「정보학연구」 제10권 제3호 2007년 9월(pp.59-75)
The Journal of Information Technology, Vol.10, No.3

자바카드기반 파일 시스템 API의 설계 및 구현

Design and Implementation of File System API based on Java Card

송 영 상 (Song, Young Sang)* · 이 지 영(Lee, Ji Young)**

목 차

Abstract

Java Card has several applets running on card, and applet can be added easily. Each applet supports different application programs, and includes file system for data management. This paper presents design and implementation of the file system API based on Java Card for the efficient data management in need of the applet. We referred the smart card international standard ISO7816-4 for the file system and used API supported at Java Card. Results show that it is easy to design and operation of file systems during the applet development, also show lighting of applet's source codes and improved access times.

Key words: Java Card, File System, Smart Card, Applet

* 인천시립전문대 겸임교수
** 세명대학교 컴퓨터학부 교수

Ⅰ. 서론

최근 인터넷 및 정보 통신의 발달로 인해 정보 보호에 관심이 높아져 가고 있는 추세이며, 개인의 정보를 보호하기 위한 많은 제품 개발 및 연구가 진행되고 있다. 가장 일반적인 소지 형태가 되어 가고 있는 스마트카드는 일반 플라스틱 카드에 마이크로프로세서와 메모리 시스템을 내장하고 있는 칩을 갖고 있어 자체적인 계산 능력과 데이터 저장 능력을 가지며 이로 인해 뛰어난 보안성과 다양한 분야에서의 응용이 가능하다. 이러한 스마트카드의 특성을 이용하여 개인 신분 증명, 접근제어 등 금융 분야 및 여러 응용 분야에서 활발히 사용되고 있다.[1]~[5]

그러나 기존 스마트카드는 발급된 이후 코드의 수정 및 추가를 통해 새로운 응용 분야에 확장 하는 것이 매우 어려우며, 또한 응용 프로그램을 작성하는데 많은 비용과 시간이 소요되는 단점이 있다. 이를 보완하기 위해 최근 스마트카드 플랫폼에 자바 가상머신을 탑재한 자바카드가 각광 받고 있는 추세이다.[3]~[15]

자바카드는 스마트카드 내에 독립적인 플랫폼이 탑재되어 하위의 운영체제 위에 자바카드 가상 기계(JCVM: Virtual Machine)가 자바카드 애플릿의 바이트코드(byte code)를 해석 및 수행하고 메모리, I/O같은 카드내의 모든 자원에 대한 접근을 제어 한다. 자바카드의 특징으로는 플랫폼의 독립성과 국제 표준인 ISO7816과 산업 표준인 EMV와 호환하는 호환성, 카드가 발급된 이후에 원하는 응용 프로그램을 다시 카드에 쉽게 적재 하여 수행할 수 있는 발급 후 적재(post-issuance)의 특징이 있다. 또한 다양한 다수의 응용 프로그램을 수용할 수 있는 유연성(flexibility)을 가지고 있다.[4]~[10]

자바카드에서 동작하는 응용 프로그램을 애플릿이라 한다. 애플릿을 구현 하기위해서는 SUN사에서 제공하는 자바카드 API (Application Program Interface)를 이용하여 설계와 구현된다. 그러나 SUN사에서 제공하는 API에는 ISO7816에서 정의하고 있는 파일 구조를 구현할 수 있는 패키지 및 API를 제공하지 않고 있다.[18] 애플릿 개발 시 대부분 파일 시스템은 바이트 배열 및 변수를 사용하여 옵셋 및 파일 크기에 대한 제어를 통해 각 애플릿에서 복잡하게 구현된다.

본 논문에서는 파일 시스템을 이용하는 자바카드 애플릿 설계를 위해 스마트카드의 4가지 파일 구조를 자바카드에서 모두 지원할 수 있는 파일 시스템 패키지 및 클래스를 설계 및 구현하였다. 설계는 각 파일 구조에 맞게 파일의 길이와 크기 및 메모리에 쓰여 지는 위치를 고려하여 설계하였고, 구현은 자바카드에서 제공하는 API를 이용하였다.

파일 시스템 API를 이용한 자바카드 응용 프로그램의 애플릿 개발 시 단일화된 파일 입출력 인터페이스를 제공할 수 있어 애플릿 개발 기간을 단축 할 수 있으며, 애플릿 코드 사이즈를 줄여 메모리 관리를 효율적으로 할 수 있고, 또한 다양한 애플릿 구현 시 각 애플릿의 용도

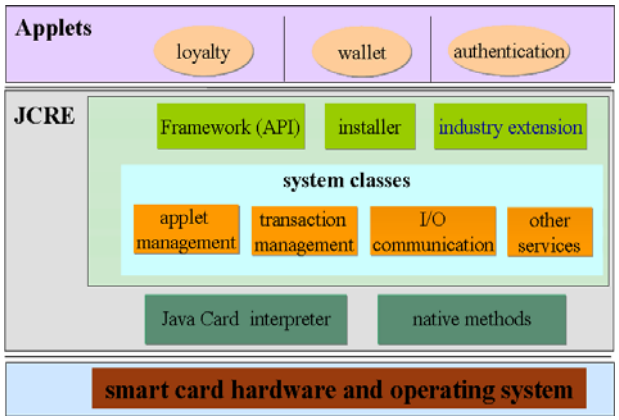
에 따라 파일 시스템을 각각 구현해야하는 단점을 보완 할 수 있게 된다.

