자바 카드를 위한 통합 개발 환경

한진희*°, 김시관**, 전성익*, 정교일*
한국전자통신연구원*, 금오공대**
(hanjh °, sijun, kyoil)@etri.re.kr*, sgkim99@naver.com**

An Integrated Development Environment for Java Card

Jin-Hee Han* °, Si-Kwan Kim**, Sung-Ik Jun*, Kyo-Il Jung*
Electronics and Telecommunications Research Institute*
KUMOH national university of technology**

요 약

이 논문에서는 자바 카드를 위한 통합 테스트 및 디버깅 환경에 대해 기술하고 있으며, 제안한 통합 개발 환경은 J-JCRE (Java language based Java Card Runtime Environment) 와 Java Card API (Application Programming Interface)로 구성되어 있다. 또한, 개발 환경은 현재 두 가지 암호 알고리즘을 지원해주며, 스크립트 실행, 시스템 클래스의 소스 레벨 디버깅, 클라이언트/서버 skeleton 자동 생성 등 다양한 기능을 제공해준다. 제안한 통합 개발 환경을 이용함으로써 사용자들은 실제 자바 카드에 응용 프로그램을 탑재하기 전에 응용 프로그램 테스트 및 디버깅을 통해 프로그램상의 오류 및 에러를 미리 점검하여 수정할 수 있게 된다.

1. 서 론

일반적으로 프로그램 작성 시 프로그램이 규격을 제대로 준수하고 있는지 정확하게 확인하기 위해서는 프로그램 테스트 및 디버그 절차가 개발 환경에 반드시 포함되어야 한다.

자바 카드 상에 탑재될 응용 프로그램 역시 자바 카드 규격을 만족해야 하기 때문에, Bull사의 *Odyssey lab™*, 슈럼버제사의 *Cyberflex™*, 젬플러스사의 *GemXpresso RAD™*, G&D사의 *Sm@rtCafé Professional™*, 그리고 Oberthur 카드 시스템사의 *GalatIC™* 과 같은 자바 카드 개발 툴이 다양하게 개발되어 사용되어왔다. 대부분의 개발 툴은 카드 리더와 카드, 소프트웨어 툴로 이루어져있다 [1].

시뮬레이션 환경은 대체로 실제 개발 환경에 비해 수행 속도가 느리지만, 스마트 카드의 경우엔 이와 정반대이다. 즉, 실제 카드의 수행 속도가 시뮬레이터에 비해 느리기 때문에 스마트 카드 응용 프로그램의 time-dependent 기능 이외의 모든 기능을 시뮬레이션 환경을 이용하여 테스트하고 디버깅할 수 있다. 이로 인해, 카드 응용 프로그램 개발자는 개발된 응용 프로그램을 실제 카드에 탑재하기 전에 시뮬레이터 환경에서 프로그램의 에러 및 오류 코드를 미리 점검, 수정할 수 있고, 이러한 개발 체계를 통해 응용 프로그램은 보다 신속하게 개발되어 time to market 이 단축되는 장점을 얻을 수 있게 된다.

본 논문에서는 자바 카드 응용 프로그램의 통합 테스트 및 디버깅 개발 환경에 대한 내용을 기술하고자 한다. 논문의 취지는 기존에 개발된 카드 개발 환경 제조업체와의 경쟁이 아닌 개발자에게 좀 더 편리하고 유용한 개발 환경을 제안하는데 있다. 논문의 구성은 다음과 같다. 2절에서는 자바 카드에 대해 간단히 소개하고, 3절에서는 제안한 시뮬레이션 개발 환경을 기술한다. 4절에서는 해쉬 알고리즘을 이용한 응용 프로그램의 테스트 및 디버깅, 그에 따른 결과를 보여주고, 마지막으로 5절에서 논문을 마무리 하고자 한다.

2. 자바 카드 (Java Card)

2.1 자바 카드 개요

1996년, 스마트 카드 제조업체인 슈럼버제사는 스마트 카드의 운영체제에 자바 바이트코드 인터프리터를 탑재하고, 카드에 적합하게 변환된 자바 클래스 파일을 다운로드 할 수 있는 자바 기반 스마트 카드를 제안하였다. 같은 해 10월에 Sun사는 자바 카드 규격을 최초로 발표하였는데, 초창기 규격은 자바 카드의 목적 및 구성에 대한 개략적인 내용으로 국한되어 있었다. 이후, 1997년에 보다 구체적인 내용을 포함한 자바 카드 2.0 규격이 발표되었으며, 1999년에 자바 카드 2.1, 2002년에 자바 카드 2.2 규격이 배포되었다 [2].

