내장형 자바 시스템을 위한 클래스 라이브러리의 특성

양희재*

Characteristic of the Class Library for Embedded Java System

Heejae Yang*

이 논문은 2002학년도 경성대학교 특별과제연구비에 의하여 연구되었음

요 약

클래스 라이브러리는 자바가상기계와 더불어 자바실행환경을 이루는 가장 중요한 요소들 중 한가지다. 통신속도나 메모리의 용량 면에서 제한을 받게 되는 내장형 시스템의 특징상 내장형 자바 시스템은 클래스 라이브러리에 대한 의존도가 매우 높다. 따라서 효율적인 내장형 자바 시스템의 구축을 위해서는 클래스 라이브러리에 대한 면밀한 분석이 필수적이다.

본 논문에서는 내장형 자바 시스템을 위한 클래스 라이브러리의 특성에 대해 분석하였다. 즉 라이브러리를 이루는 클래스 구성과, 그 클래스들을 담은 파일의 크기 및 파일 내 주요 요소인 상수풀에 대한 분석을 하였다. 또한 클래스들이 갖고 있는 필드 및 메소드의 개수, 각 메소드들이 필요로 하는 스택과 지역변수배열의 크기, 그리고 각 메소드의 바이트코드 길이 등에 대해 조사하였다. 이 연구의 결과는 내장형 자바 시스템의 클래스 적재에 따른 지연시간 해석, 인스턴스를 만들 때 소요되는 메모리의 크기 예측 등 효율적 내장형 자바가상기계의 설계에 사용될 것이다.

ABSTRACT

Class library is one of the most crucial element of Java runtime environment in addition to Java virtual machine. In particular, embedded Java system depends heavily on the class library due to having a low bandwidth communication link and a small amount of memory which are a common restriction of embedded system. It is therefore quite necessary to find the characteristic of the class libray for embedded Java system to build an efficient Java runtime environment.

In this paper we have analyzed the characteristic of the class library for embedded system. The analysis includes sorts of classes in the library, typical size of the file which contains the class, and the composition of constant pool which is a major part of the file. We also have found typical number of field and method a class contains, the sizes of stack and local variable array each method requires, and the length of bytecode in the method. The result of this study can be used to estimate the startup time for class loading and the size of memory to create an instance of class which are a mandatory information to design an efficient embedded Java virtual machine.

키워드

자바, 자바가상기계, 내장형 시스템, 클래스 파일

*경성대학교 컴퓨터공학과

접수일자 : 2003. 4. 25.

Ⅰ. 서 론

내장형 시스템에 자바 기술을 적용하는 시도가 최근 급격히 증가하고 있다. 특히 휴대폰이나 PDA 등 이동형 무선장비들이 대표적 적용 대상 시스템이 되고 있다 [1]. 내장형 시스템이나 이동형 무선장비 등은 매우 다양한 하드웨어 플랫폼과 운영체제를 가지고 있으며, 따라서 특정 내장형 시스템을 위해 개발된 응용 프로그램을 다른 내장형 시스템에 이식하는 것은 무척 어려운 일이다. 자바의 플랫폼 독립성, 즉 한 번 프로그램을 작성하면 어느 환경에서도 사용할 수 있다는 Write Once, Run Anywhere 특징은 이와 같은 이식성 문제를 해결할 수 있는 가장 유망한 해법이라는 점에서 주목을 끌고 있다. 자바는 플랫폼 독립성 외에도 객체지향성, 안전성, 용이성 등에서 내장형 시스템을 위한 적합한 기술로 인정받고 있다 [2].

이에 따라 내장형 시스템을 위한 자바 기술이 계속 개발되고 있다. 대표적인 것으로는 썬 마이크로시스템사의 J2ME (Java 2 Micro Edition) 가 있으며, 그외에도 Waba, Perc JVM, Jbed, Chi, simpleRTJ 등이 널리 알려진 것들이다.

자바 실행 환경 (Java Runtime Environment)을 이루는 요소들 중 가장 중요한 것은 자바가상기계 (Java Virual Machine) 와 클래스 라이브러리 (Class Library) 이다. 자바가상기계는 자바의 기계어에 해당되는 바이트코드의 실행 엔진 부분이며, 클래스 라이브러리는 자바 프로그램이 일반적으로 참조하는 핵심 API 클래스들, 즉 Object, Thread, Exception, String, Vector 등을 모아둔 것이다. 본 논문에서는 내장형 시스템을 위한 자바 실행 환경에서 클래스 라이브러리의 특성에 대해 분석해보고자 한다.

자바 실행 환경에서 클래스 라이브러리는 매우 중요한 역할을 차지한다. 예를 들어 컴퓨터 화면에 "Hello, World!" 라는 문자열을 찍는 간단한 자바 프로그램인 HelloWorld.java 의 경우 프로그래머가 작성하는 클래스는 HelloWorld 클래스 하나 뿐이지만, 이것이 실행 될 때에는 클래스 라이브러리에 들어있는 269개의 클래스들이 함께 적재되는 것을 확인할 수 있다 (Solaris 운영체제에서 Java 2 Runtime Environment, Standard Edition (version 1.4.0_03)을 사용하는 경우).

