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 (en adelante 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 HVM[1], que es un entorno de ejecución de *Safety Critical Java*[2] (SCJ)[2], de código abierto, diseñado para plataformas embebidas de bajos recursos. Este trabajo consiste entonces, en la implementación y validación de un ambiente de Firmware y Software, basado en HVM, para programar las plataformas CIAA-NXP y EDU-CIAA-NXP en lenguaje Java SCJ.

En particular, la implementación consiste en:

- La realización del *port* de la máquina virtual de HVM para que corra sobre el microcontrolador NXP LPC4337, que contienen las plataformas CIAA-NXP y EDU-CIAA-NXP, permitiendo la programación de aplicaciones Java.
- Un diseño e implementación de una API[3] sencilla para permitir controlar el Hardware desde una aplicación Java, que funciona además, como HAL[4].
- El *port* de la capa SCJ de la máquina virtual de HVM, para permitir desarrollar aplicaciones Java SCJ.
- La integración del *port* para la CIAA al IDE de HVM, para completar un IDE de Java SCJ sobre la CIAA.

Para validar el IDE desarrollado se presentan:

- Ejemplos de aplicaciones Java utilizando periféricos de la CIAA-NXP y EDU-CIAA-NXP mediante la API desarrollada.
- Un ejemplo de aplicación Java SCJ utilizando el concepto de Proceso SCJ para demostrar el funcionamiento del cambio de contexto.
- Otro ejemplo de aplicación Java SCJ utilizando el concepto de Planificador SCJ.
- Una aplicación SCJ completa.

[1] HVM desarrollado por Stephan Erbs Korsholm.

[2] La especificación *Safety Critica Java* es una extensión a la especificación RTSJ, una especificación de java para aplicaciones en tiempo real.

[3] *Application Programming Interface*, es decir, una interfaz de programación de aplicaciones.

[4] *Hardware Abstraction Layer*, significa: capa de abstracción de hardware.

Finalmente, se obtiene de este Trabajo Final un entorno de desarrollo para aplicaciones Java SCJ sobre las plataformas CIAA-NXP y EDU-CIAA-NXP, que además de ser software libre, cubre las necesidades planteadas, tanto al ofrecer programación orientada a objetos, así como funcionalidades de tiempo real para entornos industriales, sobre sistemas embebidos.

Agradecimientos

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN GENERAL - (8 pag)

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

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. Un ejemplo de aplicaciones con estos requerimientos son las de control a lazo cerrado. En estas aplicaciones es necesario garantizar una taza de muestreo periódica uniforme para poder aplicar la teoría de control automático.

En las siguientes secciones se expondrán las distintas tecnologías que constituyen esta temática.

1.1.1. Lenguajes de POO para sistemas embebidos

En la actualidad existen varios desarrollos de lenguajes de programación orientado a objetos para sistemas embebidos, entre ellos podemos citar:

- C++.
- Java.
- MicroPhyton.

1.1.2. Plataforma CIAA

CIAA-NXP Plataforma CIAA-NXP.

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

1.2. Justificación

Acá debería poner por que se elije Java y HVM. Arrancamos con Java, que se elije porque es superador a C++ y existe la especificación RTSJ.

Luego se descubre HVM que implementa SCJ.

1.2.1. Lenguaje de POO Java

El lenguje Java fue desarrollado originalmente por James Gosling de Sun Microsystems y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems (actualmente fue adquirida por la compañía Oracle).

Las características principales del mismo son:

- Lenguaje de programación de propósito general.
- Orientado a objetos.
- Independiente de la máquina.
- Seguro para trabajar en red.
- Potente para sustituir código nativo.
- Con comprobación estricta de tipos.
- Manejo de memoria automático mediante Recolector de Basura.
- Sin punteros, utiliza únicamente referencias a objetos.
- Permite programación concurrente de forma estándar.

