MOSAIC: MOdular System AIbo Control

Carlos Agüero, Francisco Martín, Vicente Matellán, José María Cañas
Grupo de Robótica de la Universidad Rey Juan Carlos
28933 Móstoles, Madrid (Spain)
{caguero, fmartin, vmo, jmplaza}@gsyc.es

Resumen

El objetivo de la arquitectura MOSAIC (MOdular System for AIbo Control) es disponer de un entorno modular y con un interfaz lo suficientemente general como para servir de soporte para el diseño y desarrollo de comportamientos en grupos de robots aiBo. Los comportamientos habituales en este tipo de plataformas incluyen las tareas de navegación, localización, cooperación en entornos multi-agente, etc.
La arquitectura de control presentada consta de tres módulos débilmente acoplados: MOSACHigh (tareas de alto nivel), MOSAICLow (tareas de bajo nivel) y MOSAICTcm (tareas relacionadas con las comunicaciones). El módulo MOSAICLow se encarga de resolver los aspectos relativos a la locomoción, el acceso a los sensores/actuadores y pretende ofrecer mayor grado de abstracción que el ofrecido por las imagenes sin procesar. Típicamente realiza tareas de filtrado, segmentación e identificación de objetos. El módulo MOSAICHigh se encarga de la gestión de todos los comportamientos o conductas de alto nivel. Por último, el módulo MOSAICTcm es el responsable de gestionar las comunicaciones entrantes y/o salientes.

Palabras Clave: Arquitectura Software, aiBo

1 INTRODUCCIÓN

En este artículo presentamos la arquitectura MOSAIC (*MOdular System for AIbo Control*). El objetivo perseguido con el diseño de esta arquitectura es disponer de un entorno modular y con un interfaz lo suficientemente general, como para servir de soporte para el diseño y desarrollo de comportamientos en grupos de robots aiBo. Los comportamientos habituales en este tipo de plataformas incluyen tareas de navegación, localización, cooperación en entornos multi-agente, etc. La arquitectura presentada ofrecerá el soporte para que dichos algoritmos puedan implementarse fácilmente sobre el sistema Open-R del robot aiBo de Sony.

La arquitectura MOSAIC es heredera directa de la arquitectura del TeamChaos (www.teamchaos.es), equipo conjunto para la categoría de robots cuadrúpedos formado actualmente por los grupos de robótica de las universidades de Murcia y Rey Juan Carlos, y originariamente también por las universidades de Alicante y Orebro (Suecia). La arquitectura del TeamChaos a su vez es una implementación de la arquitectura ThinkingCap (http://robolab.inf.um.es/tc2), basada a su vez en la arquitectura distribuida BGen [8]. Estas arquitecturas han sido utilizadas por el grupo de ANTS de la Universidad de Murcia en diversos robots móviles como el coche autónomo MIMICS[6], o robots comerciales como el Nomad 200, B21 o el Pioneer3 AT. MOSAIC puede considerarse de hecho un subconjunto simplificado para su uso simplificado únicamente con el robot aiBo.

La modularidad de MOSAIC es su característica más destacable, orientada a facilitar la reusabilidad de los componentes. Esta característica, general para cualquier sistema software, resulta especialmente útil en el campo de la robótica puesto que los nuevos comportamientos pueden conseguirse modulando o combinando los ya existentes. De esta forma han surgido diferentes entornos software en los últimos años para ofrecer sistemas de programación orientada a objetos para las diversas plataformas de robótica móvil.

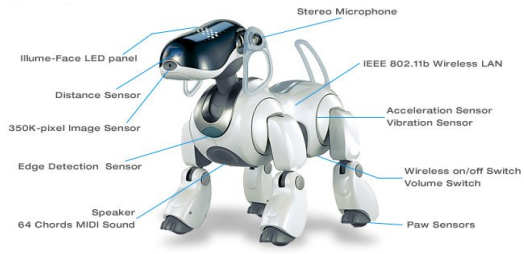

Figura 1: Sony aiBo ERS7. Vista frontal (copyright Sony Corp.)

Algunas proporcionadas por los propios fabricantes como ARIA[1] para los robots Pioneer de ActivMedia, que antes usaban las bibliotecas Saphira [7]; Mobility [11] para los B21 de iRobot (antes RWI).

