ANÁLISIS DE DOMINIO DE UN MARCO DE TIEMPO REAL PARA VEHÍCULOS AUTÓNOMOS NO TRIPULADOS

Juan Fernando Franco H. y Luís Benigno Gutiérrez Z.

Universidad Pontificia Bolivariana (UPB)
Grupo de Investigación en Automática y Diseño A+D
Circular primera #70-01 bloque 11. Medellín, Colombia.
e-mail: luis.gutierrez@upb.edu.co, juanfh@geo.net.co

Abstract: Este artículo presenta los avances de la fase de análisis de dominio de un marco de tiempo real para sistemas de control computacional embebidos en vehículos autónomos no tripulados. Los requisitos del marco se exponen basados en otros marcos utilizados en aplicaciones similares. El análisis de dominio muestra que la Ingeniería de software sugiere emplear componentes y la abstracción de objetos basados en puerto para implementarlos, concluyendo que un marco de tiempo real basado en éstos puede servir como soporte para los proyectos del Grupo de Investigación en Automática y Diseño A+D de la Universidad Pontificia Bolivariana.

Keywords: Real-time, Domain analysis, Embedded systems, Vehicles, Software engineering, Components, Computer control.

1. INTRODUCCIÓN

Booch et al. (1999) define el término marco o *framework* (en inglés) como un "patrón arquitectónico que proporciona una plantilla extensible para las aplicaciones dentro de un dominio". El mismo autor define patrón como una "solución común a un problema común en un contexto determinado". Otra definición un poco más clara indica que un marco es un programa de computador parcialmente terminado, del cual, las partes faltantes son adicionadas por el desarrollador del programa (Rogers, 1997). Algunos ejemplos de marcos populares son: *ActiveX* y *COM+* de la compañía *Microsoft*, *JAVA* de *Sun Microsystems* y *CORBA* de *Object Management Groups*. El dominio de los marcos anteriores está orientado a aplicaciones corporativas o empresariales que distan en muchos casos de las exigencias temporales de los sistemas de control.

Algunos marcos se han desarrollado específicamente para sistemas de control de tiempo real, por lo tanto se han tomado como referencia para la elaboración de la etapa de análisis de dominio: *Orocos* (Bruyninckx y Soetens, 2006), *Quantum Framework* (Samek, 2002), *OXF* (Douglass, 1999), *OCP* (Wills et al., 2001), *PBO* (Stewart et al., 1997) y *Constellation* (RTI, 2003). Sin embargo, la mayoría son complejos, costosos y de acceso limitado. Es por esta razón que el grupo de Investigación A+D de la UPB decidió construir un marco de tiempo real propio que soporte

los proyectos en el tema de vehículos no tripulados, tales como: “Investigación y diseño de un sistema automático de inspección remota para líneas de transmisión de energía eléctrica”, se trata del desarrollo de un *UAV* (*Unmanned Aerial Vehicle*), y “Desarrollo de un vehículo sumergible operado remotamente ROV, para la inspección subacuática”. En el futuro se espera continuar esta línea con el estudio de vehículos terrestres no tripulados.

La metodología empleada para el análisis de dominio, es decir, la determinación de los requisitos actuales y futuros del marco de tiempo real, se basó en una revisión del estado del arte y la literatura asociada con el tema.

Los resultados parciales obtenidos indican que el marco de tiempo real debe cumplir con los siguientes requisitos: Diseño basado en componentes e integración de los mismos, mecanismos de integración de capas de control, extensibilidad, reconfiguración en tiempo de ejecución, garantizar requisitos de tiempo real a nivel de marco, gestión de fallos y manejo de excepciones, ejecución concurrente de componentes, gestión de eventos periódicos y aperiódicos, abstracción del sistema operativo de tiempo real, documentación de diseño y utilización y plataforma de *software* que oculte y gestione la complejidad.

2. ANÁLISIS DE DOMINIO

Los sistemas de control computacional embebidos de tiempo real presentan algunas dificultades de implementación que complican notablemente su desarrollo (Berger, 2002):

- A diferencia de los computadores de escritorio, hacen parte del dispositivo que controlan y están dedicados a una tarea específica.
- Pueden implementarse en una variedad enorme de arquitecturas de procesador.
- Las restricciones de tiempo condicionan el buen funcionamiento del sistema.
- Tiene limitaciones de costo, consumo y tamaño.
- Utilizan sistemas operativos de tiempo real.
- Operan en condiciones ambientales extremas.
- Poseen recursos limitados de memoria y procesamiento.
- Requieren herramientas especializadas y usualmente costosas para su diseño.
- Para la depuración emplean interfaces especiales de hardware.

