□기 고□

Java Card Platform을 내장한 Smart Card의 구현

한국전자통신연구원 김영진* · 전용성 · 전성익 · 정교일

1. 서 론

현재 smart card는 통신, 금융, 교통, 신분 확인, 전자 화폐 등의 여러 응용 서비스에서 널리 사용되고 있으며 점차로 그 용도가 확대되어 금융면에서는 기존의 신용 카드를 대체하고 다양한 형태의 장비를 통하여 네트워크와 연결되어 사용되는 추세로 바뀌고 있다[1]. 또한, 이와 더불어 빈번한 사용에 따라 개인 정보의 안전한 저장 및 사용, 사용자 인증을 포함한 정보 보안의 문제도 크게 부각되고 있다. 이러한 조류에 발맞추어 smart card의 하드웨어도 발전하고 있다. CPU는 기존의 8bit 위주에서 보다 처리 성능이 뛰어난 32bit로 전환되고, 데이터의 암 · 복호화를 빠르게 하기 위해 특정 암호 기법에 대한 전용보조프로세서의 사용이 늘고 있다. 또, 다양한 응용 서비스용 프로그램들을 수용하기 위한 저장 공간과 수행시 필요한 임시 사용 공간으로 ROM, EEPROM 및 RAM의 메모리 크기를 늘이고 있는 실정이다. 소프트웨어 측면에서도 다수의 응용 서비스용 프로그램 수용을 가능케 하고 하위의 하드웨어에 관계없이 응용 프로그램을 작성하고 수행하기 위한 운영 체제를 구축하는 작업이 진행되었다. 또, card가 발급된 이후에도 최종 사용자(End-User)가 원하는 응용 프로그램을 다시 card에 적재하여 수행할 수 있는 발급 후 적재(Post-issuance) 개념을 도입하였다.

Java Card platform을 내장한 smart card는 현재의 smart card에 적용되는 모든 표준을 따르는 전형적인 smart card인데 하위의 운영체제 위에 존재하는 Java Card 가상 기계(Virtual Machine)가 Java Card Applet의 바이트 코드(bytecode)를 수행하고 메모리, I/O 같은 smart card 내의 모든 자원에 대한 접근을 제어한다는 점에서 차이가 난다. Java Card 기술은 플랫폼간에 이진 코드의 이식성(portability) 즉, 상호 운용성(inter-operability)이 뛰어나고 타입 검사등에 의해 악의적 코드에 대한 보안성(security)을 지닌 Java 언어를 smart card의 실시간 환경에 대해 최적화하고 있다. Java Card smart card에서의 가상 기계 이용에 의한 장점은 응용 프로그램과 운영체제를 분리하는 개방형(Open-platform) 운영체제를 가져오며 하드웨어 의존적인 어셈블리 코드가 아닌 상위 언어인 Java 언어로 쉽게 응용 프로그램을 작성 및 수행할 수 있으며 카드가 최종 사용자에게 발급된 이후에도 필요한 응용 서비스에 따른 응용 프로그램을 smart card에 적재(Post-issuance)할 수 있다는 것이다[2]. 이에 따라 다양한 다수의 응용 프로그램(Multi-application)을 수용할 수 있는 유연성(flexibility)을 가지게 한다. 또한, Java Card에서는 Java 언어 자체의 보안 특성 이외에 응용 프로그램간의 방화벽 제공함으로써 엄격한 보안성을 보장한다. Java Card platform은 위에서 기술한, 향후 smart card에서 필요로 하고 있는 내용을 충분히 안정적으로 제공하고 있으며 Java 언어 자체의 폭 넓은 프로그래밍 자원 및 대중성에 힘입어 1999년에 ETSI(European Telecommunications Standards Institute)에 의해 GSM(Global System for Mobile telecommunication) 휴대폰 내에 SIM(Subscriber Identity Module) card의 표준으로 채택되는 등 smart card 응용 서비스 시장에서의 점진적인 점유율 증가를 나타내고 있다.

본 고에서는 Java Card 2.1 platform의 특징, 구성을 살펴보고 기존 smart card와 Java Card plat-

* 정회원

form을 내장한 smart card를 비교, 고찰하고 현재 전자통신연구원에서 개발하고 있는 Java Card 2.1 smart card 및 응용 서비스와 응용 프로그램에 대해 기술한다. 또, Java Card 2.1 platform에서 제기되는 여러 가지 논의점을 살펴보고 실제 card 개발시에 발생하는 문제점들에 대한 대처 개발 방향을 논의하고 끝을 맺는다.

2. Java Card platform 개요

Java Card platform은 Java 언어와 Java Card 가상기계를 기반으로 하여 내장형 시스템 중(embedded system) 중에서 메모리, 계산 처리 능력(computing power)이 가장 작은 규모에 속하는 smart card에 대해 적용된 시스템이다. 따라서, Java Card platform은 모태가 되는 Java platform에 비해 많은 제약 사항과 수정 및 추가 사항을 내포하고 있다. 본 장에서는 먼저 Java Card의 간단한 발전 과정을 기술하고, Java platform과 Java Card platform의 전반적인 사항을 비교하여 살펴 본다. 또, 각 platform에서의 중추를 담당하는 가상기계의 구성 및 내용에 대해 비교한다.

2.1 Java Card 2.1

1996년에 Java Card 표준이 발표된 이후, Java Card는 여전히 다른 Java Card platform간의 Applet의 이식성이나 상호 운용성이 보장되지 않았는데, 특히 다양한 smart card 판매업자들의 전용 파일 시스템을 수용하기 위한 일반적인 파일 시스템 API (Application Programming Interface) 구축이 불가했다. 또, 암호 API의 기능 및 유연성이 결여 되는 문제가 있었다. 1999년 3월에 발표된 Java Card 2.1에서는 이러한 내용이 수정 · 보완되어 소프트웨어적인 방화벽이 도입되었으며 파일 시스템 API에 대한 내용이 삭제되었고 암호 API가 다양한 class로 확장되었다[2]. 본 논문에서 기술하는 Java Card에 관련된 내용은 Java Card 2.1 platform을 기준으로 기술된다.