Otras han sido desarrolladas para ser independientes del hardware, como capas intermedias, como por ejemplo Player/Stage [3], Carmen [9], Marie [4], Miro [12], CLARAty [10], etc. o la reciente propuesta de Microsoft, denominada Robotics Studio y que se basa en la tecnología .Net de la misma compañía.

Sin embargo, ninguna de estas propuestas facilita el desarrollo para el robot aiBo de Sony, cuyo

sistema operativo completamente orientado a objetos es un escalón de entrada demasiado brusco para multitud de desarrolladores. Para tratar de solventar este problema hemos diseñado MOSAIC, que facilita el desarrollo de comportamientos para la familia de robots aiBo (ERS-111, ERS-7, etc.).

En concreto, las pruebas descritas en este artículo han sido desarrolladas para el Sony aiBo ERS7 (figuras 1 y 2). Se trata de un robot móvil autónomo equipado con un microprocesador MIPS a 576MHz. y con 64MB de memoria principal. Su principal sensor es una cámara en color de *350K píxeles* colocada en el morro, 2 sensores de infrarrojos, un acelerómetro, un micrófono estéreo y sensores de contacto en las patas para saber si están apoyadas en el suelo o no. Finalmente, dispone de diversos sensores táctiles en la cabeza, barbilla y espalda.

El robot se mueve e interacciona con el entorno mediante diversos actuadores incluyendo un altavoz, 20 motores para mover cada una de las 3 articulaciones de cada pata, las 3 de la cabeza, su boca, sus 2 orejas y 2 articulaciones en el rabo. Finalmente, el robot aiBo dispone también de capacidad de comunicación inalámbrica mediante una tarjeta WiFi IEEE 802.11b.

Figura 2: Sony aiBo ERS7. Vista posterior (copyright Sony Corp.)

Como es fácil de entender, el control de un robot móvil con 20 grados de libertad es complejo, simplemente el diseño de la forma de caminar ha sido objeto de múltiples investigaciones, como por ejemplo [5] y las desarrolladas en el equipo TeamChaos para el aprendizaje automático de los parámetros[2] del que forman parte los autores de este trabajo. Igualmente, el uso de los sensores y comunicaciones del aiBo es realmente complejo, como se detallará en la sección II de este artículo, por lo que hemos creído oportuno diseñar una arquitectura que resuelva la locomoción, permitiendo comandar fácilmente al robot en velocidad, y el acceso a los sensores y comunicaciones. Ésta es la motivación principal de la arquitectura MOSAIC, facilitar el desarrollo de aplicaciones sobre esta plataforma.

El resto del artículo se organiza de la siguiente forma: en la sección II se discuten las ideas principales del sistema operativo Open-R que proporciona Sony para el robot aiBo. En la sección III se presenta la estructura de la arquitectura MOSAIC, detallando los componentes y funcionalidades básicas. La sección IV presenta un ejemplo de desarrollo de un comportamiento sencillo. Finalmente, en la sección V se presentan las conclusiones y trabajos futuros previstos.

2 EL SISTEMA Open-R

El robot aiBo se comercializó inicialmente como un sistema de entretenimiento robótico. Su atractivo diseño y su fiabilidad enseguida atrajeron la atención de los desarrolladores, lo que llevó a Sony a publicar el API de su sistema operativo. Se trata de un *FrameWork* en C++ que se sitúa sobre el sistema operativo APERTOS y que Sony ha proporcionado bajo el nombre de Open-R SDK (resumida en la figura 3).

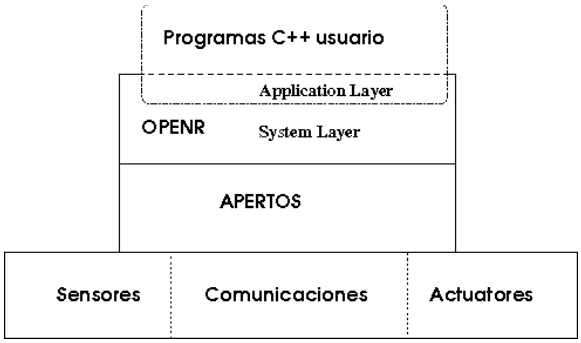

Figura 3: Arquitectura MOSAIC

APERTOS es un sistema operativo empotrado basado en una arquitectura de meta-nivel [14][15] cuyo diseño ha tenido gran influencia en la forma compleja en que se programa con Open-R. APERTOS se ha utilizado en diversos equipos de Sony, además del robot aiBo, como por ejemplo el receptor de satélite DST-MS9, aunque al tratarse de secretos comerciales hay muy poca información al respecto y fue una de las barreras que hubo que superar para publicar Open-R.

