Análisis del estado del arte en la depuración de sistemas embebidos

M. Giura, M. Prieto, N. González, L. Sugezky, M. Trujillo, J. Cruz
Departamento de Ingeniería Electrónica – Facultad Regional Buenos Aires
Universidad Tecnológica Nacional
Argentina
mgiura@frba.utn.edu.ar

***Abstract*— El proceso de depuración en sistemas embebidos permite detectar errores en la lógica de la programación. En los últimos años han surgido técnicas para sistematizar este proceso. En particular, el uso de modelos en sistemas embebidos se ha vuelto cada vez más frecuente y necesario para describir su comportamiento, por lo que contar con herramientas que permitan realizar la depuración del sistema resulta imprescindible. En el marco del proyecto de investigación “Desarrollo de software de simulación para la integración con uModelFactory” (EIUTNBA0002436) se lleva adelante el presente trabajo orientado a relevar y caracterizar las herramientas y técnicas existentes para la depuración de sistemas embebidos con el objetivo de implementarlas en el software uModel Factory.**

Keywords—UML; sistemas embebidos; modelo ; depuración

I. INTRODUCCIÓN

El uso de modelos se ha vuelto cada más frecuente ya que permiten describir el software de los sistemas embebidos, ayudan a comprender el sistema y a diseñar con un nivel de abstracción superior al de los lenguajes de programación.

Un modelo es una representación simplificada de un sistema que contempla las propiedades importantes del mismo desde un determinado punto de vista. Además de servir para lograr un conocimiento más profundo del problema, favorecen el intercambio de ideas entre las personas involucradas en el diseño.

La mayoría de los enfoques actuales en el desarrollo de software basado en modelos coinciden en [1,2]:

- Utilizar una representación gráfica del sistema a desarrollar
- Describir el sistema con un cierto grado de abstracción.
- Generar código ejecutable para el sistema embebido partiendo del propio modelo

Como representaciones de dichos modelos se destacan las máquinas de estados finitos (FSM por Finite State Machine) las cuales constituyen una herramienta gráfica que ha sido utilizada tradicionalmente para modelar el comportamiento de sistemas electrónicos e informáticos. Como una amplia extensión al formalismo convencional de estas máquinas surgieron los diagramas de estado (Statecharts) los cuales se convirtieron en parte del estándar UML (Unified Modeling Language) para describir el comportamiento de sistemas o de modelos abstractos.

Los diagramas de estado muestran el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación [3]. Para el pasaje de este objeto por los estados del modelo se analiza su respuesta a eventos y se vincula con sus respuestas y acciones. Estos diagramas normalmente contienen estados, transiciones, eventos, acciones y actividades. Un evento es una ocurrencia que puede causar la transición de un estado a otro del sistema. Esta ocurrencia se puede deber a distintas condiciones como puede ser: una condición que toma el valor de verdadero o falso (expresión booleana); la recepción de una señal o mensaje externo; o el paso de cierto período de tiempo. Una acción es una operación atómica, que no se puede interrumpir por un evento y que se ejecuta hasta su finalización.

En la actualidad existen diferentes enfoques orientados a la depuración de sistemas embebidos, principalmente pueden clasificarse en intrusivos o no intrusivos. A su vez, podemos evaluar los mismos como de tiempo real o de tiempo diferido (Figura 1). En el presente trabajo se analizan diferentes propuestas y herramientas orientadas al proceso de depuración.

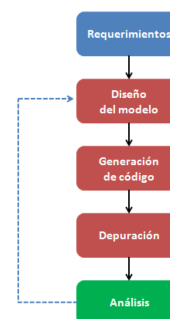

Figura 1. Enfoques utilizados en la actualidad

II. ANALISIS DE TRABAJOS

Evaluación y análisis de los datos provenientes del proceso de depuración.

Bhagyalakshmi y Tembad [4] en su trabajo: *"Trace management, Debug, Analysis and Testing Tools for embedded systems"* centran su trabajo en el rastreo y la gestión de rastreo (tracing and trace management), donde abordan los problemas actuales del proceso de depuración mediante la recopilación de información sobre la ejecución del programa embebido.

