UNIVERSIDAD DE BUENOS AIRES

Facultad de Ingeniería

Carrera de Especialización en Sistemas Embebidos

Desarrollo de Firmware y Software para programar la CIAA en lenguaje JAVA con aplicación en entornos Industriales

Alumno: Ing. Eric Nicolás Pernia.

Director: MSc. Ing. Félix Gustavo Safar.

Buenos Aires, Argentina.

Presentación:
Noviembre de 2015

RESUMEN

El propósito de este Trabajo Final es la incorporación de nuevas tecnologías en ambientes industriales mediante el desarrollo de arquitecturas de sistemas embebidos novedosas. En particular, permitir crear aplicaciones Real-Time para entornos industriales, utilizando un lenguaje de programación orientado a objetos (POO), sobre la Computadora Industrial Abierta Argentina (CIAA). Además, se espera acercar a programadores informáticos a la disciplina de programación de sistemas embebidos, permitiéndoles aplicar técnicas avanzadas de programación.

Para llevarlo a cabo se ha escogido Java como lenguaje POO y Icecaptools. Icecaptools es un plug-in de Eclipse realizado por Stephan Erbs Korsholm, que convierte al Eclipse en un IDE para la programación en lenguaje Java y permite compilar el programa de usuario para HVM. HVM es una máquina virtual de Java, libre, escrita en lenguaje C, diseñada para correr directamente sobre el hardware. Además, HVM cumple con la especificación Safety-Critical Java (SCJ), permitiendo realizar aplicaciones críticas Real-Time.

Se incluye en este trabajo la implementación y validación de un ambiente de Firmware y Software, basado en Icecaptools y HVM, para programar la Computadora Industrial Abierta Argentina (CIAA) en lenguaje Java, para aplicaciones industriales en tiempo real. Este consiste en:

- La realización del *porting* de HVM para que corra sobre el microcontrolador NXP LPC4337 que contienen las plataformas CIAA-NXP y EDU-CIAA-NXP.
- Un diseño e implementación de una HAl con API sencilla para permitir controlar el Hardware desde una aplicación Java.
- La realización del *porting* de la capa SCJ de HVM para que corra sobre el microcontrolador NXP LPC4337 permitiendo aplicaciones SCJ.
- El desarrollo de la integración en Icecaptools del *porting* de HVM para completar el IDE de Java SCJ sobre la CIAA.

Para validar el IDE desarrollado se presentan:

- Ejemplos de aplicaciones Java utilizando periféricos de la CIAA y EDU-CIAA.
- Una aplicación Real-Time (SCJ) de referencia para demostrar el funcionamiento real-time del sistema.

Agradecimientos

Agradezco a mi familia que siempre me apoya en todo lo que me propongo y ayuda para que y sea capaz de realizarlo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN GENERAL

1.1. Marco temático Programación Orientada a Objetos en Sistemas embebidos para aplicaciones industriales

Marco temático: Programación Orientada a Objetos en Sistemas embebidos para aplicaciones industriales.

1.2. Plataforma CIAA

Plataforma CIAA-NXP.
Plataforma EDU-CIAA-NXP.

1.3. Lenguaje Java

Se elije Java debido a que es uno de lenguajes de POO más utilizados en la actualidad. Es un lenguaje moderno, con manejo de memoria automático y sintaxis similar a C++. A diferencia de otros, en lugar de compilarse para la arquitectura del controlador objetivo, se compila a un lenguaje similar al assembler (bytecodes) que se ejecuta mediante una Máquina Virtual, o VM, por sus siglas en inglés (Virtual Machine). Esta máquina virtual corre habitualmente sobre un sistema operativo.

1.4. Máquinas Virtuales de Java para sistemas embebidos

Máquinas Virtuales de Java para sistemas embebidos.

1.5. Especificación SCJ

Especificación SCJ.

