임베디드 Java 라이브러리의 가변 코드 관리 방법

이승연[O] 윤석진 김철홍 양영종
한국전자통신 연구원 임베디드 SW 연구단
{coral[O], sjyoon, kch, yangyj}@kiss.or.kr

Code Variant Management of Embedded Java Library

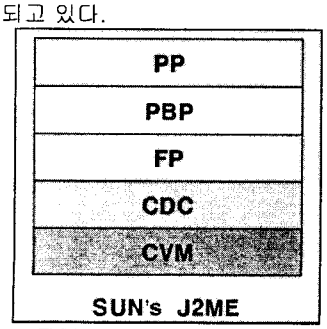
Seungyun Lee[O] Seokjin Yoon Cheol-Hong Kim Young Jong Yang
Embedded SW Laboratory, ETRI

요 약

임베디드 Java 플랫폼은 다양한 하드웨어 및 운영체제에 적용되어야 하므로, 하드웨어에 종속적인 부분과 운영체제에 종속적인 부분이 적용되는 기기마다 가변적이다. Java 라이브러리의 경우 가상머신(VM: Virtual Machine)에 종속적으로 구현되며 네이티브 코드들로 상당부분 구현되기 때문에 다양한 VM과 결합하여 임베디드 Java 플랫폼을 구성할 경우 VM에 종속적인 부분을 가변 코드로 관리할 필요가 있다. 본 논문은 임베디드 Java 플랫폼의 적용 속성을 고려하여 가변 코드를 관리하는 방법을 제안하고 이를 구현해 본다.

1. 서 론

통신기기, 정보가전, 디지털 TV 등과 같은 정보기기용 임베디드 소프트웨어 응용 분야가 급속도로 발전하면서 특정 응용 프로그램을 지원하는 라이브러리와 같은 공통 미들웨어가 필요하게 되었다. 현재 다양한 형태의 임베디드 기기들이 실제 생활에 적용되어 있으며, 그 중 Java 기반의 소프트웨어를 탑재한 기기들이 다수 존재한다. 즉, 임베디드 기기의 활용 측면에서 Java의 사용도는 높다고 할 수 있다. SUN에서 발표한J2ME(Java 2 Platform, Micro Edition)[1]은 모바일 디바이스와 같은 메모리 제약형 임베디드 시스템을 개발하기 위한 플랫폼이다. J2ME는 Configuration과 Profile을 조합하여 다양한 기기의 자원 및 요구에 적합한 임베디드 Java 플랫폼을 구성할 수 있도록 지원한다. 현재 이러한 Java 플랫폼의 특징을 이용하여 디지털 방송용 셋톱박스, 스마트폰, 텔레메틱스 단말, 등의 다양한 기기에 Java 기반 소프트웨어가 활용되고 있다.



<그림 1> SUN의 J2ME 구조

<그림 1>은 SUN의 J2ME 구조를 나타낸 것이다. CVM이라는 가상 머신 위에 다양한 프로파일들이 올라가서 자바 응용을 수행하게 되며, 이는 레이어 구조를 따르고 있다.

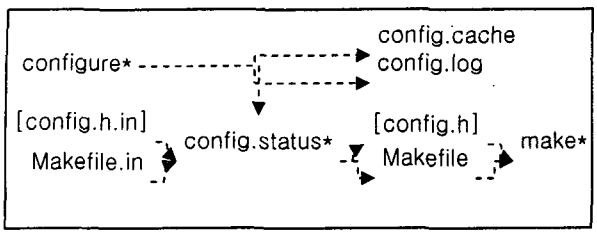
현재, J2ME 명세를 지원하는 JVM(Java Virtual Machine)들이 개발되고 있으나, 다양한 하드웨어에 포팅될 수 있도록 개발되어야 하는 어려움이 있다. VM상에서 자바 응용을 지원하는 자바 라이브러리의 경우도 VM에 의존적인 부분을 구현시 고려하여 가변코드로 관리할 필요가 있다. 그림 1에서 CDC(Connected Device Configuration)[2] 프로파일은 VM에 매우 의존적인 부분으로 VM의 방식을 그대로 반영하여 구현된다. 따라서 임베디드 자바 라이브러리를 프로파일별로 재사용할 경우 CDC 부분과 VM의 의존성을 파악하여 선택적으로 재사용하는 방법이 필요하다.

