Journal of Korea Multimedia Society Vol. 14, No. 1, January 2011(pp. 103-111)

전력공급이 안정된 장치들을 위한 자바 COS에 관한 연구

정 민 수†

요 약

기존의 스마트카드에 자바 플랫폼이 탑재된 자바 카드는 리더기를 통하여 전원이 잠시 공급될 때 어플리케이션을 적재하고 실행된다. 하지만 오늘날 여러 종류의 자바 카드는 USIM 카드와 같이 이동단말기 상에 내장되어 단말기로부터 항상 전원을 공급받을 수 있다. 이와 같이 항시 전원이 공급이 될 경우, 자바 카드들의 성능개선이 가능하다. 본 논문에서는 항시 전원 공급을 받는 자바 카드의 메모리 관리 메커니즘을 보다 효과적으로 운용함으로써 성능 향상된 애플릿 설치 및 실행이 가능한 자바카드 운영체제 시스템을 제안한다.

A Study on Java COS for Devices Which Have Safe Power System

Min-Soo Jung†

ABSTRACT

Legacy Java card which adapts a Java platform loads and executes an application when electronics power is provided. However, recently the most Java cards are embedded into a mobile terminal as USIM cards, therefore the power is continually provided for the smart cards. In this case, operation of a Java card system needs to consider its operating system to be advanced in memory management, object management and transaction mechanism. In this paper, we present a high performance Java Card system which is able to have efficient installation, loading and execution of application by applying a new memory management of the smart card that has safe power system.

Key words: Java Card(자바 카드), Mobile Terminal(이동단말기), Card Operating System(카드 운영체제)

1. 서 론

초기 스마트카드는 카드 운영체제 상에 단일 응용 프로그램만을 탑재할 수 있는 카드 플랫폼을 제공했으나, 최근에는 개방형 카드 플랫폼을 구현하고 있기 때문에 카드 한 장에 다양한 응용 프로그램을 탑재할 수 있으며, 기존의 스마트카드에 비해 성능 및 기능 안전성이 강화되었다[1,2]. 따라서, 최근에는 지갑 대신 스마트카드 칩이 삽입된 휴대폰으로 일반은행 업무는 물론 신분증, 신용카드, 교통 카드, USIM 카드 등을 이용할 수 있는 시대가 열리면서 스마트카드가 유비쿼터스 통신 기술의 생활화를 이끄는 역할을 하고 있다.

스마트 카드 플랫폼은 일반적으로 마이크로 컨트롤러와 메모리, 그리고 운영체제 및 암호 알고리즘 등으로 구성된 IC칩이 장착된 전자 카드이고, 현재 사용되는 대부분의 스마트 카드들은 자바 카드 플랫폼을 채택하고 있는 추세이다. 이는 자바가 갖는 이점들인 플랫폼 독립성과 보다 안정적인 보안 기능 때문이다[1-4].

일반적인 자바 카드가 사용하는 전원 공급 방식은 카드가 카드 리더기에 연결되었을 때, 카드 리더기로부터 약 3.5V~5V의 전원을 공급 받아서 자바 카드

※ 교신저자(Corresponding Author) : 정민수, 주소 : 경남 창원시 마산합포구 월영2동 449번지 경남대학교 제1공학관 8층(631-701), 전화 : 055)249-2217, FAX : 055)248-2554, E-mail : msjung@kyungnam.ac.kr
접수일 : 2010년 9월 2일, 수정일 : 2010년 10월 15일
완료일 : 2010년 11월 17일
† 종신회원, 경남대학교 컴퓨터공학과 교수
※ 이 논문은 2009학년도 경남대학교 학술진흥연구비 지원에 의한 것임.

운영체제가 응용 프로그램을 적재하거나 사용자로부터의 요구된 서비스들을 실행한다. 하지만 본 논문에서 제안하고자 하는 새로운 자바 카드 운영체제는 자바 카드가 항상 안정된 전원을 공급 받는다는 것을 전제로 카드 운영체제의 변형을 기술한다. 이와 같이 연구를 진행한 이유는 근래 사용되는 스마트 카드의 형태들이 단순한 플라스틱 카드 형태에서 이동단말기에 탑재되는 형태로 진화를 계속하고 있기 때문이다. 따라서, 자바 카드가 이동단말기와 같은 장치들에 탑재되어 사용이 될 경우, 단말기들은 자바 카드와 통신을 하기 위한 카드 리더 역할을 하는 것이고, 또한 자바 카드에 전력을 항상 공급해 줄 수 있다는 것이다. 2009년 현재 자바 카드의 60% 이상이 독립된 카드 형태가 아닌 단말기 부착형으로 출시되고 있다. 따라서, 본 논문에서는 이동 단말기 등에서 사용될 자바 카드들이 항상 안정된 전력을 단말기로부터 공급 받을 상황에서의 자바 카드에 관한 카드 운영체제의 개선된 방법을 제안한다[4,6]. 즉, 본 논문에서 제시한 전력 공급 방식의 변화에 따른 자바 카드 운영체제의 개선을 위해 기존 자바 카드 시스템이 사용하고 있는 메모리 관리 방법과 어플리케이션 운용 및 트랜잭션 메커니즘을 개선하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 자바 카드의 플랫폼과 자바 카드 운영체제 구성요소에서 전원과 관련된 자바 카드 운영체제의 디자인 요소들을 설명한다. 3장에서는 전력 공급이 안정된 장치들을 위한 자바 카드 운영체제를 설계하고, 4장에서는 개선된 자바 카드 운영체제에 대한 구현 및 실험 결과를 제공한다. 마지막으로 5장에서는 제안된 자바 카드 운영체제에 대한 결론 및 기대효과를 기술한다.

