자바 카드 운영체제를 위한 실시간 디버깅 방법

한진희, 전성익

한국전자통신연구원, IC카드연구팀

Real-time Debugging Method for Java Card Operating System

Jin-Hee Han, Sung-Ik Jun

IC Card Research Team, Electronic and Telecommunication Research Institute (ETRI)

요 약

본 논문에서는 자바 카드 운영체제 개발 시 실시간으로 소스 코드를 디버깅할 수 있는 방법을 제안하고, 제안한 방법을 이용하여 자바 카드 운영체제 개발 시 실시간으로 소스코드 디버깅을 수행하는 과정을 실험결과를 통해 제시하고자 한다. 논문에서 제안하는 디버깅 기능은 디버깅을 위한 클래스 및 디버깅 메소드, native 인터페이스를 통해 연결될 디버깅 함수를 자바 카드 운영체제에 구현하여 개발자가 운영체제 개발을 수행하면서 실시간으로 운영체제상의 소스코드를 원하는 형태로 디버깅할 수 있는 편리한 개발 환경을 제공해 준다. 또한, 개발자 관점의 실시간 소스코드 디버깅 기능을 지원함으로써 자바 카드 운영체제 및 응용 프로그램의 개발을 가속화시키는 부가적인 효과를 얻을 수 있다.

I. 서론

일반적으로 카드 운영체제를 개발하는 개발자는 플랫폼 규격에 적합하게 카드 운영체제를 구현해야 하기 때문에 개발 과정동안 운영체제가 통과해야 하는 다양한 테스트를 시험하고, 시험 중에 발생되는 문제점이나 에러를 수정하기 위해 디버깅 기능을 이용한다.

자바 카드 운영체제 개발자 역시 운영체제 개발을 위해 디버깅 기능을 이용해야하지만, 기존 운영 체제 개발환경에서 제공하는 디버깅 기능은 개발자의 욕구를 충분히 만족시켜줄 만큼 다양하고 편리하지 않다. 즉, 개발자 관점을 중시한 편리한 디버깅 도구가 자바 카드 운영 체제 개발자들에겐 절실히 필요한 실정이다.

이러한 자바 카드 개발 환경의 취약점을 보완하고, 자바 카드 운영체제 개발자에게 편리한 디버깅 도구를 제공하기 위하여, 본 논문에서는 실시간 소스코드 디버깅 방법을 제안한다. 논문의 2장에서는 자바 카드에 대해 간단히 소개하고, 3장에서는 제안한 실시간 소스코드 디버깅 기능을 자바 카드 운영체제 개발환경에 구현하는 방법에 대해 기술한다. 4장에서는 3장에서 언급한 디버깅 기능을 이용한 실시간 소스 코드 디버깅 기능의 테스트 및 그에 따른 결과를 보여주고, 마지막으로 5장에서 논문을 마무리 하고자 한다.

II. 자바 카드

1. 자바 카드 개요

1996년, 스마트 카드 제조업체인 슈럼버제사는 스마트 카드의 운영체제에 자바 바이트코드 인터프리터를 탑재하고, 카드에 적합하게 변환된 자바 클래스 파일을 다운로드할 수 있는 자바 기반 스마트 카드를 제안하였다. 같은 해 10월에 SUN사는 자바 카드 규격을 최초로 발표하였고, 이후 1997년에 보다 구체적인 내용을 포함한 자바 카드 2.0 규격이 발표되었으며, 1999년에 자바 카드 2.1, 2002년에 자바 카드 2.2 규격이 배포되었다[1].

자바 카드는 중앙 처리 장치, ROM, EEPROM, RAM, 암호 모듈 등이 존재하는 하드웨어 계층위

에 카드 운영체제(Card Operating System), 자바 카드 가상 기계(Java Card Virtual Machine), 자바 카드 API(Appication Programming Interface), 자바 카드 애플릿이 순서대로 탑재된다. 그림 1은 이러한 자바 카드 내부 구조를 보여주고 있다.

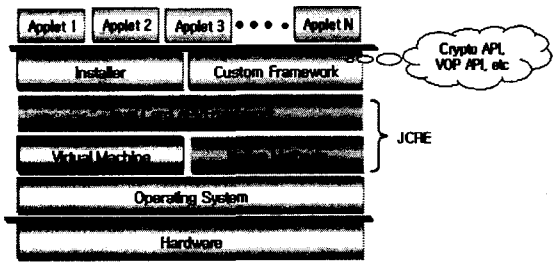

그림 1: 자바 카드 내부 구조

2. 카드와 단말기간의 통신

카드와 단말기는 APDU(Application Protocol Data Unit)를 통해 상호 메시지를 송·수신하는데, APDU는 ISO 7816 표준에서 지정한 통신 프로토콜이다.