자바 카드는 중앙 처리 장치 (Central Processing Unit), ROM (Read Only Memory), EEPROM (Electrically Erasable Programmable ROM), RAM (Random Access Memory)과 같은 메모리가 존재하는 하드웨어 계층 위에 카드 운영 체제가 위치한다. 그리고, 카드 운영 체제 위에 자바 카드상에 다양한 응용 프로그램이 존재 할 수 있도록 지원해주는 자바 카드 가상 기계 (Java Card Virtual Machine)가 탑재되며, JCVM위로 자바 카드 API (Application Programming Interface), 자바 카드 애플릿 (Applet)이 순서대로 탑재된다. 그림 1은 이러한 자바 카드 내부 구조를 보여주고 있다.

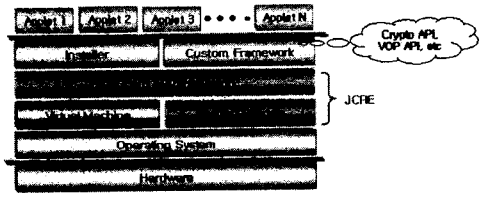


그림 1. 자바 카드 내부 구조

2.2 카드와 단말기간의 통신

카드와 단말기는 상호 통신을 위해 client-server 모델을 사용한다. 카드와 단말기의 메시지는 APDU (Application Protocol Data Unit)을 통해 전달되는데, APDU는 ISO 7816 표준에서 지정한 통신 프로토콜이다.

카드는 항상 단말기로부터 명령 APDU를 기다리며 수동적으로 동작하며, 명령어가 전달되면 해당 명령어를 처리하여 응답 APDU를 단말기에게 전송한다. 명령 APDU는 APDU의 상위 5 바이트를 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 [3].

표 1. 명령 및 응답 APDU 구조

명령 APDU						
명령 APDU 헤더				Lc	Data	Le
CLA	INS	P1	P2			
CLA: 클래스 바이트 (command-ID), INS: 명령어 코드 P1, P2: 파라미터, Lc: 명령 APDU 데이터 길이 Data: 명령APDU 데이터, Le: 응답 APDU상의 데이터 길이						

응답 APDU		
Data	SW1	SW2
Data: 응답 APDU 데이터, SW1, SW2: 상태 코드		

3. 자바 카드를 위한 통합 개발 환경

본 논문에서 제안하는 자바 카드 개발 환경은 크게 structure editor와 애플릿 테스트를 위한 시뮬레이터로 구성되고, 시뮬레이터는 또 다시 그림 2와 같은 시스템 모델로 표현된다.

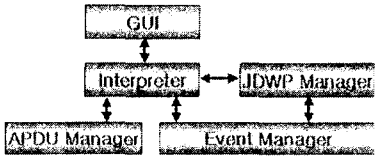


그림 2. 시뮬레이터의 시스템 구조

3.1 개요

그림 3에서 보는 바와 같이 시뮬레이터는 GUI (Graphic User Interface), 자바 카드 인터프리터, JDWP (Java Debug Wire Protocol) 관리자, APDU 관리자, 그리고 이벤트 관리자로 구성된다. JDWP는 debugger와 JCVM사이의 통신을 위해 사용되며 JCRE (Java Card Runtime Environment)상에서 개발된 후 수행되는 애플릿의 디버그를 위해 필요한 데이터 타입, 상수, 명령어들을 정의한다. JDWP 관리자는 디버깅 명령어를 JDWP 패킷으로 변환하여 적절한 명령어를 수행하고, CAP 파일로부터 다양한 디버깅 정보를 추출해낸다 [4][5][6].

자바 카드 인터프리터는 자바 카드 애플릿과 API에 의해 생성된 바이트 코드 명령어를 처리하며, 이벤트 관리자는 사용자의 명령어에 따라 생성되는 이벤트를 처리한다. 마지막으로 APDU 관리자는 명령 및 응답 APDU 처리를 담당하고 있다.

3.2 제안한 시뮬레이터 특징

통합 개발 환경은 아래와 같은 기능을 사용자에게 제공한다.

