Jdec: arquitectura basada en esquemas para aplicaciones robóticas

José .M. Cañas, Víctor M. Gómez, Pablo Barrera y Vicente Matellán
Laboratorio de Robótica, Universidad Rey Juan Carlos

Resumen

En este artículo presentamos jdec, una implementación de la arquitectura basada en comportamientos Jerarquía Dinámica de Esquemas. Esta plataforma usa esquemas como unidad básica para la programación de aplicaciones con robots. Estos esquemas se combinan en jerarquías dinámicas para desarrollar comportamientos y en jdec se han implementado como hebras. La plataforma se ha probado en distintos robots reales con aplicaciones de navegación visual, de localización y navegación híbrida en interiores. Además, proporciona una capa de abstracción del hardware, ejecución distribuida, e interfaz gráfica para depuración así como una colección de esquemas ya realizados para componer nuevas aplicaciones.

1 INTRODUCCIÓN

Más allá de las capacidades sensoriales y motoras, la inteligencia de un robot se encuentra en su software. Para comportamientos sencillos casi cualquier organización del software es válida. En cambio si queremos que el robot desarrolle comportamientos complejos o integre varias funcionalidades en el mismo sistema entonces tener una buena organización es fundamental y una buena arquitectura software marca las diferencias.

Los programadores de robots tienen que tratar con hardware y software heterogéneos. No hay estándares universalmente admitidos para la programación de aplicaciones para robots. En los últimos años se han creado varias plataformas y *middleware* para ayudar en ese desarrollo. Los fabricantes de robots y las compañías privadas proporcionan sus propios kits de desarrollo. ARIA en el caso de ActivMedia, ERSP por Evolution Robotics, Open-R por Sony y Microsoft Robotic Studio son sólo algunos ejemplos. Muchas universidades y centros de investigación también han creado sus propias plataformas, por ejemplo Player/Stage [7][9], Carmen [14], Marie [8], Miro [18], CLARAty [15], etc.

Cada plataforma encapsula la funcionalidad de diferentes maneras, dando diferentes niveles de abstracción y haciendo más fácil la generación de comportamientos en robots. Las plataformas más modernas proporcionan métodos para reusar código o comportamientos con el fin de incrementar la productividad, imponen restricciones a la hora de organizar el software del robot y dividen la funcionalidad en pequeños bloques o componentes más fáciles de reusar.

La organización de las capacidades del robot para desarrollar comportamientos autónomos ha sido tradicionalmente un objetivo de la investigación en robótica. Los paradigmas reactivo, basado en comportamientos, deliberativo e híbrido son los más relevantes. Las arquitecturas cognitivas proporcionan principios valiosos para guiar la organización del software del robot. Estas bases cognitivas son interesantes como propuesta de una metodología para afrontar la generación de comportamientos para robots y escalar en complejidad.

Además, la programación de robots puede beneficiarse de las técnicas y herramientas más avanzadas de la ingeniería del software (orientación a objetos, diseños de patrones...). Desde esta óptica surgen también ideas para orientar el desarrollo del software robótico, como la modularidad, la reusabilidad, el diseño de pruebas, etc. más allá de los requisitos clásicos como la vivacidad, la multitarea, etc.

En este artículo presentamos la plataforma *jdec* de programación de robots, basada en los principios teóricos de nuestra arquitectura cognitiva JDE [5]. *Jdec* está implementada en lenguaje C y funciona sobre el sistema operativo GNU/Linux. Lleva usándose tres años como infraestructura para el desarrollo de proyectos fin de carrera, para la realización de prácticas dentro de dos asignaturas de robótica en la URJC y para numerosos trabajos de investigación dentro del grupo (*www.robotica-urjc.es*).

La plataforma ha evolucionado en este tiempo y ha alcanzado una reseñable estabilidad, fruto de la experiencia acumulada. Este entorno académico se caracteriza por una numerosa población de usuarios y una alta rotación de programadores, que han ido ayudando a madurar *jdec* y a minimizar la curva de entrada a los nuevos usuarios.

