Journal of Korea Multimedia Society Vol. 11, No. 3, March 2008(pp. 404-413)

File Cache 및 Direct Access기능을 추가한 Java Card File System에 관한 연구

이윤석†, 전하용††, 정민수†††

요 약

유비쿼터스 사회화에 따른 개인의 정보를 보호하기 위한 방안들이 많이 제시되고 있다. 이런 방안의 한 분야로 CPU와 메모리를 가진 스마트카드가 널리 사용되고 있으며, 스마트카드 중에서도 다양한 응용 프로그램을 사용 가능하게 하는 자바카드의 사용이 확대되고 있다. 자바카드 파일 시스템의 표준은 따로 정의되어 있지 않지만, 일반적으로 스마트카드 파일 시스템 표준을 따른다. 하지만 스마트카드 파일 시스템 표준을 따름에 있어서 자바카드 가상기계의 특성상 데이터 및 코드의 중복사용으로 메모리 공간의 비효율적인 사용과 처리 속도가 늦어지는 단점을 가지고 있다. 따라서 본 논문에서는 자바카드의 이러한 단점을 해결하기 위해 File Cache 기법과 Direct Access 기법을 제안하여 최소한으로 코드 수를 줄여 메모리 공간의 효율적인 사용과 처리 속도를 개선한다.

A Study of Java Card File System with File Cache and Direct Access function

Yun-Seok Lee†, Ha-Yong Jun††, Min-Soo Jung†††

ABSTRACT

As toward a ubiquitous society, a lot of methods have been proposed to protect personal privacy. Smart Cards with CPU and Memory are widely being used to implement the methods. The use of Java Card is also gradually getting expanded into more various applications. Because there is no standards in Java Card File System, Generally, Java Card File System follows the standards of Smart Card File System. However, one of disadvantages of the Java Card File System using a standard of Smart Card File System is that inefficient memory use and increasing processing time are caused by redundancy of data and program codes. In this paper, a File Cache method and a Direct Access method are proposed to solve the problems. The proposed methods are providing efficient memory use and reduced processing time by reduce a program codes.

Key words: Smart Card(스마트카드), Java Card(자바카드), Java Card File System(자바카드 파일 시스템)

※ 교신저자(Corresponding Author) : 정민수, 주소 : 경남 마산시 월영동 449번지(631-701), 전화 : 055)249-2217, FAX : 055)248-2554, E-mail : msjung@kyungnam.ac.kr
접수일 : 2007년 10월11일, 완료일 : 2008년 2월11일
† 준회원, 경남대학교 컴퓨터공학과 석사과정
(E-mail : lysis2jt@yahoo.co.kr)
†† 준회원, 경남대학교 대학원 컴퓨터공학부 박사 수료
(E-mail : hayongj@kyungnam.ac.kr)
††† 종신회원, 경남대학교 컴퓨터공학부 교수
※ 본 연구는 2007년 경남대학교 학술연구 장려금 지원으로 수행되었음

1. 서 론

유비쿼터스 사회로 발전해 감으로 해서 개개인의 정보는 소형화된 데이터 저장소에 그 정보를 저장해 놓고 필요시에 활용하는 형태로 변화되어져 왔다. 이제는 보편화 된 신용카드와 은행에서 사용하는 금융카드 그리고 교통카드 역시 개인의 정보를 저장, 관리하고 있으며 이를 활용하여 금융 결재 및 사용자 인증 등에 사용된다. 이러한 개인의 정보를 안전하게 그리고 효율적으로 관리하는 스마트카드 중에 여러 응용 서비스를 지원하는 자바카드가 현재 널리 사용되고 있는 추세이다[1].

마이크로 프로세서를 내장한 스마트카드의 한 종류인 자바카드는 자바카드 가상기계를 탑재하여 하나의 카드를 가지고 금융, 교통, 모바일 등 각종 응용 프로그램을 탑재함으로 해서 범용성을 강조한 카드이다[2]. 하지만 자바카드의 범용성은 여러 서비스를 제공하는 다수의 응용 프로그램을 사용함에 따라 자원의 효율적 관리가 필수적으로 대두되며 또한 응용 프로그램 코드를 저장하는 메모리 공간이 한정 되어 있기에 코드를 초소형 환경인 자바카드 환경에 얼마만큼 적절하게 설계하는가가 중요한 문제가 되었다. 그리고 동일한 데이터를 사용하는 두 개 이상의 응용 프로그램의 경우에는 이 동일한 데이터를 관리하는데 상당한 어려움이 존재하게 된다[3-5].

본 논문에서는 File Cache 및 Direct Access 기능이 추가된 자바카드 파일시스템을 구현하여 하나의 카드로 여러 응용 프로그램에 사용되는 각종 정보를 효율적이고 안전하게 관리가 가능하도록 설계하였으며, ISO7816-4의 스마트카드 파일 구조를 자바카드 API로 설계 및 구현하였으며 USIM(Universal Subscriber Identity Module) 환경에도 적용하여 사용할 수 있도록 3G 표준을 준수하였다. 그리고 어플리케이션 처리 성능을 향상시키고자 File Cache를 사용하여 자주 사용하는 파일에 대한 접근을 빠르고 효율적으로 할 수 있으며, Direct Access를 사용하여 파일 참조에 따르는 다중 SELECT 명령 사용을 해결함으로서 코드의 중복 사용을 줄이고 처리 속도를 향상 시킬 수 있었다.

2. 관련연구

스마트카드는 크게 메모리 카드와 마이크로 프로세서 카드로 구분할 수 있다. 마이크로 프로세서를 보유하지 않은 메모리 카드는 단순한 데이터 저장의 기능을 가지고 있으며 여기에는 데이터가 보호되는 카드와 데이터가 보호되지 않는 카드로 구분된다. 마이크로 프로세서를 보유한 카드로는 전용 운영체제를 탑재한 카드와 오픈 플랫폼 카드로 구분되며 자바카드의 경우에는 오픈 플랫폼 카드에 속하게 된다. 이러한 자바카드의 경우에 다양한 형태의 응용 프로그램들을 애플릿 형태로 다운로드 받아 사용할 수 있으며 이를 통해서 사용자가 원하는 형태의 응용 서비스를 손쉽게 사용할 수 있게 된다[6,7].

2.1 자바카드 플랫폼

자바카드의 플랫폼은 그림 1에서 보는 것과 같이 구성되어 있으며 최하위 단인 하드웨어에는 중앙처리장치와 입출력, 롬, 램, EEPROM/플래쉬 메모리 등이 기본적으로 구성되어 있으며 또한 많은 양의 연산을 요구하는 암호화 작업을 보다 고속으로 처리해 주는 암호 전용 프로세서가 선택적으로 구성되어져 있다[3-5]. 또한 이를 제어, 관리하기 위해서 초소형의 칩 운영체제가 롬 영역에 상주해 있게 된다. 그리고 자바 플랫폼 독립의 특성을 가지게 만들어주는 가상기계는 롬 영역에 상주하게 되며 사용자가 원하는 API와 애플릿을 선 발급 또는 후 발급에 의해 카드의 롬 영역 또는 EEPROM/플래쉬 메모리 영역에 상주하여 APDU(Application Protocol Data Unit) 통신 프로토콜을 통해 호스트와 통신을 진행하게 된다[6-9].

