효율적 파일 관리를 위한 자바카드 API 설계 및 구현

송 영 상[†] · 신 인 철[††]

요 약

다양한 응용분야를 지원하기 위해 여러 개의 독립적인 애플릿이 카드에 탑재되는 자바카드에서는 각 애플릿이 응용분야에 따라 데이터를 처리 및 관리하게 된다. 본 논문에서는 각 애플릿이 데이터를 효율적으로 처리 및 관리하기 위해 자바카드에서 지원하는 기본 API(Application Program Interface)와 스마트카드 국제 표준인 ISO 7816-4에 정의되어 있는 파일 시스템을 이용하여 자바카드용 파일 시스템 API를 설계 및 구현하였다. 제안된 파일 시스템 API를 이용하여 각 애플릿에서 메소드 호출로 동일한 코드를 줄일 수 있다. 이를 통하여 처리 시간과 메모리 사용을 감소시키며, 개발 시간과 비용을 줄일 수 있을 것으로 기대된다.

키워드 : 자바카드, 파일시스템, 스마트카드, 자바카드 애플릿, 자바카드 API

Design and Implementation of the Java Card API for Efficient File Management

Young-Sang Song[†] · In-Chul Shin[††]

ABSTRACT

There are several independent applets to support various applications in a Java Card. Each applet in a Java Card processes and manages its own data without concern to other applets and their data. In this paper we proposed file system API to support efficient file management based on Java Card. Also we designed and implemented Java Card based file system API using basic API and referring to the file system standard defined in ISO 7816-4 Smart Card standard. By using proposed file system API, we can replace duplications of same code in each applet with short method call. So the used memory space and processing time is reduced and also the reduction of development time and cost will be expected.

Key Words : Java Card, File System, Smart Card, Applet, Java Card API

1. 서 론

전 세계적인 초고속 네트워크망을 근간으로 유비쿼터스 시대가 본격화 되어 가면서 다양한 형태의 응용이 개발 되고 있으며, 이에 따른 개인정보의 소지방법 또한 중요한 문제로 연구되고 있다. 초기에 강력한 정보보호 기능으로 개인 신분 증명, 접근제어 등 금융 분야 및 여러 응용 분야에서 스마트카드가 사용되었다. 그 후 국제 표준인 ISO 7816과 산업표준인 EMV(Europay, Master, Visa)와 호환성을 가지며 스마트카드 플랫폼에 JCVM(Java Card Virtual Machine)을 탑재하여 개발의 용이성, 카드발급 후 응용프로그램 갱신이 가능하고 하나의 카드에 다수의 응용 프로그램을 수용할 수 있어 융통성과 함께 이용의 다양성 및 확장성 등의 장점을 갖는 자바카드가 등장하였다. 최근에는 정보보호의 안정성 및 소지의 편리성은 물론 다양한 장점을 가진 자바카드를 이용하여 적절한 수준으로 개인정보 유출의 부작용을 방지하면서 다양한 이용을 통합적으로 지원하는 자바카드가 널리 사용되고 있는 추세이다[1-4].

자바카드의 각 응용 프로그램인 애플릿은 SUN사에서 제공하는 자바카드 API를 이용하여 설계, 구현되며, 각 애플릿은 각각 자신만의 데이터를 사용하는 것이 보통이므로 각 애플릿은 데이터를 취급하기 위해 파일의 생성, 수정, 저장, 삭제 등을 하기위한 파일 관리 시스템이 필요하다[19]. 그러나 자바카드 자체의 파일관련 표준이 없으며, SUN사에서 제공하는 API에도 역시 파일 구조를 사용 할 수 있는 패키지 및 API를 제공하지 않고 있어 애플릿 개발 시 대부분 사용하려는 데이터 처리를 위해 바이트 배열 및 변수를 사용하여 데이터의 옵셋과 파일 크기에 대한 제어를 통해 각 애플릿에서 복잡하게 구현된다. 각 애플릿에서는 파일 시스템을 구현을 위해 매번 동일한 코드를 작성해야 하는 코드의 중복사용으로 인해 카드의 메모리를 소모하고 프로그램

※ 이 연구는 2004학년도 단국대학교 대학연구비의 지원으로 연구되었음.
† 준 회 원 : 단국대학교 대학원 전자 · 컴퓨터공학과 박사과정
†† 정 회 원 : 단국대학교 전자 · 컴퓨터공학부 교수
논문접수 : 2006년 3월 31일, 심사완료 : 2006년 4월 26일

오류의 가능성이 높아지는 등의 단점이 있다.[12, 14]

본 논문에서는 스마트카드의 ISO 7816 표준을 만족하는 자바카드 파일관리 API를 제안하여 효율적인 파일관리 시스템을 설계 구현 하였다. 제안된 시스템 패키지 클래스는 ISO 7816의 4가지 파일 구조를 지원하며, 자바카드에 설치된 애플릿이 사용하도록 JCRE(Java Card Runtime Environment)에 탑재하여 동작하도록 설계 구현하였다. 파일 시스템 설계는 각 파일 구조에 맞게 파일의 길이와 크기 및 메모리에 쓰여 지는 위치를 고려하여 설계하였고, 구현은 자바카드에서 제공하는 기본 API를 이용하였다.

이를 이용하여 단일화된 파일 입출력과 인터페이스로 동작되는 애플릿을 개발 할 수 있다. 또한 매번 용도에 따라 다양하게 구현되어야 되는 애플릿이 단일화된 파일 입출력을 사용함으로써, 수행 코드와 구성 코드를 단축할 수 있다.

본 논문의 구성은 2장에서는 스마트카드에 사용되는 파일 시스템과 자바카드의 기본 구조 및 동작을 살펴보고, 3장에서는 기존의 애플릿의 파일 시스템을 이용을 살펴보고 4장에서는 본 논문이 제시하는 파일 시스템의 설계와 구현을 5장에서는 파일시스템 패키지의 동작을 검증하였다. 마지막으로 6장에서는 결론을 맺는다.