자바 애플릿의 경우도 마찬가지다. 하나의 애플릿 프로그램은 그리 많지 않은 개수의 클래스들로 구성되지만, 이 애플릿이 원격 클라이언트 컴퓨터에서 실행될 때는 그 클라이언트 컴퓨터의 자바 실행 환경에 포함되어있는 클래스 라이브러리 내 다수의 클래스들을 사용한다. 애플릿은 원격 컴퓨터의 웹 브라우저로 다운로드 되어서 실행되므로 다운로드 시간을 줄이기 위해 클래스 라이브러리를 잘 활용하도록 하는 것이 중요하다. 휴대폰이나 PDA 등과 같은 이동형 무선장비를 위한 자바 응용 프로그램도 다운로드 시간을 줄이기 위해 내장되어있는 클래스 라이브러리를 적극 사용하고 있다 [1].

본 논문의 목적은 내장형 시스템을 위한 자바 실행 환경에서 클래스 라이브러리가 어떤 특성을 가지는지를 밝히는데 있다. 즉 데스크톱 시스템을 위한 환경과 달리 내장형 시스템을 위해서는 어떤 클래스들이 라이브러리로 제공되며, 그 클래스들의 파일 크기는 얼마 정도인지에 대해 알아본다. 또한 클래스 파일의 중요 요소 중 하나인 상수풀(*constant pool*)에 대해서도 그것의 크기, 상수들의 종류 등에 대해 조사해보며, 각 클래스들이 갖고 있는 필드 및 메소드의 개수, 각 메소드들이 필요로 하는 스택과 지역변수배열의 크기, 그리고 각 메소드의 바이트코드 길이 등에 대해 분석해보고자 한다.

앞에서 살펴 본 바와 같이 자바가상기계는 클래스 라이브러리에 포함된 클래스들을 다수 사용하므로 이 연구의 결과는 내장형 자바 시스템이 필요로 하는 메모리 양의 예측, 특히 자바가상기계의 설계 시 요구되어지는 자바 스택이나 지역변수배열 등의 크기를 얼마로 할 것인가 등에 도움을 줄 수 있다. 또한 클래스 적재에 따른 지연시간 해석, 클래스의 인스턴스를 만들 때 소요되는 메모리의 크기 예측, 필요한 시간 예측 등에도 사용될 수 있을 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 분석대상으로 사용한 J2ME/CLDC 및 simpleRTJ 내장형 자바 시스템의 클래스 라이브러리에 대해 간략히 소개하며, 3장에서는 클래스 파일과 관련한 내용, 즉 파일의 크기와 파일 내부의 주요 요소인 상수풀의 특징에 대해 분석하였다. 4장에서는 클래스의 구성, 즉 필드와 메소드의 개수, 스택 및 지역변수배열의 크기, 메

소드를 이루는 코드의 길이 등에 대한 분석을 하였다. 5장에서는 본 논문과 관련된 배경 연구들에 대해 소개 및 비교를 하며, 6장에서 결론을 맺는다.

II. 내장형 자바 클래스 라이브러리

2.1 조사 대상

조사 대상으로 선택한 내장형 자바 클래스 라이브러리는 썬 마이크로시스템사의 J2ME 환경에 포함된 CLDC (Connected, Limited Device Configuration) [3] 핵심 API 클래스 라이브러리와, RTJ Computing 사의 내장형 자바 시스템인 simpleRTJ [4] 의 클래스 라이브러리이다. 그 외에도 다수의 내장형 자바 시스템이 알려져 있지만, 상용 제품으로 원천코드가 제공되고 있지 않거나 관련 문서를 발견하기 어려운 것들이었다. 독립된 이 두 가지 대상 시스템의 클래스 라이브러리의 분석 결과 매우 큰 유사성을 발견할 수 있었으며, 따라서 내장형 시스템의 특성상 다른 시스템의 라이브러리도 비슷한 결과를 나타낼 것으로 기대된다.

CLDC 는 CDC (Connected, Device Configuration) 와 함께 J2ME 에 속하는 두 가지 구성(configuration) 중 하나이며, 휴대폰이나 PDA 와 같이 160KB-512KB 정도의 적은 메모리와 16 또는 32 비트의 비교적 낮은 성능을 갖는 프로세서를 갖는 환경에 해당된다. 인터프리터 방식을 사용하는 KVM (Kilobyte Virtual Machine)을 자바가상기계로 사용한다.

simpleRTJ 는 수십 KB 정도의 매우 적은 메모리를 사용하는 내장형 자바 시스템이며, 현재 MC68302, MC68376/332, 68HC11, 68HC16, 8051, 8051XA, ARM, H8S/2241 등 다양한 8/16/32 비트 프로세서 환경에 이식되어 사용되고 있다. simpleRTJ 는 CLDC 등 대부분의 경우와 달리 동적 클래스 적재를 지원하지 않는 특징을 갖는다. 필요한 클래스들이 미리 호스트 컴퓨터 상에서 확인되고 링크 된 후 최종 클래스 이미지 파일이 ROM 등의 형태로 내장형 시스템에 적재된다.

CLDC 는 자료형 중 long 형식을 지원하지만 float 및 double 과 같은 부동소수점 형식은 지원하지 않으며, 반면 simpleRTJ 는 float 형식을 지원하고 long 및 double 형식은 지원하지 않는다.

2.2 클래스 라이브러리 구성

CLDC 는 휴대폰, PDA 등을 위한 공통적이며 핵심적인 클래스들을 정의하고 있다. 이 클래스들은 네 개의 패키지에 나뉘어 배치되어 있는데, java.lang 패키지에 38개, java.io 에 18개, java.util 에 13개, javax.microedition.io 에 10개 등 79개의 클래스와 인터페이스로 구성된다. 본 논문에서는 이들 중 인터페이스와 특수 클래스 등을 제외한 62개의 클래스들에 대해 분석하였다.