En la segunda sección se describe en detalle la arquitectura cognitiva JDE, y se presentan los conceptos de esquema y jerarquía que subyacen a la

infraestructura software. La tercera sección habla de las características concretas de *jdec*: cómo se hace la abstracción del hardware, la interfaz gráfica, herramientas desarrolladas, etc. En la cuarta sección se describen algunas de las aplicaciones para robots programadas sobre *jdec*. Y por último se resumen las principales conclusiones extraídas en estos años de experiencia con este *middleware* y posibles líneas futuras de trabajo.

2 ARQUITECTURA COGNITIVA JDE

La idea de jerarquía se ha usado abundantemente para abordar la complejidad en la robótica [3][16][19]. Por ejemplo, las arquitecturas cognitivas híbridas la han usado satisfactoriamente en los últimos años para combinar deliberación y reacción. Las arquitecturas basadas en comportamientos son otra aproximación distinta a la idea de jerarquía.

JDE es una arquitectura cognitiva jerárquica, basada en comportamientos e inspirada en la etología. Su nombre es acrónimo de *Jerarquía Dinámica de Esquemas*. El objetivo de esta arquitectura es reducir la complejidad total del sistema con un enfoque *divide y vencerás*, similar al de algunas jerarquías propuestas por la etología para explicar la generación de comportamientos en animales. Esta arquitectura tiene sus raíces en las ideas de Arbib [1] y Arkin [2].

2.1 Esquema como unidad

El principal componente de JDE es el *esquema*, que se usa como unidad del comportamiento. Un *esquema* es una pieza de software con una tarea determinada que se ejecuta de manera independiente. Todo en el sistema son esquemas, y en cualquier instante puede haber varios en ejecución. Cada uno es activado para completar una tarea particular o conseguir algún objetivo concreto. Además, los esquemas:

-Se pueden parar o reanudar.

-Se pueden *modular* mediante algunos parámetros para ajustar su comportamiento.

-Hacen su trabajo mediante iteraciones periódicas, suministrando una salida al final de cada una de ellas.

Hay *esquemas perceptivos* y *esquemas de actuación*. Los perceptivos transforman los datos sensoriales o información simple en estímulos más complejos que pueden ser usados por otros esquemas. Los esquemas de actuación acceden a los datos perceptivos y generan salidas de control que pueden ser comandos para los motores o señales de activación para otros esquemas (a su vez perceptivos o de actuación) y sus parámetros para modularlos.

La percepción y el control son dos componentes del comportamiento que están relacionados pero son diferentes. Ambos son complejos y tienen que ser solucionados en partes separadas del programa. Además, la percepción y el control se pueden fragmentar en unidades más pequeñas (los esquemas) para dividir la complejidad del problema en subproblemas más acotados. Esta fragmentación hace más fácil reutilizar el código y permite ejecutar distribuidamente varios esquemas en diferentes procesadores.

Cada esquema tiene asociado un estado que define el nivel actual de activación del esquema. Un esquema perceptivo puede estar en estado SLEPT o WINNER. Un esquema de actuación, por su parte, puede tener cuatro estados distintos: SLEPT, CHECKING, READY o WINNER. Estos estados están muy relacionados con cómo se resuelve la selección de acción en JDE.

2.2 Esquemas combinados en jerarquía dinámica

Una vez ha sido introducida la unidad básica de comportamiento, hay muchas opciones para ensamblar el sistema completo. Cada esquema JDE tiene su propio objetivo, y realiza una función particular en la que es un especialista. La jerarquía aparece porque los esquemas pueden utilizar la funcionalidad de otros para realizar su tarea por medio de activación y modulación continua de sus hijos. Esta activación puede ser repetida recursivamente, en tantos niveles como se necesite, de modo que los esquemas de bajo nivel son despertados y modulados por los de los niveles más altos. La cadena de activaciones crea una jerarquía específica de esquemas para generar un comportamiento global particular.

La jerarquía que JDE propone no es el enfoque clásico basado en invocación de funciones directas, donde el padre activa un hijo para llevar a cabo una misión y espera el resultado mientras el hijo hace el trabajo. En lugar de considerar la misión del hijo como un paso en el plan secuencial del padre, JDE entiende la jerarquía como una co-activación que solo expresa *predisposición*. En JDE un padre puede activar varios hijos al mismo tiempo, porque esto no implica que todos los hijos tomen el control del robot, sólo se les pone en alerta. Un mecanismo de selección de acción escoge qué hijo toma realmente el control del robot en ese instante. Esta selección se lleva a cabo mediante la competición entre hermanos del mismo nivel en cada una de sus iteraciones.

