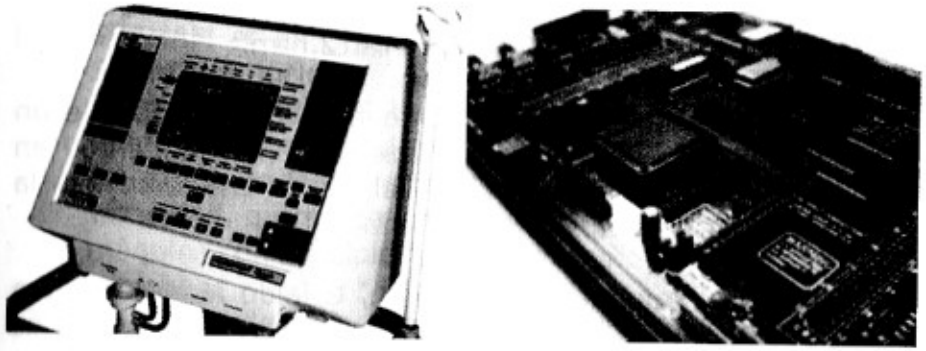
Alternativas de Diseño de Software para Sistemas Embebidos de Tiempo Real Basadas en Componentes



Juan Fernando Franco Higuita
Ingeniero Electrónico
juanfh@geo.net.co

Luis Benigno Gutiérrez Zea
Ingeniero Electrónico
PhD en Ingeniería Eléctrica y de Computadores
luis.gutierrez@upb.edu.co

Abstract. The objective of this article is to show some alternatives of design of software for real time embedded systems based on components by means of the description of some mechanisms of scheduling and interaction between components. At the present time the technology of components is being used by the Group of Investigation A+D of the UPB in the development of autonomous and unmanned vehicles.

Keywords: Real-Time, Embedded systems, Components, Software Engineering, Messages, State variable, Scheduling, Objects.

Resumen. El objetivo de este artículo es mostrar algunas alternativas de diseño de software para sistemas embebidos de tiempo real basadas en componentes por medio de la descripción de algunos mecanismos de scheduling e interacción entre componentes. En la actualidad la tecnología de componentes está siendo utilizada por el Grupo de Investigación A+D de la UPB en los desarrollos de vehículos autónomos y no tripulados.

Palabras Clave: Tiempo real, Sistemas embebidos, Componentes, Ingeniería de software, Mensajes, Variable de estado, Planificación, Objetos.

1. INTRODUCCIÓN

Los sistemas embebidos son sistemas electrónicos que incluyen un microcomputador que se programa para desarrollar una aplicación dedicada. Este tipo de sistemas están embebidos o hacen parte del dispositivo que controlan. Por ejemplo: Los sistemas de control de un automóvil, un teléfono celular, un controlador lógico programable (PLC) para aplicaciones de control de procesos industriales, entre otros [1]. Hay varias características de los sistemas embebidos que dificultan su desarrollo [2]:

- A diferencia de los computadores de escritorio, los sistemas embebidos están dedicados a una tarea específica.
- Pueden implementarse en una variedad enorme de arquitecturas de procesador.
- Usualmente son sensibles al costo.
- Tienen restricciones de tiempo real.
- Pueden utilizar sistemas operativos de tiempo real (RTOS).
- El diseño debe ser pensado para minimizar el tiempo mínimo entre fallos.
- Tienen restricciones de potencia.
- Generalmente operan en condiciones ambientales extremas.
- Tienen menos recursos de sistema que los computadores de escritorio.
- Usualmente el programa (firmware) está almacenado en memoria ROM (Read Only Memory), lo cual impone restricciones de tamaño de programa.
- Requiere herramientas especializadas y generalmente costosas para su diseño.

- El proceso de depuración y validación requiere la adición de circuitos especiales en el sistema.

Además de las características anteriormente mencionadas, es muy común que el sistema embebido deba cumplir requisitos de tiempo, convirtiéndose en un sistema de tiempo real, es decir, un sistema que debe satisfacer explícitamente restricciones de tiempo de respuesta o de lo contrario el sistema podría fallar. Un fallo quiere decir, que éste no logró alcanzar los requerimientos impuestos en su especificación [3]. Por lo tanto, en un sistema embebido de tiempo real su funcionamiento correcto depende no sólo del resultado lógico de las computaciones, sino también del tiempo en el que se producen los resultados [4].

Debido a las restricciones funcionales y temporales que presenta un sistema embebido de tiempo real, el diseño y desarrollo de software se convierte en una tarea muy compleja y costosa la cual debe repetirse cada vez que se construye un sistema nuevo, adicionalmente, en la actualidad el tiempo al mercado es en muchas ocasiones un factor de éxito definitivo en el ciclo de vida de un producto. Este escenario hace necesaria la introducción de metodologías y tecnologías propias de la Ingeniería de software para la producción de sistemas embebidos de tiempo real de manera ágil, robusta y económica, respondiendo de esa manera a las demandas del mercado. Una de las tecnologías más populares en la actualidad es la Ingeniería de software basada en componentes.