주목할 만한 점은 분석 대상 CLDC 클래스 중 거의 절반(47%)에 해당되는 29개의 클래스들이 예외(exception) 및 오류(error)의 처리를 위한 것들이라는 것이다. 핵심 API 클래스에 포함할 수 있는 여타 클래스들이 많이 있을 수 있음에도 불구하고 예외/오류 관련 클래스들이 거의 절반을 차지한다는 것은 내장형 자바 시스템에서 무엇을 중요시하고 있다는 것을 짐작할 수 있게 해준다. 3장과 4장에서 살펴보겠지만 이들 예외/오류 관련 클래스들은 그 구조에 있어서도 일반 클래스들에 비해 매우 현저한 차이점을 보이고 있음을 알 수 있다.

simpleRTJ 의 클래스 라이브러리는 java.lang 패키지에 43개, java.util 에 3 개 등 모두 46개의 클래스와 인터페이스로 구성되며, 이것은 개수 면에서 CLDC 클래스의 60퍼센트 수준에 해당된다. simpleRTJ 에서는 입출력 부분은 모두 네이티브 메소드로 처리하고 있으며, 따라서 java.io 패키지에 속하는 클래스들이 없다는 특징을 갖는다. 본 논문에서는 인터페이스를 제외한 총 43개의 클래스에 대해 연구하였다. simpleRTJ 에서는 모두 28개의 예외/오류 관련 클래스들이 있는데, 이것은 개수 면에서 전체 클래스들의 절반을 넘고있다 (65%). 핵심 API 클래스들 중에서 예외/오류 관련 클래스들이 많다는 점은 CLDC 의 경우와 정확히 일치한다.

표 1은 조사 대상이 된 전체 105개의 클래스들을 각 종류별로 분류하여 나타낸 것이다. 두 클래스 라이브러리의 내용이 많은 부분에서 일치함을 발견할 수 있다. 다만, 시스템 클래스에서의 차이는 simpleRTJ 에서 동적 클래스 적재를 지원하지 않기 때문이며, 또한 데이터 클래스에서의 차이는 CLDC 가 일체의 부동소수점 자료형을 사용하지 않고 simpleRTJ 는 64비트 정수형을 사용하지 않는 것에 따른 것이다. 두 라

표 1. 클래스들의 종류별 분류
Table 1. Sorts of Classes

클래스 종류	공통	CLDC	simpleRTJ
system	Object, String, StringBuffer, System, Thread, Throwable	Class, Runtime	-
data	Boolean, Byte, Character, Short, Integer	Long	Float, Number
math	Math	Random	-
utility	-	Hashtable, Stack, Vector, Calendar, Date, TimeZone	StringTokenizer
i/o	-	ByteArrayInputStream 등 11개	-
exception	Exception 등 15개	IOException 등 11개	-
error	Error, OutOfMemoryError, VirtualMachineError	-	AbstractMethodError 등 9개

이브러리 모두 배정밀도 부동소수점 자료형은 사용하지 않았다. 예외/오류 관련 클래스들이 다수를 차지하고 있는 것을 표에서도 확인할 수 있다.

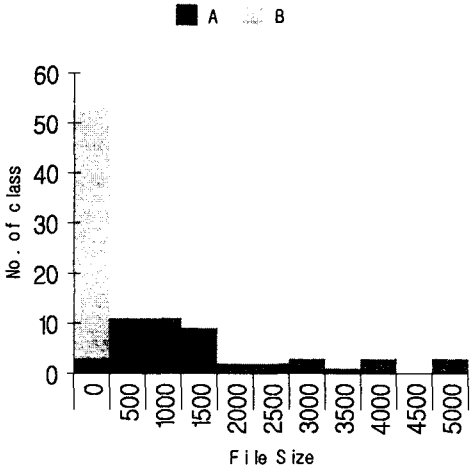

그림 1. 클래스 파일 크기(바이트)
A: 일반 클래스 B: 예외/오류 클래스
Fig. 1 Size of class file (bytes)
A: ordinary classes B: exception/error classes

III. 클래스 파일의 형식 분석

클래스 라이브러리의 특성 분석은 클래스를 담고 있는 파일 즉 클래스 파일에 대한 분석과, 클래스의 내용 즉 클래스 속성을 나타내는 필드(field) 부분 및 기능을 나타내는 메소드(method) 부분에 대한 분석으로 나눌 수 있다. 이 장에서는 먼저 클래스 파일에 대한 분석을 하며, 이것은 다시 클래스 파일의 크기에 대한 분석과, 상수풀 구조에 대한 분석으로 구분된다.

3.1 클래스 파일의 크기

클래스 파일의 크기는 내장형 시스템에서 특히 중요한 요소가 된다. 파일의 크기는 클래스 적재 시간에 직접적 영향을 끼치므로 [5] 자바와 같이 동적 클래스 적재를 지원하는 환경에서 클래스 파일의 크기를 최소화하는 것은 매우 중요한 일이다. simpleRTJ 와 같이 동적 클래스 적재를 지원하지 않는 환경이라 하더라도 클래스 파일의 크기는 메모리 사용량에 직접적 영향을 끼치므로 메모리 자원이 한정적인 내장형 시스템에서 클래스 파일 크기의 제한은 필수적이라고

할 수 있다. 조사 대상 클래스 라이브러리에 포함된 클래스 파일들에 대한 분석 결과는 다음과 같다 (그림 1).

