Journal of Korea Multimedia Society Vol. 10, No. 8, August 2007(pp. 1029-1038)

전원 공급이 지속적인 대용량 스마트 카드를 위한 JCVM 시스템 구조 개선

이동욱[†], 황철준[††], 양윤심[†††], 정민수[††††]

요 약

기존의 자바카드 플랫폼이 탑재된 스마트 카드는 전원이 잠시 공급될 때 어플리케이션을 설치하고 실행한다. 또한 예기치 않은 전원 차단에 대비하여 어플리케이션의 실행 상태와 실행 시 변경되는 모든 데이터를 비휘발성 메모리(EEPROM/Flash)의 힙(Heap)영역에 저장하고 갱신한다. 이 같은 무절제한 EEPROM의 데이터 갱신은 스마트 카드의 생명을 단축시키는 중요한 원인이 된다. 이는 항상 전원이 공급되는 환경으로 발전할 것임을 고려하지 않는 상태에서 스마트 카드를 개발했고, 또한 그 구조를 계속 유지하고 있기 때문이다. 본 논문에서는 어플리케이션 저장 메커니즘과 메모리 구조를 개선하여, EEPROM은 어플리케이션 다운로드용, RAM은 애플릿 실행용으로 사용하는 일반적인 컴퓨터 시스템 구조로 개선하여 전원이 항상 공급되는 환경에서 운용되는 고성능 자바카드 시스템을 개발한다. 제안된 기법이 적용된 자바카드 시스템을 통해 애플릿의 생성 속도가 58%, 메소드 실행속도가 33% 정도 빨라진다는 것을 알 수 있었다.

An Improvement of the JCVM System Architecture for Large Scale Smart Card having Seamless Power Supply

Dong-wook Lee[†], Chul-joon Hwang[††], Yoon-sim Yang[†††], Min-soo Jung[††††]

ABSTRACT

A smart card based on the existing Java card platform executes and installs an application only when the power is supplied for a minute. And preparing for unexpected power outrage, the execution state of an application and all the data that are modified during execution are saved in the heap. This kind of frequent data update of an EEPROM data is a main cause of reducing the life-cycle of a smart card. This is because the smart card has been developed not considering the current situation that the power is always supplied, and by this time it has continuously kept its old architecture. This paper explains the high performance Java card system free power restriction. The system improves not only application saving mechanism, but memory architecture. In special, we deploy RAM for running an applet, as well as EEPROM for downloading an application. Through proposed mechanism, we can find out performance evaluation that the creation speed of an applet and the execution speed of a method increase up to 58% and 33% respectively.

Key words: Smart Card(스마트카드), Java Card(자바카드), JCVM(자바카드 가상 기계), Java Card Applet(자바카드 애플릿)

1. 서 론

최근 지갑 대신 스마트 카드 칩이 삽입된 휴대폰으로 일반은행 업무는 물론 신분증, 신용카드, 교통카드 등을 이용할 수 있는 시대가 열리면서 스마트 카드가 유비쿼터스 통신 기술의 생활화를 이끄는 역할을 하고 있다.

초기 스마트 카드는 카드 운영체제 상에 단일 응용프로그램만을 탑재할 수 있는 카드 플랫폼을 제공했으나, 최근에는 개방형 카드 플랫폼을 구현하고 있

기 때문에 카드 한 장에 다양한 응용프로그램을 탑재할 수 있으며, 기존의 스마트 카드에 비해 성능 및 기능, 안전성이 강화되었다[1,2].

자바 카드 플랫폼이 탑재된 스마트 카드는 마이크로컨트롤러와 운영체제, 메모리, 암호 알고리즘 등으로 구성된 IC칩이 장착된 전자 카드이다. 일반적인 자바 카드는 응용프로그램인 애플릿(Applet)을 실행하는 동안 구조적인 이유로 실행속도 측면에서 다음과 같은 문제점을 가지고 있다.

첫째, 스마트 카드는 자체적으로 전원을 공급할 수 없는 구조로 외부로부터 전원을 잠시 공급받는 동안 애플릿을 실행시킨다. 이런 구조로 자바 카드는 일시적인 전원 공급 중 예기치 않은 전원차단이 발생할 경우를 대비해야 한다.

둘째, 스마트 카드는 프로그램의 실행 상태를 EEPROM에 저장하고 갱신한다. 그 이유는 일시적인 전원 공급 문제와 비용이 높은 RAM 사용을 최소화하기 위함이다.

자바 카드가 (U)SIM 표준으로 채택됨으로 인해 스마트 카드 시장에서 점유율이 증가하고 있고 응용시장이 확대되면서 휴대폰이나 가전기기 같이 항상 전원을 공급받는 환경으로 변하고 있다. 또한, 하드웨어 기술의 발전으로 RAM의 가격은 낮아지고, 전체적인 메모리 용량도 커지고 있다[표 1][3-5].