2.2 Java platform

Java language는 James Gosling 등에 의해 개발되어 1994년에 발표되었는데 처음에는 내장형 시스템을 겨냥한 간결한 code의 객체 지향 프로그래밍(OOP, Object Oriented Programming) 언어로서 개발되었다. 그러나, 이러한 간결성 및 호환성, 보안성이 인터넷상의 통신에 적합하여 인터넷을 위한 프로그래밍 언어로서 널리 사용되게 되었다.

Java 가상기계(JVM)가 Java 원천 코드를 compile한 결과인 class 화일 내의 바이트코드를 수행하는 엔진으로서 하위의 운영체제의 상위의 Layer로 존재하며, 수행엔진은 명령어(instruction)를 하나씩 수행하는 인터프리터(interprerter), 수행중 특정 기계 언어로 변화하도록 동적 컴파일(dynamically compile)을 수행하는 JIT(Just-in Time) 컴파일러 및 명령어 수행을 하드웨어를 이용하여 처리하는 Java 프로세서로 분류된다[3].

2.3 Java Card platform

메모리 및 CPU 처리 속도 등의 제약된 자원의 smart card를 겨냥하여 1996년에 탄생한 platform으로 Java Card 언어는 Java 언어의 subset이다. 개방형 플랫폼(Open-platform)을 지원하는 다중 응용 프로그램(Multi-application) smart card를 구현하는 운영체제의 상위 Layer로 Java Card 가상기계(JCVM)을 가진다. JCVM은 수행엔진으로 Java Card 바이트코드를 수행하는 인터프리터를 기본적으로 사용하며 메모리의 제한으로 인해 (특히, RAM으로 현재 최고 4k bytes 정도이며 Java Ring은 Java Card Applet을 사용하는 platform으로서 6K bytes의 RAM을 사용하는 것으로 알려져 있다.) 동적 컴파일러인 JIT 컴파일러는 실시간 수행시 각 메소드를 처음 수행할 때, 이를 컴파일한 특정 기계 코드를 임시 저장처에 보관하였다가 각 명령어 수행 대신 기계어 코드를 수행하기 때문에 많은 메모리를 요구하므로 사용이 불가하다.

2.4 JVM과 JCVM

Java Card 언어는 Java 언어의 subset인데 Java Card 바이트코드를 수행하는 수행 엔진인 JCVM은 JVM에 비해 smart card에 적합하도록 최적화되어 Applet간의 방화벽, 최적화된 명령어 등 Subset 개념이라기보다는 지원 내용은 작지만 별도의 새로운 수행환경을 구성한다. JCVM이 지원하는 내용을 JVM에 대해 나타내면 다음과 같다[4,5,8]. Smart card 상에서의 최적화를 위한 작업을 위한 내용이 JCVM에

반영되어 있음을 알 수 있다.

JVM		JCVM
- Dynamic loading	→	Pre-loading and Dynamic allocation
- ClassLoader	⟷	Off-card Installer (CAP; Convert APplet loader)
- Security Manager	⟷	Applet 간의 Firewall (context)
- Multi-thread	⟷	Single-thread
- Multi-dimensional array	⟷	1 dimensional array
- Garbage Collector	→	not supported
- char, String type support	→	not supported
- float, double support	→	not supported
- integer	→	optionally supported

2.5 JCVM의 구조

JCVM은 크게 Off-card VM과 On-card VM으로 나뉘는 분할 가상기계로 이루어지며, 좁은 개념으로는 수행 엔진인 인터프리터를 지칭하며 넓은 개념으로는 framework이 중심이 되는 system class API와 인터프리터, 메모리 관리 루틴, 예외처리 루틴 및 운영체제와의 interface 등을 포함하는 Java Card 수행환경(JCRE, Java Card Runtime Environment)을 의미한다. 분리된 JCVM의 구성은 그림 1에 보이고 있으며 각각의 VM의 기능을 간략히 기술하면 다음과 같다.

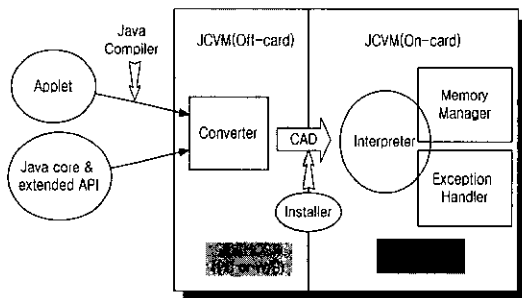

그림 1 분리된 JCVM의 구성

- Off-card VM
  - Java class loading, linking, name resolution 및 package 변환
  - verification(JCVM 사양에 대한 check 및 resolution 결과에 대한 check 포함)
  - 바이트코드 최적화
  - Java Card 파일 형식인 CAP파일(실행을 위한 바이너리 파일) 및 EXP파일(linking 및 verification을 위한 interface 바이너리 파일) 생성
- On-card VM
  - 바이트코드 수행 및 수행시 메모리 관리 (stack-machine을 에뮬레이트함)
  - 방화벽 루틴 수행

2.6 Java Card의 바이너리 형식

Java Card platform에서는 smart card에의 적재와 card상에서의 메모리 적재 및 링킹 등 운용의 편의를 위해, Java 컴파일러에 의해 생성된 class 파일을 그림 1에서 보는 것처럼 컨버터에 의해 여러 개의 컴포넌트(Component)로 구성된 CAP 파일로 변환하여 다운로드(download)한다. 컨버터는 Applet 및 class library들을 CAP 및 EXP 파일로 변환하는데, CAP은 실행하고자 하는 바이너리 파일이며 EXP는 CAP 내에서 이용하는 library들을 연결하여 위치를 알려주는데 이용되는 인터페이스 바이너리이다[5,8]. EXP 파일은 일반 class파일과 거의 유사한 반면 CAP 파일은 class, 메소드, 정적 변수 등의 분리된 11개의 컴포넌트로 구성되며 기존의 class 파일내의 스트링(String) 기반의 링킹이 아닌 컨버터에 의해 지정되는 토큰(token)과 오프셋에 의해 CAP 파일 내 바이트코드들의 내외부의 참조(reference)에 대한 링킹을 수행한다.