1) 클래스 파일의 크기는 평균 1,000 바이트 정도의 작은 값이다. 이 값은 데스크톱 자바 환경에서의 클래스 파일 평균 크기인 4,500 여 바이트[6]의 1/4 수준에 해당된다. CLDC 에서는 평균 1,168 바이트, simpleRTJ 에서는 평균 902 바이트였다.

2) 표 1에 나타낸 클래스들의 종류별 분류에서 예외/오류 클래스들은 여타 일반 클래스들과는 현격한 차이를 보였다. 즉 예외/오류 클래스 파일들은 거의 전부 500바이트 이하의 크기를 가진 반면 일반 클래스 파일들은 2,000 바이트 가까운 크기를 갖는다. 예외/오류 클래스 파일의 평균 크기는 CLDC에서 237 바이트, simpleRTJ 에서 335 바이트였고, 일반 클래스 파일의 크기는 각각 1,986 바이트와 1,959 바이트였다.

3.2 상수풀에 대한 분석

클래스 파일의 구조는 크게 다섯 가지의 부분으로 나뉘어진다 [7]. 먼저 클래스 파일을 나타내는 헤더(header) 부분이 있고, 그 뒤 상수풀이 이어진다. 상수풀(constant pool)은 클래스에서 사용되는 모든 상수 정보를 모아 둔 곳으로서 클래스 파일을 이루는 부분들 중에서 가장 큰 크기를 차지한다. 데스크톱 자바 환경에서 클래스 파일을 분석한 연구 결과에 따르면 이것의 크기는 전체 파일 크기의 60퍼센트 이상을 차지하는 것으로 알려지고 있다 [6]. 상수풀 뒤에는 클래스 자체에 대한 정보가 놓이고, 이후 이 클래스가 가진 필드에 대한 정보가 있으며, 마지막으로 이 클래스가 가진 메소드에 대한 정보가 놓인다.

첫 번째 헤더 부분은 클래스에 관계없이 8바이트의 고정 길이를 가지고, 세 번째 부분의 클래스 정보 역시 접근제어 값 등 작은 크기의 간략한 정보들로 구성되며 클래스 종류에 거의 무관하다. 따라서 본 논문에서는 두 번째의 상수풀과, 네 번째, 다섯 번째 부분의 필드 및 메소드에 대해서 분석하였다. 이 장에서는 먼저 상수풀에 대한 분석 결과를 설명하며, 나머지는 다음 장에서 설명한다.

1) 상수풀은 클래스 파일의 거의 절반에 육박하는 500 여 바이트의 크기를 가졌다. CLDC 에서는 평균 527 바이트, simpleRTJ 에서는 평균 414 바이트였다. 클래스 파일의 크기에 대한 비율로는 각각 45퍼센트와 46퍼센트에 해당된다. 이 값은 데스크톱 자바 시스템을 위한 클래스 파일에서의 비율인 60퍼센트에 비해 [6] 비교적 낮은 수준임을 알 수 있다.

2) 클래스 파일의 크기 분석에서와 마찬가지로 예외/오류 클래스들은 여타 일반 클래스들과는 구별되는 차이를 가졌다. 즉 예외/오류 클래스들의 상수풀 크기는 클래스 파일의 크기 대비 CLDC는 62퍼센트, simpleRTJ는 61퍼센트를 가지며, 일반 클래스들의 상수풀은 각각 43퍼센트와 41퍼센트에 해당되었다.

3) 상수풀에 들어있는 상수의 개수는 평균 44개이며, 클래스별로는 예외/오류 클래스들이 15개, 여타 일반 클래스들은 80개를 가지는 등 차이가 현격했다. 조사 대상별로는 CLDC에서 50/13/83 이었고, simpleRTJ에서 37/16/76 이었다 (각각 전체평균 / 예외오류클래스 평균 / 일반클래스 평균 순이다).

4) 상수풀에서 가장 큰 몫을 차지하는 것은 문자열에 해당하는 CONSTANT_Utf8 로서 전체의 56퍼센트 정도를 점유한다. 이것은 데스트톱 환경 자바 클래스 파일의 해당 값인 59 퍼센트와 비슷한 수준이다. 다음으로는 클래스명이나 필드명, 또는 메소드명의 이름과 형식을 나타내는 CONSTANT_NameAndType 이 15퍼센트, 메소드를 가리키는 CONSTANT_Methodref 14퍼센트, 클래스를 가리키는 CONSTANT_Class 9퍼센트, 필드를 가리키는 CONSTANT_Fieldref 2퍼센트 등의 순으로 나타났다. 데스크톱 환경에서의 조사값인 13, 10, 9, 4 퍼센트 등의 값들과 각각 비교했을 때 [6] 이 부분에서는 큰 차이가 없는 것으로 밝혀졌다.

5) 마지막으로 상수풀 내의 상수들을 사용 용도별로 분석해보았다. 분석 결과 상수풀에 들어있는 평균 44개의 상수들 중 바이트코드 실행 시 참조되어지는 일반 상수들은 3개(전체의 6퍼센트) 정도이며, 78퍼센트에 해당되는 34개의 상수들은 형식 및 링크 정보로만 사용되어지며, 기타 디버깅 등 직접적 필요가 없는 상수들이 7개(16퍼센트)를 차지하는 것으로 조사되었다.