2.2 자바카드의 통신방식

자바카드의 통신방식은 ISO7816-4에 정의된 스

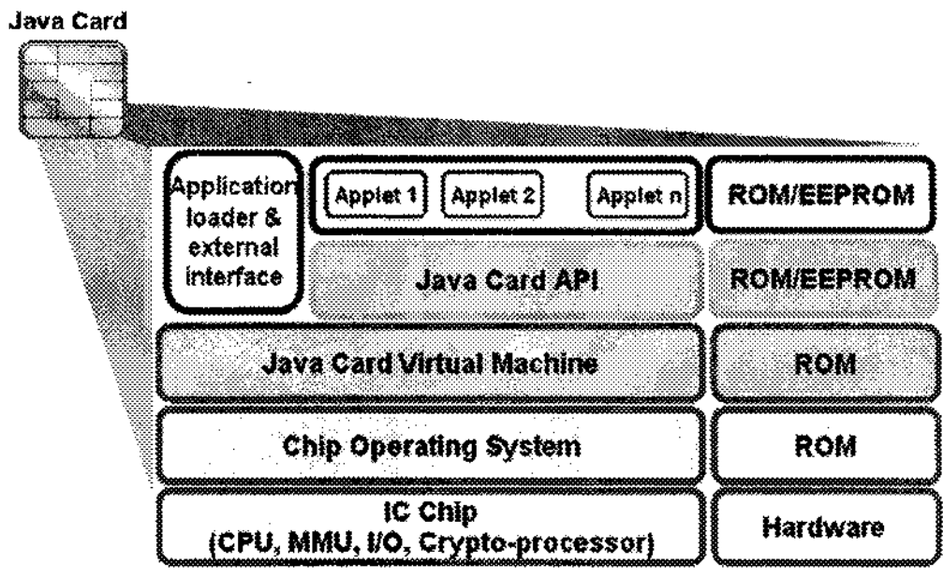

그림 1. 자바카드 플랫폼

마트카드의 표준통신 프로토콜인 APDU 통신 프로토콜을 준수한다. 그림 2와 같이 APDU 통신 프로토콜은 카드에서 단말기 또는 단말기에서 카드로 전송되는 명령어 메시지나 응답 메시지를 포함하며, 명령-응답의 쌍으로 존재하는 명령어 메시지 및 응답 메시지는 데이터를 포함 할 수 있다[6-10].

명령 APDU는 헤더와 바디로 구성되어져 있으며 헤더에 대한 설명은 다음과 같다.

(1) CLA : 명령의 클래스
(2) INS : 명령 코드
(3) P1 : 명령 파라메터1
(4) P2 : 명령 파라메터2
(5) LC : 명령어 데이터 필드 내에 존재하는 바이트 수
(6) Data : 명령어로 보내는 데이터 바이트
(7) LE : 응답 데이터 필드에서 예상되는 최대 바이트 수

응답 APDU는 바디와 트레일러로 구성되어져 있으며 바디의 경우에는 명령 APDU의 LE 길이 만큼의 데이터이며 트레일러의 경우에는 ISO7816-4에 정의된 상태 워드 또는 사용자가 정의한 상태 워드가 나타난다[6-10].

2.3 스마트카드 파일시스템

카드의 파일 시스템을 어떻게 구성하느냐가 카드 운영에 있어서 얼마만큼의 유연성과 확장성을 제공하는지를 결정하는 가장 큰 요소가 되며, 파일시스템은 카드 내의 하드웨어 사양에 맞는 적절하고 유연성 있는 프로그래밍 기법이 요구된다. IOS7816-4 스마트 카드 파일시스템은 도스와 같은 계층적인 파일 시스템과 아주 유사하다[3-5]. 각각의 파일들은 Short 형으로 선언된 파일 식별자(FID)를 가지며 이는 한 DF(Dedicated File)내에서 유일한 식별자이다. 스마트카드에서 파일의 종류는 아래 그림 3에서 보듯이 크게 3종류이다. 먼저 MF(Master File)의 경우에는 DF의 특별한 종류로 루트와 같으며 카드 내에서 유일하게 존재하며, MF의 일반적인 파일 식별자는 '3F00'를 가진다. DF는 특정 응용 프로그램이나 응용 서비스관련 EF(Elementary File)들의 그룹으로 일반적인 디렉토리나 폴더와 같은 역할을 한다. 그리고 EF의 경우 DF(MF 포함) 하위에 존재하며 파일별로 접근제어 조건이 존재하여 조건이 맞을 경우에만 파일에 대한 접근이 허용된다. DF와 MF의 경우에는 데이터를 소유하지 않으며 오로지 EF의 경우에만 데이터를 소유한다[3-5,8,9].

데이터를 소유하는 EF는 데이터를 저장하기 위해서 4가지 형태의 구조를 가지며 그 형태대로만 저장하게 된다.

(1) 트랜스페어런츠 파일 : 연속된 바이트로 구성되며 옵셋을 사용하여 파일 내에 기록된 데이터를 참조한다.
(2) 선형 고정길이 레코드 파일 : 모든 레코드가 동일한 크기를 가지고 있는 파일로서 레코드 번호는 생성 순서에 따라 연속적으로 할당된다. 레코드의 개수는 카드 발급 시에 결정된다.
(3) 선형 가변길이 레코드 파일 : 모든 레코드의 크기가 동일하지 않는 구조를 가지고 있다. 레코드 번호는 생성 순서에 따라 연속적으로 할당되며 파일의 크기는 카드 발급시에 결정된다.
(4) 싸이클릭 파일 : 선형 고정길이 레코드 파일과 비슷하지만 환형 구조이다. 각 레코드 의 크기