본 논문은 다양한 VM과 결합할 수 있는 임베디드 자바 라이브러리를 제공하기 위하여 가변 코드를 관리하고 포팅하는 방법을 제안하고 이를 구현해 본다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구에서 가변 코드를 관리하기 위해 사용하는 GNU의 AutoTool에 대해 설명한다. 3장에서는 자바 라이브러리의 가변 코드를 관리하여 다양한 VM과 연결할 수 있는 방법 및 이를 적용한 예를 기술하고, 4장에서는 결론 및 향후 연구에 대해 설명한다.

2. AutoTool을 이용한 패키징 기법

구현된 패키지를 가져와서 컴파일 할 경우 GNU의 AutoTool을 이용하여 필요한 configure 파일과 Makefile을 생성하게 된다. <그림 2>는 ./configure 쉘 스크립트를 실행시켰을 때의 내부 동작 과정이다.

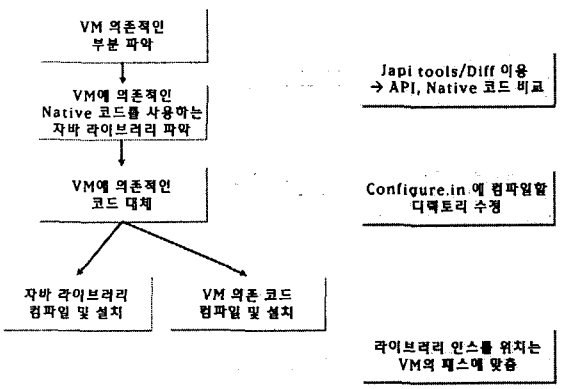


<그림 2> configure 스크립트의 동작과정

configure를 실행시키면, 일단, 시스템의 정보를 알아낸 다음 config.status라는 또 다른 스크립트를 만든다. 이 스크립트를 실행시켰을 때 미리 저장된 시스템 정보에 따라 config.h나 Makefile을 생성하게 된다. config.status는 다시 configure를 실행시키지 않고, Makefile이나 config.h를 만들어낼 때 쓰인다. 그러면 만들어야 할 것이 configure스크립트와 config.h.in, Makefile.in이 될 것이다. configure는 autoconf에 의해서, config.h.in은 autoheader에 의해 만들어지게 된다. Makefile.in인데 이것은 automake로 해결한다.

3. 임베디드 Java 라이브러리의 가변 코드 관리 기법

Kaffe[4], CVM, JikesVM[5]과 같은 다양한 가상 머신에 Java 라이브러리를 결합하는 것은 VM에 의존적인 부분을 파악하고 이를 구현해야 하는 어려움이 있다. Configuration과 Profile로 이루어진 Java 라이브러리의 구조를 살펴보면 CDC(Connected Device Configuration)에 해당하는 부분은 VM의 구현에 많은 영향을 받는다. 예를 들어, java.lang 패키지의 경우 VM 구현 메소드들을 사용하여 구현되기 때문에 제공하는 VM에 따라 구현하도록 재구현, 혹은 대체하여야 한다.



<그림 3> VM과 Java 라이브러리의 결합 과정

<그림 3>은 VM에 의존적인 부부을 파악하여 재구현, 혹은 대체한 후 자바 라이브러리와 결합하는 과정을 나타낸 것이다. CDC 스펙에 해당하는 java.io, java.lang, java.util 과 같은 패키지에 대하여 VM에 의존적인 부분을 파악하고 이 부분을 사용하는 Java 라이브러리 부분을 파악해야 한다. Japitools[6]는 기준 코드에 대하여 비교대상 코드의 구현 완성도가 얼마나 되는지 파악하는 GNU 오픈 소스이다. Japitools 와 diff[7] 를 사용하여 Java 구현과 C 구현의 다른점을 비교할 수 있으며 그 결과는 <그림 4>와 같다.

java.lang

Minor. 1 class, 2 methods. Missing. 2 methods.