2. 자바카드

2.1 스마트카드 파일 구조

스마트카드 파일 시스템은 ISO 7816-4에 규정되어 있으며 다음과 같다. 파일 종류는 MF(master file), DF(dedicated file), EF (elementary file)로 구성된다. 그 중 실질적인 데이터는 EF에 기록되어 진다. EF에 데이터를 기록하는 기본 구조는 데이터를 순차적으로 처리하는 트랜스페어런츠(transparent)와 개별적으로 동일한 구조인 레코드(record) 구조로 나눌 수 있다. 레코드는 사이즈에 따라 고정적인 것과 가변적인 것으로 구분되며, 레코드의 구조는 선형인 것과 사이클릭(cyclic)구조로 된 것으로 구분할 수 있다. 결국 4종류의 파일구조로 구분하여, 이중 하나를 선택해서 사용한다. 각 파일 구조는 다음과 같다.

- 트랜스패어런츠 파일

트랜스패어런츠 파일은 (그림 1) (a)와 같이 순차적인 바이트로 구성되며, 옵셋을 사용하여 파일 내에 기록된 데이터를 읽거나 갱신한다. 일반적으로 사용자 정보(EF_{USER}), 사용자 패스워드(EF_{PIN})등의 파일이 이에 해당된다.

- 선형고정레코드 파일

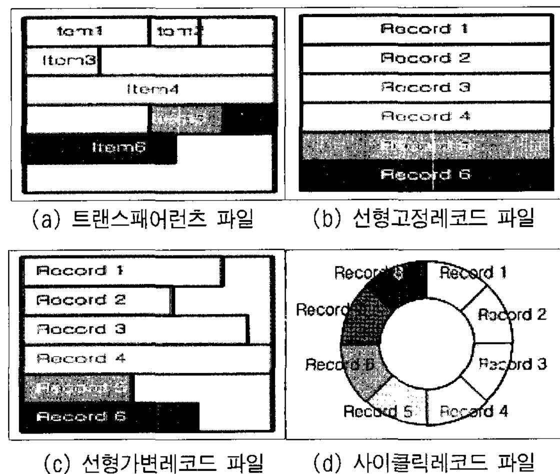
선형고정레코드(linear fixed record) 파일은 (그림 1) (b)와 같이 모든 레코드가 동일한 크기를 가지고 있는 레코드 파일로서 레코드 번호는 생성 순서에 따라 연속적으로 할당된다. 즉 첫 번째 레코드가 가장 먼저 만들어진 레코드가 되며 파일 내의 레코드 갯수는 카드 발급 시 결정된다.

- 선형가변레코드 파일

선형가변레코드(linear variable record) 파일은 선형고정레코드 파일과 비슷하나 (그림 1) (c)와 같이 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.

- 사이클릭레코드 파일

사이클릭레코드 파일은 (그림 1) (d)와 같이 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 레코드를 갖는 사이클릭 파일에 데이터를 쓰면 레코드 1이 되고 그전에 기록한 레코드 파일이 2가 되고, 가장 오래 전에 기록된 레코드가 n이 된다.



(a) 트랜스패어런츠 파일 (b) 선형고정레코드 파일

(c) 선형가변레코드 파일 (d) 사이클릭레코드 파일

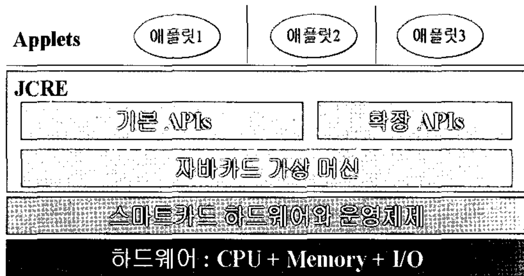
(그림 1) 스마트카드 파일 구조

2.2 자바카드

자바카드란 스마트카드 기술에 자바의 기술을 접목시킨 것이다. 일반적으로 자바카드의 메모리, 통신, 보안 그리고 어플리케이션의 실행 모델을 지원한다. 일반적으로 자바카드의 메모리는 보통 1K의 RAM과 24K의 ROM 그리고 16K의 EEPROM으로 구성된다. 자바카드는 자바 언어의 특징을 부분적으로 지원하고 이를 실행시킬 수 있는 가상머신을 필요로 한다. 자바카드 가상머신은 카드외부(off- card) 가상머신과 카드 내부(on-card) 가상머신으로 나눠는 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 JCRE을 의미한다. 자바카드의 일반적인 구조를 (그림 2)에 나타내었다. 최하위단은 자바카드의 하드웨어와 COS(card operating system)

가 위치한다. 그 윗단에는 JCRE가 위치하여 최상위 애플릿을 동작시키기 위한 작업을 수행하게 된다. 자바카드는 다음과 같은 특징을 제공한다.

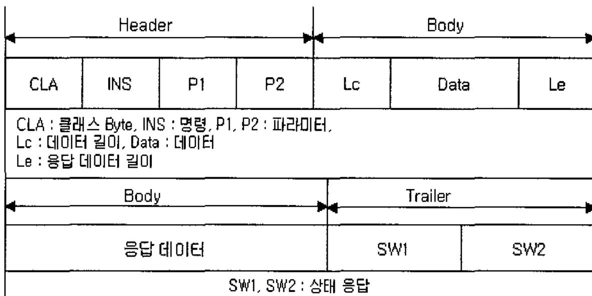
- 플랫폼 독립성(platform independence) : 자바카드의 JCRE를 기반으로 작성되어 상호 호환성이 있다.
- 복수 응용 프로그램(multiapplication program) : 하나 이상의 응용프로그램이 하나의 카드 상에서 동작가능.
- 응용 프로그램의 갱신(application program update) : 카드가 발급된 후에 응용프로그램을 설치 및 갱신가능.
- 융통성(flexibility) : 자바카드기술의 객체 지향 기술은 스마트카드 프로그래밍에 융통성을 제공한다.
- 호환성(compatibility) : 자바카드 국제 표준인 ISO 7816과 산업 표준인 EMV와 호환한다.



