Registración de Componentes para Programación de Robots Móviles Autónomos

Erika Michalczewsky **Pablo R. Fillottrani**

Departamento de Ciencias de Computación
Universidad Nacional del Sur
Av. Alem 1253 – (8000) Bahía Blanca
Argentina

e-mail: {emichal,prf}@cs.uns.edu.ar

Resumen

Se presenta el diseño de un ambiente de programación para robots móviles autónomos utilizando el concepto de registración de componentes. Mediante este concepto se dispone de la facilidad de incorporar nuevos algoritmos de planificación, navegación y/o nuevas representaciones para su utilización en el control del robot y además, su comparación. El objetivo final es presentar un ambiente lo suficientemente flexible que permita la implementación y evaluación práctica de distintos algoritmos y representaciones en la programación de robots móviles autónomos.

Palabras claves: robótica, ingeniería de software, robóts móviles autónomos, ambientes de programación.

1 Introducción

Los ambientes de programación para robots móviles autónomos son una herramienta que facilitan la tarea de controlar un robot. El objetivo de estos sistemas es ofrecer elementos para la descripción de las actividades del robot sin tener la necesidad de escribir código complejo. Muchos de éstos disponen de una línea de comandos desde donde es posible ejecutar acciones sobre el robot, sin embargo, este tipo de acciones esta restingido a un conjunto de movimientos primitivos. Por esta razón resulta muy interesante disponer de aplicaciones donde sea posible la programación de robots utilizando comandos de alto nivel para detallar las tareas. Por otra parte, los ambientes de programación que se presentan pueden ser utilizados para un sistema de control con algoritmos y representaciones particulares. El ambiente de programación que proponemos permite registrar diferentes algoritmos y representaciones con el objetivo de poder comparar y evaluar su comportamiento.

En el sistema descripto en [Michalczewsky y Fillottrani 99], en el cual se incorporaron nuevas herramientas al ambiente de programación Saphira [Konolige 98], se simplificó el trabajo del programador para definir las tareas del robot, disponiendo de actividades predefinidas y se simplificó la especificación de los mapas utilizando un editor gráfico. En este trabajo proponemos ampliar el ambiente de programación, utilizando la idea anterior de disponer de componentes predefinidas: además de tener la posibilidad de seleccionar actividades, es posible seleccionar algoritmos y representaciones de datos.

La descripción del trabajo está organizado de la siguiente manera. En la sección siguiente se expone una reseña sobre las arquitecturas más destacadas en la historia de los sistemas para robots móviles autónomos, poniendo especial énfasis en las arquitecturas híbridas. En la sección 3 se describe brevemente la arquitectura utilizada como referencia para el diseño del ambiente de programación. En la sección 4 se presenta el diseño del ambiente y por último se detallan los trabajos que se están realizando.

2 Arquitecturas de Robots Móviles Autónomos

De acuerdo a lo observado en los últimos años, los sistemas para robots móviles autónomos han evolucionado hacia un paradigma en el cual se combinan las propuestas deliberativa y reactiva.

El primer sistema para robots móviles fue implementado para el robot Shakey [Nilsson 69] con un estilo de Sensar-Planificar-Actuar (SPA). El sistema consistía en tres elementos: un subsistema de sensado, un subsistema de planificación y un subsistema de ejecución [Nilsson 80]. La tarea del subsistema de sensado era traducir los datos crudos de los sensores (generalmente datos de sonares o visión) en un modelo del mundo. La tarea del planificador era tomar el modelo del mundo y una meta, y generar

un plan para conseguirla. La tarea del subsistema de ejecución era tomar el plan y generar las acciones para el robot en relación con el plan establecido. Dos características importantes de estos sistemas eran que el flujo de control entre las componentes era unidireccional y lineal y que la ejecución del plan era análoga a la ejecución de un programa.

