Desarrollo de sistemas embebidos steer-by-wire

Por T. Erkkinen [2] y J. Langenwalter [1]

[1] Joachim Langenwalter
Director de marketing para la industria de automoción europea
jlangenwalter@mathworks.com

[2] Tom Erkkinen, director de generación de código embebido de The MathWorks
terkkinen@mathworks.com

The MathWorks
www.mathworks.com

El diseño basado en modelos permite la generación automática de software a partir de modelos para sistemas embebidos de gran producción para la automoción. Para llevarlo a cabo, es necesario contar con la estructura de ingeniería de software capaz de apoyarlo.
En este documento, se presenta una estructura de procesos, métodos y herramientas para el diseño de sistemas embebidos para automoción. Un sistema de conducción por cable (steer-by-wire) nos servirá como ejemplo.

Introducción

Recientemente, varias empresas de diferentes sectores, entre ellas Denso, Motorola y Toyota [1] han informado acerca del éxito de su código de producción. Esta tecnología se está convirtiendo en un componente importante de la evolución futura del desarrollo de software.

A pesar de que se comprende el impacto general que tendrá sobre el proceso de ingeniería de software, éste aún no se ha definido claramente. Esto es especialmente evidente para los participantes de evoluciones similares anteriores, como la migración del código máquina al código ensamblador y luego al código fuente.

La creciente abstracción y automatización trae aparejados nuevos procesos, métodos y herramientas. Los procesos en cascada se han quedado por el camino, en favor de enfoques en espiral o iterativos. Han aparecido los métodos de tiempo real, desplazando al diseño de flujo estático. Han aparecido nuevas herramientas tales como las IDE con depuradores, compiladores optimizados y herramientas de pruebas automatizadas.

No obstante, debido a la dificultad de uso, comprensión o al soporte limitado de las herramientas, no todas las buenas ideas han florecido. La experiencia nos indica que estos métodos y herramientas no siempre son prácticos para su uso en producción generalista. Por ejemplo, los métodos formales en los que se utilizan pruebas para garantizar la corrección del software están escritos en un lenguaje que sólo unos pocos expertos en el mundo pueden entender realmente. Además, las herramientas CASE de tiempo real de los años ochenta ayudaron al diseño, pero no proporcionaron una ruta fácil hacia el código final.

La generación de código de producción ha funcionado bien en la fase inicial de adopción, debido principalmente a su practicidad. Sin embargo, para su crecimiento se necesitan procesos, métodos y herramientas integrados que puedan apoyarla. Un nuevo proceso sólo tendrá éxito si cuenta con los métodos y herramientas necesarios para llevarlo a cabo. Si falta alguna de estas piezas, el esfuerzo de reingeniería que requiere un sistema embebido maduro de una empresa, ya no es factible o no es práctico.

Este documento presenta una estructura centrada en la generación de código de producción:

- *Proceso:* diseño basado en modelos
- *Métodos:* modelado, simulación, prototipaje rápido, generación de código de producción, análisis de cobertura y pruebas del modelo y pruebas in-the-loop
- *Herramientas:* herramientas de desarrollo, herramientas de verificación y validación y herramientas integrales

Proceso

El diseño basado en modelos satisface las necesidades de ingenieros de control, de sistemas DSP (procesamiento digital de señales) y desarrolladores de software al brindar un entorno común para la creación de especificaciones de manera gráfica y análisis. En este proceso, los modelos se crean y utilizan para especificar los datos en el sistema, interfaces, lógica de control realimentado, lógica discreta/de estado y comportamiento en tiempo real.

El diseño basado en modelos se usa en prácticamente todos los sectores que requieren el desarrollo de sistemas de control embebidos. Está particularmente bien afianzado en los procesos de desarrollo para aplicaciones embebidas tales como unidades de control electrónico en automoción producidas en masa d. Las aplicaciones de DSP y comunicaciones también usan este enfoque, pero enfatizan el modelado y la generación de prototipos en lugar de la generación de código de producción.