이 마지막 결과는 내장형 자바 시스템을 위한 매우 중요한 정보를 제공해준다. 즉 내장형 자바 시스템이 사용하는 클래스 파일의 상수풀에 실제 바이트코드가 참조하는 상수들은 그리 많지 않으며, 대부분(78%)은 형식 및 링크 정보를 위한 상수라는 것이다. 따라서 simpleRTJ 와 같이 동적 클래스 적재를 지원하지 않고

호스트 컴퓨터 상에서의 정적 형식 확인만 하는 환경이라면 상수풀의 대부분은 생략이 가능하며, ROM 에 적재되는 클래스 이미지의 크기는 클래스 파일의 절반 수준으로 축소될 수 있다는 것을 알 수 있다.

IV. 클래스의 구성

앞에서는 클래스를 담고 있는 파일, 즉 클래스 파일에 대해 그것의 크기 및 주요 요소 중 하나인 상수풀에 대해 분석하였다. 이 장에서는 실제 클래스 구조, 즉 클래스를 이루는 필드와 메소드에 대해 각각 분석해보았다.

4.1 필드 및 메소드 개수

먼저 각 클래스가 가지고 있는 필드 및 메소드의 개수에 대해 알아보았다. 어떤 클래스의 인스턴스가 생성될 때는 자기 클래스의 필드는 물론 상위 클래스의 필드들을 저장할 수 있는 메모리 공간의 할당이 필요하므로 [7] 필드의 개수는 메모리 사용량에 직접적인 영향을 끼친다. 메소드 개수와 메모리 사용량의 관계는 명확치 않지만, 메소드가 많을수록 메소드 테이블 엔트리 수도 늘어나므로 관련성이 있을 것으로 예상된다. 조사 대상 클래스들의 필드 개수에 대한 분석 결과는 다음과 같다 (그림 2).

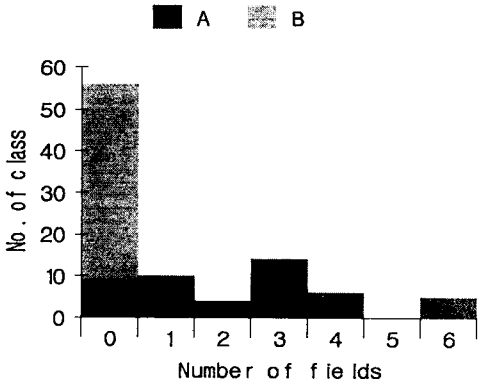

그림 2. 필드 개수 분포
Fig. 2 Distribution of number of fields

1) 필드의 개수는 평균 1.5개로 매우 적은 편이다. 즉 내장형 자바 시스템을 위한 클래스 라이브러리들은 클래스 당 평균 한 개 반 정도의 필드를 갖는다. 필드 당 4바이트에 해당되므로 이 크기는 6바이트에 해당된다. CLDC 에서는 평균 1.3개, simpleRTJ 에서는 평균 1.6개를 가졌다.

2) 필드 개수 면에서도 예외/오류 클래스들은 여타 일반 클래스들과 현격한 차이를 보였다. CLDC 에서는 29개의 예외/오류 클래스들 중 단지 한 클래스만이 1개의 필드를 가졌고, 나머지는 모두 0개의 필드를 가졌다. simpleRTJ 에서는 예외/오류 클래스들 모두가 0개의 필드를 가졌다. 일반 클래스들은 CLDC에서 평균 2.5개, simpleRTJ에서 평균 4.5개의 필드를 가졌다.

이상의 결과에 따르면 필드 사용으로 인한 메모리 소비 정도는 매우 미미한 것으로 판단된다. 특히 예외/오류 클래스들은 필드 사용이 거의 없으므로 그들의 수(전체 클래스 라이브러리의 절반에 해당)에 비해 메모리 사용에 미치는 영향은 무시할만한 것으로 보인다.

다음으로는 각 클래스들이 가진 메소드 개수에 대해 분석을 해보았다. 조사 대상 클래스들에 대한 분석 결과는 다음과 같다 (그림 3).

1) 클래스들은 평균 7.6개의 메소드를 갖는다. CLDC 에서는 평균 7.9개, simpleRTJ 에서는 평균 7.3개를 가졌다.

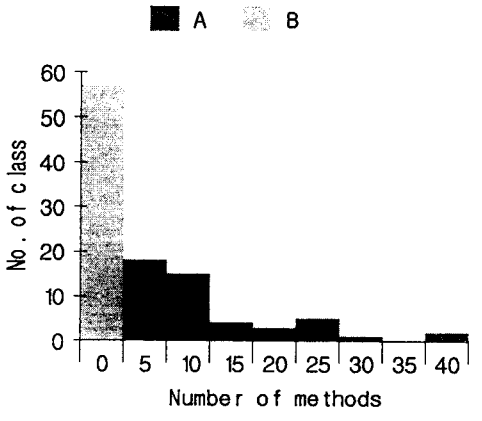

그림 3. 메소드 개수 분포
Fig. 3 Distribution of number of methods

2) 메소드 개수 면에서도 예외/오류 클래스들은 여타 일반 클래스들과 현격한 차이를 보였다. 예외/오류 클래스들의 원천코드를 살펴 본 결과 이들은 거의 모

두 두 개의 생성자 메소드만을 가지고 있을 따름이었다. CLDC 의 경우 평균 2.0개, simpleRTJ 는 평균 2.1개의 메소드를 가졌다. 이에 반해 일반 클래스들은 CLDC에서 평균 13.2개, simpleRTJ에서 평균 17.0개의 메소드를 가졌다.