Una nueva propuesta presentaba un modelo jerárquico que se denominó NASREM [Albus, McCain y Lumia 87]. Esta arquitectura constaba de niveles, cada uno con las siguientes componentes: procesamiento de sensores, modelo del mundo, descomposición de tareas y juicio de valor. Todos los niveles se encontraban unidos por una memoria global a través de la cual compartían la representación del conocimiento. Cada nivel tenía la estructura de Sensar-Planificar-Actuar, pero cada uno operaba en distintas escalas de tiempo y espacio, brindando así un equilibrio entre la asimilación de datos y la respuesta. El nivel más alto de esta arquitectura jerárquica tenía una tarea muy similar a la de los primeros sistemas, sin embargo, el lugar de tener una estructura monolítica que mapeara los planes de alto nivel directamente en comandos de nivel motor, el nivel superior pasaba sus planes a los niveles intermedios que se traducían luego en comandos de nivel inferior, que eran a su vez enviados a niveles más bajos. Aún cuando esta propuesta tiene algunos aspectos de planificación deliberativa y un control reactivo, la naturaleza *top-down* de la estructura jerárquica tiende a restringir los niveles inferiores, limitando la respuesta del sistema a condiciones externas inesperadas. Si cada nivel asume que su comando será ejecutado por los niveles inferiores, debe monitorearse y tratar constantemente con la violación de estas suposiciones.

La propuesta que revolucionó el área fue presentada por Brooks [Brooks 86], en la cual mostraba un modelo puramente reactivo basado en comportamientos. Desarrolló un lenguaje de superposición que permitía modelar un sistema análogo al comportamiento de los animales en ciclos cerrados de sensado y acción, utilizando máquinas de estado finito asincrónicas. El primer conjunto de comportamientos de un robot evitaba cualquier objeto estuviera demasiado cerca, alejándose un poco o deteniendo la marcha. Otro nivel más alto de comportamientos movía al robot en una determinada dirección. Este comportamiento dominaba al comportamiento de evitación de obstáculos suprimiendo la salida de los actuadores para el último, a menos que un objeto se encontrara demasiado cerca. Los niveles más altos se sobreponían a los niveles inferiores, de ahí el nombre de la arquitectura *Subsumption*. Estos sistemas son menos susceptibles a las fallas totales y permiten la creación incremental de los sistemas. Por otro lado, sacrifican la capacidad de razonar sobre las intenciones del sistema y sus metas.

Para lograr una simbiosis de los elementos deliberativos y reactivos, se propusieron arquitecturas híbridas ([Arkin 89] [Gat 92] [Simmons] [Alami et al.] son algunas de ellas) que incorporaban un sistema basado en comportamientos como nivel inferior de una estructura jerárquica, y un sistema deliberativo en el nivel superior. Con la capacidad de sintetizar las propuestas centralizadas y distribuidas, el paradigma híbrido se ha convertido en el método para diseñar arquitecturas para robots.

A partir de una arquitectura de control híbrido donde sus componentes son suficientemente independientes es que se propone el diseño de un ambiente de programación que permita que estas componentes puedan ser intercambiables y que permita, además, la evaluación del comportamiento de diferentes algoritmos y estructuras.

3 Arquitecturas de Control Híbrido

Este tipo de arquitectura fue inicialmente propuesto por tres grupos de investigación casi al mismo tiempo. Connell presentó su arquitectura con el nombre de *SSS* [Connell 92], la arquitectura de Gat se llamó *ATLANTIS* [Gat 92] y la arquitectura de Bonasso se denominó *3T* [Bonasso 91]. Estas arquitecturas de control híbrido constan de tres componentes, como se muestra en la figura 1, las cuales se describen a continuación utilizando la terminología de ATLANTIS, pero destacando sus características en los tres sistemas.