En particular, focalizan su trabajo en cuatro cuestiones que deben ser gestionadas después de realizado proceso de rastreo, a saber:

- La heterogeneidad de los formatos de rastreo.
- El almacenamiento de grandes cantidades de datos de rastreo.
- La gestión del análisis del rastreo.
- La visualización de la información.

Partiendo de las ventajas y desventajas de las técnicas actuales de rastreo (datos "crudos" almacenados en archivos y archivos formateados con convenientes estructuras de datos), realizan una propuesta con un nuevo enfoque: utilizar una infraestructura de gestión de trazas para los sistemas embebidos que aborde los problemas de las cuatro cuestiones ya señaladas.

Para la heterogeneidad de formatos se propone un modelo de datos genérico que represente no solo los datos crudos, sino también la meta información relacionada con el rastreo y el análisis de resultados. Con este modelo propuesto, se le da soporte a trazas "multicore" en ventanas separadas y además, se soportan accesos random o procesamientos básicos como un filtrado, se facilita el registro personalizado de mensajes de rastreo para la GUI y se provee una separación entre la ventana de comandos de entrada y la ventana de salida, lo cual facilita el trabajo del usuario con la herramienta.

El sistema propuesto está basado en una arquitectura cliente-servidor y módulos asociados para la realización de la tarea que le dan soporte a las cuestiones señaladas anteriormente.

El proceso de depuración en tiempo real. Técnicas intrusivas y no intrusivas.

Dixon y O'Keeffe [5] en su trabajo *"The advantages of Real-Time trace debug in complex embedded systems"* relatan los contrastes entre la técnica tradicional de depuración denominada Run-control Debug y la más actual Real-time trace, recalcando las claras ventajas de ésta última.

Para ello, señalan los problemas conocidos de la técnica tradicional intrusiva:

- No detecta problemas a máxima velocidad de clock
- No se posee la capacidad de detectar problemas relacionados con eventos externos en tiempo real

En contraste, la técnica de trazado en tiempo real no intrusiva, cada vez con más soporte en los nuevos procesadores, cuenta con las siguientes ventajas:

- No es necesario detener la aplicación a depurar.
- Se cuenta con la facilidad de capturar y almacenar detalles de la temporización (timing) en tiempo real de ejecución.
- Son capaces de capturar y almacenar eventos específicos de interés, descartando el resto, pudiendo observar hacia adelante o hacia atrás, en tiempo real, desde ese punto de interés (trigger condition), qué es lo que está pasando.

La traza captura el juego de instrucciones ejecutadas por el procesador en tiempo real y su temporización (alrededor de la condición de disparo o de todo el programa si no se selecciona una) y lo guarda en un buffer para ser analizadas con posterioridad. También puede capturar los datos utilizados por aquellas instrucciones. La información de temporización permite investigar "timing bugs" que de otro modo serían imposibles.

O'Keeffe y otros [6], presentan en el trabajo *"Real-Time Trace: A Better Way to Debug Embedded Applications"* retoman la técnica conocida como "Real-Time Trace" y la contrastan con técnicas tradicionales.

En primer lugar los autores se refieren a la utilización de la función "printf" como método básico de depuración, remarcando que no es aplicable en casos donde el código deba ser ejecutado en tiempo real, debido a que la impresión o transmisión de mensajes altera los tiempos de ejecución.

En segundo lugar, describen el método de depuración a través de una interfaz JTAG, también llamado "Run-Time Debug", que permite ejecutar el programa paso a paso, insertar breakpoints, visualizar y modificar variables y registros, etc. En este caso, destacan como su principal inconveniente que para poder analizar el estado del programa es necesario pausar la ejecución del mismo, y esto cambia por completo el comportamiento en tiempo real del sistema, incluso pudiendo enmascarar por completo las fallas.

Estas desventajas de las técnicas tradicionales dan lugar a éste método alternativo, donde el sistema embebido cuenta con un módulo de "Real-Time Trace" (RTT) dentro del chip, recopilando información sobre la actividad del procesador. Se pueden registrar secuencias de valores del contador de programa (PC), lecturas o escrituras de memoria con sus valores asociados, lectura o escritura de registros, etc.

Todos esos datos combinados, brindan un panorama completo de la ejecución, para conocer exactamente qué fue lo que sucedió en el sistema embebido hasta que se produjo la falla. Importando esta información en una herramienta de análisis, se puede descubrir rápidamente el origen del problema.