CAP 파일의 각 컴포넌트에 대한 간단한 기술을 하면 다음과 같다[5].

- Header 컴포넌트
  - 이 CAP 화일 및 이 화일내에 정의된 package에 대한 일반적인 정보를 가지고 있다.
  - Java Card CAP 화일을 나타내는 magic 값, 버전, package의 길이, AID(Applet IDentifier) 등을 포함한다.
- Directory 컴포넌트
  - 이 CAP 화일내의 각 컴포넌트의 크기 정보를 가지고 있다.
- Applet 컴포넌트
  - 이 package에 정의된 applet에 대한 정보를 가지고 있다.
  - 정의된 applet의 AID, install method offset을 포함한다.

- Import 컴포넌트
  - 이 package내에 정의된 각 class들에 의해 import되는 외부 package에 대한 정보를 가지고 있다.
- Constant Pool 컴포넌트
  - 이 CAP 화일의 Method 컴포넌트의 내용들 중 method의 instruction 또는 exception handler table의 exception handler catch type에서 참조되는 *class, field* 및 *method*에 대한 연결 정보를 가지고 있다.
  - 연결 정보는 class, *instance field, virtual* method, super method, static field 및 static method에 대하여 나타난다.
- Class 컴포넌트
  - 이 package에 정의된 class와 interface에 대한 정보를 가지고 있다.
  - class 정보는 super class, instance size과 Method Component로의 offset을 각 원소로 하는 virtual method table 등을 포함한다.
- Method 컴포넌트
  - 이 package에 정의된 exception handler와 각 method를 기술한다.
  - Java Card bytecode는 method 기술 내용에 나타난다.
- Static Field 컴포넌트
  - 이 package에 정의된 모든 static field 를 생성하고 초기화 하는데 필요한 정보를 가지고 있다.
- ReferenceLocation 컴포넌트
  - Method Component 내의 bytecode 중 opcode에 따르는 operand에 대한 offset list를 가지고 있다.
- Export 컴포넌트
  - 외부 package의 class가 import하는, 이 package의 모든 static 요소의 정보를 가지고 있다.
  - public class, public 또는 protected static field 및 method가 대상이 된다.
- Descriptor 컴포넌트
  - 선택적인 컴포넌트이며, 이 package에 정의된 class(또는 interface)에 대한 정보를 가지고 있다.
  - verify를 위한 class, field 및 method에 대한 type description 정보를 가지고 있다.

3. Java Card 2.1 embedded smart card의 구현

현재 개발중인 smart card는 32 bit RICS chip인 ARM7 TDMI를 사용하며 RAM 4k bytes, ROM 64k bytes, EEPROM 48k bytes를 목표로 하고 있으며 운영체제로서 Java Card 2.1 platform과 이에 인터페이스하는 OS를 가진다. 목표로 하는 하드웨어는 [6]의 8051 CPU, RAM 1.2k, ROM 32k, EEPROM 16k나 [7]의 H8 series CPU, RAM 2k, ROM 32k, EEPROM 16k에 비해 메모리가 크며 연산 성능이 뛰어나도록 설계되어 있다. smart card 구조는 그림 2에서 보는 바와 같으며 OS는 하드웨어에 의존하게 되며 JCVM과는 직접 또는 Native method를 통하여 인터페이스하게 된다. JCVM은 소프트웨어로 구현되며 JCVM은 Java Card 2.1 core API 및 extended API와 이를 이용하여 프로그래밍된 Applet을 수행하게 된다. Custom Framework API는 암호 API나 VOP(Visa Open Platform) API 등의 사용자 API 나 산업 규격용 API를 포함하는데, 구현할 암호 API는 공개키 암호 알고리즘인 RSA와 ECC(Elliptic Curve Cryptography)을 구현한 API와 대칭키 알고리즘인 DES, T-DES 및 SEED를 포함한다. 하드웨어는 RSA와 ECC용 전용 보조 프로세서를 포함한다.

이 장에서는 먼저 구현 및 개발 단계에 대해 언급하고 진행중인 단계를 smart card의 하드웨어와 소프트웨어 측면 및 off-card, 즉 host 측면에서 각각 기술하고자 한다.

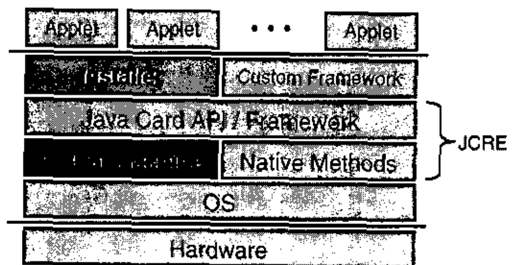

그림 2 Java Card 2.1 embedded smart card의 구조[5]

3.1 smart card의 개발 단계

개발중인 *smart card*는 그림 3에서 보는 바와 같

이 카드에 가까운 동작을 취하는 상태에 따라 대략 4 단계를 취하고 있다. 초반 단계에서는 개발에 있어서의 하드웨어에 대한 의존성이 적으므로 기능 및 동작에 대한 검증 환경이 쉽게 구성되고 디버깅에 대한 유연성이 크고 비용면에서 유리한 반면에 제한된 I/O 문제를 가지며 실시간 조건 과 실제 메모리 등에 대한 검증이 이루어지지 않는다. 이와 반대로 아래 단계들에서는 카드에서의 동작 조건이 반영된 시험 및 검증이 가능하나 디버깅 환경 구축에 대한 작업이 많이 필요하다.

첫번째 단계는 PC 또는 W/S상에서 JCVM을 포함한 Java Card 수행 환경을 소프트웨어적으로 구성하고 API 및 Applet을 수행하는 시뮬레이션 과정으로서 Java Card 바이트 코드 수행 환경 및 동작에 대한 시험, 검증을 할 수 있다.