Para satisfacer estas diferentes aplicaciones, el proceso de diseño basado en modelos debe tratar las necesidades de sistemas críticos como los sistemas de conducción por cable (steer-by-wire). El proceso debe generar un código final ejecutable que sea muy compacto, rápido y trazable. Esto se debe a la naturaleza las de unidades de control electrónico (ECU) que se producen de forma masiva y que requieren el uso de unidades microcontroladores de coma fija y procesadores DSP de bajo coste.

El diseño basado en modelos se adapta al contexto de cualquier estructura de procesos, incluyendo aquellas definidas en los estándares de ingeniería de software IEEE [2].

El estándar IEEE 730 se aplica a cualquier proyecto de software de propósito general. Se puede obtener una buena comprensión de su estructura de procesos leyendo los requisitos definidos en este estándar para la documentación de proyectos"críticos".

Los requisitos del estándar IEEE 730 incluyen:

- Especificación de los requisitos de software (SRS)
- Descripción del diseño de software (SDD)
- Plan de verificación y validación del software (SVVP)
- Informe de verificación y validación del software (SVVR)

• Documentación del usuario
• Plan de gestión de la configuración del software (SCMP)
• Otros documentos, incluido el plan de gestión del proyecto de software (SPMP)

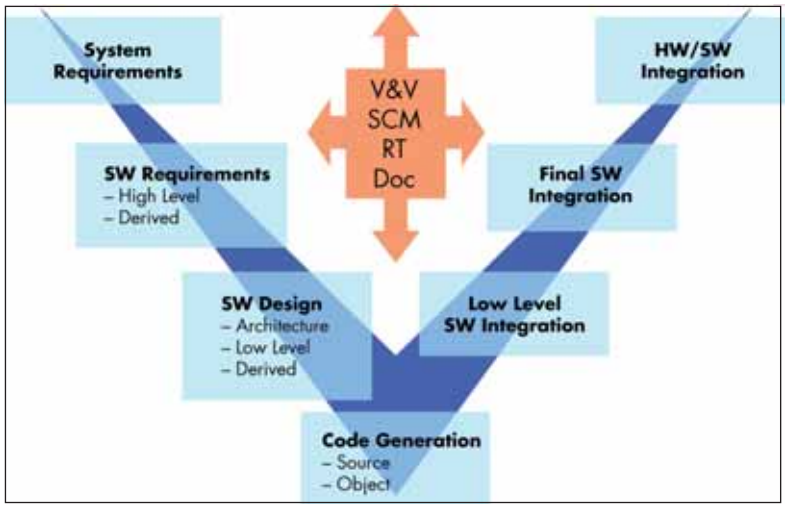


Figura 1. Diagrama V del proceso de software

Una forma habitual de ver el proceso de software es mediante el uso de un diagrama V, tal como se muestra en la figura 1. El diagrama corresponde a la mayoría de los procesos de ingeniería; no obstante, el proceso es iterativo con muchos pasos repetitivos a lo largo del ciclo de vida de desarrollo.

El proceso de software de este diagrama se compone de lo siguiente:
• Desarrollo (requisitos, diseño, codificación, integración)
• Verificación y validación (V&V)
• Integración (gestión de la configuración del software, , trazabilidad y documentación de las especificaciones)

El diseño basado en modelos pone gran énfasis en la iteración de procesos, las pruebas en las fases iniciales y la reutilización a lo largo del proceso de desarrollo, lo que lo hace excepcional y potente. La practicidad inherente a este proceso se demuestra al final del diagrama V: la generación de código de producción es una transición automática desde el diseño.

En el diseño basado en modelos, un diagrama de bloques o un modelo de diagrama de estado pueden servir como requisitos del sistema y del software, diseño de software o, con un ligero cambio de percepción, como código fuente.