Sumado a los aspectos anteriores, el diseño de sistemas de control para vehículos autónomos no tripulados debe tener en consideración otros factores, tales como (Rajive et al, 2005):

- Diseño por capas: Capa de bajo nivel, donde residen los manejadores de dispositivos, los controles realimentados básicos de los servomecanismos y el sistema de navegación. Capa media para el generador de trayectorias y capa alta para el control de misión.
- Control distribuido: Los procesos que componen la aplicación pueden estar en el mismo o varios sistemas de cómputo, incluyendo generalmente una estación de control remota. También es posible que el vehículo deba comunicarse con otros para coordinar misiones en grupo.
- Soporte para la integración de sensores y actuadores usualmente distribuidos, por medio de algoritmos de fusión de sensores.
- Uso de filtros para estimar el estado del sistema a partir de medidas con ruido, utilizando por ejemplo, filtros de Kalman.
- Soporte para diferentes modos de operación, es decir, el sistema debe garantizar cierto grado de reconfiguración para diferentes situaciones dentro de una misión o cambios en los algoritmos de control en tiempo real.
- Manejo de eventos asincrónicos como ejecución de tareas periódicas a diferentes frecuencias y manejo de eventos aperiódicos.
- Escalabilidad. El sistema debe ser extensible y robusto con respecto a la adición de nuevas funcionalidades.
- Operación por largos períodos de tiempo.
- Garantizar predictibilidad y robustez.

El *software* que constituye en si el marco de tiempo real posee unas especificaciones que son derivadas de otros marcos diseñados para aplicaciones similares.

2.1. Diseño basado en componentes

El diseño basado en componentes permite dividir una aplicación en partes que se integran a través del

marco de tiempo real. En ese sentido todos los marcos revisados mantienen esta característica común (RTI, 2003; Samek, 2002, Stewart et al., 1997; Douglass, 1999; Wills et al., 2001; Bruyninckx y Soetens, 2006). Una de las ventajas más importantes del diseño basado en componentes es la reutilización de código, la cual es deseable para la construcción de bibliotecas de componentes que puedan ser empleados en varios proyectos, minimizando de esta manera el esfuerzo y el costo de desarrollo.

2.2. Integración de componentes

Los componentes son precisamente integrados por medio del marco, permitiendo que la gestión de mantenimiento del *software* del sistema sea más fácil y robusta ya que únicamente es necesario adicionar componentes que extiendan la funcionalidad del vehículo autónomo no tripulado. Sin embargo, la integración va mucho más lejos, es decir, no sólo integra componentes sino que provee mecanismos bien definidos para el acople de las diferentes capas de control del vehículo.

2.3. Reconfiguración en tiempo de ejecución

La reconfiguración en tiempo de ejecución se logra garantizando un diseño modular y haciendo que los componentes puedan ser reemplazados sin afectar a los otros, es decir, independencia modular. La reconfiguración permite cambiar en "caliente" los algoritmos de control del vehículo, haciendo que algunos componentes se apaguen y otros se enciendan (Stewart, 1997).

2.4. Comunicación entre componentes

La estrategia de comunicación entre componentes más utilizada es la basada en mensajes utilizando el modelo *publish-subscribe* (Samek, 2002), sin embargo, este modelo presenta dificultades para analizar el comportamiento temporal del sistema (Hassani y Stewart, 1997), por lo tanto aparece otra alternativa basada en estados, la cual utiliza zonas de memoria compartida (Stewart, 1997).

2.5. Requisitos de tiempo real

Los marcos de tiempo real adicionan indiscutiblemente un retardo al tiempo de respuesta del vehículo, sin embargo, es necesario garantizar que este retardo esté acotado a un valor máximo en el peor de los casos. Usualmente los marcos corren sobre un sistema operativo de tiempo real que en últimas debe asegurar el cumplimiento de los requisitos temporales basado en alguna política de planificación de tareas.

2.6. Gestión de fallos y manejo de excepciones

Este trabajo debe ser conducido conjuntamente con el sistema operativo, particularmente detectando incumplimiento de plazos temporales y manejo controlado de fallos.

2.7. Programación concurrente

Cada componente puede estar embebido en su propio hilo de control que es planificado directamente por el sistema operativo. En caso de no contar con un sistema operativo, el marco debería gestionar la ejecución concurrente de los componentes.

2.8. Gestión de eventos periódicos y aperiódicos

El marco debe considerar los tiempos de ejecución de cada componente y junto con el sistema operativo garantizar los plazos. Adicionalmente gestionar de igual manera los eventos aperiódicos tratando de minimizar el tiempo de respuesta del componente.

2.8. Abstracción de sistema operativo de tiempo real

Para garantizar la portabilidad del marco desde el punto de vista de arquitectura de cómputo, debe existir una capa de abstracción de sistema operativo, de tal manera que sólo sea necesario rehacer esta capa y no todo el marco completo para pasar de un sistema operativo a otro.