Dentro de las ventajas claves, mencionan que este método permite ver cómo y por qué se arriba a cierto punto a través del historial de instrucciones, capturando los datos de manera no intrusiva, es decir, sin afectar el comportamiento en tiempo real de la aplicación. Además destacan que el "trace" de instrucciones permite el "replay" de ejecución en modo offline y también determinar que porciones de código se ejecutan la mayor cantidad de tiempo.

Como consideraciones a tener en cuenta, remarcan que el módulo de RTT ocupa espacio dentro del chip, consume energía y en general necesita de un módulo de comunicación externo para poder volcar la información registrada. Dado que los casos de uso pueden ser muy diversos, los módulos de RTT son configurables dependiendo de la diversidad de datos que se deban registrar. La enorme cantidad de información generada, debe ser acotarla utilizando filtros que permiten definir condiciones puntuales y regiones de interés dentro del código.

Los autores concluyen que se trata de una herramienta muy útil que permite facilitar la localización de los "bugs" más complejos y así reducir los tiempos de desarrollo, en particular en etapas críticas de un proyecto.

En el trabajo *"Tracing Interrupts in Embedded Software"* los autores Gracioli y Fischmeister [7] analizan diferentes técnicas para el registro de datos provenientes del proceso de depuración, haciendo hincapié en la optimización de la cantidad de datos a almacenar. En particular el análisis hace foco en el registro de las interrupciones. También se señala la importancia de poder realizar en tiempo diferido el proceso de depuración.

El proceso de depuración bajo un sistema operativo embebido

J. Kraft, A. Wall y H. Kienle [8] presentan en el trabajo *"Trace Recording for Embedded Systems: Lessons Learned from Five Industrial Projects"*, el análisis de técnicas de depuración debugueo para sistemas embebidos basados en un sistema operativo. SO. Sin embargo, el análisis es trasladable a un sistema embebido que no cuente con un sistema operativos dado que se basa "que" hay que registrar, "cuando" y "por qué" a pesar de no poseer un administrador de tareas (scheduler) que cambie de tareas, se puede evaluar tanto como para el caso de máquinas de estado corriendo paralelo como en el caso de cambio de estados.

Su primera propuesta consiste en registrar una marca de tiempo (time-stamp) de los eventos elegidos a almacenar y que en principio estos eventos solo sean el cambio de tareas, pudiendo también tomar como posibles eventos a almacenar las IPCs y System Calls llamadas por las tareas.

En el resto del trabajo se analiza el "que" registrar, "cuando" hacerlo y "por qué":

- Identificador de la tarea (el "que")
  Dado que el análisis está hecho para sistemas operativos proponen registrar el PID de la tarea y específicamente el STID (short task ID) para minimizar memoria de almacenamiento.

- Time-stamping (el "cuando")
  Como se dijo anteriormente se propone en el cambio de tarea y/o en el evento elegido loguear el time-stamp y con los fines de ahorrar memoria de almacenamiento se propone (ya sea por medio del RTC si se dispone o por medio del clock en caso que sea constante) así registrar la diferencia entre un evento y el otro. Eligiendo la cantidad de bits que almacenen las temporizaciones más frecuentes y utilizar bits de extensión para el caso que el tiempo del evento sobrepase los bits elegidos.

- Task-switch (el "por qué ")
  Almacenando la información del switcheo de tareas se puede identificar cuando una tarea se bloquea o la misma se suspende o se termina. También se puede ver que tareas son las que tienen mayor prioridad.

Finalmente proponen una interfaz donde se pueden gráficamente los datos almacenados e identificar posibles errores en la ejecución de las tareas.

Los autores M. Desnoyers, M. Dagenais [9] destacan en su trabajo *"Low Disturbance Embedded System Tracing with Linux Trace Toolkit Next Generation"* el uso del framework para rastreo implementado para Linux. En particular, presentan el uso de dicha herramienta, haciendo uso de la misma en la depuración de sistemas embebidos. Como parte del registro, señalan las modificaciones necesarias que implementaron para hacer uso del mismo en función de la detección de interrupciones.