Ⅱ. 자바카드

1. 자바카드

자바카드란 스마트카드 기술에 자바의 기술을 접목시킨 것이다. 자바카드 기술은 스마트카드의 메모리, 통신, 보안, 그리고 어플리케이션의 실행 모델을 지원한다. 일반적으로 자바카드의 메모리는 보통 1K의 RAM과 16K의 EEPROM, 그리고 24K의 ROM으로 구성된다. 자바카드는 자바 언어의 특징을 부분적으로 지원하고 이를 실행시킬 수 있는 가상머신(JCVM : java card virtual machine)를 필요로 한다.

JCVM은 Off-Card VM과 On-Card VM으로 나뉘는 분할 가상기계로 이루어진다. 이는 수행 엔진인 인터프리터를 지칭하며 넓은 의미로는 프래임워크가 중심이 되는 시스템 클래스 API와 인터프리터, 메모리 관리 루틴, 예외 처리 루틴 및 운영체제와의 인터페이스 등을 포함하는 자바카드 수행 환경(JCRE : Java Card Runtime Environment)을 의미한다.



<그림 1> 자바카드 구조

<Fig. 1> Java Card Structure

그림 1은 자바카드의 일반적인 구조를 나타내고 있다. 최하위 단은 자바카드의 하드웨어와 COS(card operating system)가 위치한다. 그 윗단에는 JCRE가 위치하여 최상위 애플릿을 동작시키기 위한 작업을 수행하게 된다. JCRE는 byte-code 해석기, 기본 API , industry specific extensions, 그리고 JCRE 시스템 class들로 구성된다. JCRE가 각 컴포넌트들이 적절히 분리되

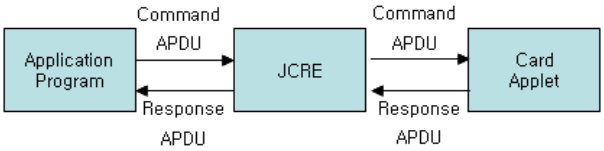
어 있고 잘 정의되어 있기 때문에 애플릿들이 다양한 스마트카드 플랫폼에서 동작할 수 있게 된다. JCRE는 명백하게 카드시스템과 어플리케이션 사이를 분리해서 설명하고 있다. 실제 어플리케이션 제작은 잘 정의된 인터페이스를 사용해서 만들어 진다.

2. 자바카드 API

자바카드 API는 스마트카드 표준인 ISO7816 에 의거해서 자바카드 애플릿 개발을 위해 만들어진 클래스들의 집합으로 이루어져있으며 3개의 핵심 패키지 java.lang, javacard.framework, java card.security와 1개의 확장 패키지 javacardx. crypto 로 이루어져있다. 그러나 자바 플랫폼 클래스들은 GUI 인터페이스, network I/O, 데스크탑 파일 시스템은 지원되지 않는다.

3. 자바카드 애플릿

자바카드는 여러 개의 응용 프로그램인 애플릿을 탑재 할 수 있으며, JCRE 내에서 규칙에 따라 동작하는 자바 프로그램이다. 애플릿은 ROM에 설치될 필요 없이 카드의 EEPROM에 다운로드 함으로써 사용가능 하다. 애플릿의 구분을 위해서 5byte 이상의 값인 AID(Application Identifier)가 주어지며, APDU(Application Program Data Unit)교환을 통해 자바카드의 JCRE와 응용 프로그램 과 통신한다.

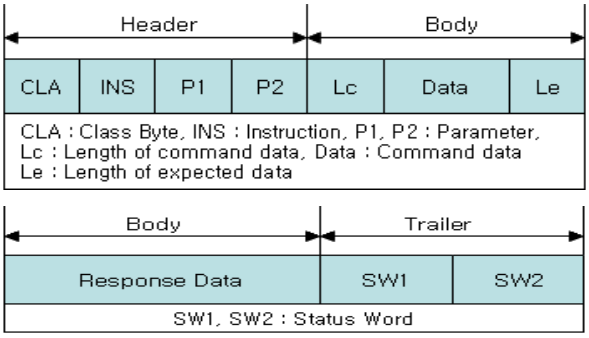


<그림 2> 애플릿 통신

<Fig. 2> Applet Communication

애플릿과 외부 응용프로그램간의 통신은 명령어(commend)APDU와 응답(response)APDU로 구성되어 APDU 교환을 통해서 이루어진다. 그림 2는 카드와 응용프로그램간의 통신을 나타내고 있다. APDU의 구조는 스마트카드 표준인 ISO7816-4에 규정되어 있다. 또한 APDU의 구조는 필수 항목인 Header 부분과 선택 항목인 Body 부분으로 나눌 수 있다.