2. 관련 연구

2.1 카드 환경의 자바 COS

자바 카드는 외부 인터페이스를 포함 1 cm^2 이하의 단일 IC 칩으로 만들어지며, 일반적인 컴퓨터와 유사한 형태의 하드웨어를 가진다. 일반적인 자바 카드의 하드웨어는 CPU와 메모리, 리더기와 카드 간의 통신모듈, 그리고 암호 프로세서 등으로 구성된다. 자바 카드에서 카드 어플리케이션과 하드웨어의 인터페이스 기능을 수행하는 자바 카드 운영체제는 하드웨어를 제어하는 각각의 모듈과 자바 어플리케이션을 실행시키는 자바 카드 가상기계이다[1,2,4].

자바 COS(Chip Operating System)로 불리기도 하는 자바 카드 운영체제는 어플리케이션 다운로드부터 설치, 적재 및 실행, 그리고 데이터 갱신, 또한 단말기로부터 입력되는 명령어를 해당 어플리케이션으로 전달하고 처리된 결과를 다시 단말기로 전송하는 등, 카드 전반에 발생되는 모든 것들을 처리한다[4,6].

2.2 상시 전원 공급시 성능을 높이기 위한 자바 COS 디자인 요소와 각 메모리의 역할

자바 COS는 기능적으로 크게 메모리 관리부분, 객체 관리부분, 통신부분의 3가지로 구성된다. 전력 공급 문제를 떠나 자바 카드에 대한 통신 방식은 ISO 7816 표준에 명시되어 있는 형태인 APDU 방식을 유지해야 한다. 또한, 현재 스마트카드로 송수신 되는 명령이나 데이터는 반드시 APDU 방식을 따르도록 되어있다. 따라서 전력 공급 방법과는 관계없이 카드에서 사용하는 ISO 7816의 APDU 통신은 계속 유지를 해야만 한다. 하지만, 통신 부분을 제외한 나머지 모듈들인 메모리 관리와 객체 관리에서는 성능 개선을 적용할 수 있다.

자바 카드가 사용하는 세 가지 메모리는 ROM, RAM, 그리고 Flash 메모리를 사용한다. ROM은 카드 사용 중에 내용이 바뀌지 않는 부분들로서 자바 카드 운영체제에 대한 코드와 자바 카드 시스템 API가 저장된다. RAM 영역은 자바 메소드 호출을 위한 실행 스택, 카드와 통신을 위한 채널 영역, 그리고 운영체제를 위한 스택으로 구성된다. 마지막으로 Flash 메모리는 자바 카드에서 저장되는 모든 어플리케이션 관련된 데이터들, 즉 패키지, 클래스, 전역변수, 그리고 객체들을 저장하고 있고, 데이터 복원을 위한 트랜잭션 영역도 포함한다.

자바 카드 상에서 프로그램 실행 중에 발생되는 모든 데이터들은 Flash 메모리에 저장 및 갱신되며, 또한 필요가 없을 경우 제거된다. 자바 카드는 자바 카드 가상기계와 자바카드 API를 기반으로 카드 어플리케이션, 즉 애플릿(Applet) 패키지들을 저장 및 관리하고 각 패키지에 대한 애플릿 객체를 통해서 사용자가 요구하는 서비스를 제공한다.

2.3 자바 카드 객체 관리 방법

자바 카드에 입력되는 애플릿에 관련된 데이터들

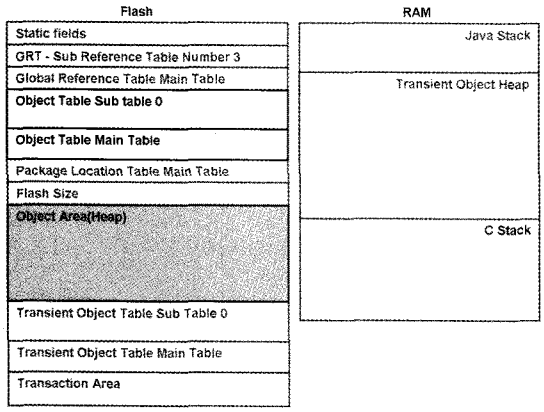


그림 1. 자바 카드에서 애플릿과 객체들을 관리하기 위한 Flash 메모리의 내부 형태

은 모두 패키지 단위로 처리되고 테이블에 의해서 관리된다. 그림 1과 같이 모든 애플릿들은 자바 카드에 패키지 형태로 입력이 되고 클래스들과 메소드들에 대한 주소값들이 각 테이블에 기록된다. 또한 자바 카드는 생성된 객체에 대한 위치 값들을 관리 테이블에 입력하여 사용한다.