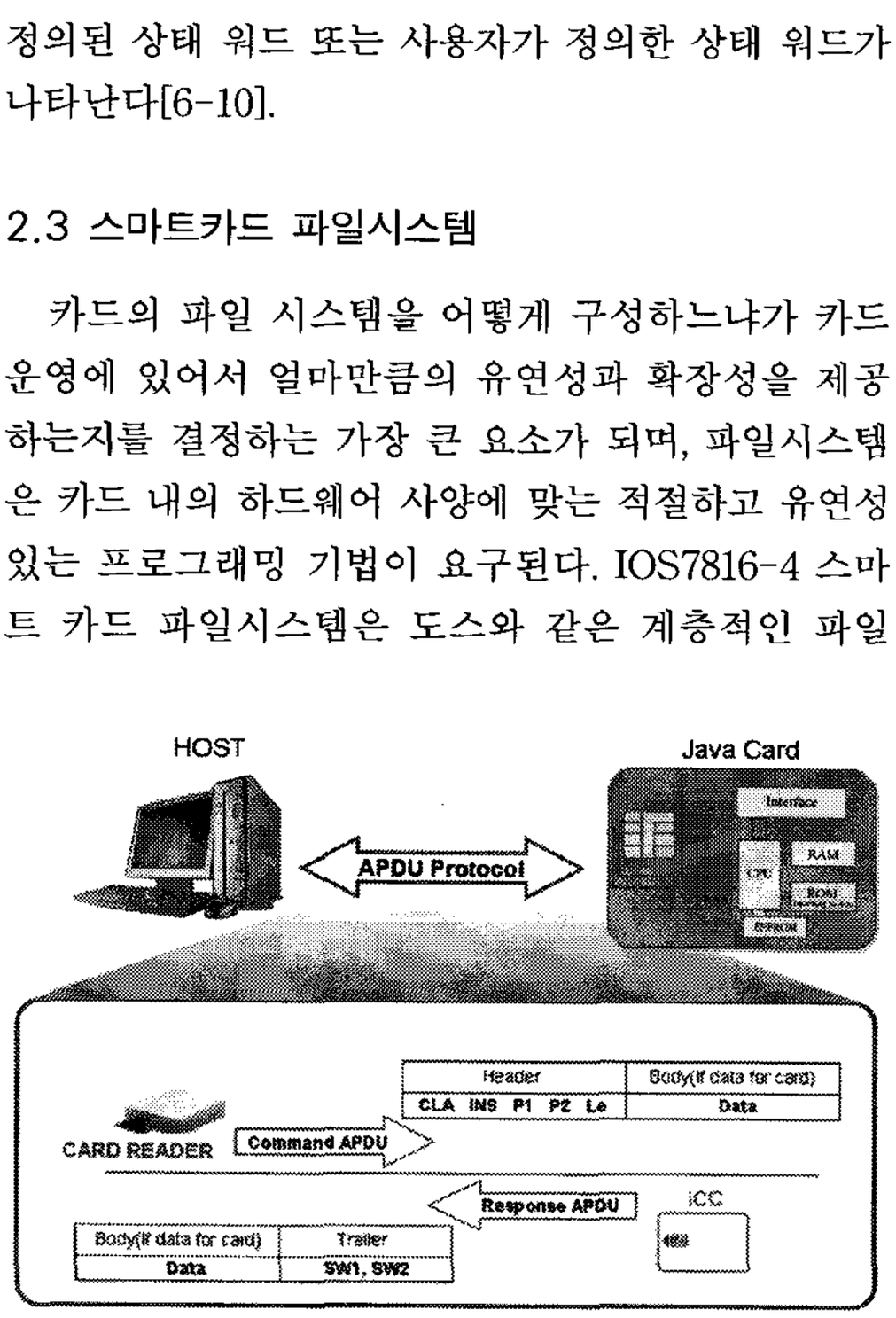

그림 2. 자바카드의 APDU 통신 방식

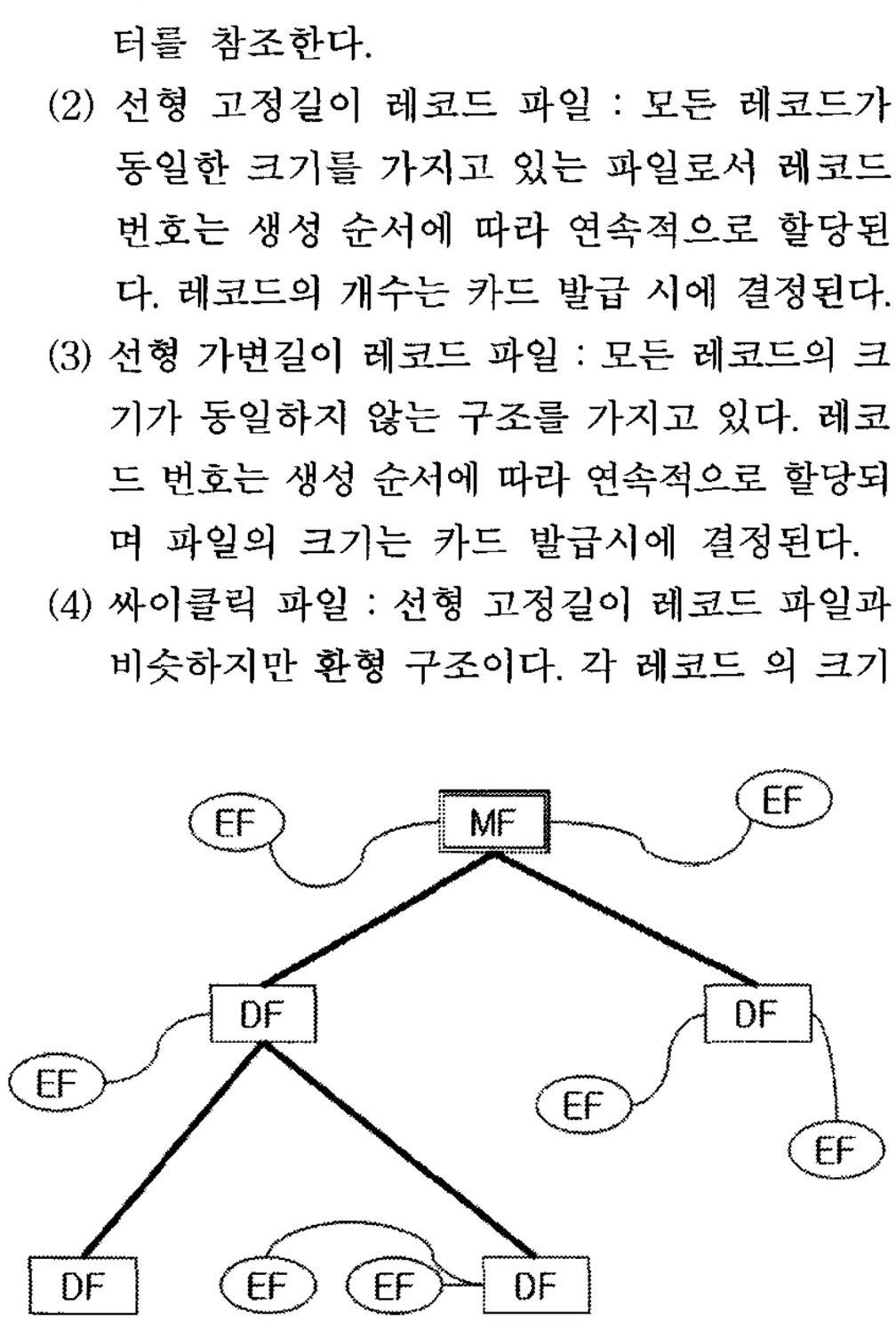

그림 3. ISO7816-4에 정의된 파일 구조

는 동일하며 레코드 번호는 역순에 따라 연속적으로 할당된다[3-5,8,9].

3. 제안 자바카드 파일시스템 설계

자바카드의 어플리케이션의 실행은 애플릿 이라는 최소단위의 프로그램으로 동작을 수행하게 된다. 이러한 애플릿은 자바카드 API를 사용하여 메모리 관리, 데이터의 저장 및 수정 그리고 암호화 작업을 수행하게 된다[7,11]. 자바카드 파일시스템이 존재하지 않는 구조에서는 동일한 데이터가 여러 애플릿에 존재하여도 이를 통합적으로 관리하는데 있어서 상당한 어려움이 있다. 또한 이런 데이터를 관리하기 위한 일련의 작업을 수행하는 메모리 관리 코드 역시 각각의 애플릿이 중복적으로 사용하고 있음으로 해서 코드의 직접적인 저장이 이루어지는 롬영역의 감소가 발생하게 되고 데이터를 저장 관리하는 EEPROM 또는 플래쉬 메모리의 감소를 유발하게 된다[12]. 이와 같은 문제를 해결하기 위해서 자바카드에 파일시스템을 API로 설계, 구현하여 애플릿에서 파일시스템 API를 통한 파일들의 제어 및 관리를 가능하도록 하였다[5].

본 논문에서는 ISO7816-4에 정의된 스마트 카드 파일시스템 표준에 따르는 자바카드 파일시스템을 설계 및 구현하였다. 하지만 이러한 스마트 카드 표준에 따르는 구현은 실행 시에 몇 가지 문제점이 발생된다. 먼저 자바카드 파일 시스템은 API로 구성되어있어 인터프리터에 의해서 번역되어야 하며 이를 통한 작업은 네이티브로 구현된 소스에 비해서는 느린 속도를 가지게 만든다[11,13]. 애플릿에서부터 빈번하게 일어나는 데이터의 저장 및 관리에 관한 각종 명령들은 이러한 API를 통해서만 수행되게 됨으로서 느린 속도의 데이터 관리는 응용 서비스 전체에 성능 저하를 유발하는 요인이 된다. 또한 파일 참조 제약에 따른 성능 저하도 유발시킨다. 느린 속도의 자바카드 API에서는 파일 시스템에 대한 명령수행이 최소한으로 그쳐야 한다. 하지만 그림 4와 같이 트리 구조를 가지는 자바카드 파일시스템에서 현재 선택된 DF의 하위 DF나 EF, 또는 상위DF, MF만 참조할 수밖에 없는 구조에서는 선택된 DF가 아닌, 다른 DF의 EF또는 DF를 선택하기 위해서 여러 경로의 SELECT명령을 수행한 후에 원하는 파일을 선택하는 작업이 필수적이다[12,14]. 이러한 코드의 중복 사용을 해결하고 파일시스템에서 가장 많이 사용되는 명령인 SELECT명령에 대한 속도 개선을 FileCache와 DirectAccess 기능을 통해 응용 서비스 전체의 성능을 향상시킬 수 있다.