4.2 메소드 정보

마지막으로 각 메소드의 세부 정보, 즉 메소드가 실행되기 위해 필요한 스택(stack)의 크기, 지역변수배열(local variable array)의 크기, 그리고 메소드에 들어있는 코드(code)의 길이 등에 대해 조사해보았다. 조사 대상 105개의 클래스들이 가지고 있는 메소드들의 전체 개수는 721개(CLDC 449개, simpleRTJ 272개)였다. 이번에도 예외/오류 클래스들에 속해 있는 메소드들(117개)과 여타 일반 클래스들에 속해 있는 메소드(604개)를 구분해서 분석해보았다.

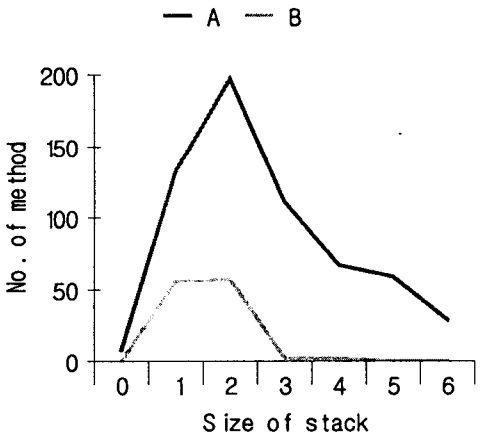

그림 4. 스택 크기의 분포
Fig. 4 Distribution of stack size

먼저 스택의 크기에 대한 분석이다 (그림 4). 어떤 메소드가 호출될 때는 그 메소드를 위한 별도의 스택이 할당되어지므로 스택의 크기는 메모리 사용량에 직접적 영향을 미친다.

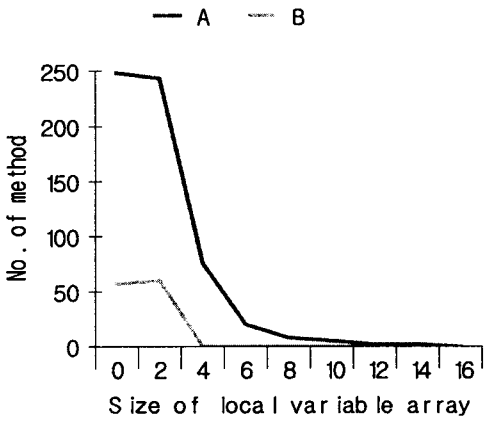

그림 5. 지역변수배열 크기의 분포
Fig. 5 Distribution of local variable array size

1) 조사 대상 메소드들이 요구하는 스택 크기는 평균 2.5였다. 스택 엔트리는 4바이트를 차지하므로 이 크기는 12바이트에 해당된다. CLDC 에서의 스택 크기는 평균 2.6, simpleRTJ 에서는 평균 2.3 였다.

2) 예외/오류 클래스들의 메소드들이 요구하는 스택 크기는 1.6 (CLDC 1.6, simpleRTJ 1.6) 였고, 여타 일반 클래스들의 메소드들이 필요로 하는 스택 크기는 2.6 (CLDC 2.8 simpleRTJ 2.4) 였다. 차이는 1.0 이므로 일반 클래스들의 메소드들이 4바이트 정도를 더 필요로 한다.

다음으로 지역변수배열의 크기에 대해 분석해보았다 (그림 5). 지역변수배열은 메소드 호출 시 각종 인자(arguments)들을 넘겨주거나 메소드 내의 지역변수를 저장하는 목적으로 사용된다. 이것 역시 메소드 호출 시 생성되며 메모리를 사용하므로 메모리 사용량에 직접적인 영향을 준다.

1) 조사 대상 메소드들이 요구하는 지역변수배열의 크기는 평균 2.1이며, 이것은 8바이트 정도에 해당되는 양이다. CLDC 에서의 지역변수배열 크기는 평균 2.4 simpleRTJ 에서는 평균 1.9 였다.

2) 예외/오류 클래스들의 메소드들이 요구하는 지역변수배열의 크기는 1.5 (CLDC 1.5, simpleRTJ 1.5) 였고, 여타 일반 클래스들의 메소드들이 필요로 하는 지역변수배열의 크기는 1.8 (CLDC 1.6, simpleRTJ

2.0) 였다. 다른 조사 결과와 달리 지역변수배열의 크기는 예외/오류 클래스와 여타 일반 클래스 사이에는 큰 차이가 없었다.

끝으로 각 메소드들이 갖는 코드 길이, 즉 바이트코드의 길이에 대해 조사해보았다 (그림 6). 분석 결과는 다음과 같다.

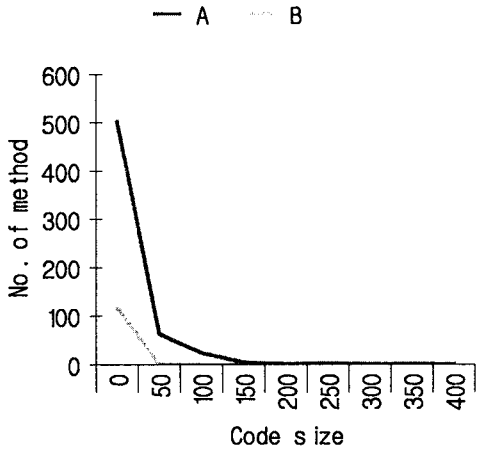

그림 6. 코드 길이의 분포
Fig. 6 Distribution of code length

1) 조사 대상 메소드들의 바이트코드 길이는 평균 25바이트 수준이었다. 메소드의 코드 길이가 생각보다 짧음을 알 수 있다. CLDC 에서의 코드길이는 평균 31.4, simpleRTJ 에서는 평균 20.0 였다.