Para lograr la independencia de la máquina posee la característica de ser un lenguaje compilado e interpretado. Todo programa en Java, se compila primero a un lenguaje similar a un *assembler* genérico basando en pila (*bytecodes*), que luego es interpretado por una máquina virtual de Java (JVM) dependiente de la plataforma.

La JVM es habitualmente un programa que corre sobre un sistema operativo, sin embargo, existen implementaciones de la JVM que corren directamente sobre el hardware (*bare-metal*) y procesadores capaces de ejecutar *bytecodes* de Java directamente, por ejemplo, el microcontrolador ARM926EJ-S.

Debido a estas características y la existencia de la especificación RTSJ que contempla aplicaciones *Real-Time* (se desarrolla en la sección [1.2.2]), se ha elegido Java como lenguaje POO para la programación de la CIAA. Además, Java es uno de lenguajes de POO más utilizados en la actualidad a nivel mundial por programadores informáticos.

1.2.2. Especificaciones RTSJ y SCJ

Especificación SCJ.

Executive Summary

This Safety-Critical Java Specification (JSR-302), based on the Real-Time Specification for Java (JSR-1), defines a set of Java services that are designed to be usable by applications requiring some level of safety certification. The specification is targeted to a wide variety of very demanding certification paradigms such as the safety-critical requirements of DO-178B, Level A.

This specification presents a set of Java classes providing for safety-critical application startup, concurrency, scheduling, synchronization, input/output, memory management, timer management,

interrupt processing, native interfaces, and exceptions. To enhance the certifiability of applications constructed to conform to this specification, this specification also presents a set of annotations that can be used to permit static checking for applications to guarantee that the application exhibits certain safety properties.

To enhance the portability of safety-critical applications across different implemen- tations of this specification, this specification also lists a minimal set of Java libraries that must be provided by conforming implementations.

1.2.3. Elección de HVM

Hardware near Virtual Machine (HVM)[1] es un entorno de ejecución de *Safety Critical* Java (SCJ) nivel 1 y 2, código abierto diseñado par plataformas embebidas de bajos recursos.

Corre directamente sobre el hardware sin necesidad de un sistema operativo (*bare-metall*). Su diseño y documentación facilita la portabilidad a nuevas arquitecturas.

Se compone de las siguientes partes:

- Icecaptools
- HVM SDK

Para desarrollar aplicaciones sobre HVM se utiliza Icecaptools, el cual es un *plugin* de Eclipse, que convierte a Eclipse en un IDE para la programación en lenguaje Java para HVM sobre sistemas embebidos.

Icecaptools genera código C a partir de la aplicación Java de usuario para correr sobre la máquina virtual de Java de HVM, así como los proprios archivos que implementan a la máquina virtual.

HVM SDK - El Software Development Kit de HVM que incluye las clases que implementan SCJ.

Por que HVM y no otra VM de Java.

1.3. Objetivos

Los objetivos del presente trabajo final son:

- Realizar el *port* de la máquina virtual de HVM para que corra sobre las plataformas CIAA-NXP y EDU-CIAA-NXP, permitiendo la programación de aplicaciones Java.
- Diseñar e implementar una API sencilla para permitir controlar periféricos del micronctrolador desde una aplicación Java.
- Llevar a cabo el *port* de la capa SCJ de la máquina virtual de HVM, para permitir desarrollar aplicaciones Java SCJ.
- La integración del *port* para la CIAA al IDE de HVM, para completar un IDE de Java SCJ sobre la CIAA.

[1] HVM desarrollado por Stephan Erbs Korsholm.

Capítulo 2

DISEÑO - (8 pag)

En particular, la implementación consiste en:

- La realización del *port* de la máquina virtual de HVM para que corra sobre el microcontrolador NXP LPC4337, que contienen las plataformas CIAA-NXP y EDU-CIAA-NXP, permitiendo la programación de aplicaciones Java.
- Un diseño e implementación de una API[1] sencilla para permitir controlar el Hardware desde una aplicación Java, que funciona además, como HAL[2].
- El *port* de la capa SCJ de la máquina virtual de HVM, para permitir desarrollar aplicaciones Java SCJ.
- La integración del *port* para la CIAA al IDE de HVM, para completar un IDE de Java SCJ sobre la CIAA.