2.9. Documentación

Un marco es útil en la medida que pueda ser utilizado por la mayor cantidad de personas. Por lo tanto además de documentar el diseño como tal, es necesario incluir guías tipo "cómo se hace" o "libros de recetas".

2.10. Plataforma de software

Es necesario incluir una plataforma de software que le sume al marco un conjunto de herramientas de

desarrollo y manejo que permitan integrar todas las capas de control. A continuación se describen algunas de esas herramientas tomadas de Rajive et al. (2003):

- Integración de capas de control: Un vehículo autónomo puede ser diseñado de la siguiente manera: las aplicaciones de bajo nivel escritas en lenguaje C, los algoritmos de control en MATLAB y Simulink, el modelo del sistema completo en UML y las interfaces de comunicación en CORBA.
- Herramientas de programación y productividad: Editor gráfico para ensamblar y conectar los componentes, herramientas de simulación y desarrollo de algoritmos como MATLAB y Simulink, modelado de sistemas y documentación basados en UML, depuradores, analizadores de uso de memoria.
- Manejador de componentes: Repositorio con manejo de versiones, exploración de componentes y su documentación, motor de búsqueda y gestión de bibliotecas de componentes.
- Monitoreo y control: Herramientas que permitan visualizar el comportamiento de la aplicación en ejecución y puedan almacenar archivos de registro en tiempo real, es decir, instrumentar la aplicación para análisis posteriores.
- Herramientas de proceso: Ambientes integrados de desarrollo multiusuario, herramientas de validación de componentes, entre otras.

3. OBJETOS BASADOS EN PUERTO (*PBO*)

A continuación se analizan las características de un modelo de objetos basados en puerto definido por Stewart et al. (1997) para implementar el concepto de componente reconfigurable. Dicho modelo, con posibles variaciones que aún no se han definido, se propone como base para la construcción de un marco de tiempo real para vehículos autónomos no tripulados.

Las características más atractivas de este modelo son:

- Escalabilidad: Desde un microcontrolador de bajo desempeño sin un sistema operativo de tiempo real hasta un multicomputador con QNX.
- Los componentes son reconfigurables.
- Está fundamentado en una teoría formal sobre autómatas de puerto definida por Streenstrup et al. (1983).

Stewart (2000) clasifica los componentes reconfigurables en tres clases:

- Componente genérico: Es un componente que no depende del *hardware* ni de la aplicación. Se puede configurar para diferentes tipos de *hardware* y puede ser usado en diferentes aplicaciones.
- Componentes *dependientes del hardware*: Solo pueden ser utilizados con un tipo específico de *hardware*. Este tipo de componente a su vez se divide en componentes de interfaz y de procesamiento. Los componentes de interfaz convierten señales específicas de hardware a datos independientes de éste. Los componentes de interfaz son empleados para reemplazar los manejadores de dispositivos y se caracterizan principalmente porque tienen asociados sus propios hilos de control. Los componentes de procesamiento se conectan a los componentes de interfaz actuando como un componentes genérico pero de mejor desempeño gracias a al uso de optimizaciones específicas de hardware.
- Componentes dependientes de la aplicación: Son componentes empleados para implementar detalles específicos de la aplicación.

3.1. Estructura básica de un componente

Como ya se mencionó los componentes son modelados como objetos basados en puerto. La figura 1. muestra un diagrama que ilustra el concepto. La comunicación entre los *PBO* se hace por medio de sus puertos. Cuando un *PBO* necesita información o genera información obtiene la versión más reciente desde sus puertos de entrada o la envía a sus puertos de salida, por lo tanto no existe sincronización entre objetos ni se conoce el origen y el destino de la información. Cada *PBO* se ejecuta como un proceso independiente. Los puertos de configuración sirven para reconfigurar componentes genéricos para ser utilizados con hardware o aplicaciones específicas. Los puertos de recursos no son implementados directamente por el *PBO*, básicamente sirven para modelar el origen o destino de los datos intercambiados con el hardware de entrada-salida. Los puertos no tienen un tipo de dato definido, es

decir, pueden manejar datos análogos, estructuras de datos, información procesada, etc., sin embargo deben ser nombrados y especificados cuando se inicializa el componente.

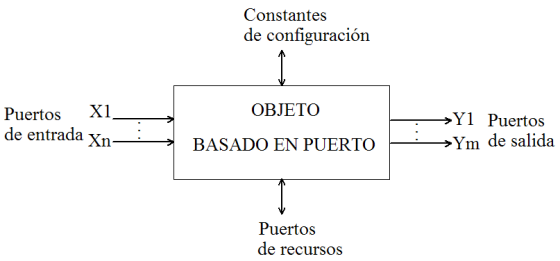

Fig. 1 Diagrama de objeto basado en puerto

Para conformar un sistema de control, es necesario interconectar varios *PBOs*, pero manteniendo transparente para el Ingeniero de control el mecanismo de comunicación entre ellos.