2) 예외/오류 클래스들에 속한 메소드의 바이트코드 길이는 6.2 바이트로 더욱 짧았으며 (CLDC 6.3, simpleRTJ 6.0), 여타 일반 클래스들의 메소드인 경우는 30.0 바이트였다 (CLDC 35.2, simpleRTJ 24.2). 예외/오류 클래스들은 앞에서 살펴 본 바와 같이 거의 0개의 필드를 가지며, 메소드 역시 6 바이트 정도로 메모리 사용량에 미치는 영향이 극히 적음을 알 수 있다.

V. 관련 연구

자바 클래스와 관련된 통계 자료를 구하는 많은 연구들이 계속 진행되어지고 있다. 이것은 클래스 파일이 자바가상기계의 오브젝트 파일일 뿐 아니라 플랫폼 독립성을 지원하는 중간 단계의 표현 수단이라는 점에서 매우 중요하기 때문이다. 이들 통계 자료들은 보다 효율적인 자바가상기계 또는 각종 자바 도구들의 개발에 사용되어진다.

본 연구는 휴대폰이나 PDA, 또는 기타 내장형 시스템을 위한 자바 환경의 경우 애플릿과 마찬가지로 다운로드에 소요되는 시간을 최소화하기 위해 클래스 라이브러리, 즉 API 를 많이 사용한다는 가정 하에서 그것들에 포함된 클래스에 대해 분석한 것이다. 응용 프로그램에 따라 다르겠지만, 애플릿이나 컴파일러의 경우 거의 대부분의 실행시간이 API 클래스의 메소드 실행에 대한 것이라는 연구 결과가 보고되어있다 [8].

Antonioli 와 Pilz [6] 는 자바 클래스 파일의 크기, 상수풀의 전체 크기 및 상수별 크기, 메소드들이 가지고 있는 바이트코드의 종류 및 사용빈도 등에 대해 해석하였다. 이들은 클래스 라이브러리 뿐 아니라 여섯가지의 각기 다른 응용 프로그램에 포함되어있는 수많은 클래스들을 조사 대상으로 한 것이다.

그들의 연구와 본 연구의 가장 큰 차이점은 그들의 경우 자바 컴파일러 (JavaCC), 웹 브라우저 (hotJava), 자바 개발환경 (Java Workshop) 등 데스크톱 환경을 위한 자바 클래스들에 대해 분석하였다는 것이며, 반대로 본 연구에서는 J2ME 등 내장형 시스템 환경을 위한 자바 클래스들에 대해 분석하였다는 점이다. 분석 대상이 다른만큼 결과도 달리 나왔는데, 3장에서 밝힌 바와 같이 파일 크기에서나 상수풀의 비율면에서 큰 차이를 나타내었다.

또한 본 연구에서는 예외/오류 클래스와 여타 일반 클래스들을 구분하여 분석해봄으로서 그들간에 존재하는 차이점을 밝힌 것도 한 특징이라 할 수 있다. Antonioli 등의 연구는 조사 대상 클래스들을 구분없이 종합적으로 통계를 내었기에 이런 차이점에 대해서는 전혀 언급을 하지 않고 있다. 그밖에도 그들의 연구는 주로 상수풀의 구성 요소에 대해서만 이루어졌으며, 반면 본 연구는 그것을 포함하여 자바가상기계의 설계에 중요한 정보가 되는 필드 및 메소드에 대한 분석을 포함하였다는 특징을 갖는다.

Antonioli 등과 유사한 분석을 또 다른 연구로서는 Waldron 의 논문을 들 수 있다. Waldron [9] 은 Antonioli 의 연구 내용 중 특히 바이트코드의 사용빈도에 대해 관심을 가졌으며, Antoniloi 와 달리 각 메

소드들이 포함하고 있는 바이트코드들에 대해 동적인 분석을 하였다. 즉 각 메소드들이 가지고 있는 바이트코드들이 실행시간 시 얼마나 빈번히 사용되어지는지에 대해 분석했다. 본 논문에서는 바이트코드 외에 실행시간 시 가장 중요한 자료구조에 해당되는 필드, 스택과 지역변수배열의 크기 등에 대해 조사하였으며, 이러한 자료들은 내장형 자바가상기계의 설계에 매우 중요한 몫을 차지하는 것들이다.

본 논문의 선행 연구는 [10][11][12] 에 각각 발표되었다. [10] 은 simpleRTJ에서 클래스 파일이 어떻게 메모리에 배치되어지는지에 대해 분석한 것이다. 2장에서 언급한 바와 같이 simpleRTJ 에서는 클래스 파일들이 정적으로 ROM 등에 적재되어지는 환경을 가정하고 있는데, 이 경우 클래스 파일의 대부분을 차지하는 상수풀을 거의 사용하지 않게 되므로 큰 폭의 메모리 절감을 얻을 수 있음을 보여 주고있다. [11] 과 [12] 는 각각 simpleRTJ 와 J2ME/CLDC 의 API 클래스들을 분석한 것으로서 본 논문의 내용과 많은 부분이 일치한다. 그러나 본 논문에서는 기 발표된 논문의 내용에 상수풀에 대한 분석을 추가하였으며, 두 가지 조사 대상 시스템에 대한 결과값의 비교와 함께 본 연구의 동기와 배경 및 중요성에 대해 보다 상세히 기술하였다.