그림 3은 APUD의 구조 형태를 보여 주고 있다. 명령 APDU의 Header 부분은 클래스(CLA), 명령어(INS), 파라미터(P1, P2)로 총4byte로 나타내고 Body부분은 전송데이터의 길이(Lc), 데이터, 응답데이터 길이(Le)로 데이터는 가변적으로 구성되어 있다. 응답 APDU의 body는 응답데이터, 상태를 표시하는 trailer로 구성되어 있다.



<그림 3> 명령, 응답 APDU

<Fig. 3> Command, Response APDU

자바카드 애플릿 설계는 javacard.framework. Applet 클래스를 상속 받아 구현된다. JCRE가 여러 개의 애플릿을 지원하므로 하나의 카드 상에 함께 존재 할 수 있으며, 여러 개의 인스턴스를 가질 수 있다. 애플릿 개발 시 JCRE에서 지원하는 API를 이용할 수 있을 뿐만 아니라 API를 설계하여 사용할 수 있다. 본 논문에서는 파일 시스템 API를 설계 및 구현하였다.

Ⅲ. 자바카드 파일 시스템 설계

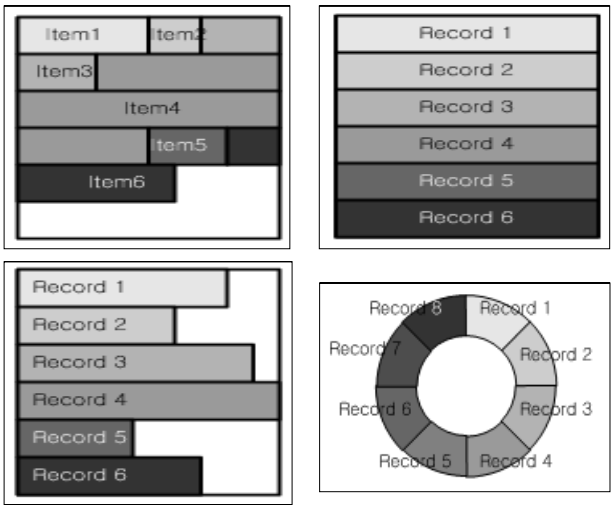
자바카드 애플릿은 여러 응용부분의 특성에 따라 설계 되어진다. 사용되는 애플릿이 데이터를 처리 하는 방법으로는 데이터를 순차적으로 메모리에 기록하는 방법과 데이터 묶음인 레코드 형태로 기록하는 방법으로 나눌 수 있다. 스마트카드에서는 이를 ISO7816-4에서 파일 구조를 정의하여 사용하고 있다. 그러나 자바카드는 국제 표준인 ISO7816과 EMV 표준과 호환된다고 하나 파일 시스템에 대한 클래스 및 API를 자바카드 API에서는 제공하지 않고 있다. 개발자가 애플릿을 설계, 구현할 때 매번 응용프로그램에 적합한 형태로 파일 시스템을 프로그래밍 하여야 하고 각 애플릿마다 구현 하는데 있어 시간 및 비용을 낭비하는 단점이 있다.

본 논문에서는 스마트카드 ISO7816-4에 정의된 4가지 파일시스템을 자바카드 애플릿 설계 및 구현 시 사용가능하도록 확장 API를 설계 및 구현하였다. 우선 스마트카드 ISO7816-4에 정의 되어 있는 파일 시스템을 살펴보면 다음과 같다.

1. 스마트카드 파일 구조

스마트카드 파일 시스템은 MF(master file), DF(dedicated file), EF(elementary file)로 구성된다. 그 중 실질적인 데이터는 EF에 기록되어 진다. EF의 구조는 데이터를 순차적으로 처

리하는 transparent와 개별적으로 동일한 구조인 레코드 구조로 나눌 수 있다. 레코드의 사이즈에 따라 고정적인 것과 가변적인 것으로 볼 수 있으며, 레코드의 구조는 linear한 것과 cyclic 구조로 된 것으로 구분할 수 있다. 그러므로 스마트카드의 EF의 구조는 transparent, linear fixed, linear variable, cyclic record의 총 4개의 파일구조로 이루어 져 있으며 이중 하나를 선택해서 사용한다. 그림 4.은 스마트카드 파일 구조의 4가지 형태를 나타내고 있다.



<그림 4> 스마트카드 파일 구조

<Fig. 4> Smart Card File Structure

1) Transparent File

연속된 바이트로 구성되며 Offset을사용하여 파일 내에 기록된 데이터를 참조한다. 일반적으로 사용자 정보(EF_{USER}), 사용자 패스워드(EF_{PIN})등의 파일이 이에 해당된다.

2) Linear Fixed Record File

모든 레코드가 동일한 크기를 가지고 있는 레코드 파일로서 레코드 번호는 생성 순서에 따라 연속적으로 할당된다. 즉 첫 번째 레코드가 가장 먼저 만들어진 레코드가 되며 파일 내의 레코드 개수는 카드 발급 시 결정된다.

3) Linear Variable Record File

Linear Fixed Record File과 비슷한 구조이지만 모든 레코드의 크기가 동일하지는 않은 구조를 가지고 있다. 레코드 번호는 생성 순서에 따라 연속적으로 할당되지만 각 레코드의 크기는 틀리다. 파일의 크기는 카드 발급 시 결정되지만 레코드 개수는 결정되지 않는다.