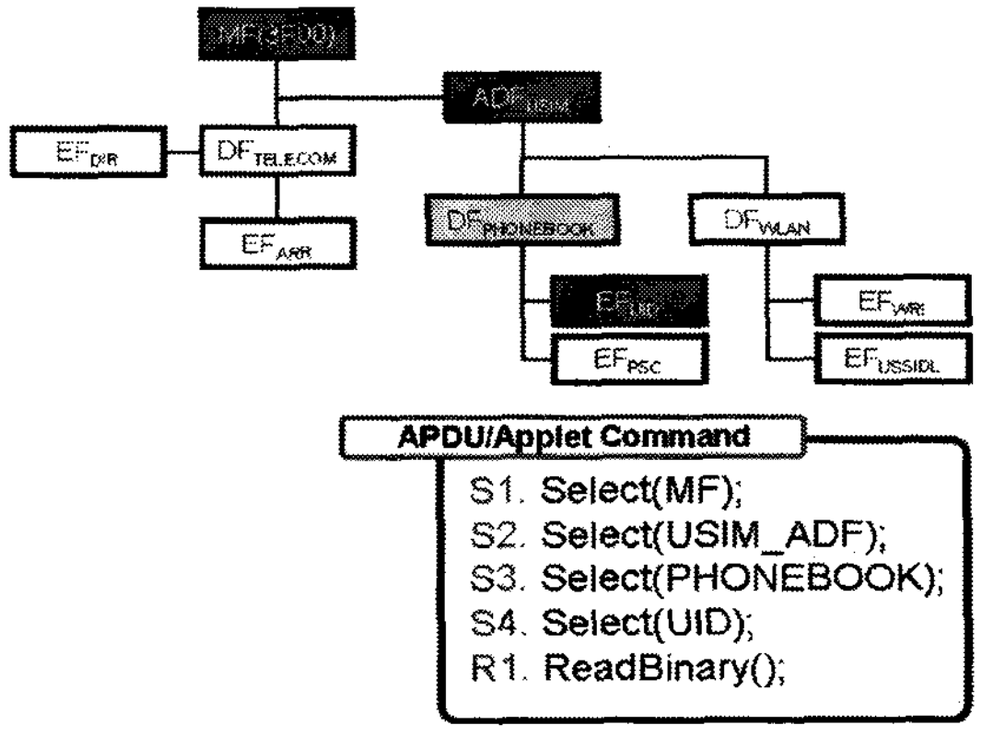

그림 4. 파일 선택을 위한 명령 예

3.1 File Cache 설계

자바카드 파일시스템에서 파일 식별자는 현재 선택된 DF내에서 파일의 검색 시에 사용 된다. 그리고 파일 검색은 파일 식별자를 통한 순차검색을 진행하게 되는데 이러한 순차 검색의 속도는 파일의 개수에 종속적일 수 밖에 없다. USIM을 사용하는 3G 모바일 폰 환경에서는 한 DF내에 적게는 2개에서 많게는 84개에 이르는 하위 파일들을 가진다. 이와 같은 파일들의 파일식별자를 일일이 자바카드 API에서 파일식별자를 통한 순차 검색을 진행한다는 것은 그만큼의 속도를 저하시키는 요인이 된다. 또한 자바카드 파일시스템에서도 자주 사용되는 파일은 항상 존재하기 때문에 이처럼 자주 사용되는 파일의 경우에 더 빨리 SELECT 되게 함으로서 그 속도를 향상시킬 수 있다.

본 논문에서 제안하는 File Cache는 선입선출 구조를 가지는 큐 형태로 설계하였으며 최근에 선택한 10개 파일의 오브젝트 ID를 소유하고 있다. 또한 동일 DF 내에서만 캐쉬가 동작하도록 설계 하였으며, 캐쉬를 통한 SELECT명령이 실패할 경우에는 자바카드 API에서 ISO7816-4 스마트카드 표준에 따라 구현된 자바카드 파일시스템의 일반적인 SELECT 명령이 수행되게 설계하였다. 아래 그림 5는 본 논문

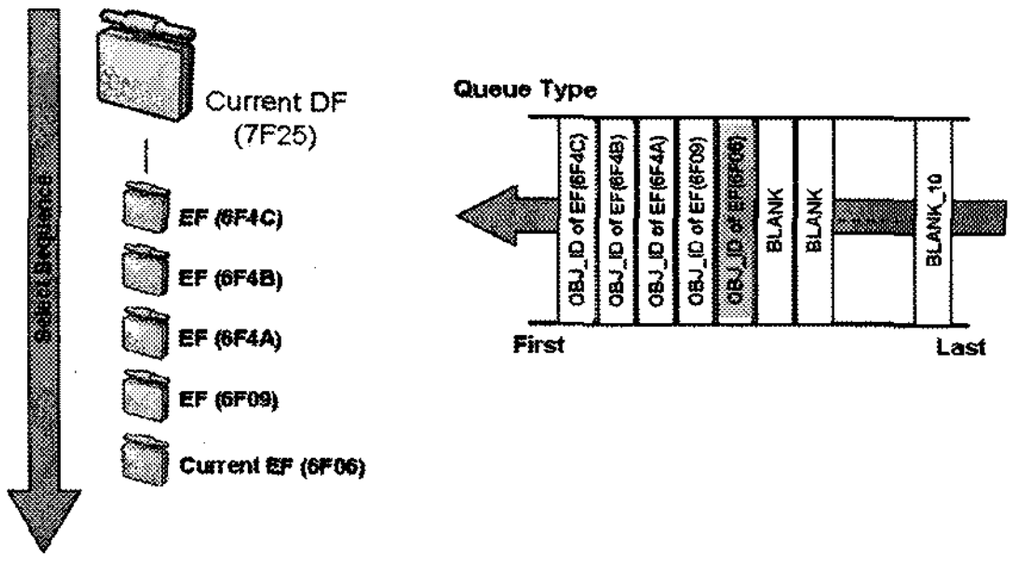

그림 5. File Cache의 설계

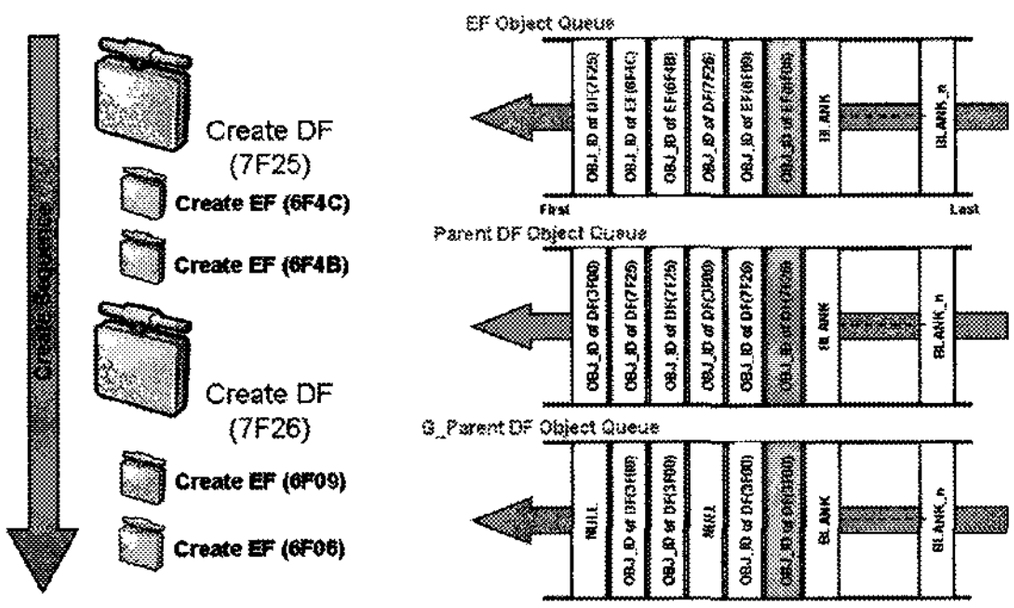

그림 6. Direct Access의 설계

에서 제안하는 File Cache로 현재 선택된 DF 내의 EF가 SELECT되는 순서에 따라 파일의 오브젝트 ID를 저장하며, 10개의 오브젝트가 가득 찰 경우에는 가장 먼저 들어온 오브젝트 ID를 삭제하고 새로운 오브젝트 ID를 저장하게 된다. 그리고 만약 SELECT를 통한 파일 SELECT작업을 수행도중 현재 DF의 변경이 일어날 경우에는 File Cache는 초기화가 된다. 이러한 형태의 File Cache의 설계는 초소형의 자바카드 환경에 따른 적은 메모리의 소모와 더불어 동일 DF내에서만 동일 파일 식별자를 가지는 특성에 따라 동일 DF내에서 파일 SELECT에 대한 최근의 10개의 파일에 대해서만 오브젝트 ID를 저장 및 관리 하도록 설계, 구현하였다.

3.2 Direct Access 설계

ISO7816-4에 따른 파일구조로 인해 현재 DF가 아닌 다른 DF의 SELECT를 요구하는 경우에는 MF로부터 시작하는 경로 전체를 SELECT하여야 한다. 이와 같은 ISO7816-4에 정의된 스마트 카드 파일시스템의 참조 제약은 READ, UPDATE등의 작업을 위해서 최소 2회 이상의 SELECT 명령을 수행해야 하는 중복 코드 사용이 발생한다[12]. 또한 접근제어 조건의 경우에도 데이터를 소유하는 EF에만 존재하는 것이기 때문에, 최종 EF를 SELECT하기 위한 경로 DF의 SELECT는 전체 응용 프로그램에서의 시간만 증가시키게 된다. 이러한 문제를 해결하기 위해 본 논문에서는 단일 명령으로 현재 DF가 아닌 다른 DF에서도 SELECT가 가능한 Direct Access를 제안한다.