VI. 결 론

클래스 라이브러리는 자바 프로그램 실행 시 가장 긴요하게 사용되는 API 클래스들의 모음이다. 애플릿과 마찬가지로 휴대폰이나 PDA, 기타 내장형 시스템 환경에서는 일반 데스크톱 환경에 비해 클래스 라이브러리의 활용도가 더욱 높다. 따라서 보다 효율적인 내장형 자바가상기계의 개발이나 소요되는 메모리 양의 예측 등을 위해서는 클래스 라이브러리를 이루는 API 클래스들에 대한 분석이 필수적이다.

본 논문은 내장형 자바 시스템의 클래스 라이브러리에 대한 분석을 하였으며, 그 결과는 다음과 같다.

1) 클래스 파일의 크기는 평균 1,000 바이트 정도로, 일반 데스크톱 환경을 위한 클래스 파일 크기의 1/4 수준이다.

2) 상수풀은 클래스 파일의 45퍼센트 정도 크기를 차지한다. 상수풀에 들어있는 상수의 개수는 평균 44개이며, 이들 중 실제 바이트코드 실행 시 참조되어지는 일반 상수들은 3개 정도이고, 나머지 41개는 형식 및 링크 정보, 그리고 디버깅 정보로 사용되는 것들이다.

3) 클래스들은 평균 1.5개의 필드를 가지며, 따라서 필드 저장을 위해 한 클래스 당 6바이트 정도의 메모리 저장공간을 필요로 한다.

4) 클래스들은 평균 7.6개의 메소드를 가진다.

5) 각 메소드들이 요구하는 스택 크기는 평균 2.5 (12바이트), 지역변수배열의 크기는 평균 2.1 (8바이트) 정도이다.

6) 메소드들의 바이트코드 길이는 평균 25바이트 수준이다.

7) 전체 클래스 중 절반 정도는 예외/오류 관련 클래스들이며, 이들은 파일의 크기나 필드 및 메소드 정보면에서 여타 일반 클래스와 현격히 구분되는 특징을 보인다.

클래스 파일의 평균 크기는 클래스 적재에 따른 메모리 공간 및 초기 설정시간의 예측을 가능하게 하며, 상수풀의 대부분이 형식 확인 및 링크를 위한 것이라는 점에서 정적 클래스 적재를 활용하면 큰 규모의 메모리 절감 효과를 거둘 수 있다는 암시를 받을 수 있다. 각 필드와 스택, 지역변수배열의 크기에 대한 정보는 각 인스턴스의 생성 및 메소드 실행에 따른 메모리 소요량을 예상할 수 있게 한다. 이런 모든 정보들은 메모리 사용을 최적화 할 수 있는 효율적인 내장형 자바가상기계의 설계에 사용될 수 있을 것이다.

참고문헌

[1] S. Helal, "Pervasive Java," IEEE Pervasive Computing, Jan-Mar 2002, pp.82-85

[2] 양희재, 자바가상기계, 한국학술정보, 2001년 3월, ISBN 89-5520-342-4

[3] Sun Microsystems, Connected, Limited Device Configuration Specification, Version 1.0a, May 2000

[4] RTJ Computing, simpleRTJ: A Small Footprint Java VM for Embedded and Consumer Devices, http://www.rtjcom.com

[5] D. Mulchandani, "Java for Embedded Systems,"

IEEE Internet Computing, May-June 1998, pp.30-39
[6] D. Antonioli and M. Pilz, "Analysis of the Java Class File Format," Technical Report, Dept of Computer Sci., University of Zurich, April 1998
[7] T. Lindholm and F Yellin, The Java Virtual Machine Specification, Second Edition, Addison-Wesley, 1999.
[8] J. Waldron, C. Daly, D. Gray and J. Horgan, "Comparison of Factors Influencing Bytecode Usage in the Java Virtual Machine," Second International Conf. on the Practical Application of Java, Manchester, UK, April 2000
[9] J. Waldron, "Dynamic Bytecode Usage by Object-Oriented Java Programs," Proc of the Technology of Object-Oriented Languages and Systems (TOOLS), France, June 1999
[10] 양희재, "simpleRTJ 자바가상기계에서 클래스 파일의 메모리 상 배치", 한국정보과학회 제 29회 추계학술대회, 2002. 10.
[11] 양희재, "simpleRTJ 클래스 파일의 형식 분석", 한국해양정보통신학회 2002 추계학술대회, 제6권 2호, 2002. 11., pp.373-377
[12] 양희재, "내장형 자바 시스템을 위한 J2ME/CLDC 클래스 파일의 분석," 대한전자공학회 컴퓨터/반도체 소사이어티 학술대회, 2002. 11. pp.29-32

저자소개

양희재 (Heejae Yang)

1985년 부산대학교 전자공학과 (공학사)
1987년 KAIST 전기전자공학과 (공학석사)
1991년 KAIST 전기전자공학과 (공학박사)
1991년-현재 경성대학교 컴퓨터공학과 교수
※ 관심분야: 컴퓨터구조, 내장형 시스템, Java