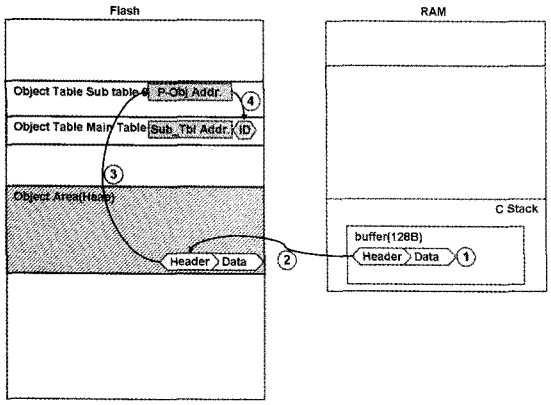
자바 카드는 일반적인 컴퓨터용 표준 자바의 형태와 달리 두 가지 형태인 영구(Persistent) 객체와 임시(Transient) 객체를 분리하여 사용한다. 이는 카드에서 사용되는 데이터들 중에서, 카드의 세션이 종료되거나 전력 공급이 끊어져도 데이터가 보존되어야 할 경우와 한 세션 중간에 잠깐 사용되고 사라져야 할 데이터를 구분하기 때문이다. 전자가 영구 객체이고 후자가 임시 객체이다. 영구 객체는 한번 기록되면 지워지지 않는 Flash 메모리에 저장되고, 임시 객체는 휘발성 메모리인 RAM 영역에 저장된다. 그림 2는 두 가지 형태의 객체들이 생성되고 각각 관리 테이블에 기록되는 과정을 보여주고 있다.

2.5 전력 공급 및 차단에 대비한 자바 COS 디자인 요소들

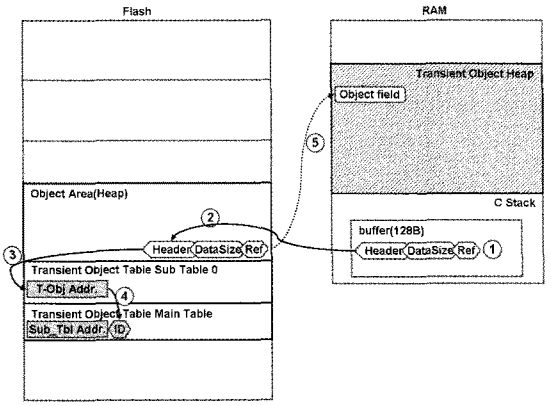
안정적인 전력 공급을 가정하면, 빠른 서비스 처리가 요구되는 작업을 기존의 자바 카드가 Flash를 이용하는 방식을 벗어나 사용되는 애플릿을 자바 카드의 RAM 영역에 상주시켜 훨씬 효율적으로 처리할 수 있다. 하지만 많은 애플릿들이 한꺼번에 RAM 상에 상주하기에는 여전히 운영상에 제약을 받는다. 또한, 전력 공급 중단 시에 RAM 상에서 동작하던 애플릿 및 각 객체들의 변경 값을 그대로 유지하기가 힘들다. 따라서 안정된 전력 공급될 때와 전력 차단이 발생할 경우에 자바 카드 운영체제에서 처리해야 할 요소들은 다음과 같다.

(1) 항상 안정된 전력이 공급이 될 경우의 고려 사항
- RAM 메모리에 상주시켜야 할 데이터의 범위
- RAM 메모리에 상주시켜야 할 데이터와 애플릿을 관리하는 테이블들과의 연관 관계

