposición la operación "ir a un lugar" que requiriera como argumentos la velocidad, y una etiqueta identificativa del sitio, se incluiría en la declaración del método del objeto local el comentario "*//JabonService*":

```
//clase propia del usuario
class NavigationServices
{
public:
      //JabonService
      int GoToPlace(float speed, string place);
      //cuantas cosas se desee, sin restricciones
}
```

El código generado incluye un servidor HTTP que puede ser lanzado con sólo 3 líneas de código. Mientras que no sea modificada la interfaz de los servicios "JabonService" del objeto que se publican como servicio web, no es necesario volver a generar el código.

El servidor HTTP responde con la descripción WSDL de las distintas operaciones disponibles por medio del protocolo GET, mientras que el método POST se reserva para la invocación específica de estos servicios. De esta forma, al utilizar los protocolos estándar, el programador puede inspeccionar cualquier servicio activo público mediante cualquier navegador web.

4.1.3. Invocador dinámico

Sin embargo una de las características más relevantes de *Jabon* es su comportamiento dinámico. Mediante la descripción WSDL de los distintos servicios, es posible realizar y configurar los servicios web durante la ejecución. El sistema incluye la aplicación *Gel* para la interfaz de comandos que permite conectarse a cualquier servidor web, descargar la descripción WSDL y de forma interactiva invocar cualquiera de los servicios ofrecidos, todo ello durante la ejecución y careciendo del conocimiento previo de los servicios disponibles. Esta herramienta es especialmente útil durante la fase de desarrollo y depuración. Una descripción detallada de estas herramientas así como el código fuente están disponibles en (Jabon, 2009).

Respecto al rendimiento, *Jabon* es capaz de conseguir tiempos de *roundtrip* para mensajes pequeños de aproximadamente 1ms, funcionando en maquina local, mientras que la transferencia de grandes cantidades de datos binarios es de 4Mb/s, incluidos los procesos de codificación y descodificación en Base64. Si bien no se garantiza el tiempo real, el rendimiento ha probado ser sobresaliente, incluido para el envío puntual de grandes cantidades de datos como imágenes y mapas. No obstante, para compartir a alta frecuencia grandes cantidades de datos (y sólo en la misma máquina), es necesario algún otro mecanismo; este es uno de los principales motivos de uso de OpenRDK para las tareas de navegación, como se ha explicado anteriormente.

4.2. Esponja, un bróker y traductor SOAP

El elemento clave de la arquitectura propuesta para el control del sistema robótico es, sin embargo, el bróker SOAP al cual hemos denominado *Esponja*, y el cual permite conectar distintos servidores de servicios *Jabon*, enlazar servicios y traducir los mensajes entre ellos.

La idea central del desarrollo es permitir que cada programador o equipo de desarrollo pueda diseñar sus agentes software de forma independiente del resto de agentes, incluso sin la necesidad de conocer las interfaces de los mismos. De esta forma cada agente puede ser desarrollado siguiendo su propia convención de nombres y parámetros en el desarrollo de los dos tipos de elementos esenciales en esta arquitectura: los *Eventos* y los *Servicios*. Así, un agente lanza *Eventos* informando a quien esté a la escucha de que algo específico ha sucedido, y permite la ejecución remota de los servicios publicados. Ambos son implementados como servicios web; pero los primeros son mensajes enviados desde un cliente a un servidor que se desconoce durante el proceso de desarrollo de la aplicación.

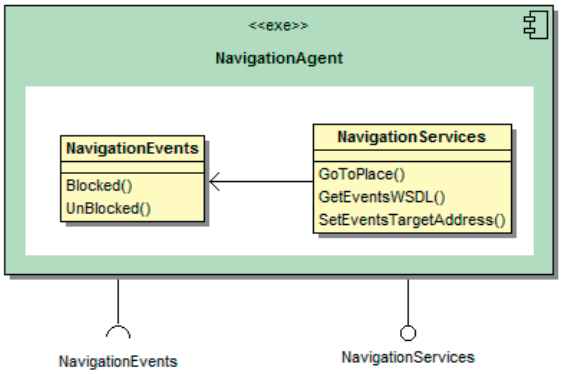

Figura 7. Ejemplo de un agente para la Navegación. Este agente es capaz de detectar que algo ha bloqueado la trayectoria del robot, en cuyo caso manda un mensaje SOAP a un servidor. El agente además incluye el servicio de "Ir a un Lugar" el cual puede ser invocado de forma remota para enviar el robot a un determinado lugar del mapa.