La depuración en embebidos utilizando herramientas de bajo nivel

En el trabajo *"Better Trace for Better Software",* el autor [10] presenta las limitaciones del proceso de depuración por software del sistema en general y presenta una herramienta de ARM y sus mejoras para dicho proceso en la depuración de las unidades del SoC completo.

Si entendemos que las aplicaciones de software para seguimiento y depuración son de corto alcance y que solo pueden dar una visión global de lo que está pasando en el sistema en lugar de una visión detallada del sistema ya que los sistemas modernos están constituidos por múltiples unidades de procesamiento, se necesita contar con una visión más detallada de cada una de ellas. Es por ello, que se presenta el

CoreSigth el cual es una tecnología propuesta por ARM la cual proporciona funcionalidades de seguimiento y depuración con el objetivo de tener información del SoC entero. CoreSight es un conjunto de componentes de harware que pueden ser elegidos e implementados apropiadamente por el diseñador del sistema para extender la depuración. Para el uso de esta herramienta a cada unidad de procesamiento (DSP, CPU, acelerador gráfico, etc) se le asocia un bloque IP con una instrucción (ETM) o programa (PTM), de esta manera se le da seguimiento a la información de las instrucciones ejecutadas o al flujo de un programa.
En orden de mejorar las capacidades del "debug" de la arquitectura CoreSight se agregaron dos nuevos componentes: System Trae Macrocell (STM) y el Trace Mamory Controler (TMC). El STM está diseñado para el seguimiento de las actividades del sistema como ser la velocidad del bus APB, identificar cuando la FIFO está llena, entre otros eventos. El TMC proporciona información sobre los eventos de memoria producidos.

III. SOLUCIONES ACTUALES

Embedded UML Target Debugger [11]

La propuesta de esta solución consiste en una técnica compuesta por un módulo para depuración del lado de la interfaz más un módulo de monitoreo del lado del microprocesador. El módulo de monitoreo está incluido en el sistema operativo de tiempo real (RTOS) que se ejecuta en el microprocesador (Figura 2). El módulo de depuración que se encuentra del lado de la interfaz gráfica recibe los datos de seguimiento recopilados por el módulo de monitoreo en tiempo real y reconstruye e interpreta dicha información por medio del uso de diagramas UML como son diagramas de tiempo o secuencia.

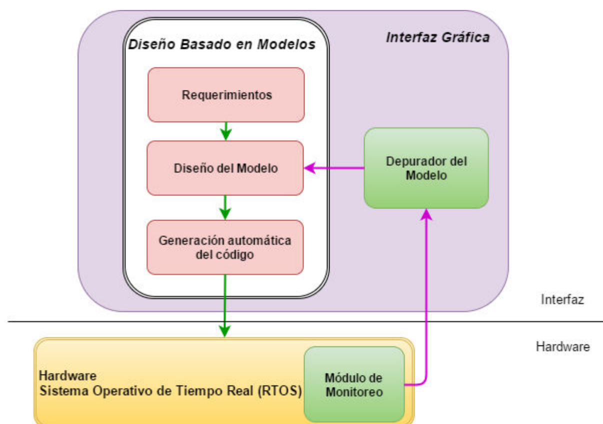

Figura 2. Diagrama de UML Target Debugger

Plantea la ventaja del desarrollo impulsado por modelos ya que el modelo puede ser probado y depurado en las primeras instancias del desarrollo utilizando la simulación. En base al trabajo sobre estos modelos se sabe que este método ayuda a identificar numerosos errores en las primeras fases del proceso. Por ello es necesario poseer en el momento de ejecución en el microprocesador una herramienta de depuración que permita la corrección de esos errores.

Otra desventaja de la simulación es la necesidad de simular todas las interfaces externas que el sistema necesita, por los cual se puede volver una tarea muy compleja. Con el UML Target Debugger el modelo se puede ejecutar en el hardware real y poseer conexión con las interfaces externas reales. A su vez, dado que toda la información que el sistema de monitoreo obtiene del sistema embebido es proporcionado en tiempo real es posible observar en la interfaz gráfica una animación en paralelo que el modelo es ejecutado en el hardware real. Finalmente la información obtenida en la depuración y evaluación hecha en la interfaz puede ser automatizada en el microprocesador por ejemplo con TestConductor.