그림 6과 같이 설계한 Direct Access는 File Cache와는 달리 모든 EF 및 DF에 대한 정보를 소유하고 있어야 한다. 그로 인해서 저장되는 데이터의 크기가 많아지게 되는데 먼저 EF 오브젝트 큐의 경우에는 CREATE명령에 의해 생성되는 순서대로의 모든 파일의 오브젝트가 저장되고 부모 DF 오브젝트 큐의 경우에는 EF 오브젝트 큐의 부모 계층인 DF 오브젝트 ID가 저장되며, 그리고 G_Parent DF 오브젝트 큐의 경우에는 부모 DF 오브젝트 큐의 부모 계층의 DF 오브젝트 ID가 저장되게 되어 있다. 서로 다른 DF내에서는 동일한 FID를 가진 파일이 존재할 수 있음으로 해서, 생성되는 모든 파일의 계층에 대한 정보를 3개의 큐에 저장하여 자바카드 파일시스템 내에서 파일의 유일성을 확보 할 수 있다.

4. 제안 자바카드 파일시스템의 구현

ISO7816-4 스마트카드 표준에 따른 자바카드 파일시스템의 느린 처리 속도를 향상시키기 위한 방법으로 앞서 설계한 바와 같이 File Cache 및 Direct Access 기능을 추가하여 파일시스템의 처리 속도를 향상시키는 방법이다. 본 논문에서의 개발 환경은 그림 7과 같이 32bit 프로세서 및 토네이도2 암호전용 프로세서가 탑재된 S3FJ9SK 보드를 사용하였으며

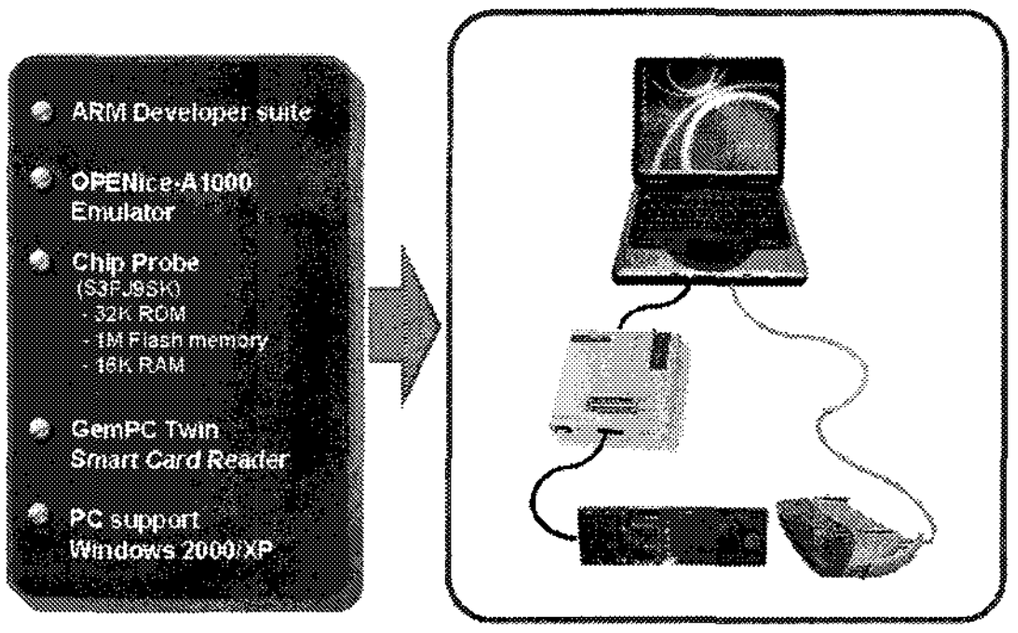

그림 7. 개발 및 테스트 환경

개발 툴로는 ARM Developer Suite V1.2를 사용하였고, 에뮬레이터로는 OpenICE-A1000을 사용하였으며 C언어와 자바를 사용하여 개발하였다.

4.1 ISO7816-4 스마트카드 표준에 따른 자바카드 파일시스템의 구현

파일을 크게 DF와 EF로 구분하였으며 아래 그림 8과 같이 DF의 경우에는 부모, 다음, 자식의 3개의 파일을 연결할 수 있도록 되어 있으며 EF의 경우에는 다음이라는 하나의 파일을 연결할 수 있는 구조이다.

이와 같은 파일 구조는 그림 9과 같이 파일이라는 최상위 클래스를 상속받아 구현되며 DF 클래스 내부에는 파일로 선언되어 있는 부모와 자식, 그리고 다음을 가지고 있다. 각각의 부모와 자식, 다음은 널 값으로 초기화 되어 있으며 새로운 파일을 생성하는 시점에 할당 연산자를 통해서 새로운 파일을 할당받음으로 해서 그림 8과 같은 파일 연결 구조를 완성할 수 있다. 아래 그림 9은 먼저 파일 이라는 최상위 클래스를 상속받아 DF와 EF를 구현하며 또한 EF에서는 트랜스페어런츠 형태와 레코드 형태로 구분하여 구현하였다. 실제 데이터를 저장하는 공간을 가진 파일은 트랜스페어런츠 파일과 싸이클릭, 선형 고정레코드, 선형 가변레코드 파일 이며 각각의 파일들은 3G USIM 환경에 맞는 파일 제어 파라메터(FCP)를 통한 접근제어 조건을 확인한다[14-16].

4.2 File Cache 구현

자바카드의 모든 명령은 명령APDU를 통해서 수행되게 된다. 이러한 명령의 분석은 자바카드 가상기계의 디스패처에서 담당하게 되고 이를 통해서 각종