Todos los esquemas despiertos (CHECKING, READY y WINNER) ejecutan concurrentemente, de manera similar a la distribución en los sistemas basados en comportamientos. Para evitar comportamientos incoherentes y la generación de comandos contradictorios a los actuadores JDE propone una jerarquía de activación de la colección de esquemas. Esta jerarquía proporciona ciertas ventajas como complejidad acotada para la selección

de acción, acoplamiento entre acción y percepción y monitorización distribuida. Todo ello sin perder la reactividad necesaria para trabajar en entornos desconocidos y dinámicos.

Un esquema de actuación manda comandos directamente a los actuadores o despierta un conjunto de nuevos esquemas hijo. Estos hijos ejecutarán concurrentemente y conseguirán en conjunción el objetivo perseguido por el padre, pues él despertó precisamente a esos esquemas y no a otros para ello. La continua competición entre todos los esquemas de actuación hermanos determina cuál de ellos finalmente pasará al estado WINNER o permanecerá en estado de CHECKING o READY. Sólo el ganador, si hay alguno, pasa al estado de WINNER y de este modo se le permite enviar comandos a los actuadores o despertar sus propios esquemas hijo. El padre activa además a los esquemas perceptivos que proporcionan la información necesaria para resolver la competición por el control entre los esquemas de actuación hijo y la información necesaria para ellos para trabajar y tomar decisiones de control.

Puede ocurrir que el mismo esquema sea activado por diferentes padres en diferentes momentos y contextos distintos. También que el mismo esquema pueda ser activado simultáneamente en diferentes niveles de la jerarquía, probablemente usando diferentes modulaciones.

Una vez el padre ha despertado a sus hijos se mantiene ejecutando, chequeando continuamente sus propias precondiciones, monitorizando los efectos de sus actuales hijos, modulándolos apropiadamente y manteniéndolos despiertos, o quizá cambiándolos por otros que puedan actuar mejor en la nueva situación. De modo que la jerarquía es dinámica. Entre los hermanos de un nivel el ganador puede cambiar si las condiciones del entorno o los objetivos del robot han sido modificados. Si el esquema ganador deja de serlo todos los esquemas activos por debajo suyo se desactivarán, y un nuevo árbol se generará debajo del nuevo ganador. Las reconfiguraciones en JDE suelen ser muy rápidas, dado que el arbitraje y las decisiones realizadas por los esquemas se producen periódicamente y a un ritmo bastante rápido.

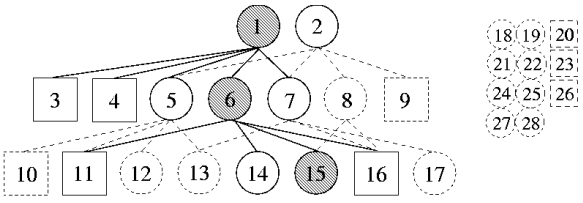

Figura 1: Ejemplo de jerarquía JDE

Las jerarquías son específicas para cada comportamiento. En la figura 1 se muestra el estado actual de una jerarquía de esquemas. Los cuadrados se corresponden a esquemas perceptivos y los círculos a esquemas de actuación, que se sombrean si resultan ganadores en la competición por el control de su nivel. Por ejemplo, el esquema 1 despierta a dos hijos percetivos (3 y 4) y tres hijos de actuación (5,6 y 7), de los cuales, gana el hijo-6 y es el único que tiene descendencia en ese momento. Además ,los esquemas con trazo discontinuo representan esquemas disponibles pero que no han sido despertados.

3 ARQUITECTURA SOFTWARE JDEC

La plataforma *jdec* es una implementación de la arquitectura cognitiva JDE. Esta implementación se ha realizado en lenguaje C y divide el software de la aplicación en dos niveles (Figura 2): la capa de abstracción del hardware y la capa de control. La primera proporciona un interfaz estable que abstrae los sensores y actuadores del robot, ya sea real o simulado. La segunda proporciona el marco de trabajo para que los esquemas de la aplicación interactúen y se comuniquen entre sí.

