Soporte a la Distribución para un Framework Basado en Componentes

Francisco Sánchez Ledesma, Juan Ángel Pastor Franco, Diego Alonso Cáceres,
Bárbara Álvarez Torres, Pedro Sánchez Palma
Grupo de Investigación "División de Sistemas e Ingeniería Electrónica"
Universidad Politécnica de Cartagena
Campus Muralla del Mar s/n, 30202 Cartagena
E-mail: francisco.sanchez@upct.es

Resumen. *Los sistemas reactivos requieren la integración de los requerimientos comportamentales y estructurales con los temporales para describir la arquitectura de la aplicación. Hemos adoptado el enfoque de Desarrollo de Software Dirigido por Modelos para tratar estos problemas de forma global: desde la definición de la arquitectura de la aplicación hasta la generación de código y modelos de análisis. Fue desarrollado un framework Orientado a Objetos para facilitar la generación de código, así como proporcionar las propiedades requeridas para la aplicación final. Este artículo describe cómo fue agregado el soporte de distribución al framework de una forma regular sin alterar el diseño, permitiendo al usuario integrar la sobrecarga de la comunicación al análisis del tiempo.*

1. Introducción

Los robots de servicios, y entre ellos los vehículos tele-operados, semi-autónomos y autónomos, son sistemas complejos con requisitos de tiempo real de todo tipo. En el trabajo que aquí se presenta se ofrece un enfoque que usa técnicas de desarrollo basado en componentes para reducir la complejidad de las aplicaciones del dominio, sin dejar de lado los requisitos de tiempo real y centrándose en los aspectos de distribución de componentes.

La mejor forma de abordar la complejidad es elevar el nivel de abstracción de los elementos de modelado, construyendo los sistemas a partir de módulos independientes que interaccionan entre sí únicamente a través de sus interfaces. No existe una definición única de lo que es un módulo y de hecho cada paradigma de programación ofrece diferentes respuestas considerando diferentes niveles de abstracción y de granularidad. El enfoque elegido en este trabajo es el Desarrollo de Software Basado en Componentes (CBSD en sus siglas inglesas), en el que un componente software es una unidad de composición con interfaces bien definidas y un contexto de uso explícito. Actualmente [1] los modelos obtenidos aplicando CBSD pueden clasificarse en: (1) modelos en los que los componentes son objetos, por ejemplo CORBA, y (2) modelos en los que los componentes son unidades arquitectónicas, en la línea de la disciplina Arquitecturas Software [2] y los ADLs (Architecture Description Languages).

Los requisitos de tiempo real de los componentes arquitectónicos no son fáciles de expresar, ya que los conceptos que se definen y utilizan en CBSD (componente y puerto principalmente) no son los mismos que modelan las características de tiempo real de dichos componentes (tareas y mecanismos de sincronización y comunicación entre tareas), aunque también hay elementos comunes (eventos, condiciones, etc.). Algunos enfoques resuelven el problema identificando los conceptos de tarea o proceso con el de componente. Sin embargo, dichas soluciones restringen excesivamente el poder de modelado de los componentes e introducen rigideces innecesarias en la planificación del sistema. Hace falta, pues, un enfoque más flexible.

Existen, no obstante, iniciativas que consideran y analizan conjuntamente los aspectos CBSD y de tiempo real, como [4] y [5] en particular para robótica. Nuestro trabajo puede considerarse enmarcado dentro de estas iniciativas.

El resto del artículo se estructura de la siguiente manera. En la sección 2 se explica brevemente el enfoque general, y las líneas de investigación abiertas. La mayor parte están referenciadas y son resultados de trabajos anteriores. En la sección 3 se explica la importancia de incorporar en el framework soporte a la distribución y se justifica el enfoque adoptado para hacerlo. Finalmente, la sección 4 presenta las conclusiones y los trabajos futuros.

2. Enfoque general de la investigación y trabajos previos.

En el marco anteriormente mencionado, se decidió seguir un enfoque de *Desarrollo Software Dirigido por Modelos* (MDSD en sus siglas inglesas), ya que MDSD proporciona el soporte conceptual y tecnológico tanto para el modelado de las aplicaciones como para la generación final de código. En este contexto se definió el lenguaje de componentes V^3CMM [6] como un meta-modelo que proporciona al diseñador tres vistas complementarias y débilmente acopladas: (1) una vista arquitectónica para definir componentes (interfaces, puertos, servicios ofrecidos y requeridos, componentes

compuestos, etc.), (2) una vista para especificar el comportamiento de los mismos utilizando autómatas temporizados, y finalmente (3) una vista algorítmica para expresar la secuencia de acciones que ejecuta un componente en función de su estado.