Jabon es utilizado en el desarrollo de los distintos agentes de la arquitectura para generar automáticamente el código del cliente responsable de comunicar los eventos y el código del servidor que implementa los servicios. Mientras que la descripción WSDL de los servicios es accesible directamente por medio del método GET, la descripción WSDL de los eventos así como la dirección a la que los correspondientes mensajes SOAP tienen que ser remitidos, debe ser especificada mediante los métodos *GetEventsWSDL()* y *SetEventsTargetAddress()*, dado que, como se ha mencionado, los eventos son gestionados por un cliente. La Figura 7 muestra una representación gráfica de un agente con sus eventos y servicios.

La conexión de los distintos agentes se realiza por medio de *Esponja*, que es capaz de generar automáticamente una representación en memoria del conjunto de servicios y eventos presentes, gracias a su núcleo dinámico expuesto anteriormente. Una vez obtenidas estas descripciones, *Esponja* puede recibir cualquier mensaje de Evento de los distintos agentes y enrutarlo y traducirlo para que sea entendido por cualquiera de los servicios de los agentes detectados (Figura 8).

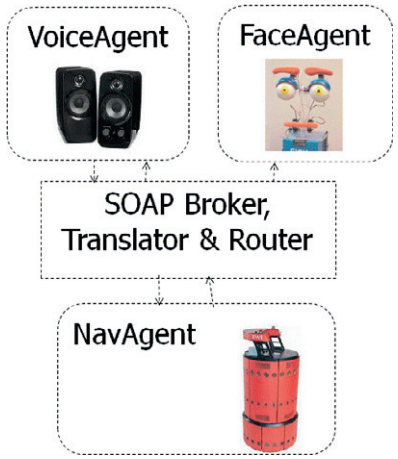

Figura 8. Los distintos agentes que componen el control del robot quedan conectados por medio de un *bróker* SOAP, de forma que no es necesario conocer con exactitud las distintas interfaces durante el desarrollo de cada uno de los agentes.

Estas asociaciones son especificadas por medio de un fichero XML leído durante la ejecución, permitiéndose además la creación dinámica de nuevas asociaciones si así se desea. El siguiente extracto de XML muestra el aspecto de una de estas asociaciones:

```
<EsponjaRules name="" xmlns:ns4="urn:VoiceEvents"
        xmlns:ns5="urn:NavigationServices">
<rule>
        <input name="ns4:RequestVisitPlace"/>
        <output name="ns5:GoToPlace"/>
        <messageTranslation>
                <parameter output="place"
                        input="placeName"/>
                <parameter output="speed" input="1"/>
        </messageTranslation>
        <responseTranslation>
        <parameter output="returns"

        input="returns"/>
        </responseTranslation>
</rule>
</EsponjaRules>
```

Ante el evento *RequestVisitPlace* -solicitar ir a un lugar- emitido por el reconocedor de voz, se activa el servicio *GoToPlace* -ir a un lugar- perteneciente al agente de navegación.

Hay que destacar que se realiza una traducción completa del mensaje SOAP, incluyendo el espacio de nombres, los mensajes de entrada y salida de ambos agentes, así como el nombre de los distintos parámetros. Estas reglas, como se ha mencionado, no sólo pueden ser creadas por medio de un fichero XML, sino que *Esponja* cuenta con una interfaz gráfica de forma que las asociaciones pueden ser creadas, configuradas o destruidas de forma interactiva mediante el ratón y el teclado, además de contar con la representación gráfica de las mismas.

La figura 9 muestra un aspecto del gráfico generado por la interfaz, en el cual se refleja cómo un evento ha sido asociado a más de un servicio. En este caso particular, la respuesta que se envía al agente que lanzó el evento queda seleccionada de entre todas por medio de un sistema de prioridades. También se refleja cómo es posible establecer asociaciones entre los eventos y servicios de un mismo agente.