(그림 2) 자바카드 구조

2.3 자바카드 APDU

카드의 사용은 자바카드의 애플릿과 응용 프로그램간의 통신을 위해 APDU(Application Program Data Unit)교환으로 자바카드의 수행환경에서 이루어진다. 애플릿과 외부 응용 프로그램간의 통신은 명령어(command) APDU와 응답(response) APDU로 구성되어 APDU 교환을 통해서 이루어진다. APDU의 구조는 필수 항목인 헤더(header) 부분과 선택 항목인 바디(body)부분으로 나눌 수 있다.



(그림 3) 명령, 응답 APDU

(그림 3)에 APUD의 구조 형태를 나타내었다. 명령 APDU의 Header 부분은 클래스(CLA), 명령어(INS), 파라미터(P1, P2)로 총4byte로 나타내는 필수 부분이다. 바디부분의 사용은 선택적이며, 구성은 전송데이터의 길이(Lc), 데이터(Data), 응답데이터 길이(Le)로 이루어진다. 데이터의 길이는 가변적이다. 응답 APDU의 바디는 응답데이터, 상태를 표시하는 꼬리(trailer)로 구성되어 있다.

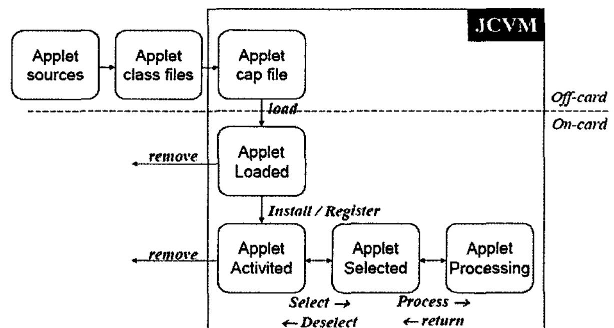
3. 기존 애플릿에서의 파일시스템

3.1 파일시스템 작성

3.1.1 자바카드 애플릿

애플릿은 자바카드 플랫폼에서 동작하는 자바 응용 프로그램이다. 하나의 카드에 여러 개의 애플릿이 탑재되며, AID(application Identifier)를 이용하여 구분한다. AID는 필수 항목인 5bytes이상의 값인 RID(resource identifier)와 0-11bytes인 PIX(proprietary identifier extension)으로 구성된다. 애플릿은 JCRE 윗단에 위치하여 카드 내에서 정해진 규칙에 따라 동작한다. 또한 ROM에 설치될 필요 없이 카드의 EEPROM에 다운로드 하여 사용한다.

기존 애플릿 작성은 카드 외부에서의 동작과 카드 내에서 동작으로 크게 두 부분으로 나누어진다. 또한 애플릿 동작을 위한 코드 작성은 SUN사에서 제공되는 API를 이용하여 소스를 작성하고 이를 컴파일 하여 클래스 파일을 만들게 된다. 만들어진 클래스 파일과 엑스포트(export) 파일을 컨버터(converter) 시켜 CAP(Converted APplet)파일을 만들어 카드에 로딩 시키게 된다. 로드된 CAP 파일은 인스톨하여 사용가능한 애플릿 상태가 되어 외부 어플리케이션 프로그램으로부터 (그림 3)과 같은 형식의 명령 APDU를 받아 처리하고, 그에 따른 응답 상태를 외부 어플리케이션 쪽으로 보내 처리 결과를 알려준다. (그림 4)에 애플릿 동작 절차를 나타내었다.



(그림 4) 자바카드 애플릿 동작 절차

자바카드의 애플릿 작성을 위해 고려해야 할 메소드는 다음과 같고, 4개의 필수 메소드가 존재한다.

- install() : 애플릿 사용 환경을 만들기 위해 카드에 설치
- select() : 애플릿이 선택되었을 때 동작을 활성화
- deselect() : 애플릿 선택을 해제될 때 활성화시킬 동작을 정의
- process() : 외부의 응용 프로그램으로 부터 명령을 받아 처리

3.1.2 자바카드 API

자바카드 API는 스마트카드 표준인 ISO 7816에 의거해서 자바카드 애플릿 개발을 위해 만들어진 클래스들의 집합으로 이루어져있으며 3개의 핵심 패키지와 1개의 확장 패키지로 이루어져있다. 그러나 자바 플랫폼 클래스들은 GUI 인터

페이스, network I/O, 데스크탑 파일 시스템은 지원되지 않는다. <표 1>에 자바카드에서 지원되어지는 기본 API 패키지를 나타내었다.

〈표 1〉 테스트 애플릿 APDU 헤더 정의

패키지	내용
· java.lang	자바카드 애플릿 개발을 위해 사용되어지는 객체 및 예외 처리를 정의
· javacard.framework	기본적인 패키지로 애플릿의 작성과 통신을 처리하기위한 클래스 및 메소드 정의
· javacard.security	카드에 사용되어지는 암호 알고리즘의 키 관리를 정의해놓은 패키지.
· javacard.crypto	암호알고리즘의 암호화 및 복호화를 제공하기 위해 정의해놓은 패키지.