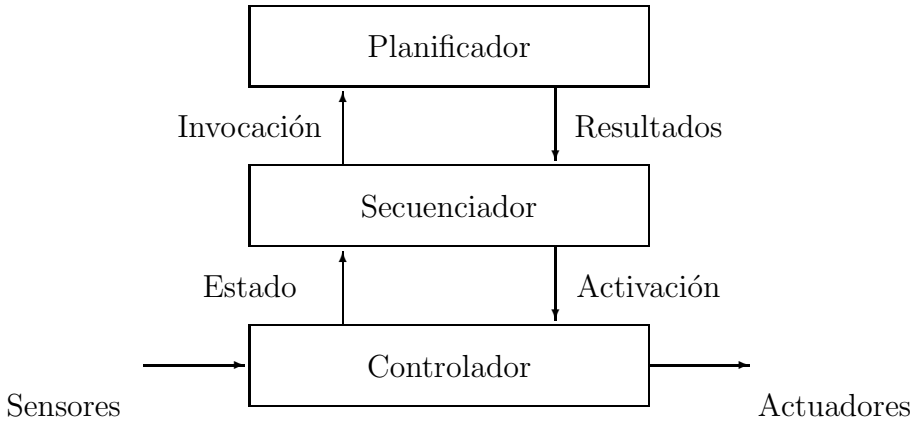

Figura 1: Arquitectura de Tres Niveles

El controlador

El controlador consiste en uno o más hilos de ejecución que implementan varios ciclos de control realimentado, acoplando fuertemente los sensores a los actuadores. La función de transferencia que calcula el controlador puede ser modificada en tiempo de

ejecución. Generalmente, el controlador contiene una librería de funciones de transferencia (llamadas *comportamientos* o *habilidades primitivas*). Los comportamientos que se encuentran activos en cada momento son determinados por una entrada externa al controlador.

Para distinguir entre el código que implementa una función de transferencia y el comportamiento físico producido por esa función de transferencia en la ejecución sobre el robot en el entorno, los comportamientos se presentan con otro tipo de letra. De esta manera un comportamiento es una porción de código que genera un comportamiento cuando se ejecuta. Los comportamientos primitivos son diseñados para producir movimientos primitivos simples que puedan ser compuestos para producir comportamientos más complejos (tarea del secuenciador). Algunos ejemplos clásicos de comportamientos primitivos son el seguimiento de una pared, moverse a un destino evitando colisiones y atravesar puertas.

Existen varias restricciones arquitectónicas importantes sobre los algoritmos que van en el controlador. Primero, computar una iteración de la función de transferencia debe estar acotado en tiempo y espacio por una constante, y esta constante debe ser lo suficientemente pequeña como para brindar una ancho de banda que soporte un control de ciclo cerrado estable para el comportamiento deseado.

Segundo, los algoritmos en el controlador deberían soportar *reconocimiento de fallas*, es decir, deberían estar diseñados para detectar cualquier falla que ocurriera al ejecutar la función para la cual fueron diseñados. En lugar de diseñar algoritmos que nunca fallen, se pueden diseñar algoritmos que nunca fallen en detectar una falla. Esto permite que otras componentes del sistema (el secuenciador y el planificador) tomen acciones correctivas para recuperarse de la falla.

Tercero, el uso de un estado interno debería evitarse siempre que sea posible. Una excepción importante son los algoritmos de filtrado, que se fundamentan en un estado interno, pero pueden de todos modos ser utilizados en el controlador. Si el estado interno es utilizado para otros propósitos, estos deben ser efímeros; es decir, deberían caducar después de un tiempo constante acotado. De esta forma, si la semántica del estado interno no refleja el estado real del entorno, por lo menos el tiempo durante el cual se manifieste este error estará acotado.

Finalmente, el estado interno en el controlador no debería introducir discontinuidades (en el sentido matemático) en un comportamiento. En otras palabras, un comportamiento (que es una función de transferencia) debería ser una función continua con respecto a su estado interno. Es responsabilidad del secuenciador el manejo de las transiciones entre los regímenes de la operación continua.

El secuenciador

La tarea del secuenciador es seleccionar cuál comportamiento primitivo (es decir, cuál función de transferencia) debe utilizar el controlador en cada momento, y proporcionar

los parámetros para el comportamiento. Modificando los comportamientos primitivos en los momentos estratégicos, el robot puede ejecutar tareas útiles. El problema es que el resultado de seleccionar una primitiva determinada en una situación particular podría no ser realmente el resultado deseado, y por lo tanto una secuencia simple de primitivas no es confiable. El secuenciador debe ser capaz de responder condicionalmente a la situación actual, cualquiera sea esta.

Una propuesta al problema es enumerar todos los posibles estados del robot en los cuales puede estar y precalcular la primitiva correcta a utilizar en cada estado para una tarea particular. Una codificación inteligente podría hacer esta tarea tratable para ciertos dominios acotados. Sin embargo, esta propuesta de *plan universal* tiene dos grandes desventajas. Primero, generalmente no es posible para un robot conocer su estado actual, especialmente cuando aparecen contingencias imprevistas. Segundo, la propuesta no tiene en cuenta la historia de ejecución del robot, que frecuentemente contiene información útil.