Además, dispone de algunas librerías para manipulación de celdillas y control borroso para hacer más fácil el desarrollo de comportamientos. Todo el software ha sido desarrollado en C sobre máquinas GNU/Linux y está disponible bajo licencia GPL

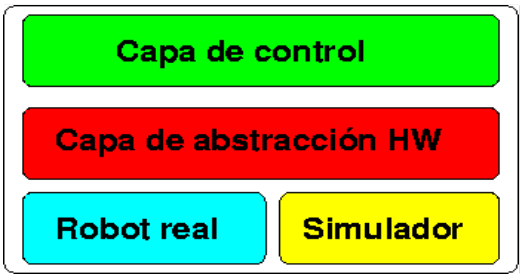

Figura 2: Infraestructura software de JDE

.

3.1 Abstracción del hardware

La capa de abstracción del hardware permite desarrollar, depurar y madurar aplicaciones robóticas sobre simulador y portarlas al robot real sólo cuando funciona bien ahí. Mientras el simulador y el robot real implementen el mismo interfaz abstracto sólo serán necesarias adaptaciones menores para que la aplicación funcione bien con el robot real. Además, este API abstracto facilita portar las aplicaciones completas a diferentes robots.

Todo el hardware es abstraído y presentado como una serie de variables compartidas de lectura/escritura. *Jdec* continuamente recoge los datos de los sensores y los escribe en las *variables sensoriales* para que los esquemas puedan leerlos. También recoge las órdenes a los actuadores que los esquemas escriben en las *variables motoras* y las

envía a los dispositivos a través del "driver" oportuno. Las variables sensoriales siempre tienen el último dato recibido, sin memoria.

Figura 3: Robot de referencia

Actualmente *jdec* incluye soporte para los simuladores Stage y SRISim, y para los robots reales ActivMedia Pioneer y RWI B21, cámaras firewire, cámaras USB, cuellos mecánicos Directed Perception y sensores láser SICK. Dentro de la capa de abstraccion del hardware *jdec* incorpora "drivers" para todos ellos. En particular el robot de referencia es el *Pioneer P3-DX* mostrado en la figura 3.

La interfaz de programación incluye las variables sensoriales: *laser*, *us* con las medidas de los sensores láser y ultrasónicos; *x,y,theta* con los datos de odometría de la base; *pan* y *tilt* con la odometría del cuello mecánico; y *colorA* y *colorB* con las imágenes capturadas por las cámaras. En la parte de actuación incluye las variables motoras *v* y *w* para indicar velocidad lineal y angular al robot, y *latitude* y *longitude* para especificar posición deseada del cuello mecánico. Para mandar órdenes al robot los esquemas de la aplicación sólo tienen que escribir los valores oportunos en esas variables compartidas.

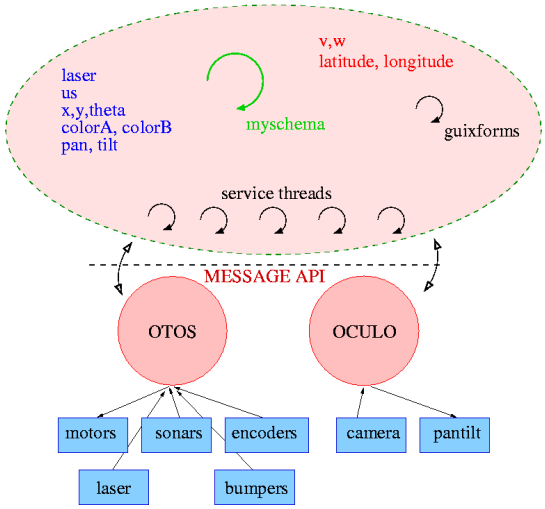

Figura 4: API estándar para aplicaciones con robots en *jdec*

Jdec dispone de un fichero de configuracion de texto (*jde.conf*) para especificar la configuración particular del robot real o simulado para cada aplicación. En ese fichero se especifica en qué dispositivo local se encuentra cada sensor o actuador del robot, o en qué máquinas se encuentran los servidores de red que veremos a continuación, etc.

