Programming
CAN-based Fieldbus Systems using Java

Manuel Joaquim Pereira dos Santos,
Ricardo José Caetano Loureiro
Universidade de Aveiro, Aveiro, Portugal,

Helmut Dispert
Department of Computer Science and Electrical Engineering,
Kiel University of Applied Sciences, Kiel, Germany
Contents

1. Introduction: 
   Embedded Intelligent Devices

2. Java in Embedded Control: 
   e.g.: Java Servlet Technology (the Embedded Java Controller - EJC)

3. Java and Native Code
The evolution of embedded intelligent devices is driven by technological advancements:

- low cost microprocessors and peripheral devices
- Internet
- Java programming language

Examples:

- new hardware platforms (e.g. Strong ARM architecture)
- Internet and the HTTP to access devices
- Development of robust and reliable software using Java.
Opportunities

Sun Microsystems, Inc.:

"Combining the strengths of these (new) technologies, there is a huge market opportunity for companies that discover how to leverage the benefits of Java technology on embedded devices."
Java

Solution:

A New Paradigm:
Write once, run anywhere

Java
History of Java (true story)

- 1990: A small group is formed at Sun Microsystems with the task to think about the future of multimedia in private households.
  
  Group Member: James Gosling

- Idea: develop a generic and simple programming language to implement intelligent electronic devices (household).

  ⇒ Embedded Programming
Why Java?

- Object-oriented (encapsulation, polymorphism, inheritance),
- No multiple inheritance,
- Platform independent bytecode,
- Java primitive data types have fixed sizes,
- Automatic run-time bounds-checking,
- True Boolean type,
- No pointer programming,
- Automatic garbage collection,
- Language support for multithreaded applications.
Java in Embedded Systems

Java Usage Models:

The proposed Java usage models fall into one of four categories:

- No Java
- Embedded Web Server Java
- Embedded Applet Java
- Application Java
These models are distinguished by two binary variables:

- location of the stored Java bytecodes
- the processor on which the bytecodes are executed

These variables can take one of two values:

- target (the embedded system)
- host (a general-purpose computer attached to the embedded system)

Ref.: Michael Barr
Java 2 Platform, Micro Edition (J2ME) encompasses VMs and core APIs specified via Configurations as well as vertical – or market-specific APIs specified in Profiles.
Java 2 Platform Editions

- Java Technology Enabled Devices
- Java Technology Enabled Desktop
- Workgroup Server
- High-End Server

Editions:
- Micro Edition
- Standard Edition
- Enterprise Edition
Java

Application Portability:
1. mechanism for executing Java bytecodes on any processor
2. common set of class libraries
Java Development Cycle

Compile-time Environment

Java Source (.java)

Java Compiler

Java Bytecode (.class)

Run-Time Environment (Java Platform) Environment

Class Loader Bytecode Verifier

Java Interpreter

Just-in-Time Compiler

Run-Time System

Operating System

Virtual Java Machine

VM

Java Classes Libraries

Operating System

Hardware
The Java Architecture

1. Programming Language
2. Virtual Machine (VM)
3. API
The basic idea of embedded control is to directly access hardware level properties:

- Registers, I/O-interfaces, sensors, actuators, etc.

Problem Basic Properties of Java:

- Platform independent bytecode,
- No pointer programming,
- Automatic garbage collection,
Access Hardware using Java

Solutions:

- Use dedicated hardware platform, dedicated OS and dedicated Java VM.
  Example:
  Snijder Embedded Java Controller (EJC).
  see: 1st Int. Workshop, Vaasa, 2002
- Use native languages that allow access to hardware level functions.
Java - Jbed

Java in a WWW browser

Java on a desktop OS

Java on a RTOS

Java on the bare metal
EJC - EW1A block diagram

E²PROM
Temp. Sensor

CPU
ARM7TDMI

DRAM
Flash
ISA-like + memory bus

COM 1
TTL
RS232
RS485
I²C master
Digital I/O
LCD
Ethernet

DIMM connector

Embedded Java Controller block diagram

Embedded Internet, © Prof. Dr. Helmut Dispert
Overview of the EJC software architecture.

User applications and servlets

EJC class library + servlet framework

Intent JVM + PJAE + javax.comm

EJC middleware and system software

Elate RTOS kernel + low-level libraries
(TCP/IP, file system architecture, graphic subsystem, etc.)

Drivers
Serial, Ethernet, I²C, Dig I/O, Flash, NV RAM, etc.

Level
Java

JVM

Kernel
Access to hardware or peripherals through native code:

Special feature of the EJC (the intent JVM):

Java classes can be directly written in VP assembler. VP assembler is a high-level assembler language which is targeted at a special Virtual Processor, and that is translated to native code either statically, at sysgen-time, or dynamically, when a class is loaded by the device (happens automatically).

⇒ Access to hardware is possible without the runtime overhead of JNI or other similar mechanisms.
JNI - The Java Native Interface

Advantages:

- Support platform-dependent features.
- Integration: Make existing libraries or applications written in another programming language available to Java.
- Speed: Implement time-critical code.
JNI

Application

C Side
- Functions
- Libraries

Java Side
- Exceptions
- Classes
- VM
JNI allows interaction in two directions:

- Java programs can use native methods written in other languages
- Native methods can use Java objects and methods out of Java applications
JNI-Example

Java-File: HelloWorld.java

class HelloWorld
{
    public native void displayHelloWorld();
    static
    {
        System.loadLibrary("hello");
    }
    public static void main(String[] args)
    {
        new HelloWorld().displayHelloWorld();
    }
}
JNI-Example

Java Code Compilation:

```bash
javac HelloWorld.java
```

Creating the C header file:

```bash
javah -jni HelloWorld
```
JNI-Example

Name of native language function that implements the native methods:

prefix + class name + _ + method name

```c
JNIEXPORT void JNICALL
Java_HelloWorld_displayHelloWorld(JNIEnv *, jobject);
```
Native Methods (C code):

```c
#include <jni.h>
#include "HelloWorld.h"
#include <stdio.h>
JNIFUN(Java_HelloWorld_displayHelloWorld
    (JNIEnv *env, jobject obj)
{
    printf("Hello World!\n");
    return;
}
```
JNI-Example

HelloWorldImp.c contains three header files:

1. jni.h
   This file contains information needed by the native language to exchange data with the Java runtime system.