Este artículo pretende hacer una revisión de varias alternativas de diseño de software para sistemas embebidos de tiempo real basadas en componentes desde la perspectiva de los mecanismos de interacción entre los componentes y las diferentes técnicas de scheduling empleadas, pero dejando al lector la tarea de profundizar en cada una de las técnicas mencionadas.
Es importante anotar que este tipo de tecnología están siendo empleada actualmente en la construcción de marcos de tiempo real para los proyectos sobre vehículos no tripulados del Grupo de Investigación A+D de la UPB, tales como: "Investigación y diseño de un sistema automático de inspección remota para líneas de transmisión de energía eléctrica", 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.

2. CONCEPTO DE COMPONENTE

Un componente es una parte física y reemplazable de un sistema que ofrece servicios y una interfaz que le permite comunicarse con otros componentes. El objetivo de la Ingeniería de software basada en componentes es lograr la producción de software de la misma manera como se producen sistemas mecánicos o eléctricos. Para lograrlo se emplean plataformas de software que permiten la conexión y configuración de los componentes utilizando marcos o frameworks y bibliotecas de componentes prediseñados. La tecnología de componentes está fuertemente relacionada con el modelamiento de sistemas basado en objetos. Algunos ejemplos de modelos de descomposición que utilizan componentes son los bloques funcionales [5] y los objetos basados en puerto [6]. Es muy usual que cada componente esté embebido en su propio hilo de ejecución y que se comunique con otros componentes, por lo tanto los temas del scheduling y comunicación entre componentes son claves en la implementación de éstos.

3. MECANISMOS DE SCHEDULING DE TIEMPO REAL.

El término scheduling, propio del vocabulario de los sistemas operativos, se podría traducir al español como planificación: es la forma como el sistema operativo planifica la ejecución de los procesos, sin embargo, en el desarrollo de software para sistemas de tiempo real es usual emplear la técnica de multihilo, es decir, una técnica en la cual "un proceso, ejecutando una aplicación, se divide en una serie de hilos, que pueden ejecutar concurrentemente" [7]. Es importante aclarar que es posible tener un mecanismo de scheduling para la planificación de varios hilos incluso en sistemas con en microcontroladores de bajo desempeño como los de 8 bits que no necesariamente cuentan con un sistema operativo.

Los mecanismos de scheduling para sistemas de tiempo real deben preservar las propiedades temporales de los componentes garantizando el cumplimiento de los plazos de ejecución.
Los scheduling de tiempo real pueden ser en general preemptive o non-preemptive, es decir, con desalojo de tareas o sin desalojo de tareas (entiéndase el término tarea como hilo en el contexto de este artículo). Los esquemas non-preemptive son usuales en sistemas microcontrolados de bajo desempeño que no cuentan con sistemas operativos. En este caso, el scheduling es cooperativo, es decir, cada tarea se apodera de la CPU un instante pequeño de tiempo y la devuelve para que otra tarea pueda utilizarla, el esquema normalmente es cíclico y las aplicaciones son del tipo background-foreground, en las cuales las interrupciones se ejecutan en el foreground (contexto de interrupción) y las tareas en background (ciclo principal del programa) siendo estas últimas de menor prioridad que las interrupciones. Los esquemas preemptive son utilizados sobre sistemas operativos de tiempo real y se basan en prioridades.

La idea es asignar a cada tarea una prioridad y ejecutar siempre la tarea de más alta prioridad. Las prioridades se pueden asignar de dos formas: estáticas, en tiempo de diseño o dinámicas, en tiempo de ejecución. Si una tarea de más alta prioridad que la actual está lista para ejecutarse, desaloja esta última y al terminar le devuelve la oportunidad de ejecución.

Aunque este esquema es adecuado para el diseño de sistemas de tiempo real, resulta complicado el tema de la asignación de prioridades. El objetivo de la asignación de prioridades en un sistema de tiempo real es lograr que todas las tareas finalicen antes del plazo establecido para cada una de ellas en la etapa de diseño [8]. Hasta la fecha, se han desarrollado varias políticas de asignación de prioridades siendo las más populares: Rate Monotonic (RM), asignación estática, y Earliest-Dead-Line-First (EDF), asignación dinámica [9].