3.2 Servidores por red

Otra característica ventajosa de *jdec* a la hora de programar robots es la posibilidad de que la capa de control y el robot real estén en máquinas diferentes sin necesidad de modificar el código. Se han desarrollado dos servidores de red para obtener en remoto la información necesaria por la capa de control. Estos servidores se llaman *otos* y *oculo*. Se ejecutan en la máquina que tiene localmente los sensores o actuadores y en otra máquina se lanza *jdec* especificando en su fichero de configuración en qué máquina de la red están los servidores remotos de sensores y actuadores.

En la figura 4 se muestra una fotografía de todo el sistema con la aplicación ejecutando en una máquina y accediendo a los dispositivos del robot a través de los servidores de red *otos* y *oculo*. *Otos* interactúa directamente con los sensores y motores de la base Pioneer o con el simulador. El laser, los ultrasonidos o la odometría y los motores de la base son proporcionados por este servidor. *Oculo* controla una o varias cámaras montadas sobre el robot o alrededor de este y el cuello mecánico, tanto su odometría como el control de sus motores.

Estos servidores ofrecen a los clientes una interfaz remota basada en intercambio de mensajes para leer valores de los sensores o enviar comandos de control a los motores independientemente. Este intercambio sigue un protocolo sencillo en el que los clientes se pueden suscribir a sensores (ultrasonidos, laser, odometria), pueden enviar comandos (a los motores) o pueden solicitar el envío bajo demanda de imágenes. En el caso de sensores "largos" como las cámaras no se admite la suscripción directa, pues requeriría un enorme ancho de banda, sino que han de pedirse y enviarse de una en una.

Jdec también incluye otra herramienta interesante, el servidor de red *reproductor*. Es posible registrar en un fichero de log toda la información proporcionada por los sensores durante una ejecución y usarlo después con el servidor *reproductor* tantas veces como se quiera. Esto resulta útil para programar sin robot real y sin simulador, y para comparar distintos algoritmos de procesamiento sometiéndolos a la misma entrada sensorial exactamente, que ha quedado capturada en el fichero de log. Este servidor sigue el mismo protocolo que los anteriores, por lo cual se puede usar del mismo modo.

3.3 Esquemas de la capa de control

La aplicación robótica en *jdec* se articula como un conjunto de esquemas que se compilan por separado y se enlazan para formar un único ejecutable. La compilación separada supone un cierto grado de modularidad. Cada esquema se implementa como una hebra independiente, que ejecuta a un ritmo controlado. En particular, se ha empleado la biblioteca *pthreads*. En un momento cualquiera la aplicación consta de varias hebras ejecutando concurrentemente e internamente organizadas en jerarquía.

La plataforma incluye un esquema de ejemplo (*myschema*) que sirve de muestra para el desarrollador de aplicaciones. Ese ejemplo incluye el esqueleto software de cada esquema que garantiza una ejecución en iteraciones periódicas a ritmo controlado. El ejemplo contiene un parámetro, el tiempo de ciclo, que permite ajustar el ritmo de las iteraciones. Respetando este esqueleto los esquemas de la aplicación no saturan el uso de procesador.

Para la comunicación entre esquemas se usan variables compartidas, de manera similar a las variables sensoriales y motoras ya mencionadas. Como los esquemas están implementados mediante hebras concurrentes se usan *mutex* para evitar las condiciones de carrera al acceder en paralelo a variables compartidas.

Cada esquema define las variables de salida que ofrece a los otros esquemas, por ejemplo aquellas en las que los esquemas perceptivos dejan sus resultados. También define variables de entrada, por ejemplo aquellas en las que los esquemas recogen su propia modulación. Normalmente estas variables se declaran en el fichero cabecera del esquema. Como se enlazan juntos, hay un espacio de nombres único para todas esas variables. Adicionalmente, cada esquema necesitará ciertas variables de otros esquemas de los que depende para realizar su labor. Por ejemplo, las de salida de los esquemas perceptivos que necesita para tomar sus decisiones o las de entrada de modulación de los esquemas hijo con los que materializa su actuación.