이 문제점들의 해결책으로 본 논문에서는 어플리케이션의 다운로드 및 설치는 비휘발성 메모리 영역을 사용하고, 실행은 휘발성 메모리 영역을 활용하는 구조로 개선하여 전원이 항상 공급되는 환경에서 운용되는 고성능 자바카드 시스템을 개발한다. 자바 카드 플랫폼의 애플릿 인스턴스 생성 및 데이터 저장 방식을 개선하고, 애플릿 실행을 위해 자바 카드의 RAM 구조를 개선하여 향상된 자바카드 시스템 성능을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로써 자바카드 시스템의 개요와 자바카드 애플릿의 실행 환경에 대하여 알아본 후, 3장에서는 본 논문에서 제시하는 개선된 자바카드 시스템 구조를 설계하고, 4장에서는 개선된 자바카드 시스템 구조를 구현하며, 5장에서는 ARM7 Core 32bits MPU[6,7]가 장착된 환경에서 제안된 자바카드 시스템의 성능평가 결과를 기술한다. 마지막으로 6장에서는 결론 및 기대효과를 제시한다.

표 1. 스마트 카드의 하드웨어 사양 비교

부품목록	기존 사양	현재 사양
CPU	8 bit	8, 16, 32 bit
RAM	1.2 Kbyte	4~10 Kbyte
ROM	47 Kbyte	128~384 Kbyte
EEPROM	16 Kbyte	64~256 Kbyte

2. 관련연구

2.1 자바카드 가상 기계(JCVM)의 개요

자바 카드 가상 기계는 일반적인 자바 가상 기계와 달리 2개의 분리된 Off-Card VM과 On-Card VM으로 구성되는데, 이는 제한된 메모리 자원을 갖는 카드의 특성을 고려하여 수행시간이 오래 걸리고, 메모리 자원을 많이 사용하는 부분은 단말기 상에서 처리하도록 하고, 카드 상에서는 메모리 자원을 적게 사용하는 부분들을 처리하게 한 것이다. 자바 카드 가상 기계의 구조는 그림 1과 같다.

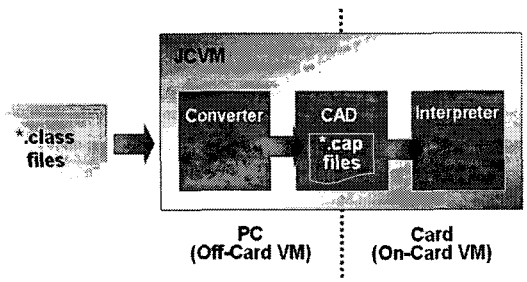

그림 1. 자바카드 가상 기계 구조

※ 교신저자(Corresponding Author) : 이동욱, 주소 : 마산시 월영 2동 449번지(631-701), 전화 : 055)249-2217, FAX : 055)248-2554, E-mail : smalldo2@hanmail.net
접수일 : 2007년 3월 19일, 완료일 : 2007년 7월 16일
† 준회원, 경남대학교 컴퓨터공학과
†† 준회원, 경남대학교 컴퓨터공학과
(E-mail : fss777@kyungnam.ac.kr)
††† 준회원, (주)이-머니금융
(E-mail : ysyang@hanmail.net)
†††† 종신회원. 경남대학교 컴퓨터공학과
(E-mail : msjung@kyungnam.ac.kr)
※ 본 연구는 2007년 경남대학교 연구비 지원으로 수행되었음

Off-Card VM은 클래스 파일 로딩, 링킹(linking), 변환(conversion), 바이트 코드 최적화(optimization), 클래스 파일 검사 (verification), 객체 및 배열 초기화 등의 기능을 담당하며, On-Card VM은 바이트코드 실행, 객체 생성 및 메모리 제어 등의 기능을 담당하는 인터프리터를 의미한다. Off-Card VM인 컨버터(Converter)는 자바카드 상에 응용 프로그램을 사후 발행(post-issuance)할 경우 사용되는 CAP 파일과 export 파일을 생성한다[3,8]. 컨버터는 일반적인 JVM의 클래스 로딩 작업을 수행하여 On-Card VM에서 발생할 수 있는 클래스 로딩 시의 보안 문제 및 하드웨어 자원 부족 문제를 동시에 해결한다고 볼 수 있다.

2.2 객체 관리를 위한 메모리 모델

스마트 카드는 ROM, RAM, EEPROM/Flash의 3가지 종류를 메모리로 가진다. 자바카드 메모리 모델은 스마트 카드의 메모리 종류와 이의 물리적 특성에 기반한다.

RAM은 단지 카드가 실행되는 동안만 유지될 필요가 있는 임시 데이터를 위해 사용된다. 즉, 자바카드 인터프리터가 바이트 코드를 실행하는 동안 생성되는 중간 값, 메소드 매개변수, 지역변수, 그리고 스택 프레임을 저장하기 위한 런타임 스택 영역이다. 애플릿의 메소드가 호출될 때 마다 스택은 스택 프레임이라 불리는 새로운 데이터 영역이 생성되고, 이 스택 프레임이 모여서 자바 스택을 형성한다. 그림 2는 메소드 호출시 스택 프레임이 할당(push)되는 과정을 나타낸다.

하나의 메소드가 호출/종료될 때마다 EEPROM의 객체 영역에 데이터를 갱신시키고 트랜잭션 영역에 사용했던 객체 로그를 저장시킨다.