- ◆ 시스템 클래스의 소스 레벨 디버깅 기능
- ◆ 클라이언트/서버 애플릿 stub/skeleton 자동 생성 기능
- ◆ Cross-reference 트리 생성 기능
- ◆ 단말기 프로그램과 카드 애플릿간의 message tracing 기능
- ◆ 스크립트 실행 기능
- ◆ 변수 값 추적 및 감시 기능
- ◆ 애플릿 별 사용 자원 모니터링 및 보고 기능
- ◆ 다양한 암호 알고리즘 지원 기능
- ◆ 애플릿 컴파일, 컨버팅 및 mask generation 기능

그림 3. 시뮬레이터의 tool bar

시뮬레이터의 tool bar는 그림 3과 같이 구성되어있다. 각각의 구성요소는 새로운 자바 파일 생성, 자바 파일 열기, 자바 파일 저장, 새로운 프로젝트 시작, 프로젝트 열기, 프로젝트 파일 저장, 프로젝트에 새로운 파일 추가, 프로젝트로부터 파일 삭제, VM 일시중지, 스크립트 실행, 브레이크포인트 설정, 메시지 추적, 디버깅 관련 수행 기능들, 그리고 APDU toolkit 실행버튼으로 구분된다. 시뮬레이터의 utility 항목은 CAP파일 생성, 스크립트 실행, 바이트코드 이용 빈도수 및 프로파일 보기 등의 기능을, project 항목은 자바 파일 컴파일, 컨버팅, mask generation, 앞의 3가지 기능을 한번에 수행하는 make all 등의 기능을 구현하고 있다. 따라서, 애플릿 개발자는 시뮬레이터의 tool bar나 project, utility 항목을 적절히 이용하여 자바 카드 애플릿을 개발하고, 테스트 및 디버깅 할 수 있게 된다.

4. 실험 및 결과

이 절에서는 시뮬레이터에서 지원해주는 두 가지 암호 알고리즘 중 해쉬 함수를 이용하는 카드 애플릿을 작성하여 애플릿 테스트, 디버깅 및 그에 따른 결과를 기술하고자 한다. 현재, 시뮬레이터는 SEED와 세 가지 해쉬 알고리즘 - MD5, RIPEMD160, SHA-1 - 을 사용자에게 제공하고 있다 [7][8].

우선, 실험을 위한 테스트 애플릿을 작성해야 하는데, 아래 해쉬 알고리즘을 이용하는 애플릿을 간략히 보여주고 있다.

```
public class Md extends Applet {
.....
public void process(APDU apdu) {
byte[] buffer = apdu.getBuffer();
.....
switch (buffer[ISO7816.OFFSET_INS]) {
case MD5:  md5(apdu);  return;
case RIPEMD160:  ripemd160(apdu);  return;
case SHA1:  sha1(apdu);  return;
case RECEIVE:  receive(apdu);  return;
case SEND16:  send16(apdu);  return;
case SEND20:  send20(apdu);  return;
..... }
private void md5(APDU apdu) {
md5 = MessageDigest.getInstance(MessageDigest.ALG_MD5,true);
md5.doFinal(orMessage,0,(short)orMessage.length, mdMessage,16,0); }

private void ripemd160(APDU apdu) {
MessageDigest ripemd160 =
        MessageDigest.getInstance(MessageDigest.ALG_RIPEMD160,true);
ripemd160.doFinal(orMessage, 0,(short)orMessage.length, mdMessage,0);}
.....
}
```

애플릿 설계 및 작성이 완료되면, 해당 자바 파일을 컴파일, 컨버팅 한 후 , mask generation 절차를 거쳐 실험을 위한 환경을 갖춘다. 그림 4는 실험을 위해 작성한 애플릿을 시뮬레이터의 다양한 기능을 이용하여 테스트하고 디버깅하는 과정을, 그림 5는 해쉬 함수를 이용하는 테스트 애플릿의 실행 결과를 보여주고 있다.

실험을 위해 사용된 스크립트는 아래와 같이 구성되어 있으며, 아래 작성된 순서대로 카드에 입력된다.

- ♦ 인스톨러 애플릿 선택
- ♦ 인스톨러 시작
- ♦ 테스트 애플릿 인스턴스 생성
- ♦ 인스톨러 종료
- ♦ 테스트 애플릿 선택
- ♦ 입력 메시지 송신
- ♦ MD5 알고리즘 수행
- ♦ MD5 알고리즘 수행 후 해쉬 값 수신 (16 바이트)
- ♦ RIPEMD160 알고리즘 수행
- ♦ RIPEMD160 알고리즘 수행 후 해쉬 값 수신 (20 바이트)
- ♦ SHA-1 알고리즘 수행
- ♦ SHA-1 알고리즘 수행 후 해쉬 값 수신 (20 바이트)

해쉬 알고리즘에 사용된 입력 데이터 및 결과 값은 표 2에 나타내었다.