Minor

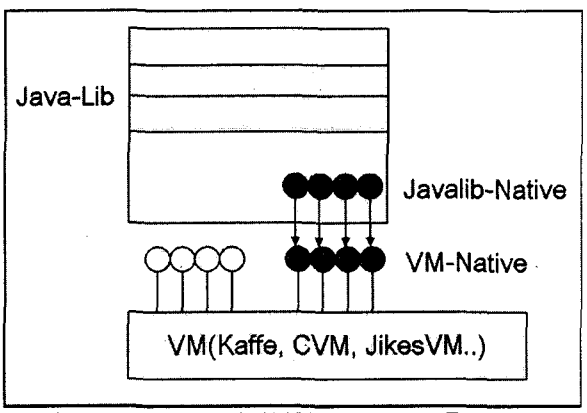
- method java.lang.Runtime.runFinalizersOnExit(boolean): deprecated in jdk14, but not deprecated in kaffe
- method java.lang.System.runFinalizersOnExit(boolean): deprecated in jdk14, but not deprecated in kaffe
- class java.lang.Class: SerialVersionUID=3206093459760846163 in jdk14, but SerialVersionUID=-6256849141973438445 in kaffe

Missing

- method java.lang.Runtime.availableProcessors(): missing in kaffe
- method java.lang.String.contentEquals(java.lang.StringBuffer): missing in kaffe

<그림 4> java.lang에 대한 Japitools 수행결과

<그림 5>는 자바 라이브러리와 VM 네이티브 코드를 결합할 때 발생하는 문제점을 나타낸 것이다.



<그림 5> VM과 Java 라이브러리의 Native 코드 결합

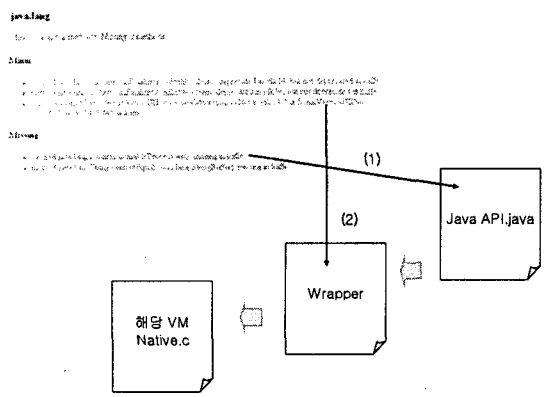
VM과 Java-Lib을 결합하기 위해서는 C코드로 짜여진 VM과 자바 라이브러리 코드를 연결시켜야 한다. Java 라이브러리에서 이용하는 네이티브 코드는 VM의 구조를 구현해야 하기 때문에 이는 결합시에 대체되거나 재구현 되어야 한다. 따라서 결합 방식은 다음과 같다.

> java.lang.Runtime → VMRuntime 호출 (native 선언) → runtime.c 에서 native 구현

VM에 의존적인 부분을 파악하고 난 후에는 의존적인 코드에 대하여 Wrapper 클래스를 두어 네이티브 메소드를

선언하게 하고, 선언된 네이티브 메소드는 제공하는 VM의 구조에 맞게 시그네이쳐를 선언하도록 한다.

또한, 자바 라이브러리와 C 네이티브 코드의 시그네이쳐를 비교했을 때 자바 라이브러리에 선언되어 있지 않거나, 선언되어 있더라도 입력이나 출력의 타입이 다른 경우는 그 타입을 맞춰서 추가해주거나 변경시켜야 한다.



<그림 6> VM과 Java 라이브러리의 결합시 해결 방법

<그림 6>은 메소드나 필드의 시그네이쳐가 일치하지 않을 경우 결합하는 방법에 대하여 나타낸 것이다. 전자의 경우(그림 6의 (1))는 자바 라이브러리에 메소드를 추가시켜 VM의 C 네이티브 코드를 호출할 수 있도록 해야 한다. 후자의 경우(그림 6의 (2))는 Wrapper 클래스에 추가로 구현을 하여 입력과 출력의 타입을 맞춰 결합할 수 있도록 지원해야 한다.