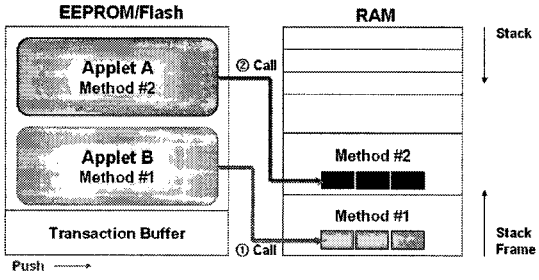

그림 2. 메소드 호출시 스택의 처리 과정

자바카드에는 영구(persistent)객체와 임시(transient)객체가 있다. 영구객체란 한 세션이 끝나거나 전원이 차단되어도 데이터가 유지되는 객체이고, 만일 영구객체의 필드를 갱신하던 중 전원이 차단되면 해당 필드는 이전에 기록 완료되었던 값으로 복구된다. 임시객체란 카드가 한 세션 동안만 데이터를 유지하는 객체를 말하며, 전원이 제거되면 임시객체는 파괴된다. EEPROM의 객체 힙(Object Heap)은 객체들을 저장하는 영역으로서 객체를 생성하면 영구객체나 임시객체의 구분 없이 힙에 저장되지만, 저장되는 형태가 다르다. 애플릿 클래스와 같이 오랫동안 저장해야하는 데이터들은 힙에 영구객체와 데이터 모두를 저장하고, 계속 저장될 필요가 없는 데이터를 임시객체라 하며, 힙에 임시객체와 레퍼런스(RAM의 주소)를 저장하고 RAM 영역에 데이터를 둔다[그림 3].

애플릿은 생명주기 동안 임시객체를 생성하고, 객체의 레퍼런스를 영구객체 필드에 저장하는데, 이는 RAM에 객체가 저장되므로 전원 차단이 발생하여 데이터가 손실되는 것을 방지하기 위함이다. 그리고 임시객체를 위해 CLEAR_ON_DESELECT와 CLEAR_ON_RESET 영역이 RAM에 존재한다[3,8,9]. 이 두 객체는 "new" 연산자를 통해 생성되고, 각각의 객체 테이블에 기록되어 관리되며, JCVM은 애플릿 실행 중 객체 접근이 필요할 때 객체마다 부여된 ID값을 이용한다. 또한, 자바카드 시스템은 데이터 무결성을 보장하기 위해 커밋(commit)과 롤백(rollback)이 가능한 트랜잭션 개념을 제공한다. 트랜잭션을 위한 영역은 트랜잭션 버퍼라 하며, EEPROM에 값(객체, 패키지, 테이블, 레퍼런스)등이 추가되거나 제거될 경우 이전 EEPROM의 값을 유지하기 위해 사용된다. 그림 4는 트랜잭션 버퍼의 구조를 나타낸다.

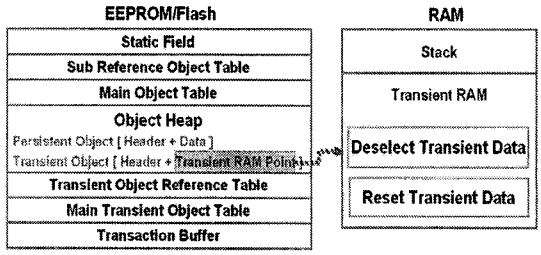

그림 3. 객체 관리를 위한 메모리 구조

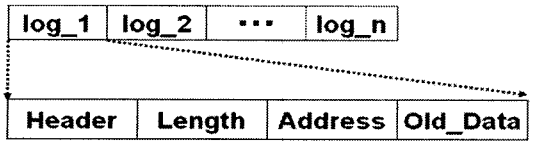

그림 4. 트랜잭션 버퍼의 구조

트랜잭션 버퍼는 4개의 필드로 구성된 log 엔트리들로 구성되는데, Header 필드는 트랜잭션 상태를 나타내고, Length 필드는 이전 데이터의 바이트 수를 나타내며, Address 필드는 EEPROM에 저장된 이전 데이터의 위치를 가리킨다. 마지막으로 old_data 필드는 EEPROM에 저장된 이전 데이터의 크기를 나타낸다.

2.3 기존의 객체 접근 방법과 애플릿 운영 방식

EEPROM과 RAM의 읽는 속도는 비슷하지만 기록 속도는 RAM이 EEPROM에 비해 약 10^4배 정도 빠르다. 이런 이유로, 많은 데이터를 EEPROM에 기록하면 자바 언어를 사용하지 않는 다른 스마트 카드(C, 어셈블리어)에 비해 상당한 성능저하를 가져올 수 있다. 그림 5에서 보듯이 애플릿 객체가 가진 필드들 중에서 영구객체에 대해서는 한 번의 접근으로 객체를 참조할 수 있지만, 임시객체에 대해서는 2번의 참조를 통해 RAM에 있는 실제 데이터에 접근할 수 있다[10].

또한, 애플릿을 운영하기 위해 EEPROM에 애플릿을 동적으로 다운받고 생성하는 모든 객체를 EEPROM에 저장한 후, 메소드의 실행이 완료 될 때까지 변경되는 모든 데이터를 EEPROM에 연속적으로 갱신한다.