카드는 항상 단말기로부터 명령 APDU가 전달되어야만 동작하는 수동적(reactive)인 매체로, 단말기로부터 명령어가 전달되면 해당 명령어를 처리하여 응답 APDU를 단말기에게 전송한다. 명령 APDU는 APDU의 상위 5바이트를 헤더로 사용하고, 응답 APDU는 상태 값으로 SW1, SW2 2 바이트를 이용하여 명령 APDU 처리 후 카드의 상태를 단말기에게 전송한다. 아래 표 1은 명령 및 응답 APDU 구조를 보여준다[2].

표 1: 명령 및 응답 APDU 구조

<table>
<tr><td colspan="6">명령 APDU</td></tr>
<tr><td colspan="4">명령 APDU 헤더</td><td rowspan="2">Lc</td><td rowspan="2">Data</td><td rowspan="2">Le</td></tr>
<tr><td>CLA</td><td>INS</td><td>P1</td><td>P2</td></tr>
<tr><td colspan="7">CLA: 클래스 바이트(command-ID), INS: 명령어 코드
P2, P2: 파라미터, Lc: 명령 APDU 데이터 길이
Data: 명령 APDU 데이터, Le: 응답 APDU 데이터길이</td></tr>
</table>

<table>
<tr><td colspan="3">응답 APDU</td></tr>
<tr><td>Data</td><td>SW1</td><td>SW2</td></tr>
<tr><td colspan="3">Data: 응답 APDU 데이터, SW1, SW2: 상태 코드</td></tr>
</table>

III. 실시간 소스코드 디버깅 기능

1. 구현 방법

이 장에서는 본 논문에서 제안한 자바 카드 운영체제를 위한 실시간 소스코드 디버깅 기능 구현 방법에 대해 자세하게 기술하고자 한다.

그림 2는 일반적인 자바 카드 운영체제 개발 과정을 보여준다[2][3][4][5].

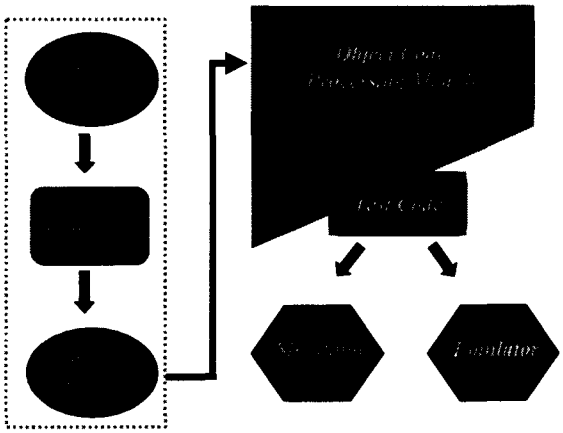

그림 2: 자바 카드 운영 체제 개발 과정

그림 2에서 보여주는 object code processing module은 object code를 이용하여 패키지 단위로 converting을 수행하고, 에뮬레이터 환경이나 시뮬레이터 환경에서 동작할 수 있는 test code를 생성해주는 도구를 포괄적으로 의미한다. 일례로, 자바 카드 애플릿을 테스트하고자 하는 경우, test code는 CAP(Converted Applet) 파일이 될 것이다. 그림 2의 일반적인 개발 과정을 기반으로 논문에서 구현하고자 하는 디버깅 기능을 구현하기 위한 개발 과정은 그림 3과 같다.

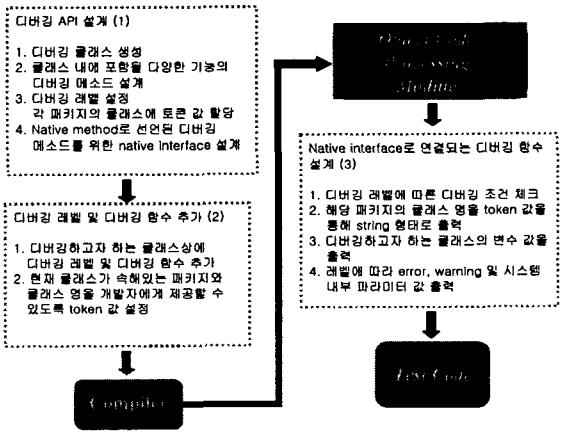

그림 3: 실시간 소스코드 디버깅 기능 구현 과정

그림 3을 살펴보면, 컴파일러를 통과하기 전과 object code processing module 이후에 소스 코드

디버깅 기능을 위한 기능 구현 과정이 어떠한 순서를 통해 개발 되어야 하는가를 보여주고 있다.