Otra característica distintiva de este proceso es el amplio esfuerzo de verificación y validación que se hace antes de llegar a la versión final. Las ventajas de la verificación y validación anticipadas es claro: se encontrarán menos errores y habrá que reescribir menos durante la integración y prueba del sistema final. Descubrir un error en el escritorio es mucho mejor que encontrarlo en una prueba de conducción de invierno en Finlandia. Otra ventaja es que se acorta el tiempo de lanzamiento al mercado.

Métodos y herramientas

Los métodos de diseño basado en modelos se emplean durante el proceso de ingeniería de software.

Los métodos de desarrollo incluyen:
1. Modelado de comportamiento
2. Diseño de software detallado
3. Diseño de arquitectura distribuida
4. Generación de código de producción
5. Integración de targets embebidos

Los métodos de verificación y validación incluyen:
a. Simulación y análisis
b. Prototipaje rápido
c. Análisis de cobertura y pruebas del modelo
d. Seguimiento y revisiones de código
e. Verificación de hardware-in-the-loop (HIL)

Los métodos integrales incluyen:
a. Interfaz de control de la fuente
b. Interfaz de gestión de especificaciones
c. Generación de informes

Más adelante se incluye una descripción concisa de cada método de desarrollo con ejemplos e información de soporte de las herramientas. Todas las herramientas que se muestran están disponibles comercialmente [1]. Las secciones siguientes se refieren a la actividad de desarrollo e incluyen métodos clave de verificación y validación. Por último, el documento concluye con los componentes integrales.

Modelado de comportamiento

Los modelos se utilizan para especificar los requisitos y el diseño de todos los aspectos de cada subsistema individual (por ejemplo, un sistema de conducción por cable).

Un sistema típico incluye:
• Entrada (por ejemplo, sensores del volante)
• Modelo del controlador o DSP
• Modelo de planta (motor de CC, cremallera y piñón, ruedas)
• Salida (cambio de dirección)

Tal como se muestra, se puede crear un modelo de sistema para representar el comportamiento deseado utilizando diagramas de bloques del sistema de control para control con retroalimentación, máquinas de estado para eventos discretos y lógica condicional y bloques DSP para filtros.

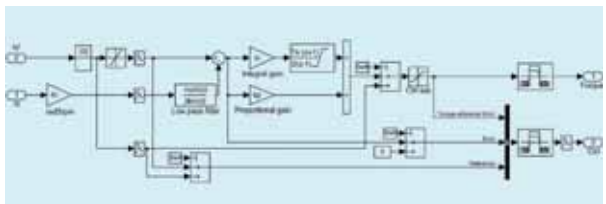
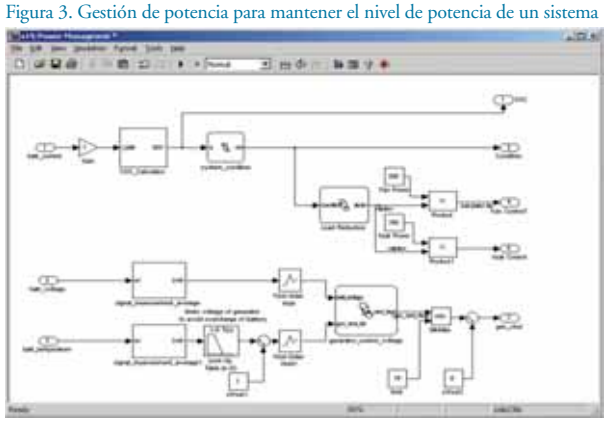


Figura 2. Modelo de un controlador PI para conducción por cable

Figura 3. Gestión de potencia para mantener el nivel de potencia de un sistema



Simulación y análisis

A continuación, el modelo se ejecuta y analiza para garantizar de que los requisitos se cumplen, usando métodos tales como las simulación basada en tiempo o en eventos y análisis en dominio de frecuencia. Por ejemplo, un sistema de conducción por cable debe responder al fallo de un sensor y "debe atenuar la respuesta de alta frecuencia por debajo de 3 db y no demorar la señal comandada más de 1,5 m/seg."