본 논문에서는 파일 시스템 API의 설계를 위해 언어에 관련된 패키지로 java.lang를 사용하였고, 자바카드 애플릿의 중요함수와 인터페이스와 클래스를 제공하는 javacard.framework 패키지를 사용하였다. 파일 시스템은 외부 APDU를 받아 처리 되므로 select(), deselect()의 메소드는 활성화시키지 않았으며, 또한 트랜스페어런츠와 레코드파일 및 사이클릭 파일을 지원하는 각각의 애플릿을 구현하였다. 파일과 레코드의 크기, 데이터를 저장하는 오프셋 위치와 크기, 파일 선택을 위한 파일 ID, 데이터 보호를 위한 atomicity처리, 총 레코드 계산 및 레코드 선택, 레코드의 추가, 레코드 사이즈 등을 고려하여 작성한다.

이와 같이 애플릿을 구성할 경우 하나의 애플릿이 서로 다른 파일 종류를 지원하고 동작하기 위해 코드는 중복 사용되어 작성된다. 뿐만 아리나 매번 애플릿을 작성할 때 동일한 파일 관리 코드가 반복되므로 메모리를 효율적으로 사용할 수 없다.

4. 제안하는 애플릿 파일 시스템

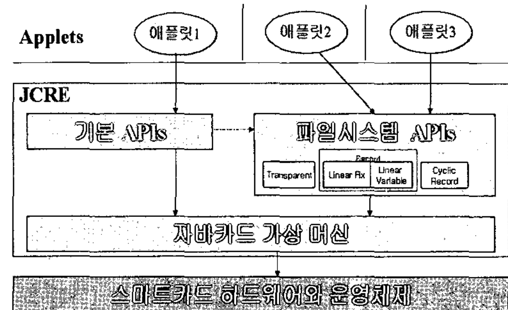
4.1 파일 시스템 제안

자바카드에는 파일시스템 관련 표준이 규정되어 있지 않으며, 또한 SUN사에서는 파일 구조를 구현할 수 있는 패키지 및 API를 제공하지 않고 있다. 자바카드의 특징인 복수 응용과 응용 프로그램의 갱신에 따라 자바카드에는 여러 개의 애플릿이 존재하거나 수시로 추가되며, 일반적으로 각 애플릿은 데이터를 관리하는 API가 필요 하다. API를 추가할 때는 자바 언어 지원과 자바카드 플랫폼에서의 동작, 스마트카드 ISO 7816 표준지원 등을 고려하여 간결하고 축약적인 API를 구성해야 된다.

본 논문에서는 위의 사항을 고려하여 기본 API를 이용하고, ISO 7816 표준을 만족하는 자바카드에서 사용가능한 파일 시스템 API를 설계 및 구현한다.

(그림 5)에서 보듯이 기본 API 클래스와 확장 API위치에 파일시스템 API를 두어 자바카드 플랫폼에서 동작 가능하게 구성한다. 또한 파일 관리를 위해 본 논문에서 제시하는 파일 시스템 API를 이용하여 좀 더 효율적인 파일 관리를 할 수 있게 한다. 3개의 애플릿의 동작은 다음과 같이 이루어진다.

- 첫 번째 애플릿 : 기본 API를 이용한 애플릿으로 여러 파일시스템을 사용할 경우 하나의 애플릿에 여러 개의 파일 시스템을 위한 코드를 작성하여야하는 어려움과 많은 코드의 작성이 필요로 하게 된다.
- 두 번째 세 번째 애플릿 : 첫 번째 보다 훨씬 간결하고 하나의 애플릿에 다양한 파일 시스템을 쉽게 적은 코드의 사용으로 사용할 수 있다.



(그림 5) 파일시스템 API 구조

4.2 파일 시스템 설계

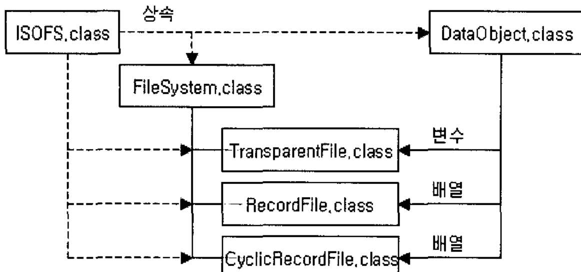
자바카드 메모리의 효율적 관리를 위해 본 논문에서는 연속적인 바이트로 구성된 트랜스패어런츠 파일과 선형 가변 파일 시스템은 비효율적 메모리 관리로 선형 고정파일 시스템을 데이터 길이를 체크 하여 사용도록, 하나의 레코드 파일과 사이클릭 파일 클래스를 설계하였다. (그림 5)에서 보듯이 설계 제안하는 파일 구조는 다음과 같다.

- 트랜스페어런츠 파일(TransparentFile) : 순차적인 바이트를 처리 하기 위한 파일 구조로 설계
- 레코드파일(RecordFile) : 고정된 레코드길이를 할당하여 고정/가변을 모두 처리 가능하게 설계
- 사이클릭파일(CyclicFile) : 레코드 구조와 동일하지만 링 구조로 구성되어 데이터 처리 가능하게 설계

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

파일 시스템에 매칭 되는 클래스에는 공통요소(데이터 멤버 및 메소드)가 발생된다. 이 공통 요소 중 가장 최소단위의 데이터 표현을 클래스 DataObject에 정의 하였다. 트랜스페어런츠파일 클래스에서 사용변수와 데이터의 길이에 대한 정보를 제공하고 있다. 또한 레코드파일 클래스와 사이클릭레코드파일 클래스는 사용되는 배열의 크기 및 길이에 대한 정의를 하고 있다. 그 외에 파일을 선택하기 위해 필요한 공통요소인 파일 아이디(FileID)와 파일 타입(FileType)은 파일시스템 클래스에 정의하였고, 세 가지 타입의 파일 클래스는 이 파일 시스템 클래스를 상속 받아서 구현하였다.