Fruto de trabajos anteriores [7] se desarrolló e implementó un framework que proporciona las clases base para implementar los componentes del diseño arquitectónico definidos con V^{3}CMM, y una infraestructura para que el usuario elija las características de concurrencia que finalmente quiere para su aplicación: número de tareas, código que ejecuta cada una de ellas, plazos, prioridades, periodos, etc. El framework proporciona una interpretación OO de los conceptos CBSD que permite trasladar los diseños basados en componentes a aplicaciones OO utilizando para ello sus hot-spots. El diseño y la documentación del framework se ha llevado a cabo mediante el uso de patrones de diseño [3]. El diseño trata de cumplir los siguientes requisitos de diseño:

- Control sobre la política de concurrencia (por ejemplo, una actividad por tarea, una única tarea para todas las actividades, o cualquier otra distribución).
- Control sobre los mecanismos de comunicación entre componentes (síncrona o asíncrona). Facilitar la instanciación del mismo a partir del modelo CBSD de una aplicación.
- Control sobre la política de distribución de los componentes en nodos computacionales.

De estos requisitos, el principal es proporcionar al usuario control sobre la concurrencia de la aplicación (tanto en el número de tareas como en sus características temporales). Para ello se eligieron patrones que permiten asignar arbitrariamente las actividades que ejecutan los componentes a un conjunto cualquiera de tareas, tal y como se muestra en la figura 1.

3. Enfoque para abordar la distribución de componentes.

La capacidad de distribución de componentes es, por diversas razones, una característica fundamental del framework, y por tanto un objetivo prioritario.

- La propia naturaleza de los sistemas robóticos, que suelen incluir varios procesadores.
- La capacidad de separar componentes con requisitos de tiempo-real estricto y no estricto, asignándolos a diferentes procesos.
- La necesidad de llevar a cabo los casos prácticos del proyecto EXPLORE, que consideran explícitamente la distribución de los componentes.
- La posibilidad de modificar el despliegue de los componentes en función de los recursos de procesamiento de los diferentes nodos y del contexto.

Con todo ello, se consideró que la mejor opción era desarrollar un middleware con los servicios mínimos necesarios: (1) creación y eliminación dinámica de componentes en nodos, (2) arranque y parada de los componentes y de la aplicación en su conjunto, (3) conexión y desconexión dinámica de componentes residentes en diferentes nodos.

Es muy importante que el código que se encarga de realizar la distribución sea un componente más de la aplicación, y por tanto, pueda incorporarse al análisis de forma regular como un componente más.

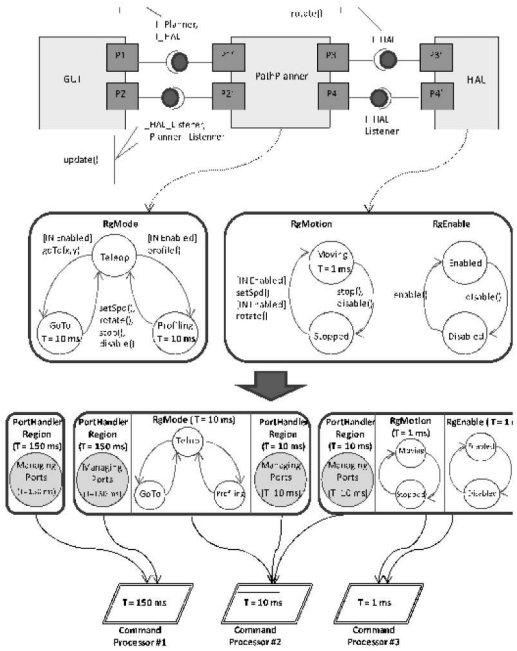

Figura 1: Un ejemplo del uso del framework. De componentes a hilos

Para hacer posible la distribución, el framework incorpora dos elementos: un componente y una clase, que se encargan de realizar el despliegue de la aplicación y de transmitir los mensajes entre los nodos.

La clase se denomina *ApplicationDeployer* y su objetivo es desplegar los componentes de la aplicación, según se especifica en un fichero de configuración. En consecuencia, para poder distribuir una aplicación, es necesario ejecutar previamente una instancia de esta clase en uno de los nodos.

El componente se llama *LocalProxyManager*, y se encarga de la creación y conexión de los componentes dentro de cada nodo y de gestionar la conexión de los componentes internos con los que se encuentran distribuidos en otros nodos. Para poder

llevar a cabo la distribución es necesario ejecutar un componente *LocalProxyManager* en cada nodo. Este componente funciona como un Gateway, que por un lado tiene puertos de componente según define el framework para poder conectarse con componentes específicos de la aplicación, y por otro lado utiliza sockets TCP para conectarse con otros nodos. En la figura 2 se puede ver la estructura de la aplicación distribuida en dos nodos.