(2) 갑자기 전력이 차단되었을 경우의 고려 사항
- 전력 차단 전까지의 애플릿의 실행 결과
- 전력 차단 전까지의 Flash 상에 기록되었던 데이터의 복원



a. 영구 객체의 생성 단계와 생성된 객체에 대한 관리 테이블에 객체 정보 추가 과정



b. 임시 객체의 생성 단계와 생성된 객체에 대한 관리 테이블에 객체 정보 추가 과정

그림 2. 두 가지 형태의 객체의 생성과정

- 전력 인가 후 최종 사용되었던 애플릿의 정보

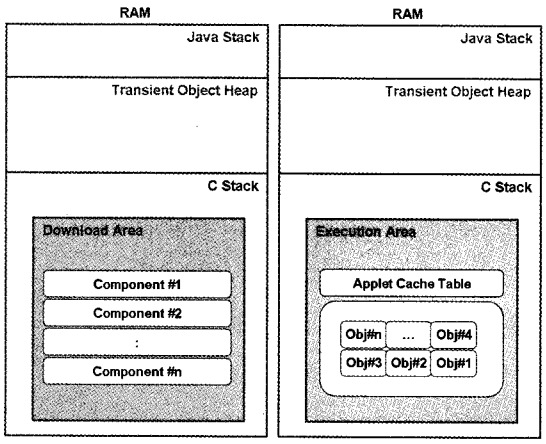
즉, 위에 나열된 고려 사항들은 자바 카드 운영체제가 얼마만큼의 데이터들을 RAM에 상주시키면서 운용하다가 만약 전력 차단이 발생하였을 경우 어떻게 대처를 하는 가를 말한다. 이는 카드 운영체제의 테이블 관리 모듈, 객체 관리 모듈, 그리고 트랜잭션 모듈에 관한 새로운 방법을 본 논문의 자바 카드 운영체제에 적용해야 한다는 것을 말한다.

3. 전력공급이 안정된 장치들을 위한 자바 COS 설계

안정된 전력이 공급이 된다면 자바 카드 운영체제의 설계 방향은 쓰기 속도가 RAM에 비해 1000배 이상 느린 Flash 메모리 대신 RAM 메모리를 최대한 활용하는 것이다. 자바 카드는 애플릿 설치와 실행에 있었어도 각 객체들이 저장된 주소 값들을 고유 테이블에서 처리한다. 따라서, RAM 상에 애플릿을 상주시켜 실행할 경우에도 항상 객체 관리 테이블과의 유기적인 연동이 고려되어야 한다.

3.1 자바 카드의 메모리 구조 설계

본 논문에서 자바 카드의 두 가지 주요 용도인 애플릿 설치와 애플릿 실행과정을 고려하여 새로운 RAM의 구조를 제안한다. 그림 3과 같이 RAM 메모리 내의 C스택 영역의 여유 공간을 자바카드 실행



(a) 애플릿 다운로드 및 설치 (b) 애플릿 실행

그림 3. 개선된 자바 카드의 RAM 구조

영역(Execution Area)으로 구분하여 애플릿이 다운로드 될 때와 애플릿이 실행될 때의 두 가지 형태로 동적으로 변형하여 사용할 수 있도록 하였다.

자바 카드에서 애플릿 다운로드와 애플릿 실행은 카드 운영체제에서 단말기로부터 입력되는 명령어에 의해 구분된다. 따라서 자바 카드 운영체제는 입력되는 APDU의 명령에 의거해 현재 동작하는 RAM 영역의 구조를 동적으로 변경할 수 있도록 설계가 되어야 한다. 그리고, 갑작스런 전력 차단에 대비하여 자바 카드에서 실행되고 있는 애플릿에 대한 데이터 보존을 위해 트랜잭션 영역 또한 RAM에 추가하여 관리한다.

3.2 RAM 상에 상주시킬 애플릿과 객체들을 위한 테이블 설계

자바 카드는 애플릿을 실행할 때 애플릿 ID 값을 운영체제에 미리 알려주고, 운영체제가 해당 애플릿의 위치를 파악하고, 필요한 경우 객체를 생성하고 메소드를 실행시킨다. 자바 카드가 갖는 RAM 용량은 제한적이기 때문에 카드 내부에 설치된 모든 애플릿을 한꺼번에 RAM에 넣고 사용하기는 불가능하다. 따라서, 애플릿 ID를 활용하여 선택된 애플릿이 사용하는 객체들을 RAM 상에 미리 적재하여 사용할 수 있게 한다. 자바 카드에 설치된 애플릿들은 하나의 패키지 형태로 Flash 메모리에 저장되고 각 애플릿 ID는 애플릿 테이블에 저장되며, 또한 자바 카드에는 추가된 패키지를 패키지 테이블에 패키지 ID와 애플릿 ID가 같이 기록되고 Flash 메모리상의 실제 주소 또한 저장된다.

이러한 자바 카드의 패키지 관리 테이블의 구조적 상관관계를 이용하여, 애플릿 선택 APUD 명령이 발생하면 한 개의 패키지를 구성하는 모든 애플릿의 위치와 크기를 패키지 관리 테이블로부터 쉽게 파악할 수 있다. 그리고 각 애플릿이 미리 생성한 객체들의 시작 주소와 크기를 구할 수 있어, 이를 그림 3(b)와 같이 RAM 상의 실행 영역에 적재하여 사용하도록 자바 카드 운영체제를 개선한다.