La aplicaciones en Open-R se realizan estructurándolas como objetos, donde no hay un programa principal, sino un conjunto de objetos concurrentes que se intercambian mensajes. Estos objetos se desarrollan en una estación de trabajo convencional y se insertan en el perro utilizando una memoria de estado sólido (*memory-stick)* junto con ciertos ficheros de configuración diversos, por ejemplo la configuración de red (ver [16] para más detalles).

Cada objeto Open-R encapsula el estado, métodos que acceden a él y un procesador virtual que ejecuta sus métodos. La comunicación entre objetos se realiza mediante paso de mensajes y la ejecución está dirigida por eventos. Así, después de la inicialización un objeto está en estado *idle* y sólo la llegada de un evento disparará la ejecución de alguno de sus métodos. Se distinguen con prioridades eventos relacionados con los mensajes entre objetos y

aquellos generados por el hardware. Un evento puede ser por ejemplo un mensaje enviado desde un objeto. La información que se pasan dos objetos se intercambia mediante una zona de memoria compartida. Cuando un objeto está realizando una operación (típicamente un método) no será interrumpido por ningún evento hasta que termina para evitar condiciones de carrera. Los mensajes que no pueden ser tratados inmediatamente se almacenan en un *buffer* de memoria compartida. Open-R no proporciona métodos transparentes de protección de la memoria compartida, lo que complica la programación en Open-R directamente.

Open-R diferencia dos niveles: el *system layer* y el *application layer*. El primero contiene los servicios necesarios para acceder al hardware del robot. El *application layer* es el programado por el usuario, como muestra la figura 3.

Los tres objetos principales que proporciona el *system layer* son:

1. `OVirtualRobotComm`: que proporciona acceso a los sensores y actuadores del robot y a la cámara.
2. `OvirtualRobotAudioComm`: que proporciona acceso al micrófono y altavoz.
3. `ANT`: que proporciona la implementación de TCP/IP.

El concepto de objeto en Open-R es similar al de proceso en UNIX, con la diferencia de que los objetos son mono-hilo (*single-thread*) y que la comunicación entre ellos se realiza mediante paso de mensajes. Los mensajes contienen los datos (cualquiera en C++) y un identificador que especifica qué método se ejecutará cuando llegue el mensaje. De esta forma, cada objeto Open-R tiene varios puntos de entrada por los que le pueden llegar mensajes, que se especifican en tiempo de compilación en un fichero de configuración similar a los IDL de CORBA que se denomina `STUB.CFG`

La exclusión mutua se resuelve utilizando la clase `RCRegion` de Open-R que puede acceder a un segmento de memoria compartida, llevando un contador de objetos que apuntan a ella.

El flujo de ejecución de los objetos Open-R es el que ilustra la figura 4 y que básicamente consiste en:

- El objeto se carga al arrancar el robot, se ejecuta el `DoInit` de cada objeto.
- Los objetos esperan la llegada de un mensaje.
- Cuando llega un mensaje, el método correspondiente es invocado.
- Cuando un método termina su ejecución, espera la llegada de un nuevo evento.

En la figura 4 se identifican dos objetos, uno que se ha marcado como *Observer* (el que recibe el mensaje) y otro como *Subject* (el que lo envía) en la nomenclatura de Open-R. Obviamente, un objeto puede tener tanto un rol como el otro. Un *Observer* se da cuenta de que ha llegado un mensaje cuando recibe un evento `Notify` del *Subject*. Este por su parte, sabrá que el *Observer* está dispuesto a recibir más datos cuando reciba un evento *Ready*, tal y como refleja la figura 4.