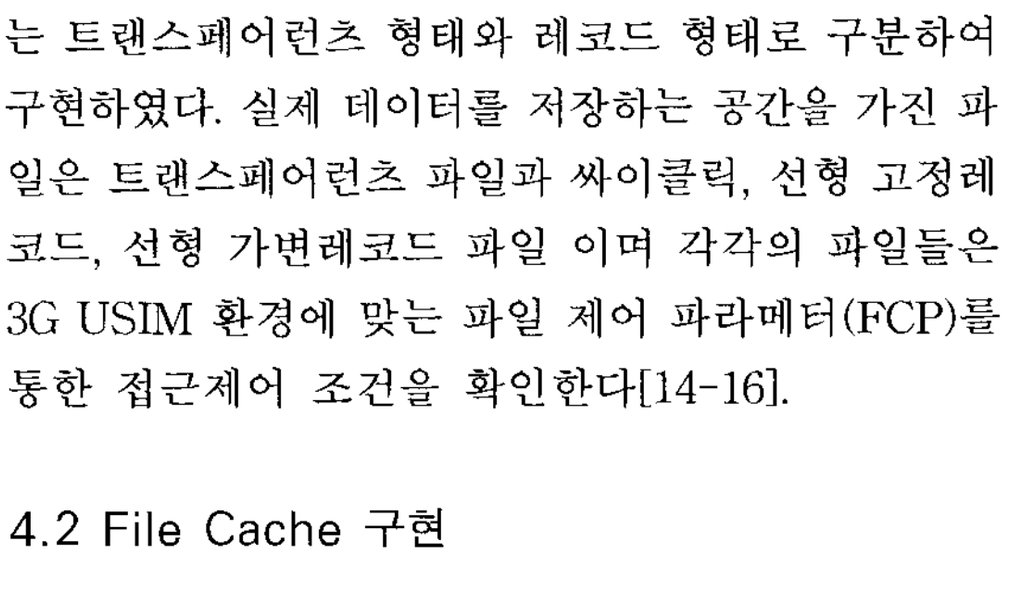

그림 8. 파일 연결 구조

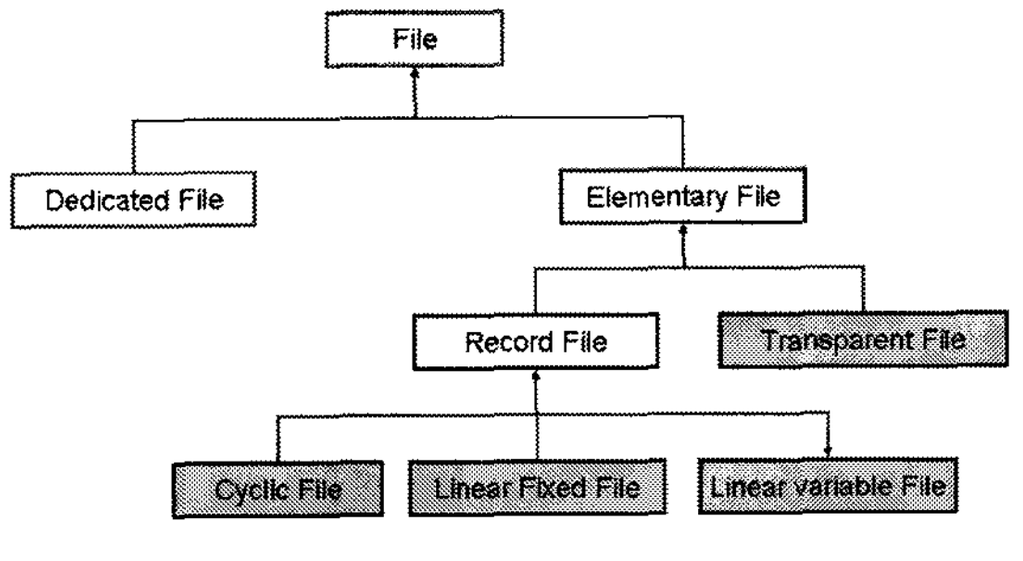

그림 9. 파일 상속 계통도

어플리케이션의 실행을 담당하게 된다. 이런 디스패처의 명령에 대한 분석 후 APDU를 통한 파일의 직접 SELECT의 경우에는 표준에 정의된 파일 SELECT 명령인 명령어 'A4'에 의해 선택 명령이 수행되게 되고 이 명령은 File Cache를 실행시키고 파일이 존재할 경우에는 응답APDU를 통해서 파일 SELECT 결과를 호스트로 송신하게 되며 File Cache내에 파일이 존재하지 않을 경우에는 자바카드 API를 통한 일반적인 파일 SELECT 명령을 그림 10과 같이 수행하게 된다. 그리고 애플릿에 의한 SELECT 명령의 경우에는 파일 검색 API를 수행하게 되며 이는 먼저 File Cache를 실행시켜 역시 존재할 경우에는 응답APDU를 통한 결과를 호스트로 그림 10과 같이 송신하게 된다.

File Cache는 적은 메모리 공간을 가지는 자바카드의 특성에 맞게 단순한 알고리즘으로 그 코드의 수를 줄일 필요가 있다. 그리하여 앞서 설계한바와 같이 큐 형태의 Cache 메모리 공간에 그림 11와 같은 알고리즘으로 구현하였다. 먼저 기본적으로 2가지 작업을 진행하게 되는데 Cache내의 파일 오브젝트 ID를 저장하는 루틴과 Cache 내의 파일 오브젝트ID

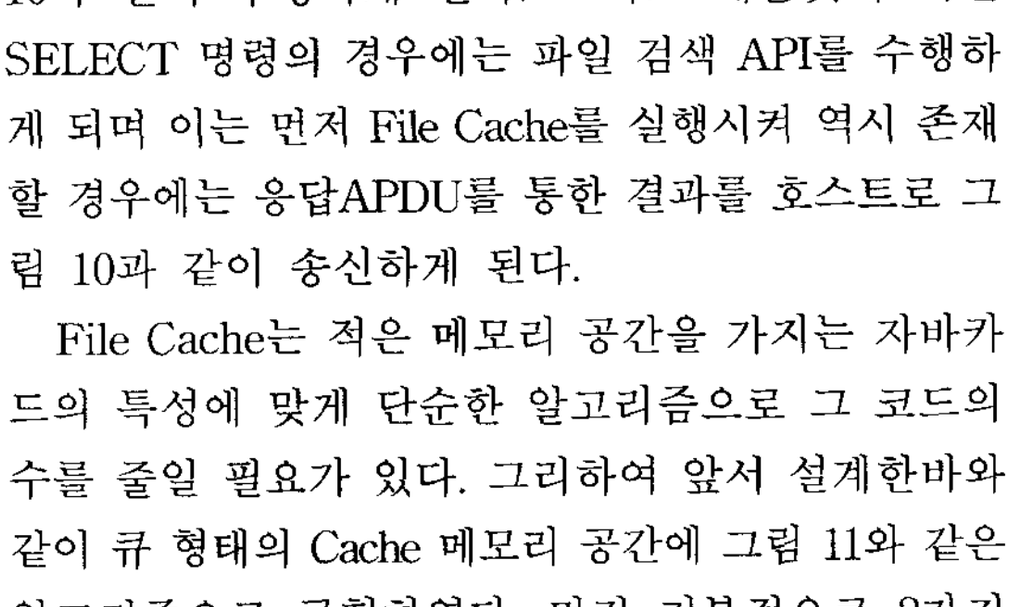

그림 10. 자바카드 File Cache의 호출 관계도

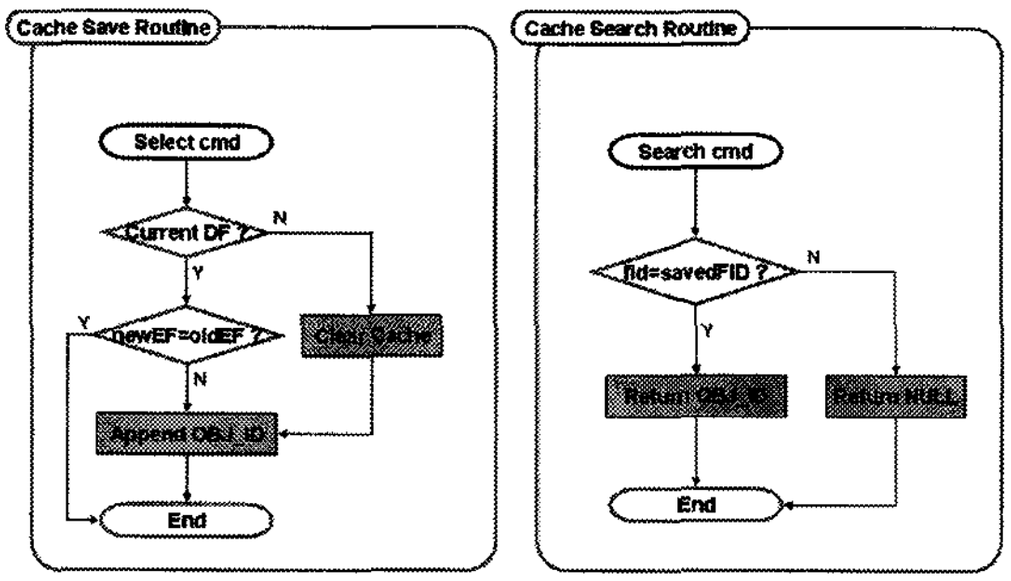

그림 11. File Cache의 구현 알고리즘

를 검색하는 루틴으로 이루어져 있다. 검색과 저장은 기본적으로 파일 식별자를 통해서 동일한 파일인지를 확인하는 절차를 가지게 되며 저장의 경우에는 현재 DF인지를 먼저 확인하게 된다. 검색의 경우에는 파일이 존재하지 않을 경우에는 기존의 자바카드 파일시스템의 일반적인 SELECT 작업을 수행하여 파일을 검색 후 현재 파일로 등록을 하는 절차를 거치도록 구현하였다.

4.3 Direct Access 구현

Direct Access의 경우에도 명령 APDU를 통한 명령어 처리와 애플릿을 통한 명령어 처리라는 두 가지 방법을 사용하는 것은 File Cache와 동일하다, 하지만 표준에 정의되지 않은 Direct Access 명령을 처리하기위해 그림 12과 같이 자바카드 파일시스템 API 중 파일 핸들러에 명령어 'A6'를 추가하여 명령 APDU를 통해서도 Direct Access 명령을 처리할 수 있도록 구현하였으며 또한 애플릿을 통한 처리에서도 Direct Access API를 자바카드 파일시스템 API에 추가 구현하여 Direct Access 함수를 실행할 수 있도록 구현하였다.