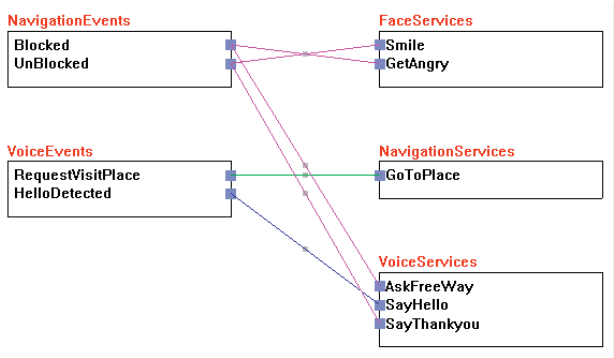

Figura 9. La interfaz de Esponja, permite la definición y visualización de las distintas asociaciones. En este ejemplo, cuando un evento de "Bloqueado" es emitido, la cara recibe una petición de "muéstrate Enfadado" y el sistema de voz recibe una petición de "solicita el paso". Una vez desbloqueado, se indica a la cara que sonría y al sistema de voz que de las gracias de forma análoga.

La traducción de los mensajes permite conectar servicios y eventos con un número, nombre y tipo de parámetros diferente. La figura 10 muestra de nuevo una representación generada por *Esponja* en el que se ha asociado el evento *RequestVisitPlace* del sistema de voz con el servicio *GoToPlace* del agente de navegación. Esta asociación corresponde además con el ejemplo del extracto del fichero XML mostrado anteriormente. Como se puede observar, el evento únicamente trasmite el nombre del lugar mediante un parámetro de tipo cadena de caracteres denominado *placeName*, mientras que el servicio requiere de dos argumentos, la velocidad -un decimal- y el lugar que es denominado como *place* y que es también una cadena de caracteres. La asociación establece la identidad entre *placeName* y *place* y fija la velocidad a un valor constante unitario.

Esponja se ha realizado en C++ y utiliza las MFCs (Microsoft Foundation Classes) para la implementación de la interfaz gráfica. Los autores consideran que es una herramienta muy útil no sólo para los sistemas robóticos y por ello se ha ofrecido como código abierto en (Jabón, 2009).

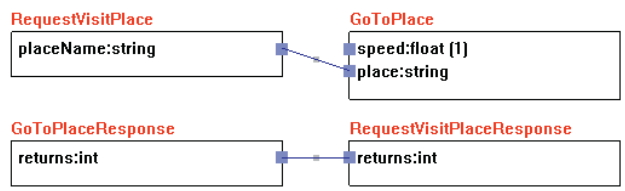

Figura 10. Esponja permite realizar asociaciones entre los parámetros de los mensajes entrante y saliente de servicios y eventos de forma flexible. En esta figura se muestra como el servicio *GoToPlace* es invocado con una velocidad unitaria y con el parámetro *place* obtenido del valor de *placeName* del evento *RequestVisitPlace*.

5. Experimentos

La arquitectura y sus funcionalidades se han probado principalmente mediante dos tipos de experimentos. En primer lugar se diseñó un experimento de laboratorio para probar la navegación, el seguimiento de personas y el mapeado 3D multi-robot. En segundo lugar se procedió a utilizar el sistema completo con los tres robots en un entorno real con usuarios ajenos al proyecto. En ambos casos, los diferentes agentes habían sido probados independientemente por los respectivos grupos de desarrollo en el laboratorio, con resultados satisfactorios. Esta fase experimental consistía en evaluar y verificar si *Esponja/Jabón* permitían una rápida integración y un correcto funcionamiento del sistema completo.

5.1. Experimento de laboratorio de mapeado multi-robot.

En este caso se trabajo exclusivamente con dos robots, *Urbano* y *Nemo*. Con ellos se procedió a realizar una instalación del sistema totalmente automática y que por tanto incluye el seguimiento de personas así como la navegación segura y mapeado, gracias a la combinación de la información obtenida por los dos robots. Para ello se seleccionó un pasillo amplio de la Universidad en el que se situaron tres obstáculos que no pueden ser detectados por el robot *Urbano* debido a la altura a la que se encuentra su laser.

Primeramente, *Urbano* es guiado por un entorno para él desconocido siguiendo a un operador. Durante esta fase, que duró algo menos de 2 minutos, el robot construye un mapa y registra la trayectoria. En la figura 3 se puede ver el mapa realizado, en el que no son visibles dichos obstáculos. Esta información es transmitida a *Nemo* el cual recorre la misma trayectoria por tramos y obtiene la representación tridimensional del entorno. En este experimento el tiempo que tardó el robot en realizar esta tarea fue de 6 minutos. La información actualizada (Figura 4) se comparte con *Urbano*, de forma que el robot es capaz de ir trazando trayectorias que esquivan los obstáculos para él invisibles.