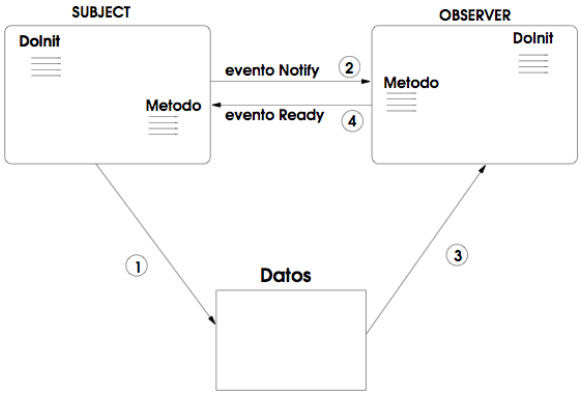

Figure 4: Flujo de datos entre objetos

Este mecanismo, completamente concurrente y orientado a objetos, hace difícil la implementación de aplicaciones en muchos casos, en especial en programadores noveles, como por ejemplo los estudiantes de últimos cursos de las titulaciones de grado, según hemos podido comprobar al intentar utilizarlo en diversos proyectos fin de carrera. En vista de lo cual hemos considerado necesario desarrollar una arquitectura que evite la complejidad de Open-R y facilite el desarrollo de aplicaciones finales sobre el robot aiBo.

3 LA ARQUITECTURA DE MOSAIC

La arquitectura presentada se puede dividir en dos grandes bloques: *Mosaic_robot* y *Mosaic_manager*. El primero de los bloques engloba el código desarrollado para ser ejecutado en el robot aiBo. *Mosaic_manager* es una *suite* de herramientas de monitorización y depuración de algunos de los módulos de la arquitectura que se ejecutan en un ordenador externo.

La arquitectura de control de *Mosaic_robot* consta de tres objetos Open-R: `MosaicHigh`, `MosaicLow` y `MosaicTcm`. La misión de cada uno de estos objetos es encapsular las tareas de alto nivel, bajo nivel, como por ejemplo el modo de caminar, y los aspectos relativos a las comunicaciones respectivamente.

Cada uno de estos objetos Open-R consta de varias clases escritas en C++, que implementan determinadas tareas. En la figura 5 se muestran las principales tareas que resuelve cada módulo de *Mosaic_robot*.

En el objeto `MosaicHigh` encontramos la clase `Controller` que se encarga de guiar los comportamientos de alto nivel. Estos comportamientos describen la conducta del robot y

típicamente es el sitio donde los futuros desarrolladores incluirán su código.

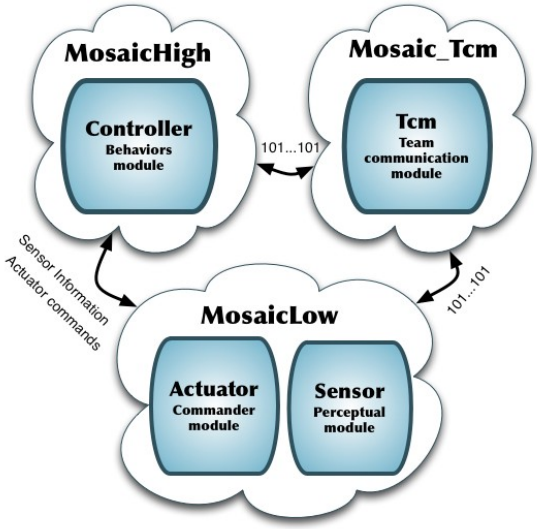

Figure 5: Arquitectura de MOSAIC_robot

El objeto `MosaicLow` contiene, entre otras clases secundarias, dos muy importantes: *Actuator* y *Sensor*. La primera de ellas proporciona un interfaz básico para acceder a los actuadores del robot, en particular ofrece un interfaz para controlar al robot en velocidad basado en el modo de caminar desarrollado por el German-Team [5]. A su vez, la clase *Sensor* ofrece un interfaz similar para el acceso a todos los sensores.

Por último, el objeto `MosaicTcm` incluye principalmente la clase *Tcm*. El objetivo de esta clase es permitir la comunicación bidireccional entre otros robots o PC's siempre que se respete el protocolo establecido.

El mecanismo de comunicación entre objetos Open-R siempre desata un evento en el objeto receptor y el intercambio de información a través de una región compartida. En el caso de comunicación desde el objeto `MosaicHigh` hasta el objeto `MosaicLow` la información intercambiada corresponde a comandos dirigidos a los actuadores del robot. Si la información fluye en el sentido inverso, lo habitual es que los datos reflejen el estado de algún sensor o la captura de una imagen. A su vez, la entidad intercambiada entre cualquier objeto y el `MosaicTcm` siempre es una secuencia de bytes, para así permitir un envío/recepción de datos más flexible.