또한 애플릿 ID를 통해 패키지 정보를 파악할 수 있기 때문에, 애플릿 선택 명령을 통해서 하나의 패키지 내에 있는 모든 애플릿의 실제 주소와 크기 정보를 가져와 각 애플릿이 사용하는 객체들을 보다 빨리 접근할 수 있도록 그림 3(b)의 애플릿 캐시 테이

블(Applet cache table)을 구성한다. 애플릿 캐시 테이블의 구성은 그림 4와 같고 16개의 애플릿들에 대한 정보를 저장하도록 구성한다. 이는 자바 카드 언어적 구성상 한 개의 패키지에 담을 수 있는 최대 애플릿의 개수가 16개로 제한되어 있기 때문이다.

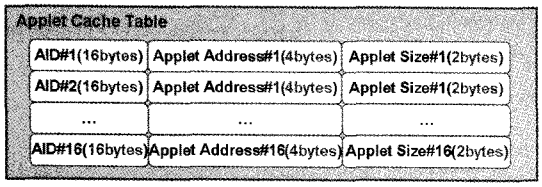


그림 4. 애플릿 캐시 테이블의 구성

3.3 기존 자바 카드 테이블 관리 방법의 개선

본 논문에서는 자바 카드가 안정된 전원이 공급이 되는 장치들에 탑재되어 사용된다는 가정 하에서 카드 운영체제에 대한 개선점들을 제시한다. 따라서 그림 2와 같이 기존 자바 카드가 사용하는 두 가지의 객체 테이블인 영구 객체 테이블과 임시 객체 테이블을 그림 5와 같이 하나의 객체 테이블로 통합하여 사용한다. 그 이유는 본 논문에서는 두 가지 객체가 모두 RAM 상에서 운영이 되기 때문이다.

기존 자바 카드의 실행과의 호환성을 위해서 객체들이 생성이 될 경우, 객체의 타입을 구분할 수 있는 코드를 추가하여 생성된 ID 값과 함께 객체 테이블에 저장한다.

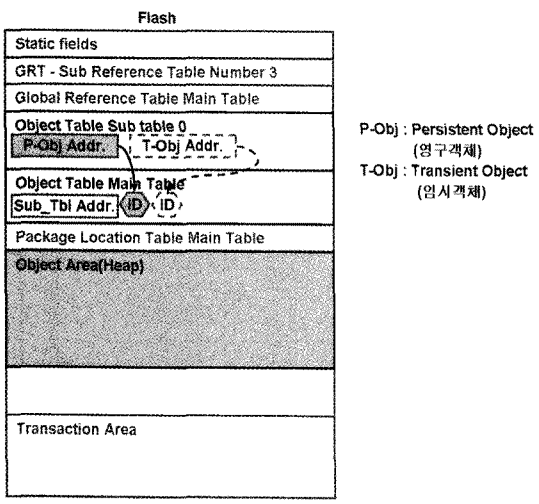


그림 5. 그림 2의 영구 객체와 임시 객체를 통합하여 관리하는 객체 테이블

3.4 전력 차단시의 트랜잭션 메커니즘

자바 카드에서 가장 중요한 부분은 데이터 무결성이다. RAM에 적재된 애플릿에 대한 객체들이 실행시에 갑자기 전력이 차단이 되는 경우, RAM 내부에 있는 모든 데이터들은 모두 소멸된다. 실행되던 모든 데이터가 사라지게 되면 자바 카드는 개인 정보를 받는 보안카드로서의 기능과 금융 관련 카드로의 기능을 상실하게 된다. 이러한 데이터 손실과 오류 등으로부터 데이터를 복원시키기 위한 방안이 트랜잭션이다. 개선된 자바카드 운영체제에서는 항시 안정적인 전력 공급이 전제지만, 만일 전력이 차단되고 다시 인가되었을 경우를 대비하여야 한다.

그러므로, 본 논문에서 제안하는 트랜잭션 메커니즘은 이동 단말기의 하드웨어적인 특성을 활용한 트랜잭션 방법을 사용한다. 오늘날의 이동 단말기는 전력이 소진되거나 차단될 경우를 대비하여, 잔여 전력을 항시 남겨 놓도록 하드웨어적으로 설계가 되어 있고 이를 단말기 시스템 프로그램에서 현재까지의 단말기 진행 사항을 메모리로 단계적으로 저장하도록 구현되어 있다. 따라서, 단말기에서 전력이 차단될 때, 남아 있는 전력을 활용하여 현재까지 자바 카드의 RAM 상에서 실행된 객체 데이터의 갱신 데이터와 카드에서 발생한 트랜잭션의 결과들을 자바 카드 내부의 Flash 메모리에 백업을 할 수 있다. 하지만, 잉여 전력이 남지 않은 경우, 즉 이동 단말기에서 강제적으로 전원이 차단이 될 경우에는 데이터를 보존시킬 트랜잭션 메커니즘을 구성하기가 어렵다.