LieberLieber Embedded Engineer for Enterprise Architect [12]

Este desarrollo propietario ofrece soluciones para sistemas embebidos y permite:

- Generación de código C.
- Generación de código C++ eficiente.
- Depuración del modelo (UML)
- Sincronización del código generado y el modelo.
- Especificación de requerimientos

LieberLieber Embedded Engineer por Enterprise Architect se diseño como parte del conjunto de herramientas creadas para el desarrollo de sistemas embebidos. Ya que es una solución integrada que posibilita realizar de manera eficiente desarrollos basados en modelos.

La versión 2.0 de Embedded Engineer ofrece la generación del código en ANSI-C a partir de las estructuras UML, las máquinas de estados y las actividades del modelo. Como nueva incorporación ofrece la generación del código que lo representa en C++. Esta solución permite generar la documentación asociada al proyecto, la integración de soluciones ya existentes del desarrollo clásico al proyecto, el fácil cumplimiento de normas de codificación ya que al haber algún cambio solo debe ser modificado el generador de código, y no se genera dependencia del proveedor de la solución ya que se obtiene el código el cual puede ser modificado ante algún error o cambio del sistema.
Gracias al módulo de depuración UML que se encuentra incluido en Embedded Engineer, LieberLieber ha resuelto un problema que venía influyendo a las herramientas de generación de código ya que el desarrollo y la depuración se daban en lugares diferentes, mientras el desarrollo se realizaba con el modelo UML la depuración se llevaba a cabo en el microprocesador en C o C++ sin conexión. Con la herramienta de depuración que se ofrece permite la conexión entre la depuración del modelo desarrollado y el código generado que se ejecuta en el hardware.

IV. CONCLUSIONES

En el presente trabajo se evaluaron las propuestas realizadas por diferentes autores como así también herramientas comerciales y bajo licencia open-source, tanto para sistemas embebidos que utilizan un sistema operativo como en el caso “bare metal”.

Dicha información será utilizada la implementación de herramientas de depuración para el software uModel Factory, orientado al diseño y simulación de sistemas embebidos utilizando diagramas de estado.

V. REFERENCIAS

[1] G. Booch, J. Rumbaugh, I. Jacobson. “El Lenguaje Unificado de Modelado”. Addison-Wesley 2nd Edition, 2006.

[2] G. Booch, J. Rumbaugh, I. Jacobson. “El Proceso Unificado de Desarrollo de Software”. Addison-Wesley 1st Edition, 2000.

[3] C. Larman. “UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado”. Prentice-Hall, 2003.

[4] C. Bhagyalakshmi, P. Tembad. “Trace Management, Debug, Analysis and Testing Tool for Embedded Systems”. Research and Reviews. International Journal of Innovative Research in Computer and Communication Engineering. ISSN ONLINE(2320-9801)

[5] B. Dixon, O’Keeffe. “The Advantages of Real Time Tracedebug in Complex Embedded Systems”. Ashling Microsystems. 2013.

[6] J. Campbell, V. Kazantsev, H. O’Keeffe. “Real-time Trace: A Better Way to Debug Embedded Applications”. White paper. Synopsys Inc. 2014.

[7] G. Gracioli, S. Fischmeister. “Tracing Interrupts in Embedded Software”. Journal of Systems Architecture. Volume 58, Issue 9, October 2012.

[8] Kraft J., Wall A., Kienle H. “Trace Recording for Embedded Systems: Lessons Learned from Five Industrial Projects”. Lecture Notes in Computer Science, vol 6418. Springer, Berlin, Heidelberg. 2010.

[9] M. Desnoyers, M. Dagenais. “Low Disturbance Embedded System Tracing with Linux Trace Toolkit Next Generation”. Embedded Linux Conference. 2006

[10] R. Mijat. Better Trace for Better Software. ARM White Paper. 2010. Disponible en
https://www.arm.com/files/pdf/Better_Trace_for_Better_Software_-CoreSight_STM_with_LTTng_-_19th_October_2010.pdf

[11] Embedded UML Target Debugger
Disponible en:
http://www.willert.de/assets/Datenblaetter/DatS-Embedded-UML-Target-Debugger-V9.0-EN-2016.pdf

[12] LieberLieber Embedded Engineer for Enterprise Architect
Disponible en:
http://www.lieberlieber.com/en/embedded-engineer-for-enterprise-architect/