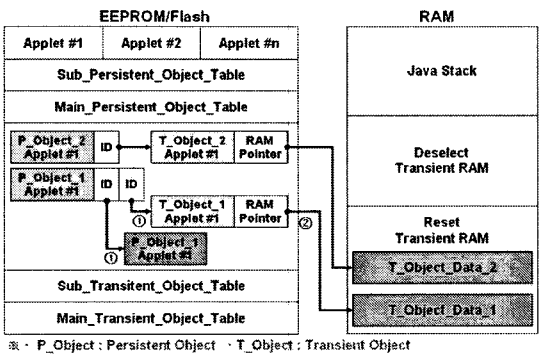

그림 5. 애플릿이 가진 객체들의 배치 방법 및 참조회수

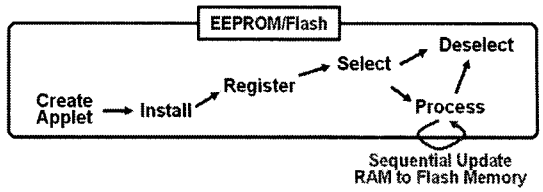

그림 6. EEPROM 기반의 기존 자바 카드 애플릿 생명주기

이러한 데이터 저장 메커니즘으로 자바카드의 속도는 현저히 떨어질 뿐만 아니라 수명 또한 단축시킨다[11-14]. 그림 6은 EEPROM 기반의 기존 자바 카드 애플릿 생명 주기를 나타낸다.

3. 자바카드 시스템 구조 개선 설계

기존 자바 카드의 애플릿 운영 방식의 문제점을 해결하고 자바 카드의 성능을 향상시키기 위해서는 애플릿 운영에 개선된 기법이 필요하다. 이에 변화하고 있는 스마트 카드 환경을 고려하여 자바카드에 "RAM기반 애플릿 실행 시스템"을 적용한 고성능 자바카드 시스템을 개발한다. 본 논문에서 제시하는 애플릿 운영정책을 적용하면 다음과 같은 이점을 얻을 수 있다.

첫째, 하드웨어의 경제적인 측면이다. 자바 카드 플랫폼에서 메모리는 매우 중요한 자원이다. 만약, 쓰기 횟수가 제한된 EEPROM에 기록 연산을 못하게 되면, 스마트 카드는 폐기되어야 한다고 볼 수 있다. 하지만 제시된 시스템을 적용하면 EEPROM의 무절제한 업데이트를 최소화하여 스마트 카드의 생명을 연장시킬 수 있다.

둘째, 자바카드 애플릿 개발 측면이다. 개선된 자바카드 시스템에서는 애플릿을 개발하기 위해 객체를 영구객체와 임시객체로 구분할 필요가 없다. 즉, 자바 카드 애플릿 개발에 전문 지식 없이도 일반적인 자바 애플릿을 개발하듯 자바 카드 애플릿을 개발할 수 있다.

표 2. 메모리별 기록횟수 비교

Type	Write Number
RAM	Unlimited
EEPROM	100,000 ~ 1,000,000
ROM	1

3.1 RAM기반 애플릿 실행 시스템

"RAM기반 애플릿 실행 시스템" 기법은 EEPROM에는 애플릿을 다운로드하고, RAM에는 애플릿 인스턴스들을 저장하는 기법이다. JCVM에 의해 생성되는 애플릿 인스턴스는 RAM에 저장되고, 애플릿을 다운로드하면서 애플릿 생성에 필요한 자바 카드 API 객체들은 EEPROM에 저장하게 하는 것이다.

기존 자바카드 시스템은 EEPROM에 애플릿 인스턴스를 생성하기 때문에 애플릿에 사용된 임시객체를 위해 RAM영역에 임시 데이터를 위한 공간이 필요하고, API에서 제공하는 별도의 메소드를 사용해야 한다[10].

그림 7은 Sun에서 제공하는 자바카드 샘플인 Wallet 애플릿의 일부를 나타낸다. 10개의 필드들이 있으며, buffer와 withdraw_money 필드에 대해서 그림 7의 (a)는 JCSystem 클래스의 임시 객체를 생성하는 메소드를 사용하여 임시 객체를 생성한다. 이 두 개의 필드는 RAM에 생성되며, 크기와 레퍼런스는 EEPROM의 Wallet 객체가 가지고 있다. 또, RAM에 만들어진 모든 임시 객체들에 대한 테이블 또한 EEPROM에 생성된다[15,16].

본 논문에서 제안하는 기법은 RAM 영역에 애플릿 인스턴스를 생성하기 때문에 RAM에는 임시 데이터를 위한 별도의 공간이 필요하지 않다. 이것은 객체를 영구, 임시 객체 2가지로 구분할 필요가 없음을 의미한다. 결국, 그림 7의 (b)와 같이 애플릿 개발자들은 임시 객체를 생성하기 위해 특별한 메소드를 사용할 필요 없이 일반적인 자바 애플릿을 개발하듯이 자바 카드 애플릿을 개발할 수 있다.