추가로 구현하거나 대체하고 난 후에는 Java 라이브러리와 VM 네이티브 코드를 컴파일하여 해당 위치에 설치해야 한다. <그림 7>은 VM과 자바 라이브러리를 컴파일하기 위한 Configure 파일의 내용이다.

```
dnl ========================================================================

dnl ========================================================================
dnl Make the necessary substitutions
dnl ------------------------------------------------------------------------

AC_SUBST(JAVA_LIBS)
AC_SUBST(with_engine)
AC_SUBST(THREAD_SYSTEM)
if test x"$program_prefix" = xNONE" ; then
        KPREFIX=""
else
        KPREFIX=$program_prefix
fi
AC_SUBST(KPREFIX)
KAFFE_ARCHOS="$khost_cpu-$khost_os"
AC_DEFINE_UNQUOTED(ARCHOS, "$KAFFE_ARCHOS", [Define the version we're compiling for])
AC_SUBST(KAFFE_ARCHOS)

dnl ========================================================================

dnl ========================================================================
dnl Checks for programs.
dnl ------------------------------------------------------------------------

AC_PROG_INSTALL
AC_PROG_MAKE_SET
AC_CHECK_PROG(ZIP, zip, zip)
if test x"$ZIP" = x"" ; then
  AC_MSG_ERROR([zip not found. You need the zip utility for the class library.])
```

<그림 7> VM과 Java 라이브러리을 컴파일하는 configure 스크립트

JNI(Java Native Interface)를 이용하여 구현된 네이티브 코드들은 실행시 로딩될 수 있도록 동적 링킹 라이브러리인 .so 형태의 파일로 생성하며, javaio.so, javalang.so, javautil.so 와 같은 이름으로 생성하도록 하였다. VM에 의존적인 네이티브 코드들은 Wrapper 클래스와 함께 컴파일하여 javabase.so 와 같은 이름으로 생성하도록 하여 해당 디렉토리에 설치되도록 한다.

VM에 의존적인 부분을 파악하고 이에 대한 Wrapper 클래스를 생성하여 재구현, 또는 대체하는 것이 VM과 Java 라이브러리의 충돌 부분을 해결할 수 있다.

4. 결 론

임베디드 Java 플랫폼은 다양한 하드웨어 및 운영체제에 적용되어야 하므로, 하드웨어에 종속적인 부분과 운영체제에 종속적인 부분이 적용되는 기기마다 가변적이다. Java 라이브러리의 경우 VM과 결합하여 임베디드 Java 플랫폼을 구성할 경우 VM에 종속적인 부분을 가변 코드로 관리할 필요가 있다.

본 논문은 임베디드 Java 라이브러리의 가변 코드를 관리하기 위하여 VM에 의존적인 부부을 파악하여 재구현, 혹은 대체한 후 자바 라이브러리와 결합하는 방법을 설명하였다. Wrapper 클래스를 이용한 추가 구현 코드 작성 및 configure 스크립트를 이용하여 동적 라이브러리로 링킹하면 자바 라이브러리의 가변 코드에 대하여 체계적으로 관리할 수 있다.

본 논문에서 Wrapper 클래스와 configure 스크립트의 재구성을 통하여 설명한 VM과 Java 라이브러리의 결합 방법을 향후, Kaffe, CVM 등과 같은 다양한 VM과 자바 라이브러리를 결합하는데 적용해 볼 것이며, 가변 코드 관리 기법을 적용한 configure 스크립트 파일을 자동 생성하는 도구를 개발할 것이다.

5. 참고 문헌

[1] Java 2 Platform, Micro Edition, Technologies, http://java.sun.com/j2me/.
[2] CDC Specification, http://java.sun.com/products/cdc/index.jsp
[3] Introduction of Autoconf,. http://www.gnu.org/software/autoconf/
[4] Kaffe, http://www.kaffe.org/.
[5] JikesVM, http://jikesrvm.sourceforge.net/
[6] Japitools, http://www.kaffe.org/compatibility_japitools.shtml
[7] Introduction of Diffutils, http://www.gnu.org/software/diffutils/diffutils.html