2. 기능 설명

그림 3의 디버깅 API 설계 부분에서는 자바 카드 패키지에 포함될 디버깅 클래스와 디버깅 클래스에 구현될 다양한 디버깅 메소드들을 설계하고, native interface를 통해 연결될 메소드 파라미터를 정의한다. 또한, string이 지원되지 않는 자바 카드 환경을 고려하여, 디버깅하고자 하는 변수가 포함되어 있는 패키지, 클래스 명을 실시간으로 출력해주기 위해 token 값을 각 패키지 별 클래스에 할당하여 추후 클래스 명을 출력하고자 할 때 활용할 수 있도록 token 값을 할당한다.

디버깅 API 설계과정이 끝나면, 운영체제 개발자는 자신이 디버깅하고자 하는 변수가 선언되어 있는 클래스 소스파일에 token 값과 디버깅 레벨을 설정한 후, 디버깅할 변수를 파라미터로 넣어 디버깅 메소드를 호출하는 코드를 추가한다.

마지막으로, object code processing module 이후, 디버깅 API 설계 시 native interface로 연결될 메소드들의 실제 처리 함수 구현 과정이 추가되어야 한다. 이 부분에서는 token 값을 클래스 명으로 변환하여 출력해주는 함수와 디버깅 레벨에 따른 디버깅 옵션을 설정해주는 함수, 개발자가 디버깅하고자 하는 변수 값을 출력해주는 함수 등을 구현한다. 디버깅 레벨에 따라 옵션 정보로 error, warning 정보 및 시스템 파라미터로 PC, SP 값 등을 출력해주는 함수도 추가적으로 구현 가능하다.

Ⅳ. 실험 및 결과

이 장에서는 제안한 실시간 소스 코드 디버깅 기능을 자바 카드 운영체제 개발 환경에 구현하여 실험한 과정 및 그에 따른 결과를 보여준다.

실험을 위해 자바 카드 운영 체제 개발 환경에 구현한 디버깅 API 클래스 및 메소드는 그림 4와 같으며, 자바 카드 암호 API로 구현된 SHA-1 알고리즘[6] 상의 변수를 디버깅하기 위해 추가한 디버깅 메소드는 그림 5와 같다. 또한, SHA-1 암호 API를 테스트하기 위해 설계한 테스트 애플릿은 표 2에서 정의한 바와 같이 CLA, INS 값에 따라 단말기로부터 송신된 명령 APDU를 적절히 처리하여 SHA-1 알고리즘의 결과 값을 단말기에게 전달해준다. 다양한 입력 값에 따른 해쉬 값을 검증하기 위해 본 실험에서는 4가지 형태의 입력 데이터를 테스트 벡터로 이용하였다[7].

```
/* Debug.java class */
/* This class provides debugging facilities  */
package com.etri.util;
import com.sun.javacard.impl.NativeMethods;
public class Debug {
public Debug() {}
public void bytetrace(byte value){
NativeMethods.bytetrace(value); }
public void bytearraytrace(byte[] array, short value){
NativeMethods.bytearraytrace(array, value); }
.... (여러가지 다양한 변수 타입에 따른 디버깅 메소드 구현)
public void settoken(byte p_token, byte c_token){
NativeMethods.settoken(p_token, c_token); }
public void setdebuglevel(byte debug_level){
NativeMethods.setdebuglevel(debug_level); }
}
```

그림 4: Debug.java 클래스 구조

```
/* SHA-1 crypto API class */
package javacardx.crypto;
import javacard.framework.*;
import javacard.security.*;
import com.etri.util.Debug;
Debug mytrace;
public final class SHA extends MessageDigest{
public SHA() {
....
mytrace = new Debug();
}
public short doFinal(byte[] input, short iOffset, short length,
byte[] output, short oOffset){
//javacardx.crypto 패키지상의 SHA 클래스 token 값 설정
mytrace.settoken((byte)9, (byte)3);
//클래스 상의 변수 값을 디버깅하기 위한 디버깅 레벨 설정
mytrace.setdebuglevel((byte)3);
...
mytrace.bytearraytrace(input, length); // 입력 데이터 디버깅
...
mytrace.bytearraytrace(output, (short)20); // 해쉬 값 디버깅
....
}
}
```

그림 5: SHA 클래스에 디버깅 메소드 추가

표 2: SHA-1 알고리즘 테스트 애플릿

CLA	INS	P1	P2	Lc	Le	비고
D0	10	00	00	1	00	입력 메시지 수신
D0	10	01	00	3	00	
D0	10	02	00	1C	00	
D0	10	03	00	1C	00	
D0	10	04	00	20	00	
D0	10	05	00	20	00	
D0	20	00	00	00	20	해쉬 값 송신
D0	A0	00	00	00	00	Run SHA-1

앞서 설계한 디버깅 API와 실험을 위해 디버깅 메소드를 추가한 SHA-1 암호 API의 실시간 소스 코드 디버깅 결과는 그림 6과 같다.