(그림 6)에서 보듯이 파일시스템 클래스 및 세 가지 파일 클래스를 사용하고, 필요한 상수는 ISOFS 클래스의 인터페



(그림 6) 파일시스템 API를 위한 각 클래스의 관계

<표 2> 파일 시스템 API를 위한 각 클래스의 속성 및 동작 정의

클래스 이름	속성(Member)	동작(Method)
DataObject	Byte 배열 : data_buffer 총 크기 : total_length 현재 사용된 크기 : cur_length	총 크기 조회 : getTotalLength() 현재 크기 조회 : getCurLength() 데이터 조회 : readData() 데이터 쓰기 : writeData()
FileSystem	파일 ID : file_id 파일 Type : file_type	파일 ID 조회 : getFILE_ID() 파일ID 쓰기 : setFILE_ID() 파일ID 일부 조회 : getFILE_SFI() 파일 Type 조회 : getFileType() 파일 Type 설정 : setFileType()
TransparentFile	파일 ID : fid 파일 Type : file_type 파일 Size : len Binary Data	파일 Type 조회 파일 ID 조회 파일 읽기 : readBinary() 파일 쓰기 : writeBinary()
RecordFile	레코드 사이즈 레코드 개수 Binary 데이터의 배열	Record 읽기 : readRecord() Record 쓰기 : writeRecord() Record 개수 조회
CyclicRecordFile	Record와 동일 Cyclic record 제어하기 위한 포인터 (current, next)	Record동일 Record 추가 : appendCyclicRecord()

이스 내에 정의 하였다. 효율적 파일 관리를 위해 설계된 각 클래스 간의 관계 및 동작을 나타내고 있으며, 파일 시스템 API를 설계하기 위해 각 클래스에 필요로 하는 속성 및 동작을 정의 하고 각 클래스의 정의 내용을 <표 2>에 나타내었다.

4.3 파일 시스템 API의 구현

파일 시스템 API의 설계를 위해 언어에 관련 된 패키지로 java.lang를 사용하였고, 자바카드 애플릿의 중요 함수와 인터페이스와 클래스를 제공하는 javacard.framework 패키지를 사용하였다. 앞에서 정의한 파일시스템 API의 구현을 위해 필요한 각 클래스의 내용은 다음과 같다.

- ISOFS.class

ISOFS 클래스는 파일시스템에 사용되는 파일 크기, 타입, 에러 코드의 상수를 정의 하고 있다. 파일 타입은 크게 트랜스페어런츠와 레코드 파일, 사이클릭 레코드 파일로 구분하였고, 최대 파일 크기는 512byte이며, 파일 최대 개수 32개로 설정하였다. 파일 사이즈 및 속성의 수정 시 파일 시스템의 ISOFS 클래스에서 수정하여 사용할 수 있도록 설계, 구현하였다

- DataObject.class

DataObject 클래스는 파일 시스템에서 데이터 처리를 위해 설계한 클래스로 데이터의 길이와 버퍼를 정의하여 데이터를 읽고 기록할 수 있는 메소드를 정의한다. 자바카드기술에서 객체의 생성은 persistent와 transient 오브젝트를 제공한다. persistent는 비활성 메모리에 저장할 수 있는 기술로 EEPROM에 데이터를 기록하고, transient는 자바카드 연산에 필요한 메모리로 RAM에 데이터를 기록할 때 사용된다. 본 논문에서는 자바카드의 기술인 atomicity를 이용하여 persistent 오브젝트에 데이터를 기록할 때 데이터의 손실을 방지 한다.

```
JCSystem.beginTransaction();
    Util.arrayCopy(value, (short)0, data_buffer, (short)0, (short)len);
    cur_length = len;
JCSystem.commitTransaction();
```

- FileSystem.class

파일시스템 클래스는 사용하는 파일의 ID와 파일 Type을 조회, 쓰기 및 설정할 수 있는 동작을 정의 하고 있다. 스마트카드에서는 파일 ID와 Type을 설정하기 위해 2byte의 필수 헤더 부분을 두고 있어, 정의된 각 비트의 값으로 파일의 ID와 Type 등을 체크 할 수 있다. 그러나 자바카드는 애플릿 단위로 데이터를 처리함으로써 파일 ID와 Type만으로도 데이터를 처리 할 수 있다.

- TransparentFile.class

TranspasrentFile 클래스는 순차적으로 들어오는 데이터를 처리 하기 위해 파일 ID와 파일 Type과 길이를 인자로 받아 데이터를 처리 한다. 길이를 체크하여 정의 된 크기보다 클 경우 에러 메시지를 띄우게 설계한다.

```
if (len > ISOFS.MAX_FILE_SIZE)
    ISOException.throwIt(ISOFS.SW_MEM_FAILURE);
setFID(fid);
setFileType((byte)(ISOFS.FILE_TYPE_TRANSPARENT|file_type));
binary_data = new DataObject(len);
```

- RecordFile.class

RecordFile 클래스는 스마트카드에서 선형 고정와 선형 가변 파일을 정의 하고 있으나 본 논문에서 설계 및 구현은 선형 가변 파일은 선형 고정를 이용하는 형태로 좀 더 효율적인 메모리 사용을 위한 설계를 하였다. 레코드 파일은 파일 ID, Type, 총 레코드 개수, 레코드 크기를 인자로 받아 처리된다. 우선 레코드의 개수와 레코드 사이즈를 체크하여 바이너리 데이터의 배열을 잡는다. 레코드의 넘버를 셋팅하는 작업을 취하여 읽기, 쓰기 메소드를 정의한다.

```
byte i;
if ((short)(total_record_count*record_size) > ISOFS.MAX_FILE_SIZE)
  ISOException.throwIt(ISOFS.SW_MEM_FAILURE);
setFID(fid);
setFileType((byte)(ISOFS.FILE_TYPE_RECORD|file_type));
record_data = new DataObject[total_record_count];
for (i=0; i<total_record_count; i++)
record_data[i] = new DataObject(record_size);
total_record = total_record_count;
```