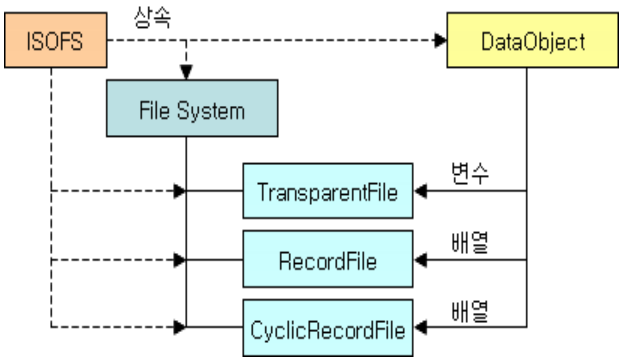
4) Cyclic Fixed Record File

Linear Fixed Record File 과 비슷하지만 링 구조로 구성되어 있다. 각 레코드 크기는 동일하며 레코드 번호는 역순에 따라 연속적으로 할당된다. 즉, 첫 번째 레코드가 가장 최근에 업데이트된 레코드이며 각 기록 프로시저에서 가장 오래된 레코드를 덮어쓰게 된다. 예를 들어 n개의 record를 갖는 cyclic 파일에 데이터를 쓰면 record 1이 되고 그전에 기록한 record 파일이 2가 되고, 가장 오래 전에 쓴 record가 n이 된다.

2. 자바카드 파일 시스템 설계

자바카드 메모리의 효율적 관리를 위해 본 논문에서는 연속적인 바이트로 구성된 transparent파일과 linear variable파일 시스템은 비효율적 메모리 관리로 linear fixed파일 시스템을 데이터 길이를 체크 하여 사용도록 하나의 레코드 파일과 cyclic 파일 클래스를 설계하였다.

레코드 파일 중 파일 시스템의 패키지 구성을 위해 기본적으로 자바카드 애플릿에서 사용되는 읽기, 쓰기, 추가의 동작과 파일에 대한 파일의 ID, 파일의 레코드 길이, 파일의 크기 등 기본정보를 정의 하였다.



<그림 5> 파일 시스템 클래스 간의 관계

<Fig. 5> Relation to File System Class

그림 5는 파일 시스템 구현을 위해 각 파일 구조들에 사용되는 데이터를 처리하기위한 변수, 배열과 쓰기, 읽기 작업을 수행할 클래스와 각 파일 구조의 특징을 위해 정의한 클래스를 설계하였다. ISOFS.class는 파일 시스템에 사용되어지는 상수를 정의 하고 있다. FileSystem.class는 각 파일의 파일 ID와 파일 Type을 나타내고 있으며, DataObjcet.class는 오브젝트내의 바이트 묶음을 나타내고 있다. 각 클래스의 속성 및 동작은 다음과 같이 정의 하였다.

1) DataObject

파일 시스템에서 데이터 처리를 위해 설계한 클래스로 데이터의 길이와 buffer를 정의하고 데이터를 읽고 기록할 수 있는 메소드를 정의하고 있다. 자바카드 객체의 기술은 persistent와 transient objcet를 제공한다. persistent는 메모리에 저장할 수 있는 기술로 EEPROM에 데이터를 기록하고, transient는 자바카드 연산에 필요로 하는 메모리로 RAM에 데이터를 기록할 때 사용한다. DataObject의 속성 및 동작을 다음과 같다.

속성 (Member)	Byte 배열 : data_buffer 총 크기 : total_length 현재 사용된 크기 : cur_length
동작 (Method)	총 크기 조회 : getTotalLength() 현재 크기 조회 : getCurLength() 데이터 조회 : readData() 데이터 쓰기 : writeData()

2) ISOFS

파일시스템의 ISOFS 파일은 파일시스템에 사용되는 파일 크기, 타입, 에러 코드의 상수를 정의 하고 있다. 파일 타입은 크게 transparent와 record file, cyclic record file로 구분하였고, 최대 파일 크기는 512byte이며, 파일 최소 사이즈는 32byte로 설정하였다. 파일 사이즈 및 속성의 수정 시 파일 시스템의 ISOFS 클래스에서 수정하여 사용할 수 있다.

3) TransparentFile

스마트카드에서 사용되어지는 일반적인 파일 시스템으로 IEF(Internal EF), WEF(Working EF)로 나눌 수 있다. IEF는 주로 사용자의 PIN, 암호에 사용되는 키를 주로 저장하기 위해 사용되어지고, WEF는 데이터를 저장하기위한 파일로 사용되어진다.

속성 (Member)	파일 ID : fid 파일 Type : file_type 파일 Size : len Binary Data
동작 (Method)	파일 Type 조회 파일 ID 조회 파일 읽기 : readBinary() 파일 쓰기 : writeBinary()

자바카드에서는 PIN을 관리하기위한 Owner- PIN API를 제공하고 있어 WEF 파일만 다루

도록 한다.

TransparentFile의 속성 및 동작은 다음과 같이 정의 하였다. 파일을 읽고, 쓰는 동작을 위해 저장되는 버퍼와 offset 및 길이를 체크하게 된다.