Se ha realizado un video demostrativo del experimento accesible en (Valero *et al*., 2012), en el que la cámara graba de forma continua para mostrar como el procedimiento de configuración se realiza sin interrupción.

5.2. Robonauta en un entorno real

Se ha realizado una demostración en un entorno real: la Ciudad de las Artes y las Ciencias de Valencia (España). Este entorno es un complejo dedicado a la diseminación científica y cultural. Los experimentos se realizaron en el museo *Príncipe Felipe*. Durante los experimentos el museo ofrecía una exposición dedicada a los premios Nobel de las ciencias para lo que estaba dotado de paneles horizontales y verticales así como escaleras y barandillas adicionales a la disposición básica del museo. El área por la que se probó el sistema fue una parte del museo de 30x5 metros en la que muchos de los expositores y elementos del entorno no eran detectables por *Urbano*.

El sistema de configuración es similar al anterior, pero ahora emplea el sistema completo con los 3 robots, incluida la interacción por voz y la gestión de visitas a la exposición una vez instalado el sistema. En primer lugar *Urbano* recorre la exposición. El procedimiento es análogo al que se seguiría en el caso de querer contratar un nuevo guía. *Urbano* realiza esta operación de forma natural, simplemente siguiendo a la persona que le está enseñando el sitio y etiquetando lugares de particular interés mediante comandos de voz, como por ejemplo, salas o lugares en los que se encuentran determinadas pinturas o esculturas. Posteriormente, *Nemo* realiza autónomamente el mapa de traversabilidad y transfiere toda la información, incluidos los lugares de interés, a *Doris*; gracias a esta información, *ésta última* puede realizar la visita de forma segura a pesar de la existencia de

obstáculos no detectables por su sistema sensorial.

En la figura 11 se representa la arquitectura concreta empleada en este demostrador, en la que se aprecia el uso de tres instancias del *bróker Esponja*, una corriendo a bordo de cada robot. Este esquema busca dotar de mayor robustez al sistema dado que así los robots podrían seguir funcionando de forma autónoma aunque se perdieran temporalmente las comunicaciones inalámbricas. Todos los computadores están por simplicidad en la misma red local, conectados mediante *hubs* a bordo de cada robot y de forma inalámbrica con el exterior. El agente *Orquestador* es una sencilla aplicación que se encarga de gestionar la secuenciación de tareas a alto nivel. Nótese que *Esponja* utiliza una GUI de configuración basada en MFC y por tanto sólo ejecutable sobre Windows. Aunque *Esponja* es multiplataforma, es conveniente disponer de una interfaz gráfica para las tareas de configuración. En el resto de plataformas *Esponja* se ejecuta sin problemas prescindiendo de esta GUI.