- **CyclicRecordFile.class**

스마트카드의 레코드 파일 시스템과 유사하며, 파일 시스템에서 가장 복잡하고 많은 코드로 설계, 구현된다. 기존 애플릿에서 사이클릭 파일 구조를 사용하게 되면 제한된 메모리 크기를 갖는 자바카드 내에서 가장 비효율적으로 메모리를 사용하게 된다.

```
byte i;
if ((short)((total_record_count+1)*rec_size) > ISOFS.MAX_FILE_SIZE)
    ISOException.throwIt(ISOFS.SW_MEM_FAILURE);
setFID(fid);
setFileType((byte)(ISOFS.FILE_TYPE_CYCLIC|file_type));
total_record = ++total_record_count;
record_data = new DataObject[total_record];
for (i=0; i<total_record; i++)
record_data[i] = new DataObject(rec_size);
record_size = rec_size;
```

본 논문에서 제시하는 파일 시스템 API를 사용할 때 메모리를 가장 효율적으로 사용할 수 있는 파일 종류이다. 사이클릭 클래스의 설계는 총 레코드 수와 레코드 사이즈를 체크하며 정해진 레코드에 추가되어지는 레코드를 쓰기위해 현재와 다음 레코드를 체크하여 데이터를 쓰게 된다. 응용프로그램 설계 시 가장 유용하게 사용 할 수 있는 구조로 초기 애플릿에서는 레코드를 추가 시켜주기 위한 작업으로 append 레코드를 선행해야 한다. 레코드 개수가 n개 이면 최근에 기록된 레코드가 1이고, 그전에 기록된 레코드가 2고 가장 먼저 기록된 레코드가 n이 된다.

4.4 파일 시스템 적용

본 논문에서 설계 및 구현된 파일 시스템 API를 카드에 탑재하여 애플릿 구현 시 참조하여 사용하게 된다.

```
import javacard.framework.*; //기본 API 참조
import dku.imagesys.FS.*; //파일 시스템 API 참조
```

파일 시스템 API를 이용하는 경우 애플릿에서 사용하려는 파일 시스템을 생성자에게 생성시켜 준 후 프로그래밍 하게 된다. (그림 7)은 객체 생성 후 각 메소드에서 파일 시스템을 이용하여 읽기 동작과 쓰기 동작 코드를 보여주고 있다.

(그림 7)에서 보듯이 기본 메소드인 install()을 실행할 때 생성자 부분을 넘겨받아 자바카드가 애플릿을 인스톨하게 되는데 이때 생성자 부분에 사용하려는 파일 종류를 지정하여 생성해준다. 예를 들어 읽기 동작을 실행하는 명령 APDU를 받아 process() 메소드에서 레코드의 넘버 및 사이즈를 읽어 cmdReadID 메소드로 넘겨 주게된다. 그 후 생성자에서 생성된 ID_File 객체의 readCyclicRecord 메소드를 호출하여 처리한다. 파일 시스템 API를 사용하지 않는다면 다음과 같은 작업을 계속적으로 반복해야한다. 그러나 파일 시스템 API를 이용하는 경우 사용할 레코드 번호를 선택하여 동일한 동작이 하나의 코드를 사용하게 됨으로써 코드의 중복을 줄일 수 있다. 쓰기 명령을 처리하기위한 방법 또한 동일하다.

```
protected FS_CyclicRecordExample(byte[] Data, short Offset, byte Length){
  ID_File = new CyclicRecordFile((short)0x3F01,
ISOFS.FILE_TYPE_WEF, NUM_ID_RECORD, SIZE_ID_RECORD);
  register();}
public void process(APDU apdu){
  byte[] buf = apdu.getBuffer();
  switch(buf[ISO 7816.OFFSET_INS])
  { case INS_READ_ID  : cmdReadID(apdu);
break;
    case INS_WRITE_ID                : cmdWriteID(apdu);
break;
     ISOException.throwIt(ISO 7816.SW_INS_NOT_SUPPORTED);
break;   }}
private void cmdReadID(APDU apdu){
  byte[] buf = apdu.getBuffer();
  // Ready Response Data
  ID_File.readCyclicRecord(buf[ISO 7816.OFFSET_P1], buf,
  SIZE_ID_RECORD);}
private void cmdWriteID(APDU apdu){
  byte[] buf = apdu.getBuffer();
  // Write CyclicRecord
  ID_File.writeCyclicRecord(buf[ISO 7816.OFFSET_P1], buf,
 (short)ISO 7816.OFFSET_CDATA, buf[ISO 7816.OFFSET_LC]);}
```

(그림 7) 파일시스템 API를 이용한 애플릿 설계 적용 예

5. 실험 및 실험 결과

5.1 파일 시스템 실험 환경 및 구현

본 논문에서의 파일 시스템 API 구현은 Pentium4 2.99GHz에 512MB의 메모리 환경에 Windows XP Professional OS 환경에서 자바카드 개발 툴 2.1.2의 스펙을 이용하였다. 자바통합 환경인 Eclipse에 JCOP Tools 3.0을 플러그 인하여 프로그래밍 하였다. 사용된 카드는 JCOP Bio31 카드로 16Kbyte EEPROM, 24Kbyte의 ROM, 2300byte의 RAM을 내장하고 16bit의 CPU로 구성되어 있는 자바카드를 사용하였다. 또한 외부 어플리케이션 프로그램과 명령 데이터를 주고받기 위해 사용한 PC/SC리더기는 JSTU 9750 모델을 사용하였다. Eclipse는 코드를 작성할 수 있는 에디터, 클래스 파일을 만드는 컴파일러과정과 CAP파일을 생성해주는 Converter 및 카드에 Loading 시켜주는 역할을 한다.

실험 방법은 기존 API를 이용해서 작성된 애플릿과 본 논문에서 제시하는 파일 시스템 API를 이용한 애플릿을 각각 트랜스페어런츠 파일과, 레코드 파일, 싸이클릭 파일을 지원하는 총 6개의 애플릿을 구현하여 동작 검증 및 파일 시스템 API의 효율성을 테스트하였다.