4) RecordFile

RecordFile은 스마트카드에서 linear Fixed와 linear Variable 파일을 정의 하고 있다. 본 논문에서 구현을 위해 linear variable 파일은 linear fixed를 이용하는 형태로 좀 더 효율적인 메모리 사용을 위한 설계를 하였다.

우선 레코드의 개수와 레코드 사이즈를 체크하여 binary 데이터의 배열을 잡는다. 레코드의 넘버를 셋팅하는 작업을 취하여 읽기, 쓰기 동작을 정의하고 있다. RecordFile의 속성 및 동작은 다음과 같이 정의 하였다.

속성 (Member)	레코드 사이즈 레코드 개수 Binary 데이터의 배열
동작 (Method)	Record 읽기 : readRecord() Record 쓰기 : writeRecord() Record 개수 조회

5) CyclicRecordFile

Record 파일 시스템과 유사하나, 파일 시스템에서 가장 복잡한 형태로 총 레코드 수와 레코드 사이즈를 체크하며 정해진 레코드에 추가되어지는 레코드를 쓰기위해 현재와 다음 레코드를 체크하여 데이터를 쓰게 된다. 응용 프로그램 설계 시 가장 유용하게 사용 할 수 있는 구조로 초기 애플릿에서는 record를 추가 시켜주기 위한 작업으로 append record를 선행해야 한다. Record 개수가 n개 이면 최근에 기록된 Record가 1이고, 그전에 기록된 record가 2고 가장 먼저 기록된record가 n이 된다.

속성 (Member)	Record와 동일 Cyclic record 제어하기 위한 포인터 (current, next)
동작 (Method)	Record동일 Record 추가 : appendCyclicRecord()

Ⅳ. 파일 시스템 구현 및 결과

1. 자바카드 파일 시스템 구현

본 논문의 파일 시스템을 구현하기 위한 환경은 Windows 2000 server 환경에서 자바카드 버전 2.1.2의 기술 스펙을 이용하였다. 자바 통합 개발환경인 Eclipse에 JCOP Tools3.0을 이용하였다. 사용된 자바카드는 JCOP bio31 카드로 스펙은 다음과 같다. 16Kbyte의 EEPROM, 24Kbyte의 ROM, 2300byte의 RAM과 16bit의 CPU으로 구성되어 있다.

2. 테스트 애플릿 구현

애플릿은 크게 파일 시스템을 이용하는 경우와 이용하지 않는 경우로 다음과 같이 구현 하였다. Transparent를 이용하는 애플릿은 파일 사이즈를 32byte의 크기를 갖고, 레코드 구조의 애플릿은 32byte의 레코드를 총 5개를 갖는 크기로 애플릿을 구현하였다. 테스트 애플릿에 공통적으로 사용되는 명령 APDU의 header 부분은 을 다음 표 1. 에 정의 하였다.

<표 1> 테스트 애플릿 APDU header 정의
<Table 1> Define APDU header

Header	값	정의
CLA	0x00	사용되는 클래스 정의
INS	0x23	데이터, 레코드 쓰기
	0x24	데이터, 레코드 읽기
	0x25	레코드 추가
P1	0x01-0x05	Transparent일 경우 0x00 Record 일 경우 Record 선택
P2	0x00	P2는 사용하지 않는다.

일반적으로 애플릿 설계를 위해 다음과 같은 사항을 고려한다.

- APDU 관련 명령(command)
- 생성자(constructor)
- Java Card 애플릿 라이프사이클
- 메소드 install( ), select( ), deselect( ), process( )
- 해당 private 메소드를 정의한다.

본 논문의 파일 시스템을 이용한 애플릿 개발 또한 동일하다. 그러나 private 메소드를 정의하는 데 있어 간결하고 효율적으로 설계할 수 있다.

테스트 애플릿 설계는 기존 자바카드에서 지원하는 API를 이용한 경우와 본 논문에서 제시하는 API를 이용하는 경우로 총 6개의 파일로 구현하였다. 각 애플릿의 패키지는 dku.image-sys.test에 순서대로 설계하였다.

<표 2> 테스트 애플릿 AID
<Table 2> Test 애플릿 AID

	File Name	AID(8byte)
Basic API	TransparentExample	0011223344556601
	RecordExample	0011223344556602
	CyclicRecordExample	0011223344556603
File System PID (A0112233445566)		
File System API	FS_TransparentExample	0111223344556601
	FS_RecordExample	0111223344556602
	FS_CyclicRecordExample	0111223344556603

표 2에서 보듯이 설계한 애플릿의 AID는 8byte로 구성하였다. 기본 API를 이용한 경우 첫 번째 바이트를 0x00으로 설정하였고, 파일 시스템을 이용하는 경우 0x01로 설정하였다. 또한 마지막 바이트는 transparent인 경우 0x01로, record를 이용하는 경우 0x02로 cyclic을 이용하는 경우 0x03으로 설정하였다. 또한 파일 시스템의 패키지 ID는 7byte로 설계 하였다.

1) TransparentFile 테스트 애플릿

그림 6은 파일 시스템을 이용하여 작성된 TransparentFile 애플릿이다. 애플릿 코드 중 읽기와 쓰기 메소드만 표기 하였다.