Uno de los principales objetivos de MOSAIC es recubrir todos los detalles escabrosos del paradigma de programación basado en Open-R y proporcionar un módulo `MosaicHigh`, donde los programadores puedan desarrollar sus conductas sin necesidad de conocer los detalles de Open-R. Es labor de la arquitectura presentada aumentar el grado de abstracción y proporcionar la transparencia suficiente para que el acceso a los sensores/actuadores y el uso de las comunicaciones sea una labor sencilla.

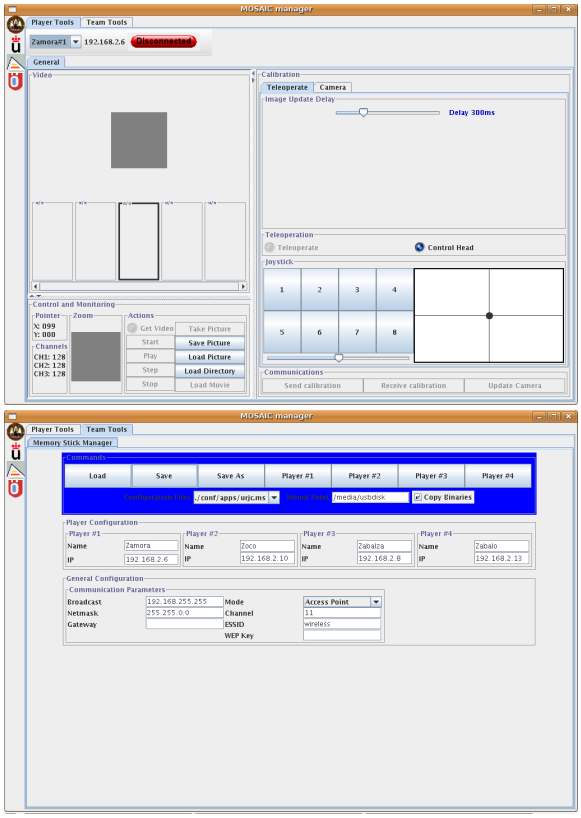

Figure 6: Capturas de la herramienta *Mosaic_manager*

Otra de las ventajas que ofrece *Mosaic_robot* frente al API crudo de Open-R es que toda la comunicación entre objetos se gestiona de manera transparente por la arquitectura de control. El objeto `MosaicHigh` dispone de una capa de software que recubre tanto la comunicación con el objeto `MosaicTcm` para el acceso a las comunicaciones, como la comunicación con el objeto `MosaicLow` para el acceso a sensores/actuadores. Por tanto, al desarrollador se le da la apariencia de que todas las primitivas están en su propio objeto Open-R.

Por otro lado, la herramienta *Mosaic_manager* ejecutandose en un PC externo permite automatizar algunas tareas, servir de herramienta visual de monitorización, etc. En la figura 6 se muestran dos capturas de la aplicación. Se dispone de un interfaz de tele-operación para mover el robot por el entorno y visualizar las imágenes obtenidas por su cámara. Además, es posible realizar una calibración de la cámara desde la aplicación. La generación de los *memory sticks* también se lleva a cabo a través del *Mosaic_manager*, configurando los parámetros de la red, punto de montaje, etc.

Otro de los *plugins* o módulos del *Mosaic_manager* permite tele-operar de manera individual cada una de las articulaciones. Una vez que se tienen todas las articulaciones en su posición

deseada se puede hacer una *foto* de esa postura y componer secuencias de movimientos. Estos movimientos fijos pueden ser ejecutados en cualquier momento, a criterio del comportamiento que esté gobernando el robot.

El diseño software está basado en pestañas, de foma que cada comportamiento implementado en el robot puede interactuar de forma independiente con la herramienta de monitorización.. Por tanto, la idea del *Mosaic_manager* es ofrecer un conjunto estándar de herramientas útiles para todos los desarrolladores y permitir que cada uno de ellos extienda su funcionalidad de manera sencilla. Así, por ejemplo, se puede disponer de módulos de monitorización de localización, de comportamientos cooperativos, etc.