Direct Access의 경우에도 두 가지 작업을 기본적으로 수행하게 된다. 먼저 파일 오브젝트 ID를 저장하는 부분과 파일 식별자를 통한 파일 검색 부분으로 아래 그림 13와 같이 나누어진다. Direct Access의 경우에는 앞서 구현한 File Cache와 달리 파일의 생성시에 데이터를 저장 관리하게 된다. 그리고 3개의 큐 형태의 메모리를 사용함으로서 현재 찾으려는 파일이 3개의 큐 형태의 메모리에 저장되어 있는 계층 정보와 파일 식별자가 일치하는 지를 확인하여 검색된 파일을 현재 파일로 지정한다.

5. 제안 자바카드 파일시스템의 성능 분석

본 논문에서의 성능분석 환경은 펜티엄4 3GHz, 512메가 램과 윈도우 XP를 탑재한 데스크 탑에서 개발 및 성능 분석을 위한 명령을 카드로 송신하였으며, 개발 보드인 S3FJ9SK 보드에 USIM 표준에 따라 구현된 애플릿을 탑재하여 각 명령에 따른 카드의 수행시간을 측정하였다.

5.1 File Cache 및 Direct Access의 수행 시간 분석

수행 시간을 측정 및 분석하기 위해서는 2가지 방법이 존재하는데, 명령 APDU를 통한 SELECT와 애플릿을 통한 SELECT이다[7,11,13]. 테스트는 동일 DF 내에 존재하는 EF를 SELECT 하는 경우에 대한 시간을 그림 15와 같이 측정 한 것이다. 명령 APDU를 통한 파일 SELECT 및 애플릿을 통한 파일 SELECT 역시 Direct Access가 가장 빠른 것을 그림 14와 표 1을 통해 확인 할 수 있었으며 다중의 SELECT명령을 단일 명령으로 축소시킨 결과