2.1. 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.

HVM provee tres formas de acceso al hardware:

- **Variables Bindeadas.** Es una variable que puedo utilizar en lenguaje Java y tiene correspondencia directa con una variable en otro lenguaje (en este caso particular, lenguaje C).
- ***Hardware Objects.*** Es una abstracción que permite acceder a registros del microcontrolador mapeados en memoria para manipularlos desde el programa en lenguaje Java. De esta forma se puede crear una biblioteca completa dependiente del microcontrolador que maneje un periférico a nivel de registros directamente en Java.
- **Métodos nativos.** Esta alternativa permite utilizar funciones realizadas en otro lenguaje como métodos en lenguaje Java. De esta manera permite ejecutar código *legacy* dando la

[1] *Application Programming Interface*, es decir, una interfaz de programación de aplicaciones.
[2] *Hardware Abstraction Layer*, significa: capa de abstracción de hardware.

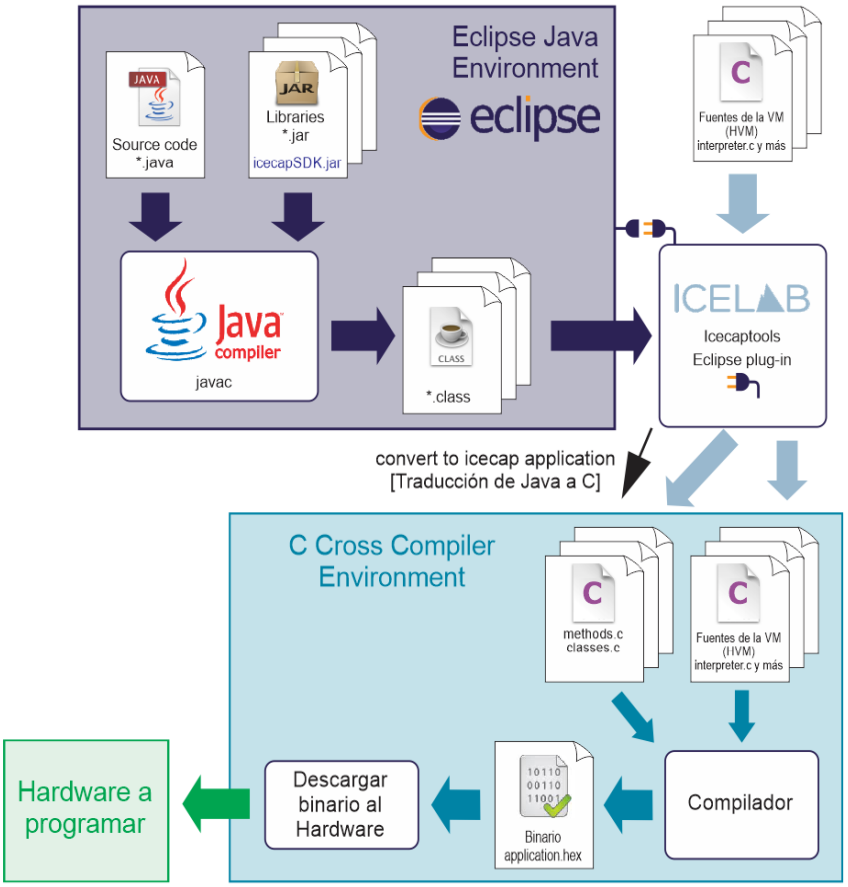

Figura 2.1: Esquema de funcionamiento de Icecaptools.

posibilidad de utilizar bibliotecas completas realizadas en otro lenguaje. Para conectar un método con una función en lenguaje C, deben respetarse ciertan convenciones de nombres y de pasajes de parámetros en las funciones realizadas para que el compilador de Java puede asociarlas.