3.4 Interfaz gráfica

Jdec incluye una interfaz gráfica que ayuda a la visualización y la depuración de la aplicación robótica. La interfaz ha sido programada usando la biblioteca *Xforms* sobre el sistema de ventanas *Xwindow*.

La interfaz monitoriza toda la información sensorial proporcionada por la capa de abstracción hardware a la capa de control. En la figura 5 los datos de los ultrasonidos están representados con rayos verdes, las medidas láser como puntos azules y la odometría con un dibujo del robot sobre cierta localización. Las imágenes tomadas por las dos cámaras del robot también se visualizan en la interfaz. Además, la interfaz gráfica proporciona una manera sencilla de interactuar con los actuadores del robot. El usuario puede teleoperar el robot o mover el cuello mecánico a voluntad usando sendos objetos gráficos.

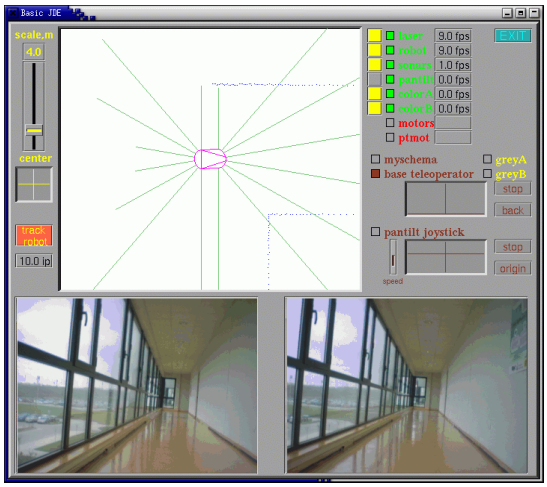

Figura 5: GUI para depuración de aplicaciones de *jdec*

Los desarrolladores de aplicaciones pueden modificar esta interfaz gráfica añadiendo controles o *widgets* para visualizar o monitorizar el funcionamiento de los nuevos esquemas. Esto es posible porque todo el código fuente de *jdec* está disponible para el programador, siendo sencillo adaptarlo para el nuevo propósito. Por ejemplo, para el comportamiento *ir-a-punto*, el GUI fue aumentado para especificar el punto de destino con un clic del ratón. Para otros comportamientos el GUI fue aumentado para ver el mapa del mundo, y para activar/desactivar algún esquema del conjunto actual.

4 EXPERIMENTOS

Durante los tres años de uso de la plataforma *jdec* se han desarrollado muchos comportamientos sobre ella (http://www.robotica-urjc.es). Algunos dentro de las prácticas de la asignatura de robótica, otros al hilo de proyectos de fin de carrera y otros más complejos como trabajos de investigación. Actualmente existe una colección de esquemas disponibles para reutilizar en nuevos desarrollos. Resaltamos algunos significativos:

4.1 Navegación híbrida

Un comportamiento interesante implementado sobre *jdec* es la navegación híbrida basada en planificación de ruta por gradiente (GPP) y en las fuerzas virtuales. El comportamiento se ha conseguido con tres esquemas [6], según muestra la figura 6. El esquema perceptivo GRADIENT-BUILDER construye un mapa siguiendo el algoritmo de planificación del

gradiente y extrae la trayectoria más idónea. Cuando no hay obstáculos alrededor el esquema de actuación FOLLOW-GRID mueve el robot siguiendo la trayectoria idónea. Si hay obstáculos cercanos el esquema de actuación VFF se activa para evitarlos manteniendo al robot lo más cerca posible de la trayectoria.

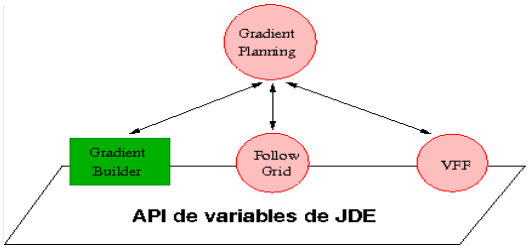

Figura 6: Esquemas para navegación global segura

4.2 Localización

Una característica imprescindible para la planificación deliberativa en robots móviles es saber dónde está el robot en cada momento. Para ello se han desarrollado dos esquemas perceptivos sobre *jdec*: VISUALLOC para localización basada en visión [12] y LASERLOC para localización basada en el sensor láser [10]. Ambos utilizan filtros de partículas para no consumir muchos recursos computacionales.