두번째 단계는 ARM7 TDMI core를 사용하는 상용 보드인 AESA(ARM Emulator System board Assembly)에 JCVM 및 COS(Chip Operating System)를 포팅(porting)하고 UART(Universal Asynchronous Receiver/Transmitter)에 card adapter를 연결하여 card reader와의 I/O를 구성함으로써 실시간 기능 시험이 가능하다. 인터럽트 처리 및 레지스터 설정 · 사용 등에 대한 펌웨어(firmware)는 보드에서 제공되는 것을 이용하여 암호 처리 전용 프로세서와 같은 사용자 모듈을 통합, 사용하게 된다. 실제 IC카드와 같은 I/O가 아니므로 card adapter에서 부가적인 작업이 필요하며 PC에서의 강력한 디버깅을 지원하는 환경을 위한, 불필요한 보드상의 자원 할당으로 인해, 원하는 자원 할당이 제한되는 단점이 있다.

세번째 단계는 ARM 7 TDMI core를 제외한 모든 모듈을 자체 개발한 보드, IESA(IC card Emulator System board Assembly)상에 JCVM과 COS를 포팅하고 smart card에서와 같은 I/O를 이용하여 실시간 기능 시험을 수행하는 단계이다. smart card의 동작에 근접하도록 하기 위해서 자체 부팅 코드와 개발된 운영체제의 연결이 필요하며 인터럽트 처리, 내부 레지스터 운용등에서의 모든 처리가 필요하다. 또, 각 제어 모듈 및 암호 전용 프로세서 모듈 등의 최적화가 필요하며 실제 EEPROM 메모리 사용 등을 통한 실제적인 실시간 기능 시험을 할 수 있게 된다. 그러나, 여전히 보드 연동 통합 개발 환경도구를 이용해서 수행시 필요한 데이터와 코드 전체를 하나의 이미지 파일로 업로드해서 사용하므로 ROM, EEPROM 및 RAM에 대한 논리적인 메모리 구분으로 인해 실제 ROM 읽기/쓰기 등에 대한 운용 시험은 불가하다. 또한, Java Card에서 지원하는, 비휘발성 메모리에서의 데이터 update인 트랜잭션(transaction) 수행시에 전원 상실 때의 smart card의 데이터 복구 동작(rollback)과 같은 운용 시험이 불가하다. 현재 개발중인 단계는 세번째 단계이며 메모리 및 칩 면적 최적화 문제는 뒤에서 논의하기로 한다.

네번째 단계는 chip 공정에 의해 세번째 단계의 모든 모듈이 하나의 chip으로 구현되어 smart card를 만드는 과정이다. 이 단계는 실제 공정시 파라미터 및 공정 업체에 의해 유동적으로 결정되므로 언급을 생략한다.

실질적인 단계로서 여기에서 제외한 내용은, 세번째와 네번째 사이에 공정전의 chip 합성 및 시뮬레이션 도구를 이용한 전체적인 smart card 하드웨어 구성 및 소프트웨어 통합 시뮬레이션 과정이다. 이 단계는 공정시의 공정 파라미터로서 미리 chip 구조를 합성하여 운영체제 및 응용 프로그램 등을 메모리에 로딩하여 수행함으로써 검증하는 단계이다. 이 단계는 실질적인 공정 내용에 의한 smart card 동작에 대한 실시간 시뮬레이션이 가능하나 세번째 단계에서 검증할 수 있는 하드웨어 동작 측면에서의 smart card 내부 동작이나 card reader 간의 동작 검증이 힘든 것이 단점이다. 이 단계는 개발상에서의 검증의 단계로서의 의미가 크므로 실제 개발 단계에서는 제외함을 밝힌다. 또한, chip 공정이후의 chip card에 대한 H/W 및 S/W의 기본 시험 및 운용 시험에 대한 언급도 생략하기로 한다.

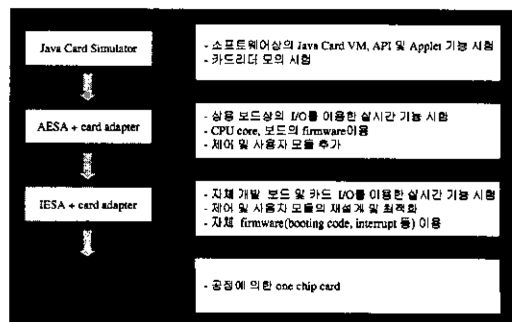

그림 3 smart card 개발 단계

3.2 Card의 하드웨어 부분

개발중인 Smart Card의 하드웨어는 현재 IESA 에뮬레이터 단계로서, CPU와 메모리 부분은 개별 소자를 사용하였고 이외의 Decoder, 메모리 인터페이스 등의 기본 모듈, 리더기와의 접촉 및 비접촉 통신을 위한 I/O Interface 모듈 그리고 암호 연산을 위한 전용 보조 프로세서 모듈 등은 FPGA로 구현하였다. CPU는 ARM사의 ARM7TDMI를 사용하고 있으며 이외의 하드웨어 모듈은 Smart Card에 적합하도록 구현하였으며 전체구성은 그림 4와 같다.