반면, RAM에 객체들을 저장하는 것은 RAM의 전체 크기를 소비할 수 있다. 기존 연구에서는 RAM의 적은 용량을 활용하기 위해 힙 영역을 EEPROM과 RAM의 일부 영역이 연접되게 할당하고, 일부 객체를 RAM에 생성하도록 하여 자바카드의 성능을 향상시켰다[15]. 하지만, 본 논문에서는 RAM에 임시 데이터를 위한 공간이었던 CLEAR_ON_DESELECT와 CLEAR_ON_RESET영역을 RAM의 애플릿 실행 영역으로 귀속시킨 후 영역을 확장시켜 사용한다.

그림 8과 같이 트랜잭션 영역은 없다. 개선된 자바카드 시스템은 항상 전원을 공급받는 환경을 위해 구현되었기 때문에 트랜잭션 처리를 하지 않는다. 인스턴스 생성부터 실행, 완료, 인스턴스 제거의 모든 작업이 RAM에서 이루어지므로 실행 중에 전원 손실이 발생한다 하더라도 RAM에서 실행하던 애플릿은 EEPROM의 데이터에 아무 영향을 주지 않는다. 단, 전원이 차단되었다가 인가되면 애플릿을 다시 생성해야 한다는 단점이 있다. 애플릿을 다시 생성하기 위해서는 자바 카드 애플릿 생명주기에 따르는 과정을 거쳐야하기 때문에 그 만큼의 시간을 소비하게 된다.

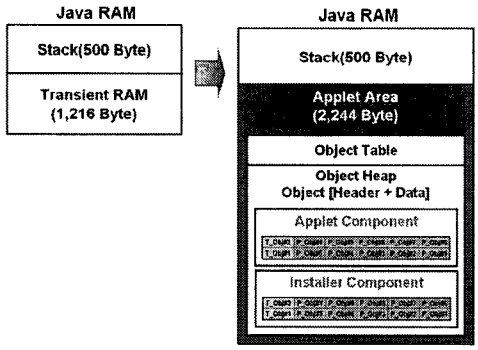

그림 8. 개선된 RAM의 구조

```
Public class Wallet extends Applet{
        short balance;
        short withdraw_money;
        byte[] buffer;
        byte incomingFlag, outgoingFlag, outgoingLenSetFlag;
        byte Irls256Flag, sendInProgressFlag, noChainingFlag;
        byte noGetResponseFlag;

        Wallet(){
                buffer=JCSystem.makeTransientByteArray
                        ((byte)0x14, JCSystem.CLEAR_ON_RESET)
                withdraw_money=JCSystem.makeTransientShortArray
                        ((short)1, JCSystem.CLEAR_ON_RESET)
        }
}
```

(a)

```
Public class Wallet extends Applet{
        short balance;
        short withdraw_money;
        byte[] buffer;
        byte incomingFlag, outgoingFlag, outgoingLenSetFlag;
        byte Irls256Flag, sendInProgressFlag, noChainingFlag;
        byte noGetResponseFlag;

        Wallet(){
                buffer=new byte[20]
                withdraw_money=new short[1]
        }
}
```

(b)

그림 7. 임시객체 생성 여부에 따른 애플릿 코드 : (a) 임시객체를 사용하는 애플릿, (b) 임시객체를 사용하지 않는 애플릿

3.2 개선된 자바카드 시스템 구조 설계

항상 전원을 공급받는 스마트 카드 환경을 고려하여 RAM에 애플릿 실행 영역을 설계하고, 인스턴스 생성과 메소드 수행을 EEPROM이 아닌 RAM에서 운영할 수 있도록 데이터 저장 메커니즘을 설계한다.

3.2.1 자바카드 시스템 메모리 구조 설계

실제 스마트 카드의 사용자가 특정 어플리케이션의 원하는 기능을 실행하기 위해서는 반드시 애플릿 인스턴스가 메소드 관련 객체들을 가지고 있어야만 한다. 즉, 애플릿 인스턴스를 생성하는 이유는 애플릿의 특정 기능에 해당하는 메소드를 실행시키기 위해서이다.

앞서 말했듯이 기존 자바 카드는 애플릿 다운로드와 애플릿 인스턴스 생성을 일련의 작업으로 처리하며 모든 내용을 EEPROM에 저장하지만, 본 논문에서는 EEPROM을 애플릿 다운로드 영역으로 사용하고, RAM은 애플릿 실행 영역으로 사용한다[그림 9].

3.2.2 애플릿 인스턴스 생성 및 저장 과정 설계

그림 10과 같이 애플릿을 EEPROM에 다운로드하고, RAM에 애플릿 인스턴스를 생성하는 방식을 제안한다.