이에 자바 카드 운영체제는 전력 차단 후 재인가 발생시, 마지막으로 실행했던 객체와 그 객체 내부의 데이터 값까지 RAM 상에 다시 적재시키도록 그림 6과 같이 RAM 기반의 트랜잭션 버퍼를 운영할 수 있도록 한다.

4. 개선된 자바 COS 구현

4.1 전력 공급이 안정된 장치들을 위한 자바 COS 구현

항시 전력을 공급을 받는 개선된 형태의 자바 카드 운영체제 구현은 다음의 상황을 고려하여 구현한다.

- 애플릿 설치와 애플릿 실행을 구분한 RAM 운영 방식
- 전원 차단시 개선된 트랜잭션 방법과 전원 공급 후의 RAM 메모리의 구성 방식

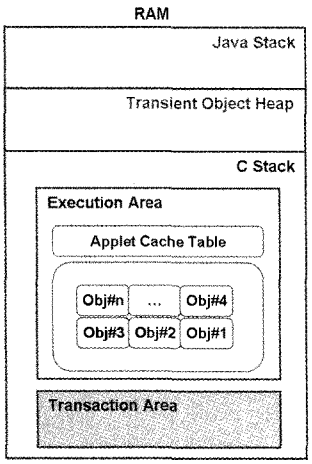


그림 6. RAM 메모리에 구성한 트랜잭션 영역

먼저, 자바 카드에 애플릿 설치와 애플릿 실행은 구분된 APUD 명령 값을 갖고 있다. 따라서 자바 카드 운영체제는 수신되는 명령을 분석하여 RAM 메모리를 그림 5와 같이 다운로드 영역으로 구성할지 실행 영역으로 구성할지를 결정하면 된다. 따라서 본 논문에서의 제안하는 개선된 메모리 운영 방식은 그림 7과 같이 구현된다.

본 논문의 자바 카드 운영체제가 사용한 트랜잭션 메커니즘은 애플릿이 실행이 될 때 RAM 메모리에 트랜잭션 영역을 그림 6과 같이 구축하고 자바 카드 객체의 인스턴스 데이터나 참조 데이터 값이 변경될 경우 변경되는 곳의 주소와 이전 값을 기록한다. 또한 전원 차단이 발생했을 경우, 단말기로부터 카드 선택 해제 명령을 전달받은 후, 실행 영역내의 객체 캐시 테이블의 데이터를 Flash 메모리 상의 실제 주소에 맞게 계산하여 기록하고 애플릿 캐시 테이블의 값을 Flash의 트랜잭션 버퍼에 기록한다. 전원 공급을 다시 받게 되면, 트랜잭션 버퍼에 기록된 애플릿 캐시 테이블을 실행 영역에 복원시키고 애플릿에 대한 객체들을 객체 캐시 테이블에 복원시킨다.

5. 실험 결과

본 논문에서 제시한 전력공급이 안정된 장치들을 위한 자바 카드 운영체제에 대한 비교분석을 위해서 삼성 S3FS9QB 개발 보드를 이용하여 실험하였다. 개발 보드의 사양은 ARM 7 Core 32 bits MCU를 사용하고 저장 매체로 RAM 10KB, 두 개의 256KB와 64KB의 Flash 메모리를 사용한다.

또한 실험에서 사용되는 애플릿은 썬마이크로시스템즈에서 제공하는 7개의 공인된 테스트용 애플릿들을 활용하였다[1,2]. 실험 구성 방식은 기존의 미국 썬사의 자바 카드 운영체제에서 애플릿 생성 및 실행 속도를 측정하고 본 논문에서 제안한 자바 운영체제를 구현하여 두 가지 Java COS의 성능을 비교하였다.

표 1에서 보는 바와 같이 애플릿 생성 시간이 평균 절반으로 줄어드는 것을 확인할 수 있다. 또한, Wallet, JavaLoyalty, ChannelDemo 애플릿들의 생성 효율성이 다른 테스트 애플릿 보다 떨어지는 이유