4 DESARROLLO DE UN COMPORTAMIENTO BÁSICO CON MOSAIC

En el diseño de la arquitectura MOSAIC se ha cuidado que la sencillez en el desarrollo de comportamientos sea un factor importante. Crear un comportamiento, por básico que sea, directamente sobre Open-R puede convertirse en una labor compleja, aún disponiendo de un módulo de movimiento previamente desarrollado.

Input: PerceptualData
Output: CommandData
Command.Data.velocidad_lineal = 250
Command.Data.velocidad_lateral = 0
if *PerceptualData.sensor_distancia < 30 cm*
then
    CommandData.velocidad_rotación=100
    if *PerceptualData.sensor_distancia < 10 cm*
    then
        CommandData.velocidad_lineal=0
    end
else
    CommandData.velocidad_rotación=0
end

Algoritmo 1: Comportamiento básico esquivar

Esto se debe a la dificultad en sincronizar los ciclos de control en un sistema asíncrono guiado por eventos tales como la recepción de mensajes de otros módulos o información sensorial del propio robot. También tal complejidad se debe a la forma de extraer la información en crudo de los sensores.

Por el contrario, realizar un comportamiento básico en MOSAIC es una tarea sencilla en la que se ha de conocer únicamente las dos abstracciones que se han desarrollado para el envío de comandos de movimiento (`CommandData`) y recepción de información de percepción (`PerceptualData`). El ciclo de control principal se sitúa en el módulo `Controller` del objeto `MosaicHigh` (figura 5). Este ciclo de control se ejecuta cada vez que se recibe información sensorial del módulo Sensor, situado en el objeto `MosaicLow`. Es aquí donde los comportamientos básicos han de ser implementados. Estos comportamientos analizarán la información recibida, crearán los comandos de movimiento adecuados y por último, se los enviarán al módulo `Actuator`, situado también en el objeto `MosaicLow`.

Un ejemplo sencillo de comportamiento básico en MOSAIC es uno reactivo de evitación de obstáculos[1] . La idea es demostrar que implementar un comportamiento sencillo con MOSAIC es muy fácil. El algoritmo implementado se describe en el algoritmo 1.

Como entrada tenemos la última percepción recibida del módulo *Sensor*, y cuando finalice este ciclo de control, los comandos de movimiento serán enviados al módulo `Actuator`. El robot mantiene un velocidad lineal constante. Si el sensor de distancia detecta un obstáculo a menos de 30 centímetros, el robot gira asignándole una velocidad de rotación no nula. Si el obstáculo está a menos de 10 centímetros, el robot únicamente gira, asignándole 0 a la velocidad lineal.

```
1  CommandData     cdata;
2  PerceptualData  pdata;

3  memcpy (&pdata, (PerceptualData *)event.Data(0),
           sizeof (PerceptualData));

5  cdata.vr.vlin = 250;
6  cdata.vr.vlat = 0;

7  if(pdata.getNHeadPSD() < 300000)
   {
8       cdata.vr.vrot = 100;
9       if(pdata.getNHeadPSD() < 100000)
10                    cdata.vr.vlin = 0;
   }
11 else
12      cdata.vr.vrot = 0;

13 subject[sbjSetCommandData]->SetData(&cdata,
                sizeof(CommandData));
14 subject[sbjSetCommandData]->NotifyObservers();
```

El código que implementa este comportamiento es el mostrado anteriormente. Una vez recibida la información del módulo *Sensor* (línea 3), el sencillo algoritmo se materializa en las líneas 5 a la 12 del fragmento de código anterior.

Las líneas 13 a la 14 son únicamente para el envío de los comandos de movimiento al modulo `Actuator`.

Con esta pequeña porción de código se ha conseguido un comportamiento aparentemente inteligente en un robot sofisticado. Cualquier comportamiento que se desee realizar puede usar de la misma manera la información de los sensores y realizar el movimiento adecuado, de una manera sencilla e intuitiva.

[1]Un vídeo demostrativo de este ejemplo puede encontrarse en http://wwww.robotica-urjc.es/videos/mosaic-demo1.mpg

5 CONCLUSIONES Y TRABAJO FUTURO

5.1 Conclusiones

En este artículo hemos presentado brevemente la arquitectura MOSAIC, orientada a facilitar el desarrollo de aplicaciones para el robot aiBo de Sony.