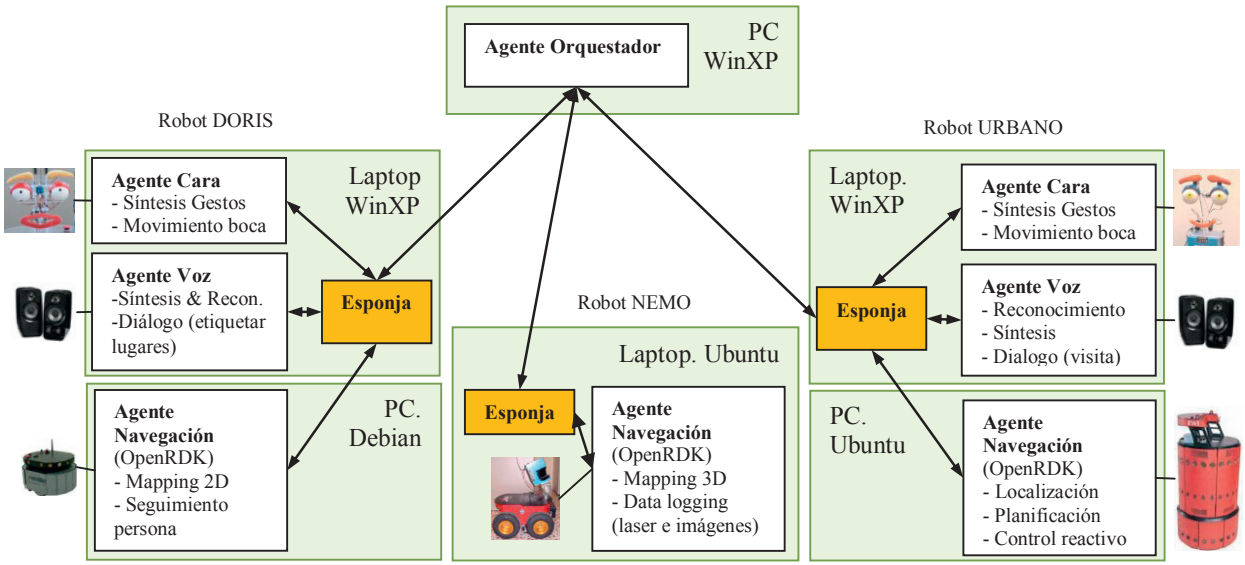

Figura 11. Diagrama de la arquitectura empleada en la demostración del museo Príncipe Felipe. Las flechas representan mensajes HTTP (llamadas a Servicios Web SOAP), todas ellos implementadas con *Jabon*. Las líneas continuas sin flecha, representan conexiones físicas (serie, audio) con los dispositivos y por tanto los PCs correspondientes están a bordo del robot respectivo.

La configuración del sistema representada en la figura 11 fue realizada íntegramente en el Museo. Nótese que cada agente se puede desarrollar de forma totalmente independiente y el código generado automáticamente por *Jabon* se compila dentro del mismo de forma sencilla. La configuración inicial de un agente para utilizar *Jabon* puede realizarse en 10 minutos aproximadamente, mientras que cada modificación de la interfaz del mismo se realiza de forma instantánea. Los ahorros de tiempo importantes en la integración del sistema no se obtienen por esta generación (otras herramientas también funcionan de forma similar, aunque algo más compleja; tareas equivalentes en gSOAP pueden tardar dos o tres veces más), sino por el hecho de que no es necesario generar ni compilar el código complementario de conexión para los otros agentes conectados. Otras herramientas requerirían el desarrollo de una aplicación centralizadora ad-hoc para cada robot que integrara los mencionados códigos complementarios interconectándolos programáticamente. Aunque dichas aplicaciones son relativamente sencillas, en el proceso de desarrollo cada cambio en un agente obliga a recompilar también dicha aplicación centralizadora, haciendo esta tarea larga y tediosa.

Hemos estimado por nuestra experiencia previa que el desarrollo necesario para implementar la arquitectura de la figura 11 sería aproximadamente de unas 9 horas.

Con las nuevas herramientas *Jabon* y *Esponja*, los tiempos se han recortado en un orden de magnitud, dado que no es necesario desarrollar dichas aplicaciones. Cada cambio de cada agente se puede hacer sin alterar en absoluto el resto de elementos, cambiando "en caliente" la configuración del bróker *Esponja* con unos "clicks" de ratón, sin necesidad de propagar cambios en ficheros de código ni recompilar. De hecho, a veces no es ni siquiera necesario cambiar la configuración, dado que *Esponja* establece mapeados de parámetros por defecto, tal y como se ha descrito anteriormente. El tiempo necesario para definir la arquitectura representada en la figura 11 es el que requiere la definición de las reglas de los *bróker Esponja* de los 3 robots, lo que puede ser realizado fácilmente en menos de una hora.

Figura 12. *Urbano, Nemo* y *Doris* antes de comenzar la configuración

La Figura 12 muestra una fotografía adquirida antes de comenzar el proceso de configuración. En (Valero *et al*., 2012) se puede ver un video del guiado de *Urbano* por el museo (Figura 13). A continuación (Figura 14) *Nemo* construye el mapa de

navegabilidad y adquiere las fotos que permiten generar un modelo texturizado del entorno (Figura 15). En (Valero *et al*., 2012) se puede ver un video del museo modelado. Finalmente *Doris* guía a una persona por el museo (Figura 16).