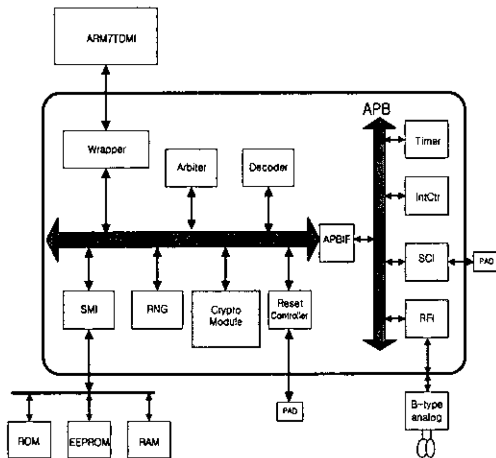

그림 4 Java Card 내장 smart card의 하드웨어 구조

3.2.1 IESA(IC card Emulator System Assembly)

Card의 하드웨어 시스템 구성은 System Bus와 연결된 ASB Bus 모듈과 Peripheral Bus와 연결된 APB Bus 모듈로 나눌 수 있다. ASB Bus 모듈은 주로 빠른 연산이 필요한 기능을 가지는 것으로서 memory interface(SMI), random number generator (RNG), 암호모듈 등이 여기에 속한다. SMI는 IC카드에 사용되는 ROM, EEPROM, RAM을 관리하는 모듈로서 메모리의 byte, halfword, word 단위의 access를 가능하게 해주며 제한된 memory protection 기능을 제공하고 있다. RNG는 난수 발생 모듈로서 32bit 의 난수를 발생시킨다. APB Bus 모듈은 주로 빠른 연산이 필요하지 않은 것으로서 Timer, 외부와의 통신을 위한 I/O 모듈 등이 여기에 속한다. 현재 개발중인 IC 카드는 접촉-비접촉 통신이 모두 가능한 smart card로서 접촉 통신 기능을 위해 ISO 7816 프로토콜을 지원하는 접촉 Interface 모듈(SCI)를 구현하였으며 비접촉 통신 기능을 위해 ISO 14443 Type-B 프로토콜을 지원하는 비접촉 Interface 모듈(RFI)를 구현하였다.

3.2.2 암호 전용 프로세서

차세대 IC카드에서는 대칭키 및 비대칭키, 서명, 해쉬 등의 암호 방식을 지원하기 때문에 높은 보안성을 가진다. 지원하는 암호 알고리즘으로는 SEED 및 triple-DES와 같은 대칭키 암호 알고리즘, RSA 및 타원 곡선 암호 알고리즘과 같은 비대칭키 암호 알고리즘, SHA-1과 같은 해쉬 함수, DSA와 같은 서명 함수를 지원한다.

RSA와 타원곡선 암호는 매우 큰 비트 값을 가지는 정수 연산을 필요로 하므로, 수행 시에 많은 시간과 높은 계산 능력을 필요로 한다. 이 때문에, 차세대 IC카드에서는 암호 처리 전용 보조프로세서를 사용하여, RSA와 타원 곡선 암호(ECC)를 고속으로 처리한다. RSA의 경우 1024bit RSA Encryption, Decryption 연산이 가능한 코프로세서를 설계하였다. 또한 ECC의 경우, RSA에서는 키 길이가 길기 때문에 연산 시간과 연산에 필요한 하드웨어가 늘어나는 단점을 극복하기 위해 163bit 타원곡선 암호연산이 가능한 코프로세서를 설계하였다.

3.3 Card의 소프트웨어 부분

Card의 소프트웨어 부분은 크게 JCVM, COS, API 및 응용 프로그램인 Applet으로 나눌 수 있다.

3.3.1 JCVM

구현중인 JCVM은 동적 자료 구조 측면에서 보면, CAP 파일내의 Java Card 바이트코드를 수행시키는 인터프리터를 중심으로 Java 스택 메모리 관리 루틴, 명령어가 저장된 데이터 공간이 유기적으로 연결되어 그림 5에서 보는 바와 같이 도시적으로 나타낼 수 있다. 이외에 방화벽 처리를 위한 컨텍스트(context) 처리 루틴 및 Native method 인터페이스가 포함된다.

• Method area 및 Heap

Method area는 ROM에 존재하며 API class library 또는 ROM에 탑재되는 Applet들의 CAP 파일 저장 구조에서 class와 메소드 내의 각 바이트코

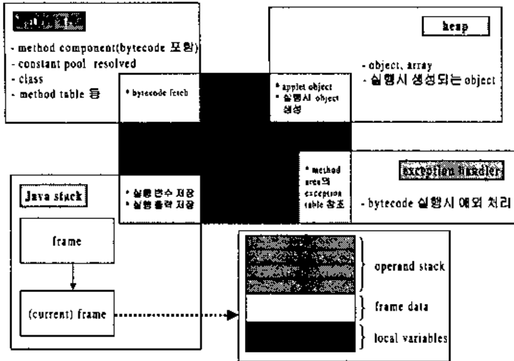

그림 5 JCVM의 인터프리터 및 주변 루틴(동적 자료 구조 중심)

드에서 필요한 class, 변수 또는 메소드에 대한 참조들을 주어진 토큰을 이용하여 레졸루션한 내용들을 포함한다. Method area에 대한 전역 레지스터로서 PC(Program Counter)를 두고 있다.

Heap은 EEPROM에 존재하며 바이트코드 수행시 필요한 class 또는 array의 객체를 생성한다. 또, Applet의 instance도 heap에 생성되고 table로 관리된다. 객체는 구별을 일반 타입 , 기본 배열 타입(primitive type) 또는 참조 배열 타입(reference type)을 구별하기 위한 헤더(header)를 가지며 class에 대한 포인터, 컨텍스트, 객체 소유의 데이터 크기 등이 포함된다.

• Java stack

Java Card platform은 단일 쓰레드(thread)이므로 하나의 Java stack을 가진다. 내부에서는 그림 5에 보는 바와 같이, 각 메소드 호출시에 넘겨받는 파라미터를 가지고서 새로운 프레임(frame)을 만들고 내부의 피연산자 스택과 로컬 변수에 초기값 및 연산 결과를 저장한다. COS 등의 외부와의 I/O가 필요한 경우는 Native method 인터페이스를 이용하여 JCVM이 볼 수 있는 메모리 주소내의(예를 들면, Java stack, APDU 버퍼 등) 값을 읽고 쓴다. 프레임에 대한 포인터로서 FP(Frame Pointer)를 두며 피연산자 스택에 대한 포인터로서 SP(Stack Pointer) 등의 가상 레지스터를 이용하고 있다.