Una alternativa es utilizar una propuesta llamada *secuenciamiento condicional*, que es un modelo más complejo de ejecución de planes motivado por el seguimiento de instrucciones de las personas. Las personas pueden hacer tareas basándose en instrucciones concisas y enfrentándose a una gran variedad de eventualidades. El secuenciamiento condicional propone una estructura computacional para codificar el tipo de conocimiento procedural que contienen las instrucciones. Difiere de la ejecución de planes tradicionales en que los constructores de control para componer las primitivas no están limitados a un orden parcial, condicionales y ciclos utilizados para construir los planes SPA. Los sistemas de secuenciamiento condicional incluyen constructores para responder a distintas circunstancias, y manejar múltiples tareas interactuando en paralelo.

Existen dos propuestas principales para el diseño de lenguajes de secuenciamiento condicional. Pueden ser lenguajes completos con una semántica de ejecución especializada, como por ejemplo RAPs (Reactive Action Packages) [Firby 89] y PRS (Procedural Reasoning System) [Georgeff, Lansky y Schoppers 87]. O también pueden ser colocados por encima de un lenguaje de programación sintácticamente extendible como LISP; así lo hacen Behavior Language y ESL. Además, la estructura del lenguaje puede tratar todos los posibles resultados de una acción de una manera uniforme, o el lenguaje puede ser estructurado para reconocer un resultado "nominal" privilegiado de una acción y tratar el resto de los resultados como "fallas". Nuevamente, RAPs y PRS implementan la primera propuesta y ESL implementa la segunda.

La propuesta a elegir depende del objetivo con el cuál se construye el sistema. RAPs y PRS son lenguajes adecuados para utilizarlos como representación para un secuenciador automático. La propuesta de ESL es más conveniente de utilizar y es más fácil de extender, pero es más difícil de analizar.

El Planificador

El planificador es el lugar donde se ejecutan los cómputos que consumen tiempo. Generalmente, esto significa planificación y otros algoritmos de búsqueda exponenciales, pero también podría incluir algoritmos de tiempo polinomial con grandes constantes como ciertos algoritmos de procesamiento de visión frente a recursos computacionales limitados. La clave del planificador es que pueden ocurrir varias transiciones de comportamientos entre el momento en que el algoritmo de planificación es invocado y el momento en el cual se produce el resultado. El planificador se ejecuta como uno o más hilos de control separados. No existen restricciones arquitectónicas en los algoritmos del planificador, que invariablemente se escriben utilizando lenguajes de programación estándar. El planificador puede hacer de interface con el resto del sistema de dos formas diferentes. Puede producir planes para que ejecute el secuenciador o puede responder a consultas específicas del secuenciador.

4 Diseño del ambiente

La propuesta que se presenta es un ambiente de programación para robots móviles autónomos que puede utilizarse para la evaluación de diferentes algoritmos y representaciones para estos sistemas. La arquitectura para sistemas de robots que se seleccionó es muy apropiada para lograr este objetivo. Las tres componentes son lo suficientemente independientes como para lograr la implementación aislada de los comportamientos individuales, los algoritmos de planificación y navegación, y las representaciones de los datos.

El ambiente de programación consta de un sistema de control principal, una interface de usuario, un repositorio de datos y los protocolos de comunicación entre componentes, como se muestra en el esquema de la figura 2. La interface de usuario es básicamente un sistema de ventanas con menúes que tiene como una de las funciones más importantes la selección y registración de nuevas componentes, ya sean algoritmos o nuevas estructuras de representación para los datos. Esta interface también dispone de un editor gráfico para trazar los mapas de los entornos, y un browser para visualizar el recorrido del robot. Además, desde el menú es posible establecer la conexión con el robot, seleccionar un mapa del entorno, o determinar los comportamientos, representaciones y algoritmos a utilizar.