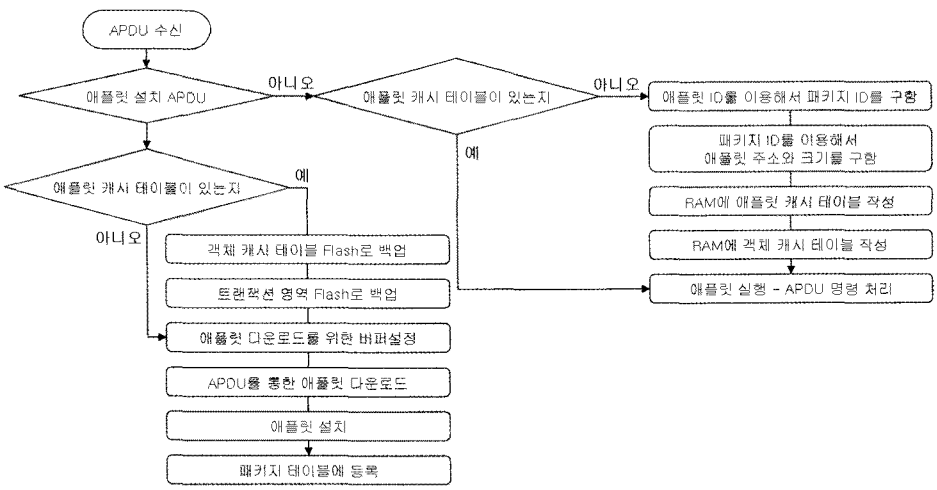


그림 7. 애플릿 설치 및 실행에 관한 RAM 메모리 운영 방식

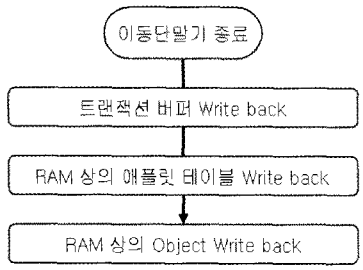


그림 8. 전력 차단 시 트랜잭션 과정

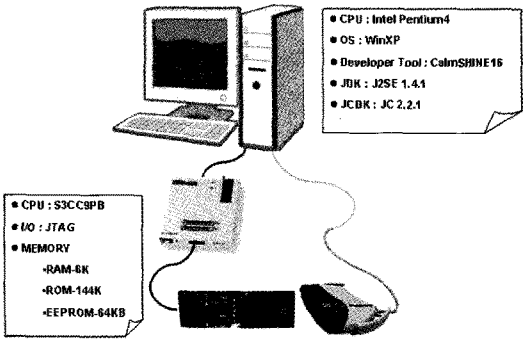


그림 9. 실험을 위한 전체 시스템 구성도

는 자바 카드에 설치되는 각 애플릿들이 정의한 메소드의 갯수와 각 메소드의 크기, 상수와 전역 변수의 갯수가 적기 때문이다.

표 2에서 제공된 결과에서는 애플릿 실행 시간이 30% 정도 줄어드는 것을 확인할 수 있다. 또한 Wallet, ChannelDemo 애플릿에서 사용되는 자바 카드 객체의 갯수가 적을수록 그 효율성이 떨어지지는 것을 알 수 있다. 하지만, Demo1이나 Demo3과 같이 사용되는 애플릿 내에서 사용되는 갯수가 많을수록 효율성이 좋아지는 것을 확인할 수 있다.

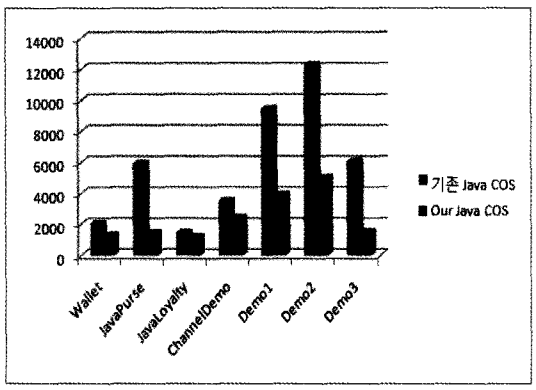
위의 실험 결과에서 나타나듯이 애플릿 생성 및 메소드 실행 시간을 측정한 결과, 본 논문에서 제시한 자바 카드 운영체제의 방식이 효율적이라는 것을 알 수 있고, 그 근본적인 이유는 애플릿 생성과 실행을 가능한 RAM 메모리에 내에서 수행을 시켰기 때문이다. 따라서, 향후 자바 카드가 항상 안정된 전력이 공급되는 장치들 내에 탑재되고, 자바 카드가 대용량의 메모리를 사용한다면 본 논문에서 제안한 방법이 자바 카드 운영에 매우 효과적임을 알 수 있다.

표 1. 애플릿 생성 시간 비교 (단위 : ms)

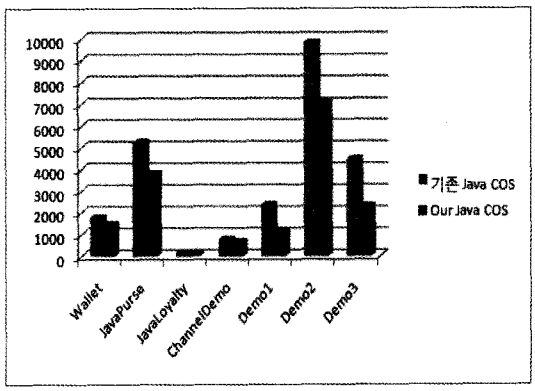
Applet	Size	기존 Java COS	Our Java COS	Reduced Rate
Wallet	3KB	2062	1297	37%
JavaPurse	12KB	5953	1453	75%
JavaLoyalty	3KB	1469	1172	20%
ChannelDemo	4KB	3516	2422	31%
Demo1	16KB	9484	3922	58%
Demo2	18KB	12281	5023	59%
Demo3	12KB	6063	1500	75%
Average				51%