〈표 3〉 테스트 애플릿 APDU 헤더 정의

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 Record일 경우 Record 선택
P2	0x00	P2는 사용하지 않는다.

애플릿 구현에 필요한 APDU의 헤더 부분을 <표 3>과 같이 정의하여 총 6개의 애플릿에 적용하였다.

트랜스페어런츠를 이용하는 애플릿은 파일 사이즈를 32byte의 크기로 정의하고, 레코드 구조의 애플릿은 32byte의 레코드를 총 5개를 갖는 크기로 정의하여 애플릿을 구현하였다. 파일 시스템 API를 카드에 탑재 시켜 이를 참조하여 애플릿을 구현할 경우 애플릿에 추가 API를 설정해주어야 한다.

5.2 실험 결과

동작에 대한 테스트는 이클립스에서 지원하는 디버깅툴을 이용하여 처리 결과 및 클래스의 정보, CAP 파일 크기, 처리 속도를 확인할 수 있었다.

일반적으로 애플릿 설계는 APDU 관련 명령(command), 생성자(constructor), 메소드 install(), select(), deselect(), process(), Java Card 애플릿 라이프사이클, private 메소드를 정의 등과 같은 사항을 고려한다.

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

```
· 애플릿 Select
=> send 00A4040008 0011223344556603 Cyclic파일 Applet AID
<= 9000
· Append Record
=> send 0025010020 11223344556677889900AABBCCDDEEFF
<= 9000
=> send 0025020020 12223344556677889900AABBCCDDEEFF
<= 9000
· Read Record
=> send 0024010020
<= 12223344556677889900AABBCCDDEEFF9000
=> send 0024020020
<= 11223344556677889900AABBCCDDEEFF9000
· Write Record
=> send 0023010020 A1223344556677889900AABBCCDDEEFF
<= 9000
=> send 0023020020 A2223344556677889900AABBCCDDEEFF
<= 9000
· Read Record
=> send 0024010020
<= A2223344556677889900AABBCCDDEEFF9000
=> send 0024020020
<= A1223344556677889900AABBCCDDEEFF9000
```

(그림 8) 파일시스템 AIP를 이용한 애플릿 동작 결과

〈표 4〉 애플릿 테스트 결과

<table>
<tr><td colspan="5">File System Package(code size : 789byte)
File name : FSISO 7816
Package : dku.imagesys.FS
PID : 00 11 22 33 44 55 (6byte)</td></tr>
<tr><td colspan="2" rowspan="2">파일종류 / 항목</td><td colspan="3">기존 파일시스템 애플릿
/(본 논문제안) 파일시스템 API 사용 애플릿</td></tr>
<tr><td>Tran.</td><td>Rec.</td><td>Cyc.</td></tr>
<tr><td colspan="2">Code Size on card(byte)</td><td>251 / 251</td><td>294 / 253</td><td>486 / 295</td></tr>
<tr><td rowspan="2">속도</td><td>Write data(ms)</td><td>110 / 109</td><td>109 / 109</td><td>125 / 110</td></tr>
<tr><td>Read data(ms)</td><td>94 / 79</td><td>82 / 78</td><td>94 / 78</td></tr>
</table>

(그림 8)은 FS_CyclicRecordExample의 애플릿을 테스트한 결과이다. 카드의 동작은 사용 애플릿을 선택하고 첫 번째 레코드를 생성하기 위한 추가 명령과 기록 될 32byte의 데이터를 카드에 전송하면 카드는 사이클릭 파일의 레코드를 생성하여 기록하게 된다. 레코드의 길이가 잘못되었을 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 응답 APDU를 전송하게 된다. 예를 들어 크기 32byte의 레코드면 0x6C20의 응답 APDU가 전송된다. 정상적으로 동작하였을 경우 0x9000의 응답 APDU가 전송된다.

<표 4>에 기존 API를 이용한 경우와 본 논문에서 제시하는 파일 시스템 API를 이용한 애플릿 테스트 결과를 보였다. 하나의 애플릿이 하나의 파일 구조를 갖는 형태로 작성하였으며, 코드 길이와 데이터를 카드에 쓰고, 읽는 속도를 비교하였다. 본 논문에서 구현된 파일 시스템 패키지의 코드 길이는 총 789byte였다.

자바카드 EEPROM의 16Kbyte중 4Kbyte에는 기본 API가 탑제되어 있다. 기본 API를 이용하여 Cyclic 파일을 구현할 경우 대략 500byte 정도의 코드를 사용하게 된다. 그러나 본 논문에서 제시하는 파일 시스템 API를 사용할 경우 300byte 정도의 코드로 사이클릭 레코드 파일을 구현 할 수 있다.

만약 사이클릭 레코드 구조를 갖는 애플릿 10개를 카드에 내장할 때 메모리 사용은 다음과 같다.

- 일반적으로 애플릿 설계할 경우

  486 × 10 = 4860 byte

- 파일시스템 API를 이용하여 애플릿을 설계할 경우

  295 × 10 + 789 = 3739 byte

위와 같이 약25%가 감소되었다. 또한 파일 시스템 API를 사용하지 않고 하나의 애플릿에 여러 파일 구조를 지원해야 하는 경우 코드의 길이가 더욱 늘어나게 될 것이다. EEPROM에 데이터를 쓰는 속도 또한 향상됨을 볼 수 있다. 데이터를 EEPROM에 쓰기 전에, EEPROM에 기록되어지는 곳의 내용을 먼저 지우고, 데이터의 내용을 쓰게 된다. 스마트카드의 경우 4byte 기록을 위해 3.5ms의 시간이 소요된다. 본 논문에서 사용되어지는 자바카드 처리는 사용 데이터나 어플리케이션 APDU에 따라 속도 차이는 조금씩 있으나, 128byte를 기록하기 위해 소요된 시간이 스마트카드를 기준으로 헀을 때 약 7% 향상된 결과를 얻을 수 있었다.