En el repositorio de datos se almacenan las componentes que son registradas en el sistema. Cada componente es identificada a través de un ID y debe ofrecer un conjunto de servicios que es determinado por el ambiente de programación, esto último permitirá que cualquier algoritmo o estructura que se registre pueda ser activado sin dificultades. Las componentes pertenecerán a grupos diferentes según su propósito, ya que cada grupo tiene definido un conjunto de servicios particulares que debe brindar.

Estos servicios fueron definidos luego de analizar varias representaciones utilizadas para modelar los entornos y diferentes algoritmos de planificación y navegación. Una vez incorporadas al sistema, las componentes se encuentran disponibles para ser activadas, a través del sistema de control.

El sistema de control tiene como tarea controlar el flujo de información entre las componentes, utilizando los protocolos de comunicación. La función de los protocolos de comunicación consiste en comunicar las componentes con la ayuda del sistema de control. Estos protocolos definen la información que podrán intercambiar las componentes de los distintos grupos.

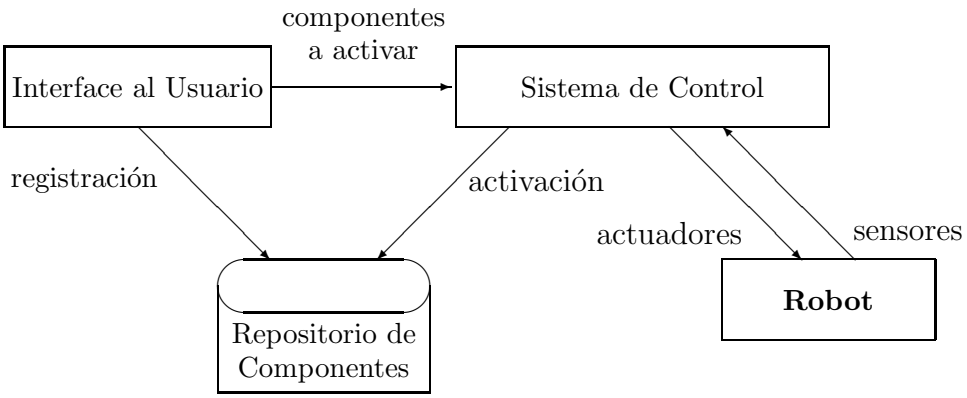

Figura 2: Ambiente de Programación para Robots Móviles Autónomos

Un ejemplo de una situación particular en este ambiente sería primero establecer la tarea que debe realizar el robot. Luego, el sistema comenzaría controlando al robot con un conjunto de algoritmos y representaciones definidas por defecto. Este conjunto podría ser modificado desde el menú, antes de iniciar la comunicación con el robot. Una vez seleccionado el tipo de control sobre el robot se iniciaría la comunicación con el mismo, el cual sería controlado utilizando los algoritmos seleccionados. El sistema podrá determinar si el robot logró su objetivo o si la tarea no pudo ser concretada.

La arquitectura de control del robot que se diseñó, utiliza las ideas de la arquitectura de tres niveles. En la propuesta presentada, el controlador esta conformado por un conjunto de comportamientos elementales como seguir una dirección, pasar por una puerta, etc. El comportamiento activo en un momento particular es determinado por el secuenciador. Sin embargo, es posible elegir el conjunto de comportamientos que se quieren que sean considerados para el control del robot. Todos los comportamientos están fuertemente acoplados a los datos que se reciben de los sensores.

Para el secuenciador se está estudiando la posibilidad de incluir el lenguaje Colbert [Konolige 97], que es un lenguage con las ideas de PRS, de las cuales comparte las máquinas de estado finito y las actividades concurrentes. Es un lenguaje que proporciona constructores condicionales, de iteración y de secuencia, con una semántica

basada en máquinas de estado finito. La elección de este lenguaje no es determinante y podría ser reemplazado por alguna de las otras alternativas.

Para el planificador, se está considerando una propuesta de planificador con una estructura muy sencilla en la cual la dinámica del entorno se muestra como un diagrama de transición donde los estados son situaciones y los arcos representan acciones simples o compuestas [Baral y Gelfond 99]. Además, es posible realizar tareas correctivas, ejecutando acciones y recordando la historia del dominio. Se mantiene la información sobre las situaciones y las acciones sobre el entorno y la definición de los estados posibles que se pueden suceder después que se ejecute una acción. También se definen una colección de caminos que pueden interpretarse como las trayectorias posibles en el entorno. Las tareas pueden formularse en forma de preguntas, y pueden responderse utilizando programas lógicos. El lenguaje que se utiliza para la programación y representación de la información en el planificador es *A-Prolog*.