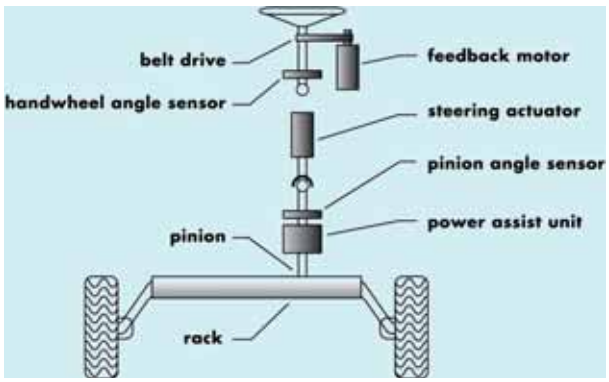


Figura 4A. Sistema de conducción por cable

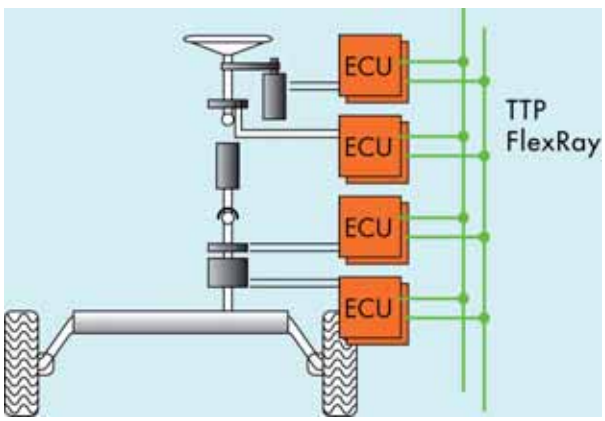


Figura 4B. Sistema de conducción por cable con sistema de bus redundante con tolerancia de errores (FlexRay)

El modelado y la simulación del sistema de conducción por cable de las figuras 4A y 4B determina si estos requisitos están en conflicto o son válidos. La simulación es una actividad de validación esencial y garantiza que se pueda crear un sistema que cumpla los requisitos.

Prototipaje rápido

Debido la imprecisión de los modelos de planta y a que la potencia de procesamiento es insuficiente para obtener una solución de trabajo en el chip de producción, el modelado por sí solo no proporciona una solución total.

Para solucionar estos inconvenientes, el prototipaje rápido es muy útil porque sustituye el modelo de planta por el modelo físico. En el ejemplo de la conducción por cable, la planta sería un coche y, en este caso, se utiliza un coche real. Sin embargo, debido a que el sistema aún no se ha construido, una plataforma de tiempo real o embebida ejecuta el software del controlador e interactúa con la planta.

Existen dos formas de prototipaje rápido: funcional y en el target. El prototipaje funcional usa un potente ordenador en tiempo real, como por ejemplo un PowerPC multiprocesador de coma flotante o un sistema DSP. El propósito es determinar si el sistema controla el coche físico del mismo modo que controlaba el coche modelado. Si lo hace, se demuestra que las imprecisiones del modelo de planta son insignificantes, y se valida la estrategia de control.

El prototipaje rápido sobre el objetivo (on-target) ejecuta el software en el mismo MCU o DSP, en lugar de en el núcleo de un PowerPC de gama alta u otro tipo de hardware de prototipaje rápido de gama alta. El propósito es descargar el código en el objetivo de producción real para hacer pruebas rápidas con la planta física. Si su comportamiento es correcto, el controlador, no sólo queda validado, sino que también se confirma que puede llevarse a la producción.

Diseño de software detallado

La actividad de diseño de software incluye la especificación de datos de coma fija, las tareas de tiempo real, los tipos de datos, las pruebas integradas y los diagnósticos.

Con el diseño basado en modelos, los ingenieros de software refinan y restringen el mismo modelo que se usa para la especificación de algoritmos y la validación como parte del proceso de generación del código de producción.