자바카드에서 제공되는 API를 사용하기 위해 javacard.framework.*와 파일 시스템을 사용하기 위해 dku.imagesys.FS.*를 추가시켜 준다.

TransparentFile을 이용하여 새로운 생성자 ID_File을 생성하여 읽기 쓰기 작업을 수행할 수 있다. 읽기 메소드의 정의는 파일 크기를 체크 하여 버퍼에 있는 데이터를 순서대로 읽어들인다. 쓰기 메소드는 카드로 전송된 데이터(OFFSET_ CDATA를 받아 버퍼에 순서대로 작성하게 된다. 그림 6.에서 보듯이 간단한 코드로 transparent 파일을 사용할 수 있다. 그러나 파일 시스템을 이용하지 않을 경우 코드길이는 늘어나게 된다.

```
import dku.imagesys.FS.*;
private TransparentFile ID_File;

// ---- Read ID Command ----------------
private void cmdReadID(APDU apdu){
 ...........
 // Ready Response Data
 ID_File.readBinary((short)0, buf, SIZE_ID_FILE);
 ...........
}
// ---- Write ID Command ----------------
private void cmdWriteID(APDU apdu){
 ...........
 // Update Data
ID_File.writeBinary(buf, (short)ISO7816.OFFSET_CDATA, (short)0, (short)SIZE_ID_FILE );
}
```

<그림 6> TransparentFile 코드

<Fig. 6> TransparentFile Coding

2) RecordFile 테스트 애플릿

RecordFile은 5개의 Record를 기본으로 설계 하였고 레코드의 크기는 ISOFS에서 변경할 수 있다.

```
import dku.imagesys.FS.*;
private RecordFile ID_File;

// ---- Read ID Command ----------------
private void cmdReadID(APDU apdu){
 // Ready Response Data
 ID_File.readRecord(buf[ISO7816.OFFSET_P1],
        buf, SIZE_ID_RECORD);
}
// ---- Write ID Command ----------------
private void cmdWriteID(APDU apdu){
 // Update Data
ID_File.writeRecord(buf[ISO7816.OFFSET_P1], buf,
                (short)ISO7816.OFFSET_CDATA, SIZE_ID_RECORD);
}
```

<그림 7> RecordFile 코드

<Fig. 7> RecordFile Coding

그림 7에서 Record 데이터를 읽고 쓸때는 P1을 체크하여 지금 선택된 레코드의 위치를 파악하게 된다. 읽기 메소드는 레코드의 길이를 체크하여 P1이 지정한 레코드의 데이터를 버퍼로 옮겨 데이터를 읽을 수 있다. 쓰기 메소드는 카드로부터 전송된 데이터의 지정된 레코드에 쓰

게 된다. 이때 에러 발생 시 응답 APDU는 0x6A83이 전송되게 된다. Transparent와 마찬가지로 파일 시스템을 이용하면 한 라인으로 record가 처리 된다. 만약 파일 시스템을 이용하지 않고 코드를 작성하게 되면 파일 크기와 record를 선택하기 위한 코드가 추가 되어야 함으로 복잡해지게 된다.

3) CyclicRecordFile 테스트 애플릿

CyclicRecordFile은 선택되는 레코드의 넘버를 체크하여 선택된 레코드에 데이터를 기록하게 된다. 이때 레코드의 개수가 n이라고 하면, 레코드선택은 최근에 쓰여진 레코드가 1이 되고 가장 먼저 쓰여진 레코드가 n이 된다. 그림 8.은 파일 시스템을 이용하여 작성된 CyclicRecordFile 애플릿의 일부이다. 이 애플릿에서는 읽기, 쓰기, 추가 메소드로 구성된다. 총 메소드 크기를 5개로 잡았으므로 5개 내에서는 읽기, 쓰기, 추가가 가능하다. 읽기 메소드는 앞의 RecordFile과 동일하게 동작한다. 쓰기 메소드는 현제 레코드위치와 이전에 쓰여진 레코드 위치를 파악하기 위한 메소드가 추가되어 지정된 레코드에 데이터를 쓰게 된다. 추가 레코드 또한 쓰기 레코드와 동일하나 애플릿에서 레코드를 추가 시켜 줄때 사용한다.

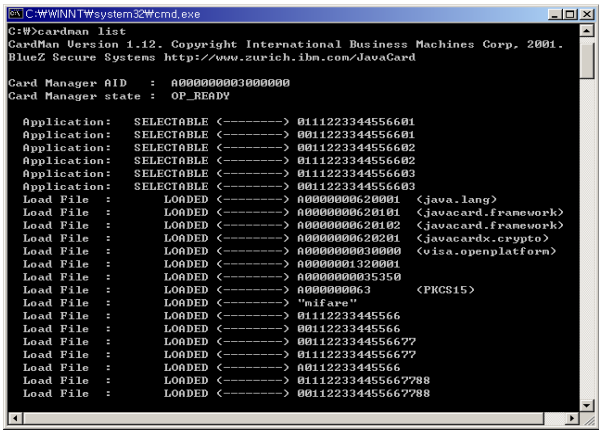
```
import dku.imagesys.FS.*;
private CyclicRecordFile ID_File;

// ---- Read ID Command ----------------
private void cmdReadID(APDU apdu){
 // Ready Response Data
 ID_File.readCyclicRecord(buf[ISO7816.OFFSET_P1], buf, SIZE_ID_RECORD);
}
// ---- Write ID Command ----------------
private void cmdWriteID(APDU apdu){
 // Update Data
ID_File.writeCyclicRecord(buf[ISO7816.OFFSET_P1], buf,(short)ISO7816.OFFSET_CDATA,
       buf[ISO7816.OFFSET_LC]);
}
// ---- Append ID Command ----------------
private void cmdAppendID(APDU apdu){
// Lc Check
if ( buf[ISO7816.OFFSET_LC] != SIZE_ID_RECORD )
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
// Append CyclicRecord
ID_File.appendCyclicRecord(buf, (short)ISO7816.OFFSET_CDATA, buf[ISO7816.OFFSET_LC]);
}
```