Se elije métodos nativos como alternativa para proveer al programa de usuario en lenguaje Java el acceso a los periféricos del microcontrolador. Esto es porque ya existen bibliotecas completas de manejo de periféricos y además *stacks* y *file systems* entre otras.

2.2. 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.
- 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

2.3. Diseño de HAL para manejo de periféricos

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

2.4. Diseño del IDE

Diseño del IDE.

Capítulo 3

IMPLEMENTACIÓN - (14 pag)

En particular, la implementación consiste en:

- La realización del *port* de la máquina virtual de HVM para que corra sobre el microcontrolador NXP LPC4337, que contienen las plataformas CIAA-NXP y EDU-CIAA-NXP, permitiendo la programación de aplicaciones Java.
- Un diseño e implementación de una API[1] sencilla para permitir controlar el Hardware desde una aplicación Java, que funciona además, como HAL[2].
- El *port* de la capa SCJ de la máquina virtual de HVM, para permitir desarrollar aplicaciones Java SCJ.
- La integración del *port* para la CIAA al IDE de HVM, para completar un IDE de Java SCJ sobre la CIAA.

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 API para manejo de periféricos

Implementación de API para manejo de periféricos CIAA-NXP.
sAPI

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

Enumerar el hardware que permite atacar

[1] *Application Programming Interface*, es decir, una interfaz de programación de aplicaciones.
[2] *Hardware Abstraction Layer*, significa: capa de abstracción de hardware.

3.5. Port de HVM SCJ para CIAA-NXP

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

3.6. Implementación del IDE

Implementación del IDE.

3.6.1. Plugins desarrollados

Plugins desarrollados.

Capítulo 4

RESULTADOS - (8 pag)

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.

- Ejemplos de aplicaciones Java utilizando periféricos de la CIAA-NXP y EDU-CIAA-NXP mediante la API desarrollada.
- Un ejemplo de aplicación Java SCJ utilizando el concepto de Proceso SCJ para demostrar el funcionamiento del cambio de contexto.
- Otro ejemplo de aplicación Java SCJ utilizando el concepto de Planificador SCJ.
- Una aplicación SCJ completa.

Capítulo 5

CONCLUSIONES Y TRABAJO A FUTURO

5.1. Conclusiones

En el presente Trabajo Final se ha logrado obtener un entorno de desarrollo para aplicaciones Java SCJ sobre las plataformas CIAA-NXP y EDU-CIAA-NXP, que además de ser software libre, cubre las necesidades planteadas, tanto al ofrecer programación orientada a objetos, así como funcionalidades de tiempo real para entornos industriales, sobre sistemas embebidos.

Como subproducto, se obtiene además una API sencilla para el manejo de periféricos de las plataformas CIAA-NXP y EDU-CIAA-NXP, que puede utilizarse en aplicaciones Java, o directamente en lenguaje C, debido a su diseño como módulo de Firmware de la CIAA. Se ha tenido especial cuidado en el diseño de esta API para que la misma sea lo más genérica posible logrando que además se comporte como una HAL.

El desarrollo de este Trabajo Final demandó la articulación de conocimientos adquiridos a lo largo de la Carrera de Especialización en Sistemas Embebidos, en particular, las asignaturas:

- **Arquitectura de microprocesadores**. Se utilizaron de esta asignatura los conocimientos adquiridos sobre la arquitectura ARM Cortex M necesarios para implementar en lenguaje *assembler* las funciones que realizan el cambio de contexto, necesarias para que funcione el concepto de Proceso SCJ.