4. ARQUITECTURAS DE TIEMPO REAL

Como se mencionó anteriormente la conexión e interacción entre componentes se hace a través de marcos de tiempo real. Actualmente se destacan dos tipos de arquitecturas: Arquitecturas manejadas por tiempos y arquitecturas manejadas por eventos. Ambas manejan el concepto de componente y la interfaz de puerto para la comunicación entre ellos. Las arquitecturas manejadas por tiempo (Time-Triggered Architectures: TTA) disparan la ejecución de componentes por medio de eventos temporales, esto es tienen un período de muestreo, presentando dificultades en el manejo de eventos aperiódicos. Usualmente en cada período de muestreo, el componente consume los valores de los puertos de entrada, los procesa y los envía a sus puertos de salida de manera cíclica, sin embargo, además de los problemas de manejo de eventos aperiódicos, es usual que las salidas de los componentes presenten jitter debido a que el tiempo de procesamiento de las entradas pueden ser variable, es decir, aunque el componente se ejecuta de manera periódica las salidas pueden ser producidas en cualquier instante durante el tiempo de ejecución del componente [6].

Las arquitecturas manejadas por eventos (Event-driven Architectures: EDA) ejecutan los componentes cuando se presentan eventos en sus puertos de entrada. Este esquema es más general que el anterior porque los eventos podrían ser temporales o aperiódicos. Un ejemplo interesante de implementación de este esquema es la arquitectura Timed Multitasking [8]. En esta arquitectura, los componentes se activan por el arribo de eventos en sus puertos de entrada, pero se controla el instante en el cual se producen las salidas de los componentes, manteniendo de esta manera un consumo y producción regular de las entradas y las salidas respectivamente.

5. MECANISMOS DE INTERACCIÓN DE COMPONENTES

Los mecanismos de interacción de los componentes corresponden a los diferentes patrones por medio de los cuales se comunican o intercambian información. Los patrones más utilizados son: cliente-servidor, productor-consumidor y

publicador-subscriptor. En estos patrones la información se intercambia en forma de mensajes. En el modelo cliente-servidor, un componente llamado cliente solicita información a un componente llamado servidor, por lo tanto el esquema de comunicación es uno a uno y posiblemente no sería adecuado para la implementación de sistemas de tiempo real reconfigurables, adicionalmente debido al carácter sincrónico de la comunicación, podría presentar problemas de bloque o retardos inaceptables en sistemas distribuidos. En el modelo productor-consumidor, un componente produce los mensajes que son consumidos por otro u otros componentes pero de manera anónima, es decir, el productor no sabe quién consumirá los mensajes y el consumidor no conoce quién los produce. La interfaz de comunicación entre los componentes se hace a través de puertos que se enlazan en tiempo de diseño, es decir, el enlace entre puertos es estático. En el modelo publicador-subscriptor, el publicador genera de manera espontánea mensajes, mientras que los subscriptores deben informar su interés en determinados tipos de mensajes. Un mecanismo de intermediación debe escuchar todos los mensajes publicados y luego informar a los subscriptores interesados la disponibilidad de un mensaje debidamente filtrado.

Como se mencionó antes, los patrones expuestos se basan en el intercambio de mensajes, sin embargo, este no es el único modelo utilizado. Existe otro concepto denominado comunicación basada en estados [6], en la cual existe una zona de memoria compartida donde residen todos los datos que los componentes pueden leer y escribir, evitando de esta manera copiar varias veces la misma información como en el caso de la comunicación por medio de mensajes.

6. CONCLUSIONES

El desarrollo de software para sistemas embebidos de tiempo real basado en componentes permite el concepto de producción de software tal como se producen sistemas mecánicos o eléctricos. En este artículo se mostraron algunas alternativas de scheduling e interacción entre componentes que deben ser evaluadas a la hora del diseño de aplicaciones de tiempo real basadas en componentes. En la actualidad la tecnología de componentes está siendo utilizada por el Grupo de Investigación A+D de la UPB en los desarrollos de vehículos autónomos y no tripulados.

7. REFERENCIAS

[1] J.W. Valvano. "Introducción a los sistemas de microcomputadora". México, D.F.: Thomson, 2004. p. 55.

[2] A. Berger. "Embedded Systems Design: An Introduction to Processes, Tools, and Techniques". Berkley: CMP Books, 237p.

[3] P. A. Laplante. "Real-Time Systems Design and Analysis: An Engineer's Handbook", 3era edición. Marzo, 2004.

[4] A. Burns. y A. Wellings, "Sistemas de tiempo real y lenguajes de programación". Madrid: Addison Wesley, 2003. 806 p.

[5] IEC 61499-1. "Function Blocks-Part 1: Architecture". IEC Standard, 2005.

[6] D. Stewart. "Software Components for Real Time". Embedded system Programming. v.13 , no.13. Dec. 2000.

[7] W. Stallings. "Sistemas Operativos: Aspectos internos y principios de diseño", 5ta edición. Madrid: Prentice-Hall, 2005. p 60.

[8] J. Liu y E.A. Lee. "Timed Multitasking for Real-Time Embedded Software". En Control System Magazine, IEEE, v.23, n.1, Feb. 2003. pp. 65-75.

[9] C. Lie y J. Layland. "Scheduling algorithms for multiprogramming in hard real-time environment" En Journal of ACM, v.10, n.1, 1973. pp. 46-61