La implementación de estos artefactos (clase y componente) no ha requerido modificar la estructura original del framework, sino que simplemente se han añadido nuevos elementos, instanciando las clases base que proporciona el framework.

Para las comunicaciones entre los componentes que se ejecutan en diferentes nodos se utiliza actualmente el protocolo orientado a la conexión TCP/IP, aunque se prevé utilizar buses deterministas como CAN bus. El *ApplicationDeployer* también se encarga de informar a cada uno de los *LocalProxyManager*s de en qué puerto y en qué dirección IP se encuentran el resto de nodos del sistema, para que sean ellos los que directamente establezcan y gestionen la comunicación.

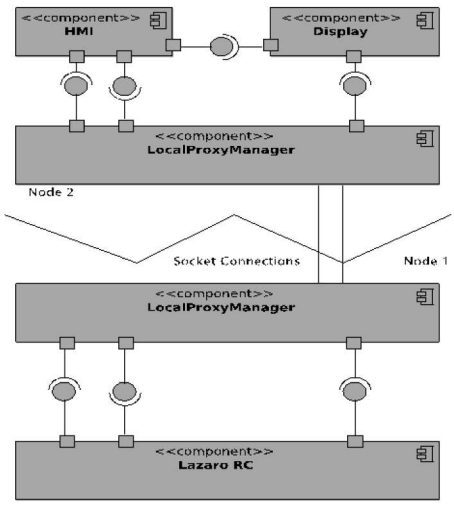

Figura 2: Arquitectura de una aplicación distribuida en dos nodos.

4. Conclusiones y trabajos futuros

En este artículo se ha descrito la evolución de un trabajo en curso, incorporando la capacidad de distribuir componentes a un framework OO de soporte al desarrollo basado en componentes. La capacidad de distribución se ha añadido de forma regular al framework, respetando su diseño original, lo que permite analizar el impacto que sobre las características temporales de la aplicación tiene una determinada distribución de sus componentes. Los servicios de distribución incorporados han sido los estrictamente necesarios para poder realizar el despliegue de los componentes, teniendo en cuenta que la arquitectura está definida previamente, y que por tanto es un parámetro de entrada al nodo maestro.

Para el tipo de aplicaciones que se han realizado hasta el momento, estos servicios han demostrado ser más que suficientes, si bien hay que comprobar cómo se comportan a medida que la complejidad de las aplicaciones va aumentando.

El trabajo descrito en este artículo es un trabajo en marcha. Actualmente se sigue trabajando en ampliar el framework con capacidades adicionales siguiendo un enfoque dirigido por patrones. Entre estas extensiones hay que destacar (1) la adición de heurísticos para determinar el número de tareas y llevar a cabo la asignación de las actividades de los componentes a dichas tareas, (2) el desarrollo de transformaciones de modelos que permitan instanciar el framework a partir de un modelo de componentes de entrada, (3) el refinamiento y la mejora de los patrones utilizados para implementar máquinas de estados jerárquicas y autómatas temporizados, y (4) la adaptación de la implementación para que sea conformes con el perfil Ravenscar para diseñar aplicaciones de tiempo-real estricto. Por último, pero no menos importante, estamos trabajando en la generación de modelos de entrada para herramientas de análisis a partir de las instancias del framework. Se ha hecho un estudio preliminar con la herramienta TimesTool, pero el objetivo final es utilizar el perfil MARTE de UML.

Referencias bibliográfica

[1] Lau, K. and Z. Wang (2007). Software component models. IEEE Trans. Software Eng. 33(10), 709–724.

[2] Shaw, M. and P. Clements (2006). The golden age of software architecture.. IEEE Softw. 23(2), 31–39.

[3] Brugali, D. and G. Menga (2000). Frameworks and pattern languages: an intriguing relationship. ACM ComputingSurveys 32(2), 1–6.

[4] Schlegel, C. (2008). The Challenge of Real-Time Robotics Behavior: An Applied Research Perspective. In Proc. SDIR-III, IEEE ICRA 2008, Pasadena, 2008.

[5] López, P. (2010). Desarrollo de sistemas de tiempo real basados en componentes utilizando modelos de comportamiento reactivos. Tesis Doctoral, Santander 2010.

[6] Iborra, A. et als (2009). Design of service robots: Experiences using software engineering. IEEE Robotics & Automation Magazine, DOI: 10.1109/MRA.2008.931635, marzo 2009.

[7] Pastor, J. et als (2010). Towards the Definition of a Pattern Sequence for Real-Time Applications Using a Model-Driven Engineering Approach. 15th International Conference on Reliable Software Technologies, Ada-Europe 2010.