Pruebas del modelo

Es mucho mejor probar el modelo en el escritorio que implementarlo en hardware para su construcción e integración. Las pruebas basadas en el código fuente han existido durante muchos años, pero los métodos más recientes permiten la realización de pruebas en los modelos y análisis de la cobertura estructural. El escenario de uso es que un desarrollador "ejercita" al controlador completamente para verificar la integridad de su diseño por medio de la simulación y la cobertura. Otro tipo de pruebas es el análisis de modos y efectos de fallos (FMEA, del inglés, Failure Mode Effect Analysis), para garantizar el funcionamiento seguro en caso de fallo de un sistema de conducción por cable , tal como se muestra en la figura 5.

Por ejemplo, el desbordamiento numérico o el código muerto serían dos ejemplos de diseño pobre desde el punto de vista de la integridad. La realización de pruebas de estrés del modelo con valores numéricos máximos y mínimos ayuda a evitar que se produzcan condiciones de desbordamiento. Este tipo de pruebas es fácil de realizar con la simulación, pero lo que no es fácil es detectar el código

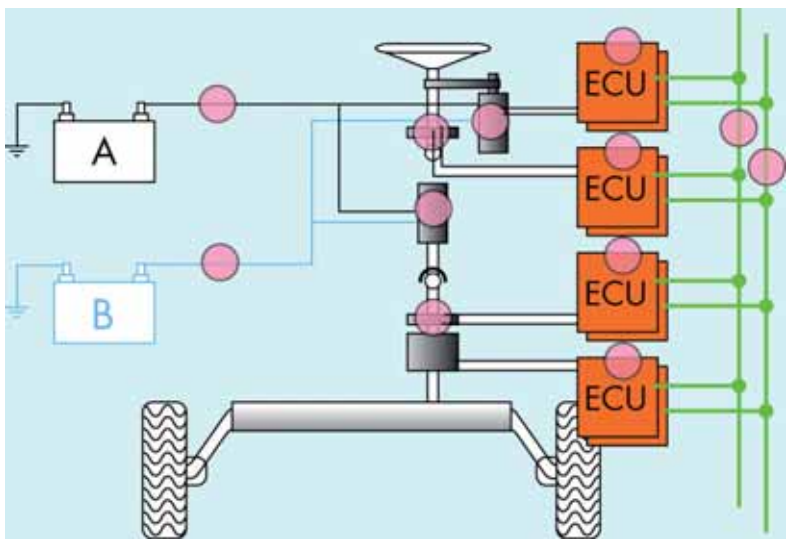


Figura 5. Posibles fallos de un sistema de conducción por cable

muerto. Para ello es necesario llevar a cabo pruebas de cobertura estructural. La diferencia entre el código muerto y el código desactivado radica en que el desarrollador es consciente de que el código desactivado existe y lo ha desactivado por alguna razón. En cambio el código muerto, significa que algo se ha "escapado" durante la especificación.

El análisis de cobertura del modelo evalúa los resultados acumulados de un procedimiento de prueba para determinar los bloques que no fueron ejecutados y los estados a los que no se llegó durante una simulación.

Ciertos tipos de cobertura están bien establecidos en los lenguajes de código fuente (como C, C++) pera ahora el análisis de cobertura está disponible para el modelo [3]. Este esfuerzo requería nuevas teorías y herramientas que no eran necesarias (ni factibles) para C, debido a que estos lenguajes no utilizan construcciones como bloques o estados.

La Administración Federal de Aviación de Estados Unidos (FAA) considera la cobertura de decisión por condición modificada (MD/DC) como el nivel de cobertura más riguroso que deben cumplir los sistemas en los que la seguridad es vital [4]. Esta cobertura, entre otras, ahora está disponible en el diseño basado en modelos y, en muchos casos es necesario utilizarla en los diseños de X-by-wire