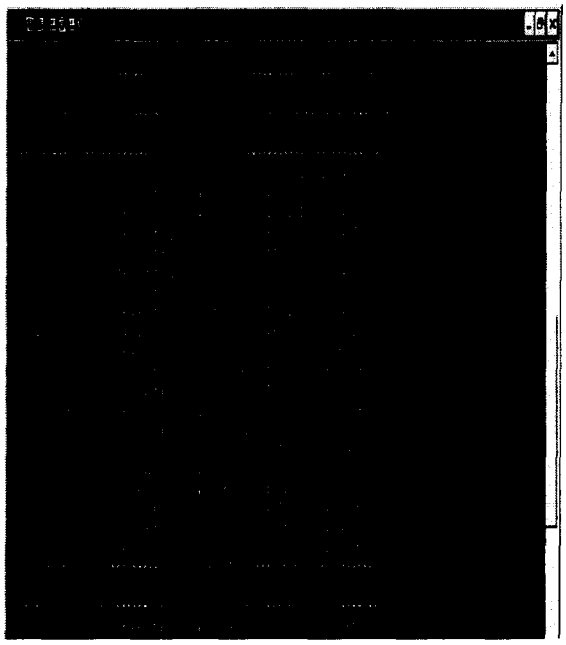

그림 6: 실시간 소스 코드 디버깅 실험 결과

그림 6의 실험 결과를 보면 알 수 있듯이, token 값을 이용하여 현재 디버깅하는 변수가 위치해있는 패키지와 클래스 명, SHA-1 알고리즘에 입력으로 들어가는 데이터와 20 bytes의 해쉬 값이 실시간으로 DOS 창에 출력되고 있다. 본 논문에서는 이해를 돕기 위해 간단한 예제를 들어 설명하였지만, 실험에서 이용한 디버깅 레벨이 아닌 개발자 요구사항에 맞는 디버깅 레벨을 설정하여 디버깅 옵션을 자유자재로 설계하여 이용하는 것도 가능하다.

그림 7은 앞서 표 2에서 정의한 명령 APDU에 따른 카드의 응답 APDU를 보여주고 있다.

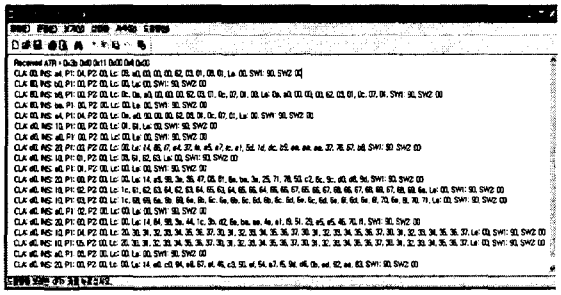

그림 7: 명령 APDU와 응답 APDU

V. 결론

본 논문에서는 자바 카드 운영체제 개발환경에 실시간 소스코드 디버깅 기능을 구현하는 방법과 구현된 디버깅 기능을 이용하여 실험하는 과정 및 그에 따른 결과를 보여주었다. 논문에서 제안한 실시간 소스코드 디버깅 방법은 개발자 관점을 고려하여 설계되었으며, 자바 카드 운영체제를 개발하는 개발자들이 자신의 요구사항에 맞는 디버깅 옵션을 설정하여 편리하게 소스코드를 디버깅할 수 있는 개발 환경을 제공하는데 목적을 두고 있다. 또한, 제안한 디버깅 기능을 이용함으로써 다양한 자바 카드 운영체제 및 응용 프로그램의 개발이 가속화될 수 있을 것이다.

참고문헌

[1] Michael Caentsch, "Java Card-From Hype to Reality," *IEEE Concurrency*, pp. 36-43, 1999.

[2] Chen, Zhiqun, Java Card Technology for Smart Cards, Addison-wesley, 2000.

[3] Sun Microsystems Inc., Java Card ™ 2.2 Application Programming Interface Specification, 2002.

[4] Sun Microsystems Inc., Java Card ™ 2.2 Runtime Environment Specification, 2002.

[5] Sun Microsystems Inc., Java Card ™ 2.2 Virtual Machine Specification, 2002.

[6] Menezed, A., van Oorschot, P., Vanstone, S., Handbook of Applied Cryptography, CRC Press, 1997.

[7] http://java.sun.com/products/javacard/