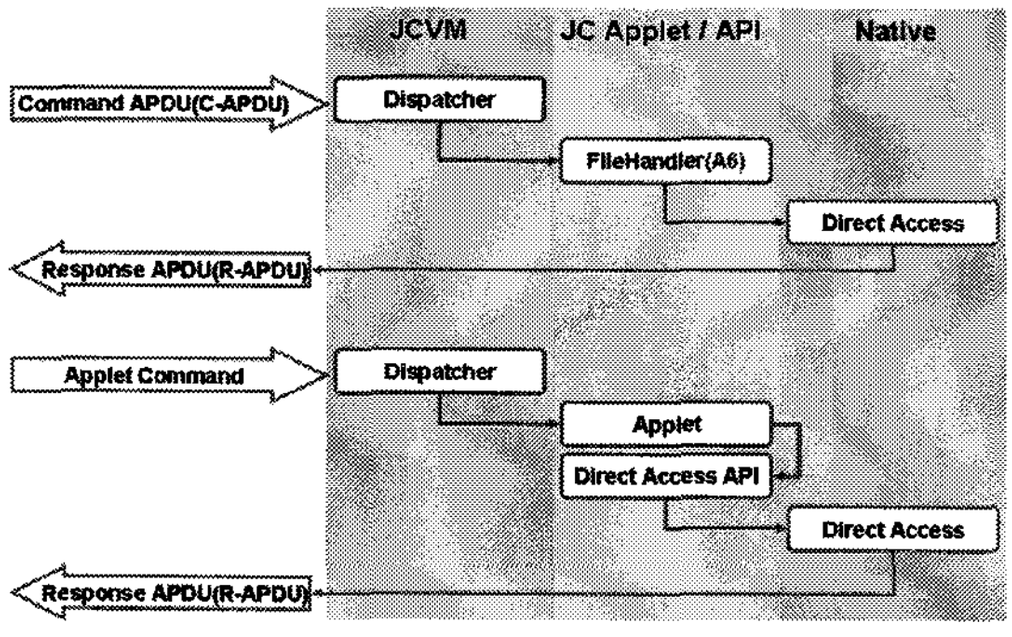

그림 12. 자바카드 Direct Access의 호출관계도

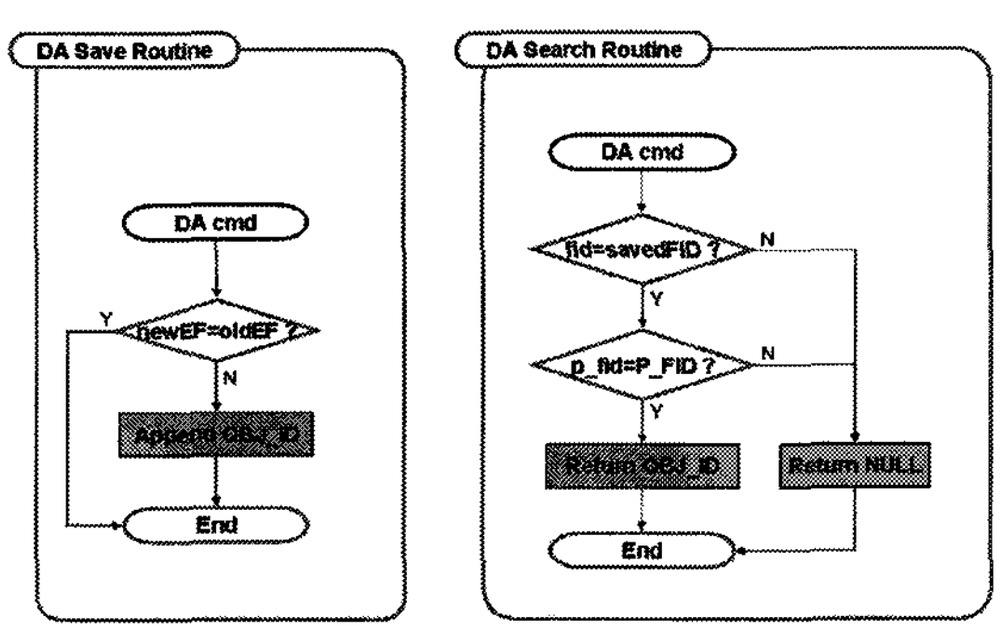

그림 13. Direct Access의 구현 알고리즘

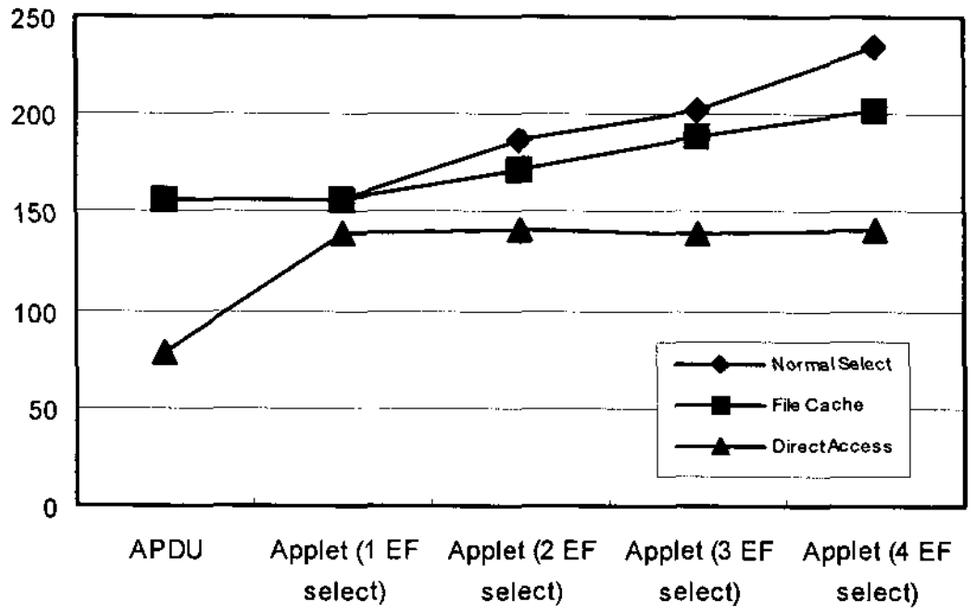

그림 14. Select 방식에 따른 성능 비교

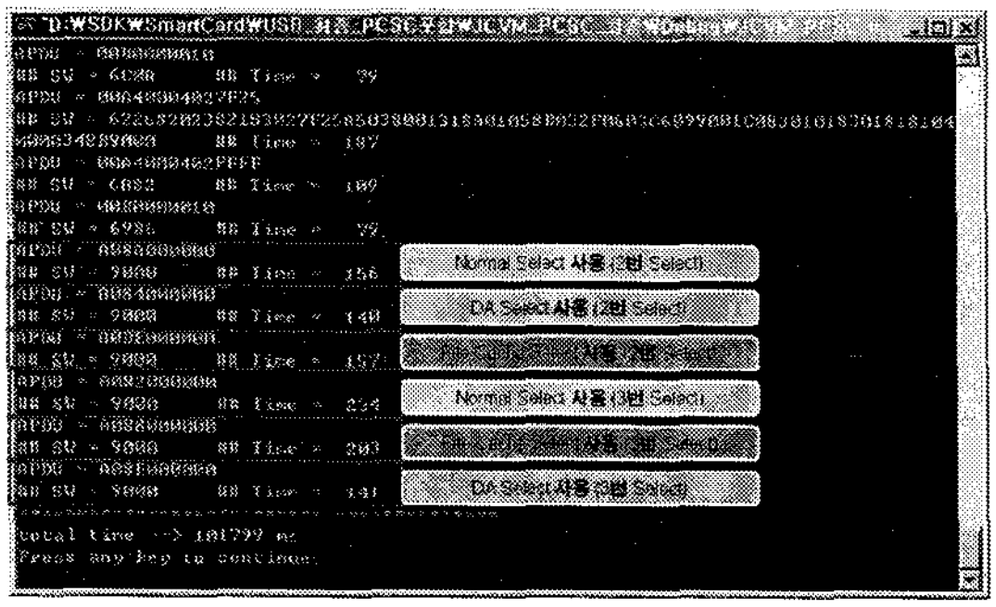

그림 15. 각 명령별 수행시간 확인

Direct Access의 수행 시간이 거의 동일한 것을 표 1과 같이 확인할 수 있었다. 그리고 4번 EF를 SELECT한 결과로 보아 여러 파일을 SELECT할 경우에는 File Cache가 일반적인 SELECT보다 속도가 빠름을 확인 할 수 있었다. File Cache의 경우에는 현재 사용하는 자료 교체 정책인 FIFO 가 아닌 LRU 방식의 교체 정책을 사용하여 Cache의 적중률을 향상시키는 방법을 고려해 볼 필요성이 있다.

5.2 코드 사용횟수 분석

각각의 명령에 따른 코드의 사용횟수는 여러 명령을 하나의 명령으로 처리 가능한 Direct Access가 가장 효율적이었으며 File Cache의 경우에는 선택하려는 파일이 모두 적중했을때는 최소한의 선택코드로 사용이 가능하였다. 이와 같은 코드의 최소 사용은 자바카드 내에서 코드의 저장 공간인 롬 영역을 더 확보 할 수 있게 해주며 특히 Direct Access의 경우에는 동일 DF가 아닌 다른 DF에 대한 SELECT의 경우에는 현저하게 그 코드 수를 줄일 수 있는 효과가 있다.

6. 결 론

개인의 중요정보를 저장하고 관리하는 자바카드 시스템에서는 그 정보를 안전하고 효율적으로 저장하고 관리하는 것이 가장 중요하다. 이러한 데이터의 저장 및 관리 기법으로 기존에는 애플릿을 통한 데이터 저장 및 관리를 진행해 왔으나, 이러한 방법으로는 동일한 데이터의 중복사용으로, 데이터 저장 공간 감소와 더불어, 데이터를 관리하기 위한 코드들이 애플릿 별로 별도로 작성되어야 하는 문제로 프로그램 영역인 롬의 메모리 공간 감소가 발생하게 된다. 이러한 문제를 해결하고 효율적이고 안전한 관리를 이루기 위해서 자바카드에서는 ISO7816-4에 정의된

표 1. Normal Select, File Cache, Direct Access의 EF Select 시간 (단위 : ms)

	C-APDU	Applet (1 EF Select)	Applet (2 EF Select)	Applet (3 EF Select)	Applet (4 EF Select)
Normal Select	156	156	187	203	234
File Cache	156	157	172	188	203
Direct Access	78	140	141	140	141

표 2. 각 명령별 사용 코드 수 비교

	C-APDU	Applet (1 EF Select)	Applet (2 EF Select)	Applet (3 EF Select)	Applet (4 EF Select)
Normal Select	4	4	5~8	6~12	7~16
File Cache	4	4	5	6	7
Direct Access	1	1	2	3	4

스마트카드 파일구조를 자바카드 API로 설계 및 구현하였다[5]. 하지만 스마트 카드 표준에 따른 파일 구조를 설계한 결과 SELECT 명령이 빈번하게 발생함으로 해서 자바 카드에서는 응용 프로그램의 처리 효율이 떨어지는 문제가 발생하였다.

본 논문에서 제안한 File Cache및 Direct Access를 사용하면 먼저 자주 사용되는 파일에 대한 빠른 SELECT가 가능해 진다. 그리고 파일 참조 제약에 따른 다중 SELECT명령의 사용역시 Direct Access를 사용하여 단 하나의 SELECT명령으로 해결됨으로 코드의 사용을 줄일 수 있고 다중의 SELECT명령을 사용함에 따르는 느린 처리 속도를 개선시킬 수 있었다.

다양한 응용 프로그램을 탑재하여 사용자로 하여금 편리함과 더불어 안전함을 제공하기 위해 그 사용이 확대되고 있는 자바카드는 본 논문에서 제안한 File Cache 및 Direct Access가 추가된 파일시스템을 사용함으로 해서 다양한 응용 서비스의 중요한 데이터들을 기존의 자바카드 보다 향상된 속도로 데이터를 안전하게 그리고 신속하게 관리 할 수 있게 될 것이다.

참 고 문 헌

[1] R&DBiZ, 스마트 카드(Smart Card)[시장동향 리포트 2006], 2006.

[2] http://www.javacardforum.org Java Card Forum Home Page

[3] Chung-Huang Yang et al., "A Smartcard-based Framework for Secure Document Exchange," *IEEE 32nd Annual International Carnahan Conference on, Security Technology*, pp. 93-96, 1998.

[4] Lodovic casset, "Formal Development of an Embedded Verifier for Java Card Byte Code," *International Conference on Dependable System and Networks*, pp. 51-56, 2002.

[5] 송영상, "자바카드 파일 시스템 API에 관한 연구," 2007년 단국대학교 대학원 박사학위 논문

[6] 양윤심, "대용량 램 기반 스마트 카드를 위한 자바 카드 시스템 구조 개선," 2007년 경남대학교 대학원 박사학위 논문

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

[8] W.Rankl, W.Effing, Smart Card Handbook, JOHNWILEY & SONS

[9] 탁승호, 스마트 카드, 성안당, 서울, pp. 85-115, 2004.

[10] Zhang Jianjie, Li Feihui, Ge yuanqing, Yue Zhenwu, and Yang Zhilian, "A Java Processor suitable for applications of smart card," *4th International Conference* on 23-25, pp. 736-739, 2001.

[11] Sun Microsystems, Inc, The Java Card™ 2.2.1. Runtime Enviroment(JCRE) *Specification*, SUN, 2003.

[12] 정임영 외, "스마트카드에서의 메모리 관리 기법," 한국정보처리학회 추계학술발표대회 논문집, 제9권, 제2호, pp. 1-4, 2002.

[13] Sun Microsystems, Inc, The Java Card™ 2.2.1 Virtual Machine Specification, SUN, 2003.

[14] TTAT.3G-31.102, IMT-2000 3GPP - Characteristics of the USIM Application(R7), pp. 6-117

[15] ETSI 102.221, UICC-Terminal interface: Physical and logical characteristics, pp. 44-92. 2005.

[16] ETSI 102.222, Administrative commands for telecommunications applications, pp. 9-30, 2005.

이 윤 석

2006년 2월 경남대학교 컴퓨터공학부 학사

2006년 3월~현재 경남대학교 컴퓨터공학과 석사과정

관심분야 : Java Technology, Java Card, HomeNetwork Security

전 하 용

2003년 경남대학교 정보통신공학부 졸업(공학사)
2005년 경남대학교 대학원 컴퓨터공학부 졸업(공학석사)
2007년 경남대학교 대학원 컴퓨터공학부 박사 수료
관심분야 : 정보보호, 자바카드, 인증 프로토콜

정 민 수

1986년 서울대학교 컴퓨터공학과 학사
1988년 한국과학기술원 전산학과 석사
1994년 학국과학기술원 전산학과 박사
1990년~현재 경남대학교 컴퓨터공학부 교수
관심분야 : Java Technology, JavaMachine, HomeNetworking