Diseño de arquitectura distribuida

Los sistemas embebidos modernos contienen varias unidades de control electrónicas (ECU) distribuidas, que se comunican entre sí a través de un sistema de comunicaciones con tolerancia de errores como FlexRay™. El último sistema de control de estabilidad dinámica (DSC del inglés Dynamic Stability Control) de BMW contiene el ABS como una de las 15 subfuncionalidades [5]. Al añadir bloques de componentes de red tales como hosts, tareas, señales, etc. de DECOMSYS [6] a los subsistemas individuales, las funciones embebidas se pueden conectar y correlacionar con una arquitectura de ECUs. Además, facilita la simulación del comportamiento temporal de activaciones de tareas de un sistema operativo dirigido por tiempo como el OSEKtime/OS. Los clusters, hosts, tareas y conexiones se diseñan y simulan en el entorno de MATLAB / Simulink. Por último, el diseño completo se integra sin problemas con el producto DECOMSYS::DESIGNER para interactuar con el depósito de datos de diseño xCDEF de FlexRay.

Y las soluciones de diseño distribuido en red de Vector (DaVinci) y Cadence (SysDesign) integran el código generado por el RTW / RTW-EC de SL/SF (Simulink/Stateflow) a partir de susbsistemas y los correlacionan con las arquitecturas para su verificación.

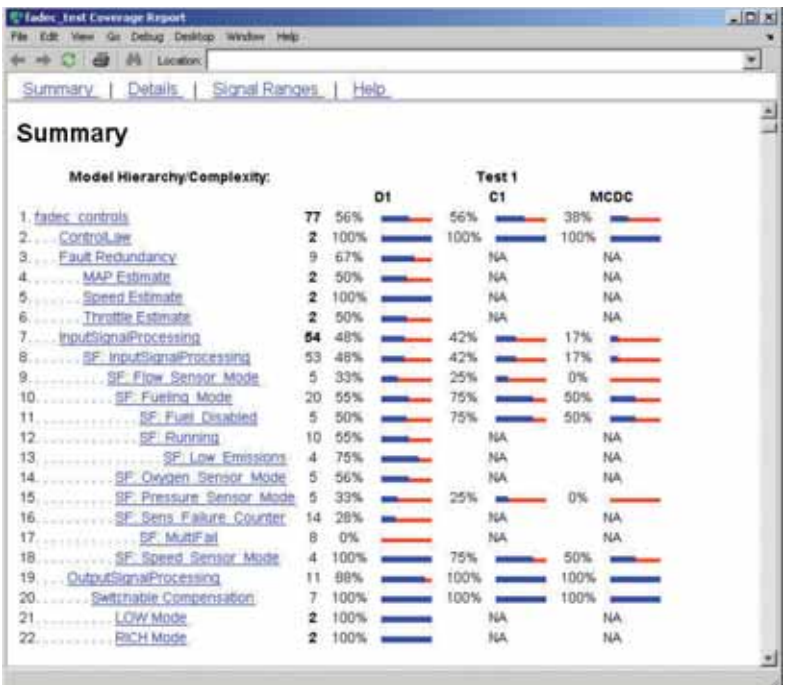


Figura 6. Cobertura del diseño de gestión de potencia de la figura 3

Generación de código de producción

Una vez que se ha verificado y validado el modelo, es el momento de generar el código. Este es un proceso es sencillo, similar a la utilización de un compilador. Existen varios parámetros de optimización y opciones de configuración del usuario. La clave está en conseguir que el código sea eficiente, preciso y que se integre con el código heredado y otras herramientas. También es importante poder ver la trazabilidad del código en el diagrama, para poder revisarlo y verificarlo.

Seguimiento y revisiones de código

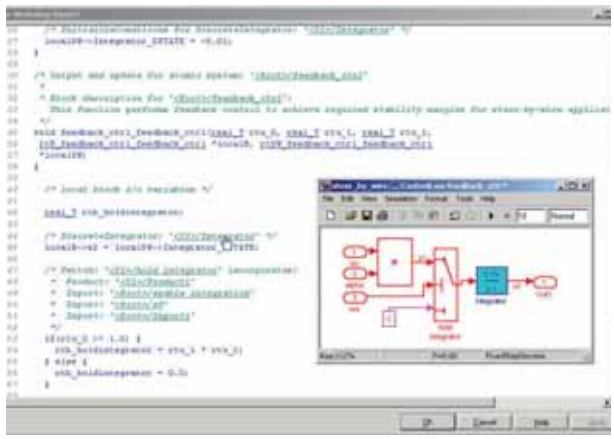


Figura 7. Revisión del código

La figura 7 muestra un informe en formato HTML enlazado automáticamente. Tal como se muestra a continuación, cuando el desarrollador selecciona el bloque Sum en el código, el bloque Sum se resalta en el diagrama.