Una de sus características principales es su simplicidad, de hecho ese es su objetivo principal. En particular, en la actualidad está siendo empleada por diversos alumnos de grado para el trabajo final de carrera. Hasta ahora, los alumnos de grado han sido prácticamente incapaces de usar el sistema Open-R por la gran barrera que supone comprender su sofisticada arquitectura.

5.2 Trabajo Futuro

La arquitectura MOSAIC está todavía dando sus primeros pasos, por lo que los primeros trabajos se centrarán en garantizar su estabilidad y depurar las posibles erratas que aparezcan. Simultáneamente se irá desarrollando un repertorio de comportamientos básicos, que aprovechando las características de modularidad de MOSAIC podrán utilizarse para construir comportamientos más sofisticados. Finalmente, estamos estudiando el rendimiento de la arquitectura, lo que habrá que completar y formalizar.

Agradecimientos

Los autores quieren agradecer especialmente el trabajo del grupo ANTS de la universidad de Murcia donde se ha desarrollado gran parte de la arquitectura del TeamChaos, de la que MOSAIC es un subconjunto.

Este trabajo ha sido parcialmente financiado por la Comunidad de Madrid dentro de la red RoboCity 2030 (S-0505/DPI/0176) y por el Ministerio de Educación dentro del proyecto ACRACE (DPI2004-07993-C03-01).

Referencias

[1] ARIA reference manual. ActivMedia Robotics.

[2] Walk calibration in a four-legged robot. B. Bonev, M.A. Cazorla y H. Martínez. Proceedings of the 8th Int. Conf. on Climbing and Walking Robots (CLAWAR 2005), pp. 493-500, Londres 2005.

[3] T. Collett, B. MacDonald and B. Gerkey, Player 2.0: Toward a Practical Robot Programming Framework, *Australasian Conf. on Robotics and Automation (ACRA 2005)*, Sydney (Australia), 2005.

[4] C. Cote, Y. Brosseau, D. Letourneau, C. Raievsky adn F. Michaud, Robotic software integration using MARIE, *Int. J. of Advanced Robotic Systems.*, vol. 3, no. 1, 2006, pp 55-60.

[5] Reliable and Precise Gait Modeling for a Quadruped Robot. U. Düffert y J. Hoffmann. RoboCup 2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence (LNCS 4020).

[6] MIMICS: Exploiting Satellite Technology for an Intelligent Convoy, A. Gómez, H. Martínez, M. Zamora, B. Úbeda, F. Gómez de León, L.M.Tomás, *IEEE Intelligent Systems* , 17(4):85-89, 2002.

[7] The Saphira architecture for autonomous mobile robots. Konolige, Kurt and Myers, Karen L. En *Artificial Intelligence and Mobile Robots: case studies of successful robot systems*. MIT Press. Editores David Kortenkamp, R. Peter Bonasso y Robin Murphy. pp. 211-242. 1998.

[8] A Framework for Defining and Learning Fuzzy Behaviours for Autonomous Mobile Robots, H. Martínez Barberá, A. Gómez Skarmeta, *International Journal of Intelligent Systems* , 17(1):1-20, 2002.

[9] M. Montemerlo, N. Roy and S. Thrun, Perspectives on standarization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) toolkit, *2003 IEEE/RSJ Int. Conf. on Intelligent Robot Systems (IROS 2003)*, Las Vegas (USA), 2003, pp 2436-2441.

[10] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons and T. Estlin, CLARAty and challenges of developing interoperable robotic software, *2003 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS-03)*, vol. 3, 2003, pp 2428–2435.

[11] Mobility Robot Integration software, User's guide. RWI, Real World Interface.

[12] Utz, Hans, Stefan Sablatnög, Stefan Enderle and Gerhard Kraetzschmar. Miro – middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures* **18**(4), 493–497, 2002.

[13] Open-R SDK, Programmer's guide. Sony Corporation. 20030201-E-003.

[14] Apertos92 Yasuhito Yokote. "The Apertos Reflective Operating System: The Concept and its implementation". *Sony Computer Science Laboratory Inc. Japan 1992.*

[15] Jun-ichiro Itoh, Yasuhito Yokote. "Concurrent Object-Oriented Device Driver Programming in Apertos Operating System". *Sony Computer Science Laboratory Inc. Japan 1994*.

[16] Francisco Martín Rico. "Open-R: un enfoque práctico". Reports on Systems and Communications (ISSN 1698-7489). http://gysc.es/tr .