4.3 Navegación visual

La figura 7 muestra la arquitectura de esquemas para el comportamiento sigue-persona con visión monocular. El esquema perceptivo FILTER identifica la persona en la imagen en color obtenida por la cámara. El esquema de actuación VISUAL TRACKING controla los motores del cuello mecánico para mantener la persona centrada en las imágenes, y el SEARCH hace un barrido con el cuello mecánico si la persona desaparece de la imagen. Cuando no hay obstáculos alrededor el esquema BASE TRACKING alinea al robot con la orientación actual del cuello mecánico. En caso de que haya obstáculos y en función de lo cerca que se encuentren, los esquemas AVOID-OBSTACLES, VFF o STOP se activarán. Se pueden encontrar más detalles de esta aplicación en [4].

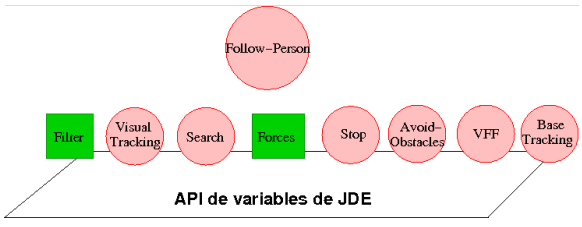

Figura 7: Esquemas para la aplicación de sigue persona

4.4 Procesamiento y atención visuales

Se han desarrollado varios esquemas que extraen de las cámaras información relevante para la navegación. Por ejemplo, el esquema perceptivo FRONTERA estima visualmente la distancia a los obstáculos en el suelo. El esquema perceptivo PASILLO analiza esos obstáculos para extraer las estructuras geométricas con forma de pasillo. También se ha desarrollado una aplicación para que el robot identifique distintos objetos (paredes, personas, puertas y otros robots) a la vez que navega [17]. Se ha programado un esquema perceptivo para la identificación de cada objeto: ID_PARED, ID_PERSONA, ID_PUERTA e ID_CONGENERE Mientras que para navegar se recurre al esquema de actuación VFF que fue desarrollado anteriormente.

Para ampliar la información más allá del campo visual instantáneo de las cámaras se han desarrollado varios trabajos de atención y memoria visuales. El trabajo [13] implementa un algoritmo capaz de mantener actualizada una representación de los objetos relevantes de la escena a través de visión monocular activa. La aplicación tiene un esquema perceptivo llamado SCENE que se encarga de procesar las imágenes capturadas por la cámara para buscar y seguir los objetos relevantes de la escena (de color rosa o verde). Este esquema supone un ejemplo de percepción activa, puesto que se encarga de comandar movimiento a un cuello mecánico en el que va montada la cámara para seguir a los objetos de interés.

El proyecto [11] se sirve de la visión estéreo para que el robot sea capaz de navegar siguiendo los objetos relevantes de la escena en tres dimensiones. Este proyecto tiene varios esquemas perceptivos: HSIFILTER extrae los colores relevantes de la imagen, SEGMENTATION agrupa esos píxles interesantes en objetos, PROJECT3D reconstruye parte de la escena en tres dimensiones a partir de las imágenes segmentadas y ATTENTION que se encarga de mantener una representación de la escena global alrededor del robot. También incorpora dos esquemas de actuación: PAN_TILT comanda el movimiento al cuello mecánico para no perder los objetos de interés, y VFF implementa la navegación con evitación de obstáculos.

5 CONCLUSIONES

En este artículo hemos presentado la plataforma software *jdec* para el desarrollo de aplicaciones robóticas. Esta plataforma es una implementación de la arquitectura cognitiva JDE, que propone el esquema como unidad de construcción de comportamientos y los combina en jerarquías para escalar en complejidad.