표 2. 애플릿 선택 후 메소드 실행 시간 비교 (단위 : ms)

Applet	Size	기존 Java COS	Our Java COS	Reduced Rate
Wallet	3KB	1766	1469	16%
JavaPurse	12KB	5249	3812	27%
JavaLoyalty	3KB	156	109	30%
ChannelDemo	4KB	765	657	14%
Demo1	16KB	2359	1097	53%
Demo2	18KB	9827	7077	27%
Demo3	12KB	4453	2312	48%
Average				31%



(a) 애플릿 생성 시간



(b) 메소드 처리 시간

그림 10. 애플릿 생성 및 메소드 처리 시간에 대한 비교 (단위 : ms)

6. 결 론

본 논문에서는 USIM과 같이 이동 단말기에 탑재되어 단말기로부터 항시 안정된 전력을 자바 카드가 공급을 받을 수 있다는 것을 전제로 하였고, 따라서 현재 자바 카드에서 운영되고 있는 방식이 미래지향적으로 바뀌어져야 한다는 취지에서 개선된 자바 카드 운영체제의 방식을 제안하였다. 또한, 카드의 하드웨어적인 요소 중에서 RAM 메모리가 점점 대용량으로 바뀌어 감에 따라 이를 최대한 활용할 수 있는 방안을 제시하였다.

본 논문의 주된 아이디어는 스마트카드가 일반 개인용 컴퓨터 형태처럼 점진적으로 진화하고 있다는 것을 착안하여 다양한 형태의 애플릿 설치는 비휘발성 메모리인 Flash에 저장하고 실행 시 관련된 객체 데이터들을 RAM 메모리에 항상 배치시키고 관리함으로써 성능 향상의 효과를 얻고자 하는 것이고, 실험을 통하여 그 성능을 입증하였다.

참 고 문 헌

[1] Sun Microsystems, Inc., "Virtual Machine Specification, Java CardTM Platform, Version 2.2.1," October, 2003.

[2] Sun Microsystems, Inc., "Runtime Environment Specification, Java CardTM Platform, Version 2.2.1," October, 2003.

[3] U. Hansmann, "Smart Card Application Development Using Java," *Springer*, August 26, 2002.

[4] Z. Chen, "Java Card Technology for Smart Cards: Architecture and Programmer's guide (Java Series)," *Addison-Wesley*, June 16, 2000.

[5] D. Davis and D. Balaban, "Wake Up and Smell The Java!," *Card Technology Magazine*, February 2002.

[6] G. Grimaud and J.J. Vandewalle, "Introducing Research Issues for Next Generation Java-based Smart Card Platforms," *the Proceeding of Smart Objects Conference, Grenoble, France*, May 15-17, pp. 138-141, 2003.

[7] W. Rankl, and W. Effing, "Smart Card Handbook, the third Edition," *John Wiley & Sons*, January 16, 2004.

[8] M. Baentsch, P. Buhler, P., T. Eirich, F. Hring, and M. Oestreicher, "Java Card - From Hype to Reality," *Concurrency, IEEE*, Volume: 7, Issue: 4, October 6, pp. 36-43, 2002.

[9] B. Venners, "Inside the JAVA Virtual Machine Second Edition," *McGraw-Hill Companies*, January 6, 2000.

[10] Samsung Electronics, "User's Manual S3FS9QB 32-Bit CMOS Microcontroller for Smart Card Revision1," September, 2005.

[11] B. Cahoon and K. McKinley, "Data Flow Analysis for Software Prefetching Linked Data Structures in Java", *the Proceedings of*

International Conference on Parallel Architectures and Compilation Techniques, ACM, Barcelona, Spain, 2001.

[12] G. Selimis, A. Fournaris, G. Kostopoulos and O. Koufopavlou, "Software and Hardware Issues in Smart Card Technology," *IEEE Communications surveys & Tutorials*, Vol. 11, No.3, pp. 143-152, 2009.

[13] K. Mayes and K. Markantonakis, "Smart Cards, Tokens, Security and Applications," *Springer*, January 2008.

[14] M. Hendry, "Multi-Application Smart Cards: Technology and Applications," *Cambridge University Press*, July 2007.

정 민 수

1982년 3월~1986년 2월 서울대학교 컴퓨터공학과 학사
1986년 3월~1988년 3월 한국과학기술원 전산학과 석사
1988년 3월~1994년 2월 한국과학기술원 전산학과 박사

1990년 9월~현재 경남대학교 컴퓨터공학과 교수
관심분야 : 자바기계, 임베디드시스템, 컴파일러