El comportamiento del *PBO* se codifica como una máquina de estados finitos que puede tener una estructura fija o variable dependiendo de la forma como se implemente el marco. La máquina de estados es controlada directamente por el marco y se le asocia un hilo de control, sin embargo, estos detalles de implementación deben quedar ocultos al Ingeniero de control cuya responsabilidad llega hasta la descomposición del sistema en componentes, definición de tiempos, plazos y eventos, codificación del comportamiento de los *PBOs* y finalmente la unión de los diferentes componentes para formar el sistema de control.

Una de las tareas importantes en el proceso de definición del marco de tiempo real será establecer el mecanismo de comunicación entre *PBOs*.

4. CONCLUSIONES

El análisis de dominio realizado indicó que la utilización de marcos facilita la labor del Ingeniero de control aislando los detalles de implementación.

Se concluye que el marco propuesto debe cumplir con las siguientes especificaciones: basado en componentes, mecanismos de integración de capas de control, extensibilidad, reconfiguración en tiempo de ejecución, garantizar requisitos de tiempo real a nivel de marco, gestión de fallos y manejo de excepciones, ejecución concurrente de componentes, gestión de eventos periódicos y aperiódicos, abstracción del sistema operativo de tiempo real, documentación a nivel del diseño y utilización y plataforma de *software* que oculte y gestiones la complejidad.

Se propone implementar la abstracción de componente por medio de objetos basados en puerto para soportar los proyectos sobre vehículos autónomos no tripulados del Grupo de Investigación en Automática y Diseño A+D de la UPB.

5. ESTUDIOS POSTERIORES

La fase de análisis de dominio es apenas la primera etapa en el proceso de diseño de un marco, los trabajos posteriores estarán encaminados a:

- Terminar la fase de análisis de dominio identificando claramente los puntos fríos y calientes.
- Diseño.
- Implementación.
- Pruebas.
- Aplicaciones.
- Desarrollo de la plataforma de *software*.

REFERENCIAS

Berger, A. (2002). *Embedded Systems Design: An introtroduction to Processes, Tools, and Techniques.* Berkely: CMP Books, 237 p.

Booch, G. , Rumbaugh, J. y Jacobson, I. (1999). *El lenguaje unificado de modelado*. Madrid: Addison Wesley, 432 p.

Bruyninckx, H. y Soetens, P. (2006). *Whats is orocos ?*. [online]. Disponible en: http://www.orocos.org/stable/rtt/current/doc/orocos-overview.html. [citado en noviembre 19 de 2006].

Burns A. y Wellings, A. (2003). *Sistemas de tiempo real y lenguajes de programación.* Madrid: Addison Wesley, 806 p.

Douglass, B.P. (1999). *Doing hard time: development real-time embedded systems with*

UML, objects, frameworks and patterns. Massachussets: Addison Wesley, 749 p.

Hassani, M. and Stewart, D. (1997). *"Mechanism for Communicationg in Dynamically Reconfigurable Embedded System". Proc. of High Assurance System ENgineering Workshop, Washington DC, August 1997.*

Rajive, J, Bose, A. and Breneman, S. (2005). *Software Plataforms for Unmanned Systems*. [online] mayo de 2005. Disponible en: http://www.rti.com/products/robotics_controls/. [citado en noviembre 18 de 2006].

RTI,. (2003). *Constellation, A Real-Time software framework, overview, capabilities and benefits*. [online]. Mayo de 2003, Disponible en: http://www.rti.com/products/robotics_controls/. [citado en noviembre 19 de 2006].

Rogers, G. (1997). *Frameworks-Based Software Development in C++*. N.J.: Prentice Hall.

Samek, M. (2002). *Practical Statecharts in C/C++*. Kansas: CMP Books, 387 p.

Stewart, D. (2000). *Software Components for Real Time". Embedded system Programming.* Vol 13, no. 13. Dec. 2000.

Stewart, D.B, Volpe, R.A y Khosla, P.K. (1997).*"Design of dynamically reconfigurable real-time software using portbased objets". IEEE Trans. on Software Engineering*, v.23, n12, Dec. 1997.

Steenstrup, M., Arbib, M.A. y Manes, E.G. *"Port Automata and Algebra of Concurrent Processes". Journal of Computer and Systems Sciences*, v.27, n. 1, Aug. 1983, pp. 29-50.

Wills, L., Kannan, S., Sander, S., Guler, M., Heck, B., Prasad J.V.R., Schrage D., and Vachtsevanos G. (2001). *An Open Platform for Reconfigurable Control*. IEEE Control Systems Magazine. June 2001.