1.6. Justificación

Con la creciente complejidad de las aplicaciones a realizar sobre sistemas embebidos en entornos industriales, y el aumento de las capacidades de memoria y procesamiento de los mismos, se desea poder aplicar técnicas avanzadas de programación para realizar programas fácilmente mantenibles, extensibles y reconfigurables. El paradigma de Programación Orientada a Objetos (POO) cumple con estos requisitos.

Para aplicaciones industriales es requerimiento fundamental cumplir con especificaciones de tiempo real. Es por esto que se debe elegir un lenguaje POO que soporte de manejo de threads real-time. Ejemplos de aplicaciones con estos requerimientos son, control a lazo cerrado, etc.

1.7. Objetivos

Los objetivos del presente trabajo final son:

- Realizar el *porting* de HVM para que corra sobre el microcontrolador NXP LPC4337 que contienen las plataformas CIAA-NXP y EDU-CIAA-NXP para permitir correr aplicaciones Java sobre la CIAA.
- Diseñar e implementar una HAL para controlar el Hardware desde una aplicación Java mediante una API establecida.
- Realizar el *porting* de la capa SCJ de HVM para que corra sobre el microcontrolador NXP LPC4337 permitiendo aplicaciones Java SCJ sobre la CIAA.
- Integrar en Icecaptools el *porting* de HVM para completar el IDE de Java SCJ sobre la CIAA.

Capítulo 2

DISEÑO

2.1. Elección de la VM de Java, HVM

HVM[1] son las siglas de Hardware Virtual Machine. Es una máquina virtual (VM) de Java libre, diseñada para ejecutarse bare metal sobre microcontroladores y portable a diferentes arquitecturas. Su implementación de referencia fue realizada por Stephan Erbs Korsholm (Icelab, Dinamarca) para su tesis doctoral. Esta VM cumple con la especificación Safety-Critical Java (SCJ) Level 0, 1 y 2 permitiendo realizar aplicaciones en tiempo real para sistemas críticos. Provee tres formas de acceso al hardware:

- Variables Bindeadas.
- Hardware Objects.
- Métodos nativos.

La última forma, métodos nativos, se elige como alternativa para proveer al programa de usuario en lenguaje Java el acceso al hardware.

2.2. IDE Icecaptools

Icecaptools se distribuye como un plugin de Eclipse realizado por Stephan Erbs Korsholm, que convierte al Eclipse en un IDE para la programación en lenguaje Java y permite compilar el programa de usuario para HVM. Funciona realizando una traducción de un programa de usuario escrito en lenguaje Java, a un programa en lenguaje C que incluye el código de dicho programa y el código C generado de HVM. De esta manera, logra portabilidad entre diferentes microcontroladores y permite integración con programas escritos previamente en lenguaje C, como por ejemplo, el firmware de la CIAA. Requiere un toolchain de lenguaje C, para el microcontrolador objetivo, que permita compilar el código, generando un binario ejecutable, y su posterior descarga a dicho dispositivo. En la figura [2.1] se incluye un esquema gráfico de su funcionamiento.

2.3. Porting de HVM e icecaptools a una nueva arquitecura

Para portar el Firmware HVM a una nueva arquitectura deben realizarse los siguientes pasos:

- Conseguir un ambiente de desarrollo en lenguaje C para la arquitectura.
- Estandarizar los tamaños de tipos de datos.

[1] Significa que se ejecuta directamente sobre el hardware, sin necesidad de un sistema operativo.