6. 결 론

자바카드는 스마트카드의 플랫폼에 JCVM을 탑재하여 개인의 인증 및 정보 보호를 위해 각광받고 있다. 자바카드의 운영은 사용 응용 분야에 따라 애플릿을 작성하여 동작된다. 애플릿 작성을 위해 SUN사에서는 API를 지원하고 필요에 따라 추가할 수 있게 되어있다. 최근 개인의 많은 정보 데이터를 처리하기 위해 파일 관리가 필수적이다. 그러나

SUN사에서 제공하는 기본 API에서는 파일 관리를 위한 API를 지원하지 않고 있다.

본 논문에서는 애플릿이 데이터를 처리하는데 있어 효율적인 데이터 처리를 위해 파일 시스템 API를 설계 및 구현하였다. 설계한 파일 시스템 API는 기존 스마트카드에서 사용 중인 파일 시스템과 호환성을 위해 ISO 7816의 파일 시스템 구조를 참조하였다. 파일 시스템 API는 트랜스페어런츠, 레코드, 사이클릭 레코드 3개의 파일 타입을 정의하였고, 파일 구현 시 파일의 ID, 파일의 Type, 길이, 레코드 사이즈, 레코드 크기 등을 고려하여 공통된 부분 및 동작을 정의하였다.

본 논문에서 설계 구현한 파일 시스템 API의 크기는 789byte이었고 동작을 검증하였다. 만일 10개의 애플릿을 사용한다고 가정하여 파일시스템을 이용하게 되면 코드의 크기가 약25%의 감소됨을 확인하였다. 또한 카드 애플릿과 터미널 간의 파일 처리 및 전송 속도도 적은 코드를 수행하게 되는 파일 시스템 API를 이용한 경우 7%의 효율적인 속도 향상을 볼 수 있었다.

본 논문에서 제안한 파일 시스템 API를 사용하는 경우 애플릿 개발 시 단일화된 파일 입출력과 인터페이스를 제공할 수 있어 개발에 필요한 노력과 시간을 줄 일 수 있을 뿐만 아니라 개발의 편이성, 데이터의 캡슐화 시켜 다양한 애플릿을 개발 하는데 로딩 되는 시간을 줄 일 수 있다. 또한 atomicity를 이용하여 카드의 EEPROM에 데이터를 기록할 때 데이터 보호를 할 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.

참 고 문 헌

[1] W. Rankl, W. Effing, "Smart Card HandBook," WILEYVCH, 2000.

[2] Timothy M. Jurgensen, Scott B.Guthery, "Smart Cards," Person Education, 2002.

[3] S. Oaks, "JAVA Security," O'REILLY, 1998.

[4] Z. Chen, "Java Card Technology for Smart Cards," Addison Wesley, 2000.

[5] V. Hassler, M. Manninger, M. Gordeev, C. Muller, "Java Card for E-Payment Application," Artech House, 2002.

[6] M. Oestreicher, "Transactions in Java Card," Annual Computer Security Application Conference, pp.291~298, 1999.

[7] L. Casset, L. Burdy, A. Requet, "Formal Development of an Embedded Verifier for Java Card Byte Code," International Conference on Dependable System and Networks, pp.51~56, 2002.

[8] 김호원, 최용제, 김무섭, 박영수, "비대칭키 암호 알고리즘을 고속으로 수행하는 자바카드 구현 및 성능 평가", 대한전자공학회 하계종합학술대회 논문집 제24권 제1호, pp.55~58, 2001.

[9] 문상재, 이필중, "차세대 IC 카드를 이용한 정보보호 신기술 시스템 개발", 정보통신부 보고서, 1997.

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

[11] 김성준, 이희규, 조한진, 이재광, "자바카드 기반 공개키 암호 API를 위한 임의의 정수 클래스 설계 및 구현", 정보처리학회, 9권 2호, pp.163~172, 2002.

[12] 임현준, 김현아, 정재우, 김광훈, "Java Card SIM API의 Toolkit Registry 구현에 관한 연구", 정보처리학회 추계학술 발표대회 제9권 제2호, pp.1~4, 2002.

[13] 김도우, 정민수, "자바카드 플랫폼상에서 자바 클래스 파일의 최적화 연구", 멀티미디어학회 논문지, 6권 7호, pp.1200~1208, 2003.

[14] 황선명, 염희균, "자바카드 애플릿의 검증 방법", 정보처리학회 소프트웨어공학연구회지, 제5권 1호, pp.36~46 2002.

[15] 이정우, 전성의, "자바카드에서 Post-issuance API에 관한 연구", 정보과학회 가을 학술발표 논문집 2002권 pp.583~585, 2002.

[16] Uwe Hansmann 외, "Smart Card Appli- cation Development Using Java," Springer, 2002.

[17] SILBERSCHATZ외, "Operating System Concepts," WILEY, 2002.

[18] Jess Garms, Daniel Somerfield, "Java Security," 정보문화사, 2002.

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

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

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

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

송 영 상

e-mail : yssong@dankook.ac.kr

1998년 삼척산업대학교전자공학과(공학사)

2000년 단국대학교(공학석사)

2000년~현재 단국대학교 박사과정

관심분야 : 정보보안, 전자상거래, 스마트카드, 자바카드 등

신 인 철

e-mail : char@dankook.ac.kr

1973년 고려대학교 전자공학과(공학사)

1978년 고려대학교(공학석사)

1986년 고려대학교(공학박사)

1979년~현재 단국대학교 전자·컴퓨터공학부 교수

관심분야 : 정보보안, 스마트카드, 디지털 워터마킹, 전자상거래