Integración de objetivos embebidos

La figura 7 utiliza bloques de transición de tasa de ejecución (Rate Transition Block); no obstante, también existen enlaces directos a RTOS (sistemas operativos de tiempo real) comerciales, incluidas variantes de VxWorks y OSEK. Tal como se muestra a continuación, también es necesaria la integración del controlador de dispositivo.

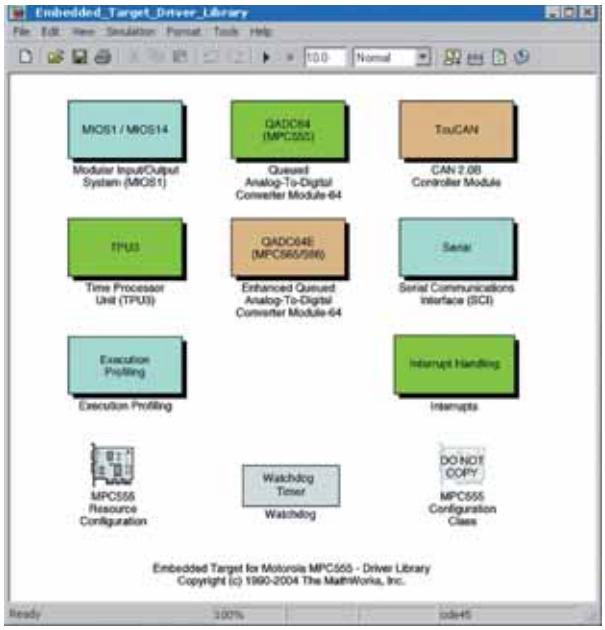


Figura 8. Blockset de controlador de dispositivo para el Motorola MPC555 utilizado en el sistema de conducción por cable

Componentes integrales

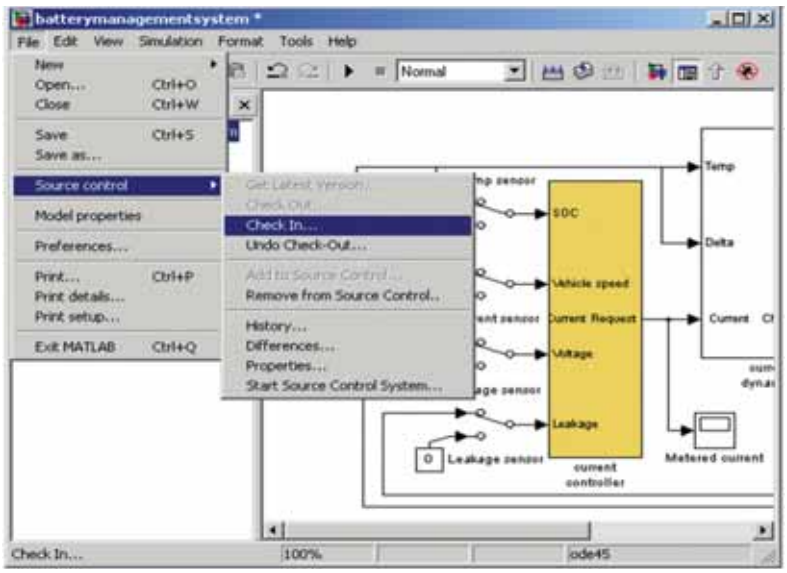
La mayoría de los estándares de software requieren trazabilidad de los requisitos, que posiblemente se hayan originado en otras herramientas de requisitos, a lo largo del desarrollo.