표 2. 해쉬 함수에 사용된 입력 및 출력 데이터

	입력	해쉬 값
MD5	abc	900150983cd24fb0d6963f7d28e17f72
RIPEMD 160	abc	8eb208f7e05d987a9b044a8e98c6b087f15a0bfc
SHA-1	abc	a9993e364706816aba3e25717850c26c9cd0d89d

5. 결 론

앞서 설명한 바와 같이, 프로그램이 해당 규격을 제대로 따르고 있는지 정확하게 측정하기 위해서는 프로그램 개발 과정에 테스트 단계가 필수적으로 구현되어야 한다. 이러한 개념을 바탕으로 본 논문에서는 자바 카드 응용 프로그램을 위한 통합 개발 환경을 제안하였으며, 간단히 시뮬레이터의 기능 및 디버깅 과정, 테스트 결과에 대해 기술하였다.

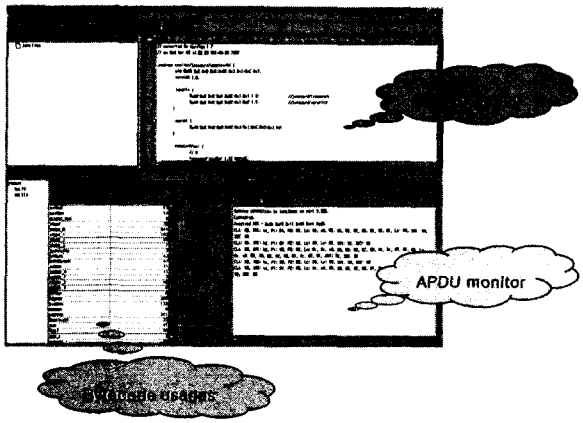
제안된 통합 개발 환경은 현재 2가지 암호 알고리즘을 제공하고 있으며, 자바 카드 애플릿을 위한 테스트 및 디버깅 기능도 다양하게 지원하고 있다.

향후, 개발 환경에 RSA, ECC (Elliptic Curve Cryptography)와 같은 공개키 알고리즘 및 다양한 암호 알고리즘을 추가 할 예정이며, Open Platform 2.1 규격도 구현할 것이다.

6. 참고문헌

[1] Isabelle, A. Et al., *An integrated development environment for Java Card*, Computer Networks, pp. 391-405, 2001.

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



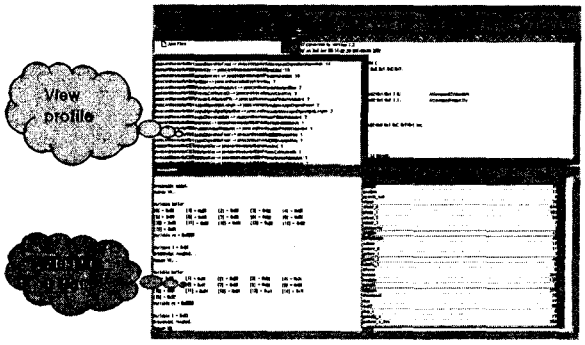


그림 4. 애플릿 테스트를 위한 다양한 디버깅 기능

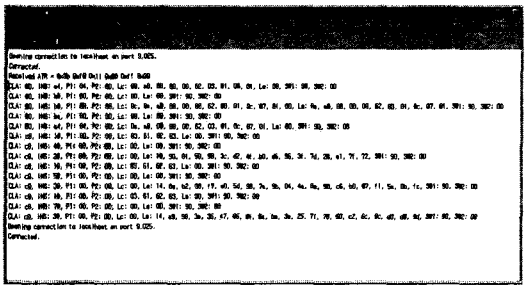


그림 5. 해쉬 함수를 이용하는 애플릿 테스트 결과

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

[4] Sun Microsystems Inc., *Java Card™ 2.1.2 Development Kit User's Guide*, 2001.

[5] Sun Microsystems Inc., *Java™ Debug Wire Protocol Java Card™ Extensions*, 2001.

[6] Sun Microsystems Inc., *Java Card™ 2.1.1 Runtime Environment Specification*, 2000.

[7] Sun Microsystems Inc., *Java Card™ 2.1.1 Application Programming Interface Specification*, 2000.

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