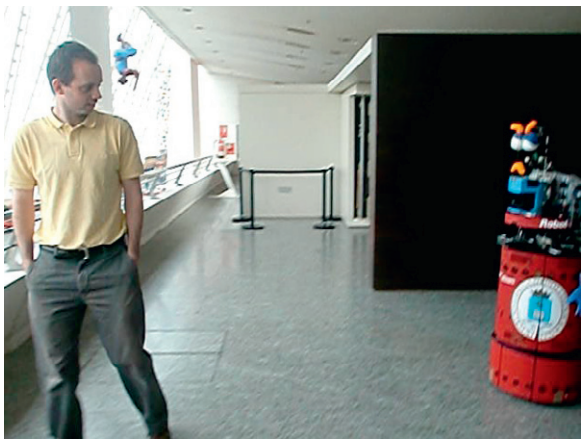

Figura 13. *Urbano* siguiendo a una persona

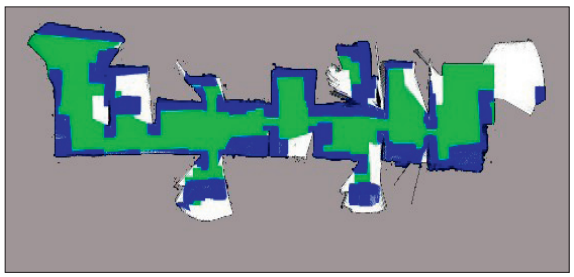

Figura 14. Mapa de navegabilidad construido por *Nemo*. En verde se refleja la zona navegable, en azul la no navegable, en la que se aprecian algunos expositores pegados a las paredes que no eran visibles para *Urbano* ni *Doris*, y sin embargo han sido eficazmente etiquetados como obstáculos.

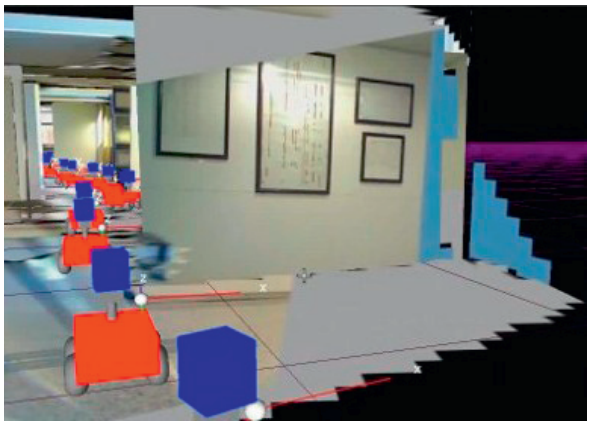

Figura 15. Mapa 3D texturizado del entono generado por *Nemo*. Se han sobreimpreso las posiciones sucesivas utilizadas por el robot para capturar la información 3D.

6. Conclusiones

Este artículo ha presentado una novedosa arquitectura basada en los estándares de los servicios web que permite la integración de agentes totalmente heterogéneos en tiempo de ejecución, permitiendo el desarrollo desacoplado de agentes por diferentes equipos de investigación, de una forma ágil y sencilla. Aunque es difícil de valorar cuantitativamente, se podría estimar que el tiempo dedicado a tareas de integración se ha visto reducido en un orden de magnitud, incluido el tiempo necesario de aprendizaje de la nueva arquitectura.

Figura 16. *Doris* guía a una persona hacia el punto de interés que se le ha solicitado por medio de un comando de voz. Para ello *Doris* planifica sobre el mapa de navegabilidad generado en la fase de configuración inicial.

La arquitectura aquí presentada parece tan apropiada que ya ha sido adoptada por los otros equipos de investigación para proyectos ajenos a la robótica. Por último, se han liberado las herramientas software de dicha arquitectura como *open-source*, esperando que sean de beneficio para una comunidad más amplia y que otros puedan contribuir en la herramienta mejorándola o completándola con funcionalidades que no se habían contemplado. El código de Jabón/Esponja se puede descargar de (Jabon, 2009), mientras que OpenRDK está disponible en (Calisi *et al.*, 2008). Se están estudiando nuevas formas genéricas de intercomunicación de información entre procesos y desarrollo desacoplado, intentando extender la arquitectura a un público más amplio.

Las herramientas desarrolladas han sido utilizadas con notable éxito en una demostración del proceso de instalación de un robot guía interactivo en un nuevo lugar mediante la colaboración de diferentes robots móviles y de forma natural, únicamente mediante la presencia y la voz de un guía, de tal forma que dicha instalación pueda ser ejecutada por personal no técnico con un mínimo entrenamiento.