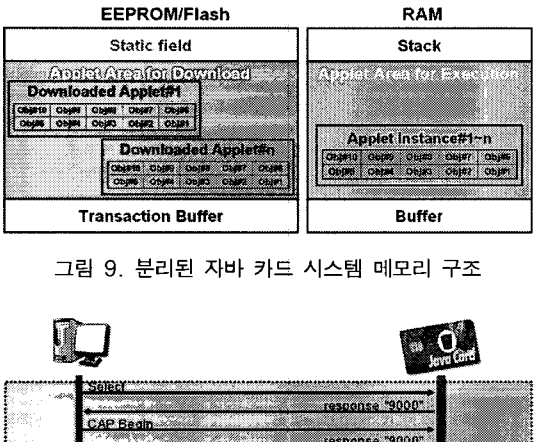

그림 9. 분리된 자바 카드 시스템 메모리 구조

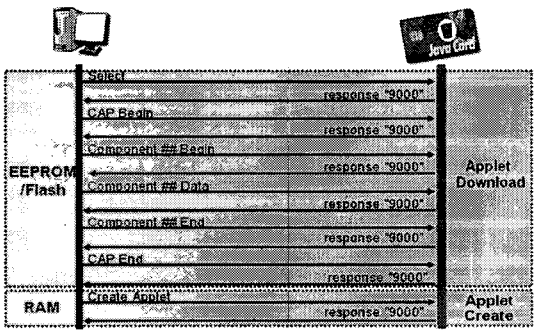

그림 10. 제안된 방식의 애플릿 다운로드 및 생성

애플릿 다운로드 절차는 기존 시스템과 동일하다. 카드가 CAD(Card Acceptance Device)로부터 "Install Applet" APDU를 수신하면 EEPROM에 애플릿을 다운로드하는데, 그 동안 APDU(Application Protocol Data Unit)형태로 전달되는 컴포넌트 정보들을 카드의 EEPROM 영역에 저장하고, 전달받은 패키지 정보들을 로딩 하면서 링킹을 수행 한다. 다운로드가 완료된 후, "Create Applet" APDU 수신시는 RAM 영역에 애플릿 인스턴스를 생성한다. 그리고 "Execute Method" APDU 수신시는 RAM 영역에 생성된 인스턴스로 애플릿의 메소드를 실행한다.

4. 개선된 자바카드 시스템 구조의 구현

자바카드 시스템 구조 개선을 위해서 RAM에 애플릿 실행 영역을 운영한다. 이것은 하드 디스크에는 응용프로그램이 저장되고, 해당 응용프로그램 실행은 RAM에서 이루어지는 일반 컴퓨터 시스템과 같은 구조이다.

4.1 구현 알고리즘

앞선 설명대로 애플릿 다운로드는 EEPROM을 활용하고, 실행은 RAM을 활용한다. 그림 11은 변경된 자바 카드 RAM 구조를 나타낸다.

애플릿 실행은 CAD에서 "Create Applet" APDU를 자바 카드로 전송하여 RAM의 애플릿 실행 영역에 인스턴스를 생성시키고, 메소드를 수행하게 한다. 그림 12는 본 논문에서 제안한 기법의 알고리즘이다.

4.2 개선된 애플릿 인스턴스 생성 절차

자바카드 가상 기계의 인터프리터가 "new"명령

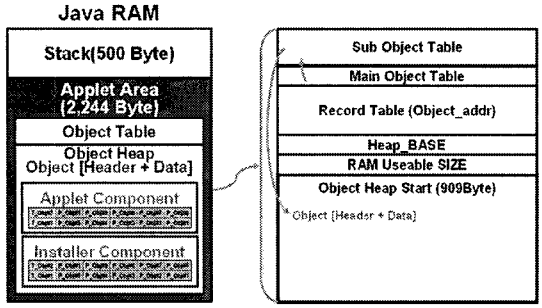

그림 11. 변경된 자바 카드 RAM 구조도

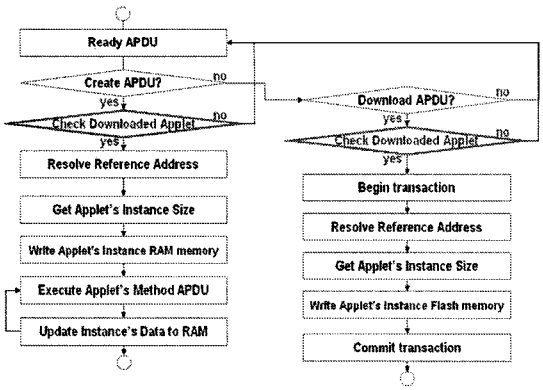

그림 12. 구현 알고리즘

을 통해 객체들을 생성한다. 그림 13은 기존 자바카드 시스템의 객체 생성 과정과 개선된 자바카드 시스템의 객체 생성 과정을 비교하고 있다.

기존 자바카드 시스템은 전원 손실에 대한 데이터 무결성을 보장하기 위해 EEPROM에 애플릿 실행상태를 저장하고, 트랜잭션 영역에 객체의 로그를 기록해야 하지만, 개선된 자바카드 시스템은 항상 전원을 공급받는 환경을 위해 구현되었기 때문에 트랜잭션 처리를 하지 않는다.