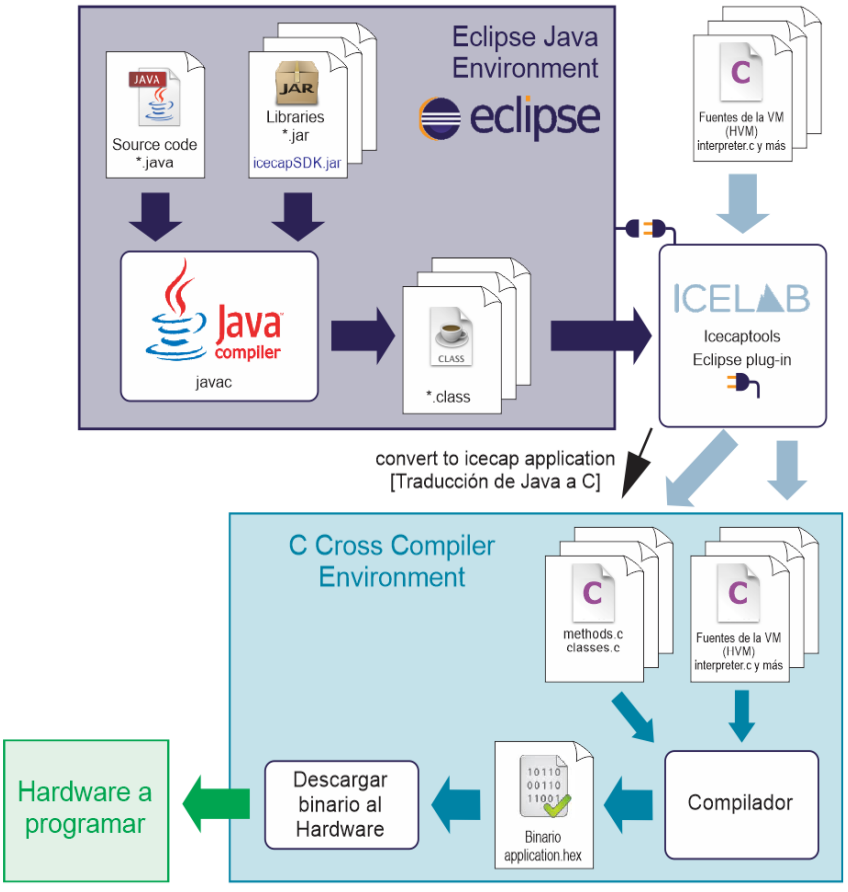

Figura 2.1: Esquema de funcionamiento de Icecaptools.

- Implementar una HAL para la arquitectura.
- Implementar el acceso al hardware.

Para facilitar la inclusión de nuevas arquitecturas a icecaptools y automatizar tareas habituales, que en caso contrario debería realizar el usuario para cada nueva aplicación, se propone:

- Adaptar Icecaptools para la arquitectura ARM mediante su extensión a través de un plug-in que llamaremos HvmForArmCortexM.
- La programación de un segundo plug-in más específico llamado HVM4CIAA.

Finalmente, en la figura 2 se gráfica la solución propuesta para lograr utilizar icecaptools y HVM con la CIAA.
FIGURAAAAAAAAAAA

2.4. Diseño de HAL para manejo de periféricos

Diseño de una HAL para manejo de periféricos con una API sencilla.

2.5. Diseño del IDE

Diseño del IDE.

Capítulo 3

IMPLEMENTACIÓN

3.1. Paradigmas y lenguajes de programación utilizados

Paradigmas y lenguajes de programación utilizados.

3.2. Entorno Eclipse

Entorno Eclipse.

3.3. Port de HVM (Java básico) para CIAA-NXP

Port de HVM (Java básico) para CIAA-NXP.

3.4. Implementación de una HAL para manejo de periféricos CIAA-NXP

Implementación de HAL para manejo de periféricos CIAA-NXP.

3.5. Ejemplos de uso de periféricos desarrollados

Ejemplos de uso de periféricos desarrollados.

3.6. Port de HVM (Java SCJ) para CIAA-NXP

Port de HVM (Java SCJ) para CIAA-NXP.

3.7. Ejemplo de aplicación de tiempo real desarrollado

Ejemplo de aplicación de tiempo real desarrollado.

3.8. Plugins desarrollados

Plugins desarrollados.

3.9. Implementación del IDE

Implementación del IDE.

Capítulo 4

RESULTADOS

4.1. Acerca del IDE obtenido

Acerca del IDE obtenido.