<그림 8> CyclicRecordFile 코드

<Fig. 8> CyclicRecordFile Coding

3. 테스트 애플릿 실험 결과

자바카드 통합 개발 환경인 Eclipse를 이용하여 테스트 애플릿을 구현하여 CAP 파일을 생성하였다. 생성된 CAP 파일은 Eclipse의에서 플러그인 되어 있는 JCOP Tools를 이용하여 카드에 로드시킬 수 있을 뿐만 아니라 제공되는 Simulator를 이용하여 확인 할 수 있다. 본 논문에서는 실제 카드에 로딩한 내용으로 전개한다.



<그림 9> 자바카드 상태

<Fig. 9> Status Java Card

IBM에서 제공하는 JCOP 카드의 상태를 확인 할 수 있는 cardman.exe 프로그램을 이용하여 카드 상태를 확인 할 수 있다. 그림 9.은 카드에 파일 시스템 API와 테스트 애플릿을 로딩한 결과를 보여 주고 있다. SELECTABLE로 표시되어 있는 것이 테스트 애플릿이다.

명령 데이터를 주고받기 위해 사용한 PC/SC리더기는 JSTU 9750 모델을 사용하여 명령 APDU와 응답 APDU를 확인 할 수 있었다.

그림 10은 FS_CyclicRecordExample의 애플릿을 테스트 한 결과이다. 카드의 동작은 사용 애플릿을 선택하고 추가 명령 APDU와 32byte의 데이터를 카드에 전송하면 카드는 cyclic 파일의 레코드를 생성하여 기록하게 된다. 레코드의 길이가 잘 못되었을 경우 응답은 0x6700이고, 정의된 레코드의 범위를 벗어날 경우 0x6A83의 상태 응답이 전달된다. 또한 명령 APDU의 Lc의 값이 정의된 레코드의 크기를 벗어날 경우 0x6C00에 정의된 레코드 크기를 더하여 응답APDU를 전송하게 된다. 예를 들어 크기 32byte의 레코드면 0x6C의 응답 APDU가 전송된다. 정상적으로 동작하였을 경우 0x9000의 응답 APDU가 전송된다.

본 논문에서 제시하는 파일 시스템의 설계 및 구현 카드에 쓰여 진 코드는 총 789byte였다.

```
· 애플릿 Select
=> send 00A4040008 0011223344556603
<= 9000
· Append Record
=> send 0025010020112233445566778899 00AABBCCDDEEFF
<= 9000
=> send 0025020020122233445566778899 00AABBCCDDEEFF
<= 9000
· Read Record
=> send 0024010020
<= 13223344556677889900AABBCCDDEEFF9000
=> send 0024020020
<= 122233445566778899 00AABBCCDDEEFF9000
=> send 0024030020
· Write Record
=> send 0023010020A1223344556677889900AABBCCDDEEFF
<= 9000
=> send 0023020020A2223344556677889900AABBCCDDEEFF
<= 9000
· Read Record
=> send 0024010020
<= A3223344556677889900AABBCCDDEEFF9000
=> send 0024020020
<= A2223344556677889900AABBCCDDEEFF9000
```

<그림 10> Cyclic 애플릿 테스트 결과

<Fig. 10> Cyclic 애플릿 test result

<표 3> 애플릿 테스트 결과

<Table 3> Applet Test Result

		일반적인 애플릿 설계			File System API		
		Tran.	Rec.	Cyc.	Tran.	Rec.	Cyc.
Code Size on card(byte)		251	294	486	251	253	295
속도	Write data (ms)	110	109	125	109	109	110
	Read data(ms)	94	82	94	79	78	78

표 3은 파일 시스템의 검증을 위해 메모리에 쓰여지는 코드 사이즈와 애플릿이 터미널과의 동작시간을 체크 한 결과 이다. 파일 시스템을 이용하지 않은 애플릿 코드 사이즈는 Transparent가 적은 코드 사이즈를 보였고, Cyclic 사이즈가 가장 큰 사이즈를 차지하는 것을 볼 수 있다. 그러나 파일 시스템을 이용한 애플릿은 거의 비슷한 파일 사이즈를 나타냈다. 예를 들어 cyclic- record 파일을 갖는 애플릿 10개를 카드에 내장 할 때 메모리 사용은 다음과 같다.