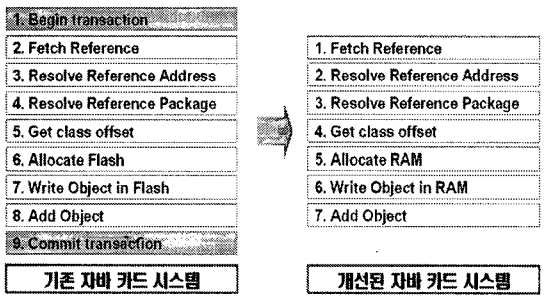

그림 13. 자바카드 시스템의 객체 생성 과정 비교

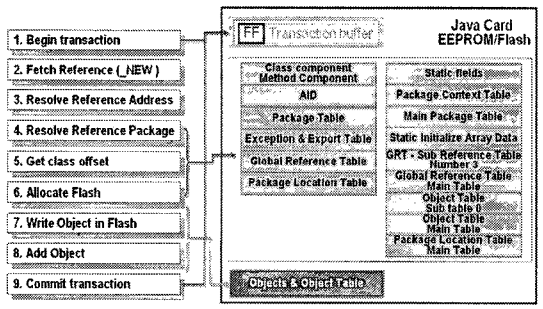

그림 14. 기존 자바카드 시스템의 객체 생성 관련 영역

자바카드 가상 기계는 생성될 객체에 관한 정보를 ROM에 기술되어 있는 코드들을 분석하여 생성할 객체의 형태를 정하고, 객체 내에서 사용될 지역 변수의 타입과 개수를 파악한 다음 실제 객체의 헤더와 연산을 하여 객체의 크기 값을 구한다. 그 후, 가상 기계가 객체를 위한 메모리 공간을 요구하여 EEPROM의 객체 영역에 생성한다. 객체 생성의 마지막 과정이 객체를 관리하는 객체 테이블에 생성된 객체의 위치 정보를 기록함으로써 객체 생성이 완료한다. 기존 자바카드 시스템의 객체 생성 과정은 애플릿 다운로드에 필요한 객체들과 애플릿 실행에 필요한 객체들은 모두 이 절차를 따른다. 그림 14는 기존 자바카드 시스템의 객체 생성 절차에 따른 EEPROM 영역을 나타낸다.

본 논문에서 제안한 기법은 기존 절차대로 EEPROM의 객체 테이블에 애플릿을 다운로드하고, 애플릿 실행에 필요한 애플릿 인스턴스를 개선된 절차에 의해 RAM 영역의 객체 테이블에 생성한다. 그림 15는 개선된 자바카드 시스템의 객체 생성 절차에 따른 RAM영역을 나타낸다.

그림 16은 개선된 자바카드 시스템에서의 애플릿 생명주기를 나타낸다.

개선된 애플릿 실행 환경은 애플릿 인스턴스가 RAM에 생성되기 때문에 실행 중 EEPROM으로 데이터를 갱신시키는 반복 작업이 발생하지 않으므로 자바 카드 실행 속도를 향상시킬 수 있다.

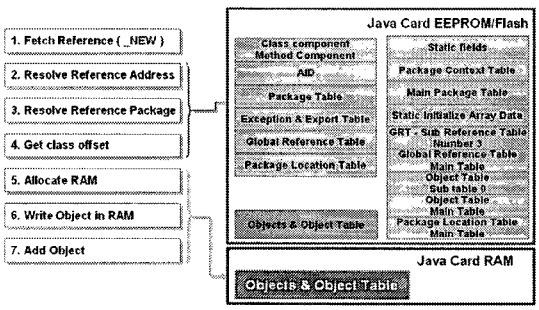

그림 15. 개선된 자바카드 시스템의 객체 생성 관련 영역

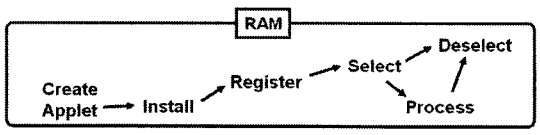

그림 16. RAM 기반의 개선된 자바 카드 애플릿 생명주기

5. 실험결과

본 논문에서 제시한 자바카드 시스템 구조 개선에 대한 정확한 측정을 위해 스마트 카드 개발 보드인 S3FS9QB 모델을 이용하여 실험하였다[6,7]. 이 모델은 RAM 10KB, Program Flash 256KB, Data Flash 64KB의 메모리를 가지며, IC 칩으로 ARM7 Core 32bits MCU를 사용한다.

실험에서는 Sun에서 제공하는 ChannelDemo, JavaLoyalty, JavaPurse, Wallet 등 4개의 자바카드 애플릿과 테스트용으로 제공하고 있는 Demo1, Demo2, Demo3등 3개의 테스트 집합을 이용하였다. 실험은 기존 시스템의 애플릿 생성 및 메소드 실행 속도와 비교했고, 특히 메소드 실행 속도 측정 방법은 데이터를 기록하거나 갱신시키는 메소드들을 중심으로 측정하였다. 그 이유는 표 3에서 보듯이 JavaLoyalty 애플릿의 잔액을 읽는 메소드의 속도는 기존 시스템과 동일하기 때문이다.

기존 시스템의 애플릿 생성 및 메소드 실행 속도와 비교한 결과는 표 4와 같다. 표 4에서와 같이 RAM에 애플릿을 생성하는 기법이 기존 EEPROM/Flash에 생성하는 기존 자바 카드의 생성 속도에 비해 약 58% 정도 효과적이고, 메소드 실행 속도는 33% 정도 효과적인 것으로 나타났다.