4.2. Comparaciones entre Java y C

4.3. Aplicación de referencia SCJ

Aplicación de referencia SCJ.

Se ha completado el porting de HVM e Icecaptools para la CIAA y se ha creado un conjunto de clases para manejo básico del hardware desde el programa en lenguaje Java. De esta manera puede crearse una aplicación en lenguaje Java para las plataformas CIAA-NXP y EDU-CIAA-NXP, permitiendo controlar sus periféricos.

Enumerar las clases desarrolladas

Como subproducto, se obtiene además un nuevo módulo de Firmware para la CIAA nombrado sAPI (simple API) que permite encapsular los periféricos y con una interfaz muy sencilla pensada para el programador con perfil informático no experto en Sistemas embebidos.

Enumerar el hardware que permite atacar

Se ha logrado portar además portar la capa de HVM que implementa la especificación SCJ obteniendo una plataforma de firmware y software que permite la programación de aplicaciones Real-Time para la CIAA en lenguaje Java.

Para probar el correcto funcionamiento de las herramientas, ha realizado una aplicación de referencia con especificadores de tiempo real y se ha comprobado su correcto funcionamiento.

Poner diagramas temporales de mediciones en ejecución?

Capítulo 5

CONCLUSIONES Y TRABAJO A FUTURO

5.1. Conclusiones

En este trabajo se ha logrado llevar a cabo un diseño completo de un micro PLC, incluyendo especificaciones de Hardware, Software base (sistema operativo, drivers, implementaciones de funciones standard, etc.), y formato que debe respetar el resultado de la compilación de programas definidos en los lenguajes de la norma IEC61131-3, para poder ejecutarse en equipos que se correspondan con el modelo diseñado.

Por otra parte, se definen en detalle los siguientes aspectos salientes para la implementación de un entorno de edición de programas para PLC: comportamiento esperado de la interfaz de usuario, y requisitos que debe cumplir un modelo computacional de los conceptos y lenguajes de programación descriptos en la norma citada.

Se buscó generar especificaciones que no dependieran de determinados componentes o fabricantes para el Hardware, ni de un Sistema Operativo específico para el entorno de edición de programas. Permitiendo así, construir un PLC de bajo costo de Hardware, de forma tal que pueda ser utilizado en ámbitos académicos y que, al mismo tiempo, permitan la edición de programas en forma ágil y cómoda, en particular para los lenguajes gráficos (Ladder y FBD) incluidos en la norma.

Como parte del trabajo, se desarrolló una implementación que cumple con las pautas y definiciones incluidas en las distintas partes del diseño, utilizando componentes de Hardware económicos y fácilmente obtenibles; y un entorno de programación (para desarrollar el entorno de edición de programas) que puede obtenerse desde Internet en forma gratuita.

Por lo tanto, se llega a la conclusión que los objetivos planteados al comenzar el trabajo han sido alcanzados satisfactoriamente.

La definición del diseño fue llevada a cabo en conjunto, y en muchos casos influida por la construcción de implementaciones de referencia. Se culmina este Trabajo Final con la férrea convicción que esta manera de realizarlo, permitió llegar a diseños más acabados, brindando pautas concretas y realmente útiles para el desarrollo de futuras implementaciones.

Por otro lado, el trabajo resultó sumamente arduo en varios aspectos; entre los se destacan:

- La gran cantidad de iteraciones y versiones preliminares que culminaron en la especificación presentada del modelo computacional referente a los conceptos y lenguajes de programación incluidos en la norma IEC61131-3.
- La complejidad inherente a algunos detalles de la implementación de la interfaz de usuario para lenguajes gráficos, en particular el recálculo de las posiciones ante distintas acciones que puede llevar a cabo el usuario en un segmento de programa en lenguaje Ladder.