· 기존 API를 이용하는 경우

486 × 10 = 4860 byte

· 파일시스템을 이용하는 경우

295 × 10 + 789 = 3739byte

이와 같이 약25%가 감소되며, 또한 카드 애플릿과 터미널 간의 데이터 전송 속도도 적은 코드를 수행하게 되는 파일 시스템을 이용한 애플릿이 대략 10ms정도 효율적인 것을 볼 수 있다.

Ⅴ. 결론

본 논문에서는 자바카드에서 지원하지 않는 파일 시스템을 자바카드에 적용하여 카드의 사용을 효율적으로 하기 위해서 자바카드 파일 시스템 API 패키지를 설계 및 구현하였다. 파일 구조는 스마트카드 국제 표준인 ISO7816-4에서 제시하는 4가지의 파일 구조를 이용하였다.

ISO7816-4의 파일시스템을 자바카드에서 사용하기 위해 각 파일 시스템의 특성을 분석한 결과transparent 파일은 일반적으로 데이터를 읽고, 기록함으로써 binary 데이터를 기록하면 되었고, 나머지 파일 시스템은 레코드형태로 linear fixed와 liner variable은 레코드의 크기와 파일의 길이를 체크 해줌으로써 하나의 형태로 설계하였으며, cyclic 파일은 record의 선택이 현재, 다음을 고려해서 설계 하였다.

본 논문에서 구현한 파일 시스템 API의 크기는 789byte이었다. 테스트를 위해 기존 API를 이용한 애플릿과 파일 시스템 API를 이용한 애플릿으로 설계 하여 테스트한 결과 좀더 효율적인 것을 확인하였다. 10개의 애플릿을 사용한다고 가정하여 파일시스템을 이용하게 되면 코드의 크기가 약25%의 감소됨을 확인하였다. 또한 카드 애플릿과 터미널 간의 데이터 전송 속도도 적은 코드를 수행하게 되는 파일 시스템을 이용한 애플릿이 10ms의 효율적인 속도를 볼 수 있다. 그러므로 파일 시스템 API를 이용함으로써 애플릿을 개발 하는데 필요한 노력과 시간을 줄 일 수 있을 뿐만 아니라 개발의 편이성, 데이터의 캡슐화 시킬 수 있을 뿐만 아니라 다양한 애플릿을 개발 하는데 로딩 되는 시간을 줄 일 수 있다.

향후 연구과제로는 본 논문의 파일시스템의 최적화 및 여러 응용 분야의 연계성을 위해 파일 시스템을 이용한 애플릿 간의 공유 파일 관리 시스템은 연구 과제로 남긴다.

참고문헌

[1] WILLIAM STALLINGS, “Network and Internetwork Security”, PRENTICE HALL, 1995

[2] Alfres J. Menezes, “HandBook of APPLIED CRYPTOGRAPHY”, CRC, 1997

[3] Scott Oaks, “JAVA Security”, O′REILLY, 1998

[4] Zhiqun Chen, “Java Card Technology for Smart Cards”, Addison Wesley, 2000

[5] Vesna Hassler 외, “Java Card for E-Payment Application”, Artech House, 2002

[6] Marcus Oestreicher, “Transactions in Java Card″, Annual Computer Security Application Conference, pp.291, 1999

[7] Ludovic Casset 외, “Formal Development of an Embedded Verifier for Java Card Byte Code″ International Conference on Dependable System and Networks, pp.51, 2002

[8] 문상재 외, “차세대 IC 카드를 이용한 정보보호 시스템 개발”, 정보통신부, pp.17, 1997

[9] 김연선, 이창욱, “자바카드 애플릿 설계 및 검증에 관한 연구”, 한국통신정보보호학회 종합학술 발표회논문집, Vol.10, No.1, pp.805, 2000

[10] 김성준 외, “자바카드 기반 공개키 암호 API를 위한 임의의 정수 클래스 설계 및 구현”, 정보처리학회, 9권 2호, pp.163, 2002

[11] 임현준 외, “Java Card SIM API의 Toolkit Registry 구현에 관한 연구”, 정보처리학회 추계학술 발표대회 제9권 제2호, 2002

[12] 김도우, 정민수, “자바카드 플랫폼상에서 자바 클래스 파일의 최적화 연구”, 멀티미디어학회 논문지, 6권 7호, pp.1200, 2003

[13] 황선명, 염희균, “자바카드 애플릿의 검증 방법”, 정보처리학회 소프트웨어공학연구회지, 제5권 1호, 2002

[14] 이정우, 전성의, “자바카드에서 Post-issuance API에 관한 연구”, 정보과학회 가을 학술발표 논문집 2002권 pp.583, 2002

[15] Uwe Hansmann 외, “Smart Card Appli- cation Development Using Java”, Springer, 2002

[16] SILBERSCHATZ외, “Operating System Concepts”, WILEY, 2002

[17] Jess Garms, Daniel Somerfield, “Java Security”, 정보문화사, 2002

[18] http://java.sun.com/products/javacard/datasheet.html

[19] http://www.zurich.ibm.com/jcop/order/tools.html

[20] http://www.eclipse.org/downloads/index.php

[21] http://kr.sun.com/korea/sun_info/2004/web_spring/sunintech/tech02.html