La plataforma define un interfaz abstracto de acceso al robot basado en variables e incorpora heramientas como el interfaz gráfico de usuario y los servidores remotos de sensores y actuadores. También incluye un conjunto de esquemas ya desarrollados disponibles que pueden reusarse en nuevas aplicaciones.

Jdec ha sido probada correctamente con comportamientos reactivos, secuencias flexibles de comportamientos y jerarquías simples que no requieren una arquitectura complicada. No ha sido probada con escenarios más complejos, en parte porque hemos detectado algunas limitaciones de *jdec* a la hora de reusar e integrar código. Por ejemplo, la reutilización nunca es automática, siempre requiere de cierta adaptación y puede haber colisión de nombres entre esquemas.

Desde el punto de vista del desarrollador, *jdec* aporta beneficios como la abstracción del hardware y poder reciclar en cierta medida componentes ya desarrollados. Resulta muy ventajosa la capacidad de depurar la aplicación en el simulador y portarla al robot real simplemente cambiando la configuración, sin modificar el código fuente.

Actualmente estamos trabajando en la carga de los esquemas de manera dinámica, como *plugins*. También estamos explorando la posibilidad de una visualización realmente distribuida en la que cada esquema lleve su propio interfaz gráfico y no requiera la modificación de la interfaz gráfica común.

Agradecimientos

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] M. Arbib and J.S. Liaw, *Sensorimotor Transformations en the Worlds of Frogs and Robots*. Artificial Intelligence, 72 (1995) 5379

[2] Ronald C. Arkin, Masahiro Fujita, Tsuyoshi Takagi and Rita Hasegawa, *An Ethological and Emotional Basis Human-robot Interaction.* Robotics and Autonomous Systems, vol 49, pags 191-2001, 2003

[3] S. Behnke and R. Rojas, *A hierarchy of reactive behaviors handles complexity.* Balancing Reactivity and Social Deliberation en Multi-Agent Systems, LNCS 2103 Springer, 2001, pp 125-136.

[4] R.Calvo, J.M.Canas, L.García-Perez, *Person following behavior generated with JDE schema hierarchy*. Actas de ICINCO 2nd Int. Conf. on Informatics en Control, Automation and Robotics. Barcelona, 2005.

[5] J.M Cañas. *Jerarquía Dinámica de Esquemas para la generación de un comportamiento.* Tesis Doctoral, Universidad Politécnica de Madrid, Diciembre de 2003.

[6] J.M.Cañas, J.Raul Isado and Lía García, *Robot navigation combining the Gradient Method and VFF inside JDE architecture.* Actas del VI Workshop de Agentes Físicos, Congreso Nacional de Informática CEDI-2005, Granada 2005.

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

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

[9] B. Gerkey, R. Vaughan and A. Howard, *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems.* Int. Conf. on Advanced Robotics (ICAR 2003), Coimbra (Portugal), 2003, pp 317-323.

[10] Redouane Kachach, *Localizacion del robot Pioneer basada en laser.* Proyecto Fin de Carrera ITIS, URJC, 2005.

[11] Olmo Leon, *Navegacion de un robot con un sistema de atencion visual 3D.* Proyecto Fin de Carrera Ing. Informática, URJC, 2006.

[12] Alberto López, *Localización de un robot con visión.* Proyecto Fin de Carrera ITIS, URJC, 2005.

[13] Marta Martínez, *Sistema de atención visual en escena.* Proyecto Fin de Carrera Ing. Informática, URJC, 2005.

[14] 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.

[15] 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.

[16] M. Nicolescu and M. Mataric, *A hierarchical architecture for behavior-based robots.* Int. Joint

Conf. on Autonomous Agents and Multiagent systems, Bologna (Italy), 2002, pp 227-233.

[17] Ricardo Palacios, *Representación rica de la escena 3D alrededor de un robot móvil.* Proyecto Fin de Carrera ITIS, URJC, 2006.

[18] H. Utz, S. Sablatnög, S. Enderle and G. Kraetzschmar, *Miro-Middleware for mobile robot applications.* IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures, vol. 18, no. 4, 2002, pp 493-497.

[19] H. Utz, G. Kraetzschmar, G. Mayer and G. Palm, *Hierarchical behavior organization.* 2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS-05), Edmonton (Canada), 2005.