Considerando este último aspecto, si bien algunas características del Framework Morphic ayudaron en la implementación de referencia de la GUI, la poca documentación existente sobre esta herramienta, hizo que en muchas oportunidades fuera necesario revisar el código de sus clases constituyentes, recurriendo a las capacidades avanzadas de depuración de Pharo, que permiten examinar y modificar el código de su propia interfaz en tiempo de ejecución.

Cabe resaltar que la utilización de los conceptos principales del paradigma de programación orientada a objetos, facilitó el desarrollo del diseño y su implementación de referencia. En particular, contribuyó a manejar la gran complejidad del modelo computacional de una Configuración de Software y de los programas incluidos, permitiendo la obtención de componentes bien definidos e interfaces claras entre los mismos, logrando que componentes de distintas características (como por ejemplo, los segmentos que corresponden a distintos lenguajes de programación IEC61131-3) puedan ensamblarse en forma sencilla dentro de un mismo diseño general.

Además, el desarrollo de un modelo computacional adecuado referente a los conceptos de distintos lenguajes de programación, involucró investigar y adquirir conocimientos avanzados de programación, por ejemplo; tipos de datos, definiciones, declaraciones, parseo, compilación, etc.

Otra herramienta clave para llevar a cabo esta labor, fue la experiencia adquirida en programación de Microcontroladores, siendo, el autor de este trabajo, alumno y auxiliar académico en las materias correspondientes, hecho que contribuyó en la investigación de uno de los Microcontroladores (y su IDE de desarrollo) más novedosos disponible en el mercado, utilizado para la implementación de referencia del Hardware.

Concluyendo, la realización de este Trabajo Final demandó la articulación entre los conocimientos adquiridos a lo largo de la carrera y los aprendizajes incorporados, habilitando su puesta en práctica.

5.2. Trabajo a futuro

Como labor a futuro, pueden realizarse las siguientes tareas:

- Modelado del lenguaje ST (Structured Text) y de los elementos SFC (Secuencial Function Charts), definidos en la norma IEC 61131-3.
- Modelado de la opción Retenibilidad de variables como indica la norma IEC61131-3. Este tipo de variables se mantienen en memoria del PLC incluso luego de la pérdida de alimentación del equipo PLC.
- Diseño de Entradas y Salidas Analógicas.
- Diseño de módulos de comunicaciones industriales compatibles, siguiendo las pautas expuestas en la norma IEC 61131-5.
- Diseño de un simulador de PLC para ensayar el programa de usuario sin necesidad de poseer el Hardware específico.
- Diseño de un sistema de depuración en caliente y observación de variables internas del PLC desde la computadora, mientras el programa de usuario se ejecuta en el PLC.

REFERENCIA BIBLIOGRÁFICA

1. Barry, R. (2011).*Using the FreeRTOS TM Real Time Kernel NXP LPC17xx Edition ed. 3*: Real Time Engineers Ltd.

2. Bergel, A., Cassou, D., Ducasse, S., y Laval, J. (2013). *Deep into Pharo*. Switzerland: Square Bracket Associates. Consultado en 10-10-2013 en http://pharobooks.gforge.inria.fr/PharoByExampleTwo-Eng/latest/PBE2.pdf.

3. Black, A., Ducasse, S., Nierstrasz, O. y Pollet, D. (2009). *Pharo por Ejemplo*. Switzerland: Square Bracket Associates. Consultado en 10-10-2013 en http://pharobyexample.org/es/PBE1-sp.pdf.

4. Claus Gittinger Development & Consulting (1996). *Stream classes*. Consultado en 10-10-2013 en http://live.exept.de/doc/online/english/overview/basicClasses/streams.html.

5. Ducasse, S. (2008). *[Pharo-project] (Foro) - Collapsing panes*. Consultado en 10-10-2013 en http://lists.gforge.inria.fr/pipermail/pharo-project/2011-January/040520.html.

6. Gamma, E., Helm, R., Johnson, R. Vlissides, J. (1995). *Patrones de Diseño: Elementos de software orientado a objetos reutilizable*. Madrid: PEARSON - Addison Wesley.