Para la navegación de los mismos se han utilizado principalmente algoritmos existentes y conocidos en la comunidad robótica, como GMapping, y se han realizado algunos desarrollos propios auxiliares para completar el demostrador, que aunque necesarios, no aportan una especial contribución al área de la navegación autónoma. No obstante nos gustaría resumir aquí nuestra perspectiva y lecciones aprendidas al respecto.

Posiblemente uno de los puntos más débiles de la navegación autónoma con los algoritmos y soluciones empleados reside en su falta de robustez: la mayoría de los sistemas apenas son capaces de gestionar errores imprevistos, como un laser que deja de funcionar enviando siempre los mismos datos, o simplemente un grupo de gente que rodea totalmente al robot durante un largo

periodo de tiempo bloqueando su visibilidad casi totalmente, y haciendo fallar típicamente los algoritmos de localización sin tener generalmente información de dicho fallo.

En entornos estructurados existen obstáculos a diferentes alturas que un sensor 2D puede no detectar. Sin embargo, es posible llegar a conseguir una navegación robusta con percepción puramente bidimensional en ciertos entornos muy controlados como un museo, asumiendo la hipótesis que los obstáculos peligrosos (escaleras abajo, barandillas y postes, cristales no detectados por el laser) son estáticos y su posición puede ser anotada en el mapa del robot.

No obstante, la aplicabilidad de los algoritmos existentes bidimensionales no es factible en la mayoría de los entornos reales no planos (garajes, museos con exhibiciones a diferentes alturas). Nuestro trabajo de investigación actual se basa en navegación autónoma y robusta basada en modelos geométricos tridimensionales (como el de la figura 15), y de forma totalmente independiente del *framework*, de tal forma que se pueda reutilizar también en otros sistemas como ROS.

Entre los trabajos futuros se incluye una experimentación no sólo en un entorno real, como se ha presentado en este artículo, sino también durante su uso habitual, es decir, con público.

English Summary

Web Services based integration architecture for heterogeneous and distributed systems: application to interactive mobile robots

Abstract

Development of complex robotics systems usually requires tools and software frameworks such as the WillowGarage Robot Operating System (ROS) to ease code integration and reuse tasks. Nevertheless, research in robotics is increasingly becoming multidisciplinary, involving areas of voice technology, artificial intelligence or web systems, which can make researchers highly reluctant to use such robotic-specific systems. Even existing constraints, such as using different Operating Systems, can make this task impossible. This paper presents a novel architecture completely based on standards such as Web Services, that allows a totally decoupled development of agents with minimal intrusiveness and high simplicity by using a dynamic broker to interconnect agents at runtime. This architecture, which does not replace other systems but complements them. It has been used successfully inside a complex multi-robot demonstrator in a fully automated system deployment of an interactive tour guide robot in a new environment.

Keywords: software architecture, system integration, robot control.

Agradecimientos

Este proyecto ha sido financiado parcialmente por el plan nacional de I+D (Proyectos Robonauta: DPI2007-66848-C02-01, Arabot: DPI 2010-21247-C02-01) y supervisado por CACSA, a quien agradecemos su colaboración.

Referencias

Alonso-Martín F., Gonzalez-Pacheco V., Castro-González A., Ramey A. A., Yébenes M. y Salichs M.A. , 2010. Using a Social Robot as a Gaming Platform. Lecture Notes in Computer Science. Volume 6414/2010, 30-39.

Bruyninckx H., 2001. Open robot control software: the OROCOS project. . IEEE International Conference on Robotics and Automation ICRA., Vol. 3, pp. 2523-2528. doi:10.1109/ROBOT.2001.933002

Burgard W., Cremers A.B., Fox D., Hähnel D., Lakemeyer G., Schulz D., Steiner W., Thrun S. ,1999. Experiences with an interactive museum tour-guide robot. Artificial Intelligence. Vol. 1-2 N. 114. pp. 3-55.

Calisi D., Farinelli A., Iocchi L., and Nardi D. 2005. Autonomous navigation and exploration in a rescue environment. In Proceedings of IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR), pages 54–59, Kobe, Japan, June 2005. ISBN: 0-7803-8946-8.