• 수행 엔진

Java Card에서의 수행엔진은 소프트웨어 인터프리터로서, method area에서의 PC에 대응하는 하나의 바이트코드를 가져와서(fetch) 수행하면서 필요한 스택 연산을 수행한다. 각 메소드 간의 연결은 _invokeXX 명령어를 사용하며 타입에 따라 달라진다. 메소드 내는 각 명령어들이 연속적인 연결되어 수행된다. 메소드는 앞에서 기술한 바와 같이 프레임을 Java stack내에 새로 만들고 수행에 필요한 파라미터와 내용을 프레임내의 스택에 두고 이를 이용하여 명령어들을 수행한다.

• 인터럽트 핸들러

인터럽트는 JVM에서처럼 필요한 시점에서 새로이 인터럽트 처리 class의 객체를 생성하는 것이 아니라 JCVM 초기화 시에 미리 필요한 객체들을 생성시켜둔 뒤에, 적절한 인터럽트 타입에 따라 각 객체의 처리 함수를 호출한다.

• 방화벽

방화벽은 각각의 Java Card Applet 간의 분리된 data 저장 공간에 대한 접근 제어인 컨텍스트 변환 및 관리에 의해 구현되며 컨텍스트는 각 패키지(package)와 패키지 내의 Applet에 대한 정보를 결합하여 구성한다.

• Native method 인터페이스

Java Card API에서 수행되는 외부 I/O 또는 메모리 사용 함수에 대한 Native method를 JCVM에 구현하여 수행한다.

동작의 측면에서 APDU 처리와 관련하여 살펴보면, JCVM은 내부적으로 크게 초기화부분과 Java Card API 및 Applet을 수행하는 루프(loop)로 구성되며 API는 호스트의 터미날 프로그램(terminal program)과 맞물려 명령 APDU를 받아 들이고 처리하여 응답 APDU를 내보내는 내부 루프를 크게 구성한다. 이 과정은 도식적으로 표현한 것이 그림 6이다.

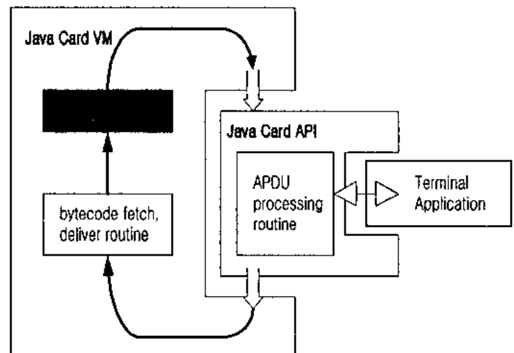

그림 6 JCVM의 동작 구조

3.3.2 COS

COS는 계층적으로는 H/W와 JCVM사이에 존재하며 기능상으로는 JCVM에서 요구하는 가상 메모리의 실제 메모리 입·출력 및 관리를 담당하며 시스템 함수 및 Care reader와의 통신과 같은 I/O 드라이버를 포함한다. 또, 전원이 공급될 때, H/W 초기화, 인터럽트 처리 벡터 처리 및 자체 스택 메모리 설정 루틴 등을 포함한 부팅 코드를 포함하여 구현하고 있다.

I/O측면에서 생각하면, Card Reader에 연결된 호스트 프로그램과의 통신 프로토콜인 APDU(Application Protocol Data Unit)의 내용 분석 및 처리는 API, JCVM을 포함한 JCRE에 의해 이루어지며 H/W에 인터페이스하여 APDU를 구성하는 데이터를 직접 송·수신하는 것은 COS에서 담당한다. 또한, COS는 전원 공급시의 리셋 신호에 대한 응답인 ATR(Answer-To-Reset) 송신 루틴을 구현하고 있다.

3.3.3 API

구현된 Java Card API는 크게 lang, framework, security class들을 포함하는 core API 와 암호 API와 같은 사용자 API로 나뉜다. Applet 의 안전한 적재 및 관리를 위한 VOP API는 구현중에 있으며 이에 대해서는 4장에서 언급한다. core API는 Java 언어 기능을 구현하는 기본 API, Java Card 기능 API, 시스템 API와 예외처리 API 등으로 구성되며 구현된 암호 API는 DES, 3-DES 및 RSA(1024bit 키)와 해시 함수 MD5, SHA-1, RIPEMD160이며, 암호 모듈과의 인터페이스를 고려하여 타원 곡선 암호알고리즘(ECC) 구현을 진행하고 있다.

3.3.4 Applet

Applet은 Applet API class를 반드시 계승하여야 하므로 기본적으로 install(), select(), process() 및 deselect() 메소드를 가지며 install() 메소드가 Applet에 대한 대외적인 창구 함수가 된다. Card에 ROM화 되거나 CAP 파일 형태로 탑재되는 Applet은 사용될 응용 서비스에 의존하게 된다. 모든 package는 고유한 식별 이름인 AID가 할당되어 Card에 적재되며 각 package내의 Applet도 역시 고유한 식별 이름인 AID가 할당되며 선택되어 사용시에 반드시 AID로서 접근된다. Java Card에서 사용되는 AID는 5바이트의 RID(Resource Identifier) 와 11바이트 이하의 PIX(Proprietary Identifier eXtension)로 구성된다[4]. IESA상의 Java Card platform에서 적재되어 사용되는 package 및 Applet에 대한 AID 할당 예는 그림 7에서 보는 바와 같으며 Md는 해시 함수 시험용 Applet 파일 이름이다.

package	com.sun.javacard.samples.Md	
	RID	PIX
	0xa0 0x0 0x0 0x0 0x62	0x3 0x1 0xc 0x7

Applet	com.sun.javacard.samples.Md.Md	
	RID	PIX
	0xa0 0x0 0x0 0x0 0x62	0x3 0x1 0xc 0x7 0x1

그림 7 package 및 Applet의 AID 할당 예

3.4 호스트(host) 부분

개발하고 있는 호스트 부분의 프로그램은 크게 Java Card 발급 및 시험 프로그램과 시범용 응용 서비스 프로그램으로 나뉜다.