7. Gemtalk Systems (2011). *Pharo the collaborActibe book*. Consultado en 10-10-2013 en http://pharo.gemtalksystems.com/book/.

8. Giner, G., Rafael, J. (2008). "*Programación estructurada en C ed. 1*". Pearson Prentice Hall.

9. Gough, B. (2005). An Introduction to GCC. Network Theory Ltd.

10. IEC (2003). *IEC 61131-3 Programmable controllers - Part 3: Programming languages ed2.0*. International Electrotechnical Commission.

11. Juarez, J. (2012). *UNQ - Apuntes de cátedra: Sistemas Digitales*. Consultado en 10-10-2013 en http://iaci.unq.edu.ar/materias/sistemas_digitales/index.htm.

12. Kosik, M. (2008). *Pluggable Morphs Demo*. Consultado en 10-10-2013 en http://wiki.squeak.org/squeak/2962.

13. Lewis, D. (2008). *OSProcess*. Consultado en 10-10-2013 en http://wiki.squeak.org/squeak/708.

14. LSE (2012). *UBA - Apuntes de cátedra: Sistemas embebidos*. Consultado en 10-10-2013 en http://laboratorios.fi.uba.ar/lse/.

15. Moreno Gomez, J. (2009). *What is the difference between Sinking and Sourcing Input Configuration - PLC?*. Consultado en 10-10-2013 en http://reliability-maintenance.blogspot.com.ar/2009/07/what-is-difference-between-sinking-and.html.

16. National Instruments (2011). *Digital I/O Sinking and Sourcing*. Consultado en 10-10-2013 en http://www.ni.com/white-paper/3291/en.

17. Radioaficionados (2010). *Protección contra inversiones de polaridad*. Consultado en 10-10-2013 en http://www.radioelectronica.es/radioaficionados/19-inversion-polaridad.

18. Ritchie, D. (1993). *The Development of the C Language*. Consultado en 10-10-2013 en http://cm.bell-labs.com/cm/cs/who/dmr/chist.html.

19. Siemens Industry Online Support (2013). *¿Qué significan los términos "sumidero"(alemán: "P-schaltend") y "fuente"(alemán: "M-schaltend") en los módulos digitales de SIMATIC?*. Consultado en 10-10-2013 en http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&objId=42616517&load=treecontent&lang=es&siteid=cseus&aktprim=0&objaction=csview&extranet=standard&viewreg=WW.

20. Stallman, R., Pesch, R., Shebs, S., et al. (2011). *Debugging with GDB*. Free Software Foundation.

21. Stallman, R. (2001). *Using and Porting the GNU Compiler Collection*. Free Software Foundation. Consultado en 10-10-2013 en http://gcc.gnu.org/onlinedocs/gcc-2.95.3/gcc.html.

22. Szirty (2013). *PLC programozás sokféleképpen (La programación de PLC de muchas maneras)*. Consultado en 10/10/2013 en http://szirty.taviroda.com/lang/index.html.

23. Wikipedia (2013). *Analizador sintáctico*. Consultado en 10-10-2013 en http://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico.

24. Wikipedia (2013). *Drop-down list*. Consultado en 10-10-2013 en http://en.wikipedia.org/wiki/Drop-down_list.

25. Wikipedia (2013). *Framework*. Consultado en 10-10-2013 en http://es.wikipedia.org/w/index.php?title=Framework&section=10.

26. Wikipedia (2013). *Programación orientada a objetos*. Consultado en 10-10-2013 en http://es.wikipedia.org/wiki/Programacion_orientada_a_objetos.

27. Wikipedia (2013). *Smalltalk*. Consultado en 10-10-2013 en http://es.wikipedia.org/wiki/Smalltalk.

28. Wikipedia (2013). *Tubería (informática)*. Consultado en 10-10-2013 en http://es.wikipedia.org/w/index.php?title=Tuber%C3%ADa_%28inform%C3%A1tica%29.