5 Trabajos Futuros

Se presentó la descripción del diseño de un ambiente de programación que en este momento se está implementando. Esta herramienta ofrecerá un escenario en el cual será posible programar actividades para el robot de una manera sencilla. Hasta el momento se encuentran realizadas la especificación y el diseño del sistema, y se está comenzando con su implementación. Se eligió como lenguaje de programación para el ambiente, J++, considerando sus facilidades para la generación de interfaces gráficas y para la comunicación de bajo nivel con el robot. Se está trabajando en una registración de componentes basada en DLL, la cual no es totalmente dinámica como es el propósito inicial. Sin embargo, se está evaluando la alternativa dinámica utilizando procesos independientes para cada algoritmo. Una vez implementado este nuevo ambiente, se prevee utilizarlo como herramienta didáctica en varios cursos relacionados. Como ambiente de programación dispone de la facilidad para incorporarse como instrumento de estudio para el aprendizaje de técnicas de planificación y control en sistemas de robots móviles autónomos.

Referencias

[Alami et al.] R. Alami and R. Chatila and S. Fleury and M. Ghallab and F. Ingrand. *An Architecture for Autonomy*, en *International Journal of Robotics Research*, 1998. Special issue on integrated architectures for robot control and programming.

[Albus, McCain y Lumia 87] James Albus, H. McCain y R. Lumia. *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*, Technical Note 1235, 1987.

[Arkin 89] Ronald C. Arkin. *Towards the Unification of Navigational Planning and Reactive Control*, notas del trabajo, en *AAAI Spring Symposium on Robot Navigation*, 1989.

[Arkin 98] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[Baral y Gelfond 99] Chitta Baral y Michael Gelfond. *Reasoning Agents in Dynamic Domains*, en http://earth.cs.ttu.edu/'mgelfond/papers/Ibai.ps, 1999.

[Bonasso 91] Peter R. Bonasso. *Integrating Reaction Plans and Layered Competences Through Synchronous Control*, en Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, páginas 1225–1231, 1991.

[Brooks 86] Rodney A. Brooks. *A Robust Layered Control System For A Mobile Robot*, en *IEEE Journal of Robotics and Automation*, 1986.

[Connell 92] Jonathan Connell. *SSS: A Hybrid Architecture Applied to Robot Navigation*, en *Proceedings of the IEEE International Conference on Robotics and Automation*, páginas 2719–2724, 1992.

[Firby 89] James R. Firby. *Adaptive Execution in Complex Dynamic Worlds*, Reporte Técnico Nro. YALEU/CSD/RR 672, del Computer Science Department, Yale University, 1989.

[Gat 92] Erann Gat. *Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots*, en *Proceedings of the Eleventh AAAI*, 1992.

[Georgeff, Lansky y Schoppers 87] M. Georgeff, A. Lansky y M. Schoppers. *Reactive Reasoning and Planning*, en Proceedings of the AAAI-87, páginas 677–682, 1987.

[Konolige 97] Kurt Konolige. *COLBERT: A Language for Reactive Control in Saphira*, 1997.

[Konolige 98] Kurt Konolige. *Saphira Software Manual, version 6.1*, 1998.

[Michalczewsky y Fillottrani 99] Erika Michalczewsky y Pablo Fillottrani. *Un Ambiente para Programación y Control de Robots Móviles Autónomos*, en *Proceedings CACIC'99 Congreso Argentino de Ciencias de la Computación*, 1999.

[Nilsson 69] Nils J. Nilsson. *A Mobile Automaton: An Application of AI Techniques*, en *Proceedings of the First International Joint Conference on Artificial Intelligence*, 1969.

[Nilsson 80] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga Press, 1980.

[Simmons] R.G. Simmons. *Structured Control for Autonomous Robots*, en *IEEE Transactions on Robotics and Automation*, 1994.