Calisi D., Censi A., Iocchi L., y Nardi D., 2008. OpenRDK: a modular framework for robotic software development. In Proceedings of the International Conference on Intelligent Robots and Systems (IROS). Nice, France. pp. 1872-1877. http://openrdk.sf.net

Doucet, A.; De Freitas, N.; Gordon, N.J. (2001). Sequential Monte Carlo Methods in Practice. Springer

de la Puente P., Rodriguez-Losada D., Valero A., Matia F., "3D Feature Based Mapping Towards Mobile Robots' Enhanced Performance in Rescue Missions," in Proc. Of the IEEE Int. Conference on Intelligent Robots and Systems (IROS) St. Louis, Missouri, USA, October 2009.

Gross H.-M., Bohme H.-J., Schroter C., Müller S., König A., Martin C., Merten M., and Bley A., 2008. "Shopbot: Progress in developing an interactive mobile shopping assistant for everyday use," in Proc. of the Int. Conf. on Systems, Man and Cybernetics.

Grisetti G., Stachniss C., y Burgard W., 2005. "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," IEEE Intern. Conf. on Robotics & Automation.

Jabon, 2009. Jabon a C++ SOAP toolkit. www.intelligentcontrol.es/jabon

Lacey, G., Rodriguez-Losada, D. 2008. The evolution of Guido: a smart walker for the blind. Robotics & Automation Magazine,. Volume: 15, Issue: 4. pp.: 75-83. ISSN: 1070-9932. DOI: 10.1109/MRA.2008.929924

Makarenko A., Brooks A., Kaupp T., 2007. On the Benefits of Making Robotic Software Frameworks Thin. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Workshop on Evaluation of Middleware and Architectures.

Montemerlo M., Pineau J., Roy N., Thrun S., and Verma, V., 2002. Experiences with a Mobile Robotic Guide for the Elderly. Proceedings of the AAAI National Conference Artificial Intelligence, Edmonton, Canada.

Nourbakhsh I., Bobenage J., Grange S., Lutz R., Meyer R., and Soto A. ,1999. An Affective Mobile Educator with a Full-time Job. Artificial Intelligence, Vol. 114, No. 1 - 2, pp. 95-124.

Pedraza L., Rodriguez-Losada D., Matia F., Dissanayake G., y Valls Miro J., 2009. Extending the Limits of Feature-Based SLAM With B-Splines. IEEE Transactions on Robotics. Volume: 25 Issue: 2. pp 353 - 366. ISSN: 1552-3098 Digital Object Identifier: 10.1109/TRO.2009.2013496

Robert A. van Engelen and Kyle Gallivan, 2002. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks, in the proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002), pages 128-135, May 21-24, Berlin, Germany.

Rodriguez-Losada D., Matia F., Pedraza L., Jimenez A., Galan R.. 2007. Consistency of SLAM-EKF Algorihtms for Indoor Environments. Journal of Intelligent and Robotic Systems. Springer. DOI: 10.1007/s10846-007-9171-8. ISSN 0921-0296, Vol. 50, Nº. 4, pags. 375-397

Rodriguez-Losada D., Matia F., Galan R., Hernando M., Montero J. M. y Lucas J.M., 2008. Urbano, an Interactive Mobile Tour-Guide Robot.. Advances in Service Robotics. Edited by: Ho Seok Ahn. ISBN 978-953-7619-02-2. Hard cover, 342 pages. InTech Education and Publishing.

Thrun S., Bennewitz M., Burgard W., Cremers A.B., Dellaert F., Fox D., Hahnel D., Rosenberg C., Roy N., Schulte J., Schulz D. ,1999. MINERVA: A Second-Generation Museum Tour-Guide Robot. IEEE International Conference Robotics and Automation. Vol.3, pp. 1999-2005.

Valero A. and Randelli G. and de la Puente P. and Calisi D. and Rodrìguez-Losada D. and Matìa F. and Nardi D. 2009. "UPM-SPQR Rescue Virtual Robots 2009 Team Description Paper." Robocup 2009, Graz, Austria.

Valero A., de la Puente P., Rodriguez-Losada D., Hernando. M. Videos demostrativos, 2012: http://goo.gl/PuWaO, http://goo.gl/eHXFO, http://goo.gl/wtorB, http://goo.gl/H30fN