3.4.1 발급, 시험 프로그램

Java Card 발급 프로그램은 Applet의 card 발급 후 적재를 위한 프로그램으로 On-card installer에 대응되는 CAP 파일 로더(loader)이며 필요한 초기화 데이터를 함께 로드하게 된다. 물론 발급 프로그램 또한 Applet과 마찬가지로 구축하는 응용 서비스에 맞추어 구현되어야 한다. 시험 프로그램은 API 또는 Applet에서 지원하는 APDU 명령을 단수 또는 복수로 card로 전달하여 응답 APDU를 받음으로써 card의 기능을 시험하는 프로그램이다. 현재 개별 APDU 명령 기반 프로그램과 APDU 배치(batch) 처리 프로그램을 작성하여 시험하고 있다.

3.4.2 응용 서비스 프로그램

시범용 응용 서비스는 작년에 이은 뱅킹 응용 서비스와 RF ID(Identifiaction) 서비스를 자체적으로 모두 개발 중에 있으며 전자 지갑 기능, 은행 계좌 연동 입·출금 기능, 로열티 기능 등이 구현 예정 내용이다.

4. 현안 및 논의

이 장에서는 Java Card 탑재 smart card 개발 중에 쟁점이 되고 있는 내용 및 필수적인 내용에 대해 논의하고 이에 대처하는 개발 방향에 대해 기술하고자 한다.

4.1 메모리 및 칩 크기 최적화

실제 smart card내의 chip 크기는 ISO 7816 규격에 따라 $25mm^2$이내이어야 하는데, 목표로 하는 RAM, ROM, EEPROM 크기 및 IESA상에서 구현되어 동작하는 여러 모듈들의 로직 크기가 이 크기 이내로 가능하기 위해서는 실제 메모리 크기 및 각 하드웨어 모듈들의 구현 크기가 최적화 되어야 한다. 특히, 실제 필요한 메모리 크기는 card 운영체제 및 응용 프로그램과 연관이 매우 깊으므로 소프트웨어 측면에서의 최적화 작업이 상당히 중요하다.

JCVM 및 주변 루틴들은 ARM용 C로 되어 있으며 하위의 COS는 ARM용 C와 Thumb Assembly로 되어 있으며 ARM 방식과 Thumb 방식 연동(interwork) 모드로서 오브젝트 파일을 생성하고 있다. JCVM 및 COS 각각에 대한 코드 및 메모리 사용에 대한 최적화 작업은 ARM용 C 컴파일러의 적절한 최적화 옵션을 이용한 Thumb Assembly의 복합 사용을 더욱 늘이는 방향에서 계속되어야 할 것으로 본다. 이것은 ARM용 C 컴파일러가 충분히 코드 최적화를 수행하고 있어서 프로그래밍에 많은 시간과 노력이 드는 Assembly 사용이 덜 필요하기 때문이며, Assembly를 사용하는 경우는 ARM 7 TDMI가 지원하는 16bit Thumb 명령어 방식은 일부 제약 기능[예를 들면, 명령어의 피연산자(operand) 오프셋(offset)의 범위 제한]이 있지만 메모리 크기를 ARM 방식에 비해, 줄이면서도 내부에서는 32bit로 동작을 하므로 처리 속도는 같으면서도 메모리 차지 용량은 감소시킨다는 장점이 있기 때문이다.

하드웨어 모듈 중에서 암호 API와 더불어 필요한 암・복호화를 수행하는 암호 전용 프로세서의 로직 수에 대한 최적화도 큰 문제로 작용한다. 이것은 암호 API와의 기능 설계 및 분배에 직접적으로 관련되어 있으므로 신중한 판단이 요구된다. 그 이유는, 암호 API에서 소프트웨어적으로 많이 처리하면 ROM에서의 코드 크기가 커지고 처리 속도가 느려지는 반면에 전용 프로세서에서 많은 연산을 수행하도록 하면 속도는 빨라지는 대신에 구현을 위한 전체 로직수가 늘어나 결과적으로 칩 크기에 영향을 주기 때문이다. 다만, CPU 자체를 기존의 8bit 보다 연산 처리 성능이 뛰어난 32bit를 사용하므로 소프트웨어적인 암호 처리에 대해서, 기존의 처리보다 신속한 연산을 기대할 수 있을 것이다.

4.2 Java Card 바이트코드 및 인터프리터 최적화

실제 smart card내에서 수행되는 내용은 ROM에 존재하는 API들의 바이트코드로서 바이트코드의 최적화 및 이의 수행 엔진인 인터프리터의 최적화문제는 ROM내의 메모리 차지 공간뿐 아니라 card 성능에 영향을 미치는 중요한 문제이다. 최적화를 고려할 수 있는 부분은 최적의 Java class 생성, Off-card VM에서의 CAP 파일로의 변환시의 최적화, CAP 및 EXP 파일간의 링킹후 최적화 문제를 들 수가 있다. 특히, ROM 상에 존재하는 API들의 Java Card 바이트코드는 링킹 후에도 불필요한 코드의 반복이나 불필요한 Java 스택(stack) 동작을 가져오는 코드들의 연결이 있으므로 Off-card에서의 여러 가지 최적화 기법들이 요구된다. 또한, On-card에서도 인터프리터를 최적화하는 작업을 통해서 스택 구조를 개선하고 가상 메소드(virtual method) 호출에 대한 오버헤드(overhead)를 줄임으로써 메모리 크기 및 속도를 동시에 개선할 수 있는 연구가 필요하다. 실제 수행의 측면에서 살펴보면, Java Card VM은 스택 기반 가상 기계이므로 실제 동작이 일어나는 레지스터 기반의 CPU와의 매칭(matching)이 성능에 큰 영향을 준다. [6]에서는 4개의 레지스터를 이용하여 스택 크기를 줄임으로써 그만큼 스택 사용 횟수를 줄이고 있다.