- **Programación de microprocesadores**. De esta asignatura se aprovecha la experiencia sobre lenguaje C para microcontroladores de 32 bits y el manejo de sus periféricos. Fue de especial importancia, debido a que, a excepción de unas pocas, todas las funciones para portar HVM a la CIAA debían realizarse en lenguaje C. Este lenguaje también se utilizó en la creación de la API para el manejo de periféricos.

- **Ingeniería de software en sistemas embebidos**. Se aplican de la misma las metodologías de trabajo, provenientes de la ingeniería de software, que aportan calidad y eficiencia la desarrollo. En particular, diseño iterativo y manejo repositorios de software, diseño modular y en capas.

- **Gestión de Proyectos en Ingeniería**. Durante esta se desarrolló el Plan de Proyecto del Trabajo Final, permitiendo desde un principio tener una clara planificación del trabajo a realizar.

- **Sistemas Operativos de Proposito General**. Se aprovechan los conocimientos adquiridos sobre Linux para probar las herramientas desarrolladas sobre este sistema operativo.

- **Sistemas Operativos de Tiempo Real (I y II)**. De estas asignaturas se aplica el conocimiento obtenido sobre planificadores de tareas expropiativos y la manera en que trabajan. Esto ha sido muy importante para la realización de este Trabajo Final. También la creación de módulos de Firmware para la CIAA.
- **Desarrollo de Sistemas Embebidos en Android**. La plataforma Andoid es un claro caso de éxito de la aplicación de un lenguaje POO en sistemas embebidos (aunque no sea para aplicaciones induistriales). Estas aplicaciones se realizan en lenguaje Java. Si bien se contaba con experiencia en programación orientada a objetos en otros lenguajes, esta asignatura fue para el autor el primer acercamiento a dicho lenguaje. En consecuencia, mucho de lo aprendido colaboró en la decisión de llevar a cabo este trabajo.
- **Diseño de Sistemas Críticos**. Los conceptos aprendidos en esta asignatura contribuyeron a comprender, inmediatamente, las importantes implicancias de poder programar aplicaciones SCJ en sistemas embebidos, que provee HVM, para aplicaciones industriales.

Además, se han adquirido aprendizajes en las temáticas:

- Programación de aplicaciones en lenguaje Java.
- Especificaciones de Java, entre ellas, RTSJ y SCJ.
- Programación de aplicaciones SCJ.
- Experiencia en construcción de planificadores expropiativos con restricciones de tiempo real.
- Máquinas Virtuales de Java para sistemas embebidos y su funcionamiento interno.
- Desarrollo de API para sistemas embebidos.
- Desarrollo de *plugins* para Eclipse y la aplicación de patrones de diseño orientado a objetos para lograr una exitosa solución.

Por lo tanto, se llega a la conclusión que los objetivos planteados al comenzar el trabajo han sido alcanzados satisfactoriamente, y además, se han adquirido conocimientos muy importantes para la formación profesional del autor.

5.2. Trabajo a futuro

Como labor a futuro, pueden realizarse las siguientes tareas:

- Programar los *Launchers* para las plataformas CIAA y EDU-CIAA-NXP para permitir *debuggear* directamente en Java.
- Integrar el port de la CIAA al repositorio oficial de HVM.
- Investigar HVM-TP, el nuevo desarrollo de Stephan Erbs Korsholm. Es una máquina virtual de Java *Time Predictable* y Portable para sistemas embebidos *Hard Real-Time*, que toma como base a HVM, extendiéndola y mejorando sus capacidades.

REFERENCIA BIBLIOGRÁFICA

1. The Open Group (2013). *Safety-Critical Java Technology Specification JSR-302*. Version 0.94, 25-06-2013. Oracle. On line, última consula 12-10-2015 http://download.oracle.com/otn-pub/jcp/safety_critical-0_94-edr2-spec/scj-EDR2.pdf.

2. Stephan E. Korsholm (2014). *The HVM Reference Manual*. Icelabs, Dinamarca. On line, última consula 05-10-2015 en http://http://icelab.dk/resources/HVMRef.pdf.