2. HelloWorld.h
   The generated header file.

3. stdio.h
   Contains the printf function.
JNI-Example

1. Write Java Code
   HelloWorld.java

2. Compile with javac
   HelloWorld.class

3. Generate header file with javah
   HelloWorld.h

4. Write implementation of native method
   HelloWorldImp.c

5. Compile native code, load shared library
   hello.dll

6. Run program using java interpreter
   "Hello World"

Required libraries:
- jni.h
- stdio.h
Java in Embedded Systems

Why Embedded Java?

- **Hardware Independence** (greater than with C/C++)
- **Downloading capabilities:**
  - Software downloaded into device
- **Internet Connectivity**
- **Security**
- **Better Productivity**
Application: CAN-Bus

CAN-Bus $\Rightarrow$ Controller Area Network

Serial 2-wire bus system
aimed at automobile applications,
developed by Bosch.
International Standard (ISO 11898)
Field Bus Systems

Field Bus Organizations

Field Bus Systems

Sensor/Actuator Bus
- ASI
- INTERBUS-S
- PROFIBUS DP

Process Bus
- CAN
- DIN-Bus
- BITBUS
- PROFIBUS FMS
- P-NET

Embedded Internet, © Prof. Dr. Helmut Dispert
Virtual CAN Interface

- Control Functions
- Monitor Functions
- CAN Analyzer
- User Application
Java based CAN Program

class: VCIJNI.java

** Function: VCI2_PrepareBoard
** Native method declaration

```java
public native int VCI2_PrepareBoard(CAN_Analyser myApp,
                                   int board_type,
                                   short board_no,
                                   String AddInfo,
                                   byte AddInfoLength,
                                   JCallback receive,
                                   byte [] m_callback_databytes);
```
Dynamic Link Libraries

Code in class `VCIJNI.java` to load DLL (`VCIJNI.dll`) into memory

```java
static {
    System.loadLibrary("VCIJNI");
}

public VCIJNI() {
}
```

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>.dll</td>
</tr>
<tr>
<td>Unix</td>
<td>.so</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>.dylib</td>
</tr>
</tbody>
</table>

**JNI-function**

**for loading of the VCIJNI.DLL**
Dynamic Link Libraries

**Java Code Compilation:**

```
javac VCIJNI.java
```

**Creating the C header file:**

```
javah -jni VCIJNI
```
Dynamic Link Libraries

Generated Header File IXXAT_VCIJNI.h

/* DO NOT EDIT THIS FILE – it is machine generated */
#include <jni.h>
/* Header for class IXXAT_VCIJNI */

#ifndef _Included_IXXAT_VCIJNI
#define _Included_IXXAT_VCIJNI
#endif

#ifdef __cplusplus
extern "C"
#endif

/* Class:     IXXAT_VCIJNI
 * Method:    VCI2_PrepapreBoard
 * Signature:
 (LIXXAT/CAN_Analyser;ISLjava/lang/String;BLIXXAT/JCallback;[B)I
 */

JNIEXPORT jint JNICALL Java_IXXAT_VCIJNI_VCI2_1PrepareBoard
(JNIEnv *, jobject, jobject, jint, jshort, jstring, jbyte,
jobject, jbyteArray);
Dynamic Link Libraries

File VCIJNI.cpp implementing the native code for function
Java_IXXAT_VCIJNI_VCI2_1PrepareBoard

```c
JNIEXPORT jint JNICALL Java_IXXAT_VCIJNI_VCI2_1PrepareBoard
    (JNIEnv * env,
jobject jObj,
jobject myApp,
jint board_type,
jshort board_no,
jstring AddInfo,
jbyte AddInfoLength,
jobject JReceiveCallback,
jbyteArray callback_databytes)
{
    int i_test;
    jboolean *isCopy = NULL;

    //Produce a new global reference to Object " JReceiveCallback "
g_JObj_JCallback = env->NewGlobalRef(JReceiveCallback);
```
Dynamic Link Libraries

Create Object:

```java
// Object with the Java Native Interface
static VCIJNI vci = new VCIJNI();

// prepare board for further configuration
i_test = vci.VCI2_PrepareBoard(
    myApp,
    BoardConfStruct.board_type,
    BoardConfStruct.board_no,
    AddInfo,
    AddInfoLength,
    JReceiveCallback,
    JReceiveCallback.m_a_data);
```
a) CAN Interface:

**USB-to-CAN Module**

Intelligent CAN module for the USB-Port
b) Connected Device:

**Absolute Rotary Encoders**

Allows a direct read-out of the angular position
Program Code: Java
CAN Code: C++
GUI: Java Swing

Future Expansion:
Internet Connectivity: Java
Using Java to Control CAN

Some more important applications

CAN Bus

CAN Device

ON

Summer Cottage in Vaasa
The end

Thank you!
Contact:
Prof. Dr. Helmut Dispert
University of Applied Sciences Kiel
Faculty of
Computer Science and Electrical Engineering
Grenzstr. 3
24149 Kiel, Germany
Tel.: +49-431-210-4114/4123
Fax.: +49-431-210-64114
E-Mail: helmut.dispert@fh-kiel.de