4.3 VOP(Visa Open Platform)

Java Card 탑재 smart card에서의 Applet의 안전한 적재 및 이의 효율적인 관리를 위해 Visa에 의해 개발된 VOP는 현재 GlobalPlatform에 의해 관리되고 있으며 OP로 공식명이 바뀌었다. OP는 다중 응용 프로그램용 smart card의 관리에 대한 표준으로서 자리 매김을 하고 있어 Java Card에서는 필수적이다

[2]. OP는 구현측면에서 Card상의 Applet 적재 및 관리를 포함한 content management, security management를 담당하는 Card Manager와 암호화 서비스를 위한 key management 응용 프로그램인 Security Domain가 주축을 이루어 Java Card API와 결합하여 구현할 수 있는데 Card와 호스트간의 상호 인증(mutual authenticate)를 위해 API 또는 COS에서의 난수 발생기(RNG, Random Number Generator)와 3-DES 암호 알고리즘의 사용이 요구된다. Applet delete을 구현하기 위해서는 JCRE에서 관리하는 Applet table에서의 Appet 상태(state)를 APPLET_LOGICALLY_DELETED으로 설정한다. 실제 Card에서의 사용시는 Applet 인스톨(install)시에 Applet 객체 및 인스턴스(instance) 변수의 생성이 EEPROM 상에서 이루어 지므로, 완전한 Applet 삭제 기능의 구현을 위해서는 JCRE가 COS의 메모리 관리 루틴에 이들에 대한 삭제 정보 등을 제공하여 EEPROM 메모리의 재사용이 가능하도록 해야 하는 점을 염두에 두어야 한다. 이것은 결국 가비지 콜렉션(garbage collection)의 간단한 기능이라고 볼 수 있으며 제한된 자원에 대해 효율적인 메모리의 사용을 위해서는 추후에 smart card에 적합한 효율적인 가비지 콜렉션 기법에 대한 연구도 필요할 것이다.

5. 맺음말

본 고에서는 Java Card platform에 대한 내용을 구현의 측면에서, 전반적으로 살펴 보았으며 이 내용을 바탕으로 실제로 개발 중인 Java Card 2.1 탑재 smart card에 대해 기술하였다. 또한, 지금 현안이 되고 있는 내용들을 제시하고 논의해 보았다.

Java Card는 쉬우면서도 강력한 객체 언어인 Java 언어를 바탕으로 하여 개발된 platform으로서, 지속적인 버전업(version-up)을 통한 platform의 안정성과 유연성을 더욱 높이고 있고 네트워크에 맞물리는 인터페이스와의 연계가 쉬워서 향후 smart card 시장에서 더욱 널리 사용될 것으로 판단된다. 또한, 향후 신용 카드가 smart card로 바뀔 것을 감안한다면 Java Card 탑재 smart card의 개발은 국내에서의 Java Card 기술의 확보라는 측면에서 매우 의미 있는 일로 여겨지며 나아가 경쟁력있는 smart card를 만들기 위한 많은 작업과 연구가 요구된다.

참고문헌

[1] 한국전자통신연구원, 스마트 카드 기술 사장 보고서, 1999.12.

[2] Michael B., Peter B., Thomas E., Frank H., Marcus O., "JavaCard-From Hype to Realty", IEEE Concurrency, Oct - Dec. 1999.

[3] Vijaykrishnan N., Ranganathan N., Gadekarla R., "Object-Oriented Architectural Support for a Java Processor", ECOOP '98, LNCS 1445, pp. 330-355, 1998.

[4] Java Card™ 2.1 Virtual Machine Specification, Sun Microsystems, Inc., Final Revision 1.1, June 7, 1999.

[5] Joshua S., Judy S., "The Java Card™ Virtual Machine", JavaOne Conference, 1999.

[6] Michael B. et al., JetZ, IBM Technical Report, 1999.

[7] LIU S., MAO Z., YE Y., "Implementation of Java Card Virtual Machine", Journal of Computer Science and Technology, vol. 15, no. 6, Nov. 2000.

[8] Zhiqun Chen, Java Card™ Technology for Smart Cards : Architecture and Programmer's Guide, ADDISON-WESLEY, 2000.

김 영 진

1997 서울대학교 전기공학부(공학사)
1999 서울대학교 전기공학부(공학석사)
1999~현재 한국전자통신연구원 정보 보호연구본부 IC카드OS연구팀 연구원
관심분야:IC Card, Java Card VM, Java VM, 객체지향언어 등
E-mail:youngjk@etri.re.kr

전 용 성

1990 경북대학교 전자공학과(공학사)
1992 경북대학교 전자공학과(공학석사)
1992~1999 국방과학연구소 선임연구원
1999~현재 한국전자통신연구원 정보보호연구본부 IC카드OS연구팀 선임연구원
관심분야:IC Card, Security, Security, 생체인식, 영상처리 등
E-mail:ysjeon@etri.re.kr

전 성 익

1985 중앙대학교 전산과(이학사)
1987 중앙대학교 전자계산학과(공학석사)
1987~현재 한국전자통신연구원 IC카드OS연구팀 팀장 책임연구원
관심분야:IC Card 기술 , 실시간 시스템소프트웨어, 객체지향시스템, 병렬처리 등
E-mail:sijun@etri.re.kr

정 교 일

1981 한양대학교 전자공학과(공학사)
1983 한양대학교 산업대학원 전자계산학과(공학석사)
1997 한양대학교 대학원 전자공학과(공학박사)
1981~현재 한국전자통신연구원 정보보호연구본부 기반연구부 부장 책임연구원
관심분야:IC Card, Security, Biometry, 정보전, 신호처리 등
E-mail:kyoil@etri.re.kr

● 제28회 정기총회 및 추계학술발표회 ●

- 일 자 : 2001년 10월 19일(금) ~ 20일(토)
- 장 소 : 서울여자대학교
- 논문모집 및 발표일정
  1) 접수기간 : 2001년 8월 6일(월) ~ 25일(토)
  2) 심사결과 통보 : 2001년 9월 15일(토)
  3) 수정논문 접수마감 : 2001년 9월 22일(토)
  4) 논문발표 : 2001년 10월 19일(금), 10월 20일(토)
- 문 의 처 : 한국정보과학회 사무국
  Tel. 02-588-9246/7, 4001/2
  http://www.kiss.or.kr, E-mail:kiss@kiss.or.kr