Además, la gestión de la configuración de software (SCM) es necesaria para almacenar, gestionar versiones y recuperar diferentes componentes de desarrollo.

La documentación procesada mediante programas de generación de informes garantiza que la dirección, los clientes y los proveedores verán el modelo. La interfaz SCM se muestra en la figura 10.

llevar a cabo las actividades de la ingeniería de software. La mejora de aspectos puntuales pero dispersos es insuficiente. En este documento se describe la estructura de ingeniería de software necesaria para el diseño de software basado en modelos y la generación de código de producción. Se muestran métodos y herramientas específicos para ilustrar que no se trata simplemente de teoría, sino que ya está disponible y es práctica.

Cada tema presentado es tan extenso, que podría convertirse fácilmente en un documento o libro separado. Animamos a aquellos lectores que deseen más información o que estén interesados en el intercambio de ideas sobre métodos adicionales a que se pongan en contacto con los autores.



Verificación de hardware-in-the-loop (HIL)

Una vez construido el controlador, se pueden realizar en el bucle una serie de pruebas de bucle abierto y bucle cerrado con el modelo de planta de tiempo real.

El primer ejemplo se refiere sólo al procesador y se conoce como prueba "processor-in-the-loop".

El segundo ejemplo utiliza el hardware de ECU realmente construido, y se conoce como "hardware-in-the-loop". En ambos casos, el controlador físico se prueba con el modelo de planta. Mediante una serie de pruebas, probablemente las mismas que se realizaron durante la validación de los requisitos, se debe demostrar que el controlador es aceptable para el cliente.

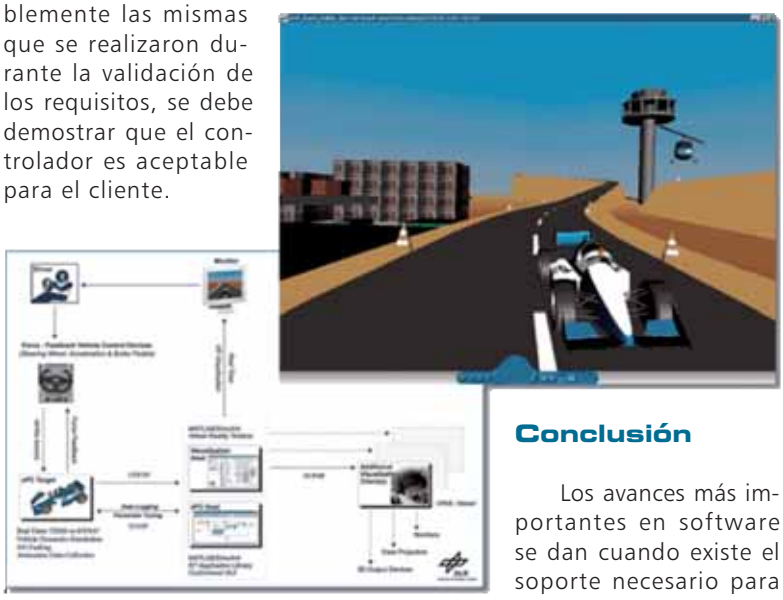


Figura 9. Prueba HIL en la DLR de Alemania con retroalimentación de fuerza

Conclusión

Los avances más importantes en software se dan cuando existe el soporte necesario para

Referencias

[1] www.mathworks.com
[2] www.ieee.org
[3] B. Aldrich, "Using model coverage analysis to improve the controls development process," AIAA 2002
[4] "Software considerations in airborne systems and equipment certification," RTCA/DO-178B, RTCA Inc., diciembre1992
[5] Dr. Michael von der Beeck, ARTIST Industrial Seminar, París, 23.4.2002 http://www.artist-embedded.org/PastEvents/Kickoffs/BMW.pdf
[6] www.decomsys.com
[7] Paul Yih, Jihan Ryu, J. Christian Gerdes, Modification of Vehicle Handling Characteristics via Steer-by-Wire, Dept. of Mech. Eng, Stanford University