표 3. JavaLoyalty 애플릿의 읽기와 기록 속도
(단위 : ms)

JavaLoyalty Applet	기존 시스템	개선된 시스템	Reduced Rate
Create JavaLoyalty	1562	1203	23%
Read Balance	203	203	0%
Reset Balance	156	109	30%
Total	1921	1515	21%

그림 17과 그림 18은 표 4의 수치를 그래프로 나타낸 것이다.

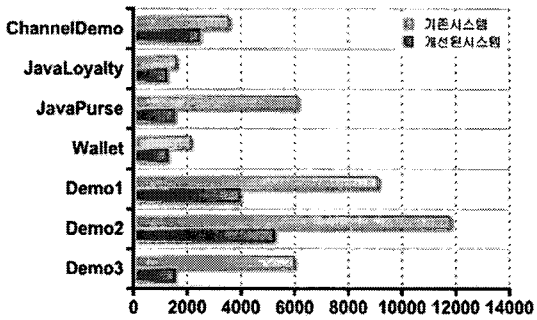

그림 17. 애플릿 생성 시간 비교

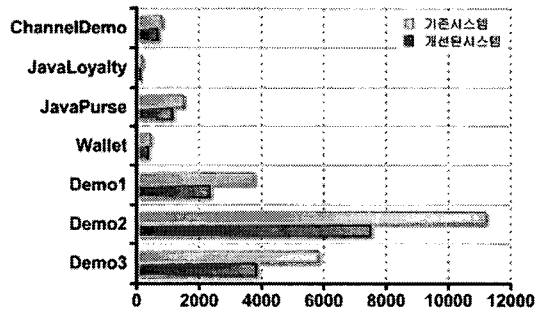

그림 18. 메소드 실행 시간 비교

위 결과에서 애플릿 생성시간은 약 58% 감소한 반면, 애플릿 실행시간은 약 33% 밖에 감소되지 않았다. 그 이유는 테스트 샘플에 있는 메소드들은 갱신시키는 데이터가 아주 적기 때문이다. 즉, 갱신 데이터가 많은 메소드들을 테스트하면 더욱 효과를 볼 수 있다. 또한, Channel Demo, JavaLoyalty, Wallet과 같이 단일 애플릿이거나 크기가 작은 샘플보다

표 4. 애플릿 인스턴스 생성 및 메소드 실행 시간 비교 (단위 : ms)

Applet	애플릿 크기	애플릿 생성 속도			메소드 실행속도		
		기존시스템	개선된시스템	Reduced Rate	기존시스템	개선된시스템	Reduced Rate
Channel Demo	4KB	3516	2422	31%	765	657	14%
JavaLoyalty	3KB	1562	1203	23%	156	109	30%
JavaPurse	12KB	6063	1468	76%	1469	1124	23%
Wallet	3KB	2094	1234	41%	422	344	18%
Demo1	16KB	9062	3906	57%	3719	2313	38%
Demo2	18KB	11766	5203	56%	11171	7483	33%
Demo3	12KB	5953	1500	75%	5812	3812	34%
Total		40016	16936	58%	23514	15842	33%

JavaPurse, Demo1, Demo2, Demo3와 같이 여러 애플릿으로 구성되고 크기가 큰 샘플일수록 생성시간과 실행시간에 더욱 효과적임을 확인 할 수 있다. 그러나 제안된 기술을 적용시키기 위해서는 자바카드 시스템이 대용량 RAM을 지원해야 한다.

6. 결론 및 기대효과

스마트 카드 개발 초기 환경과 달리 메모리 용량의 증가 및 지속적인 전원 공급과 같은 하드웨어와 환경적인 요인의 변화에 따라 스마트 카드도 개선되어야한다. 만약 자바카드 및 SIM카드 사용자가 인증 어플리케이션을 다운로드 받고 설치하기 위해 많은 시간을 소비한다면 어플리케이션 설치를 중단해 버리는 일이 빈번히 발생할 것이다. 본 논문은 변화하고 있는 스마트 카드 환경을 고려하여 "RAM기반 애플릿 실행 시스템"을 적용한 고성능 자바카드 시스템을 제시하였다. 이는 자바 카드 애플릿을 동적 다운로드하고, 자바카드 가상 기계가 생성하는 모든 객체를 비휘발성 메모리에 저장함으로 자바 카드의 속도를 현저하게 떨어지게 하는 기존 시스템 구성을 개선시키기 위함이다.

개선된 자바카드 시스템 구조는 어플리케이션 실행에 관련된 모든 객체를 비휘발성 메모리가 아닌 RAM에 생성함으로써 애플릿 생성 및 메소드 실행 속도를 향상시킬 수 있고, 전원 차단 시 RAM에 생성된 객체가 제거되기 때문에 데이터에 대한 보안측면도 강화시킬 수 있다. 그리고 스마트 카드의 생명 주기 기준은 비휘발성 메모리의 마모 정도이기 때문에 본 논문에서 제시하는 시스템을 적용하게 되면 EEPROM/Flash의 무절제한 업데이트를 최소화하여 스마트 카드의 생명을 연장시킬 수 있다. 그리고 객체의 구분을 제거함으로써 애플릿 개발자들이 자바카드의 전문적인 지식 없이도 애플릿을 개발할 수 있게 된다. 하지만 전원이 차단되었다가 인가되면 애플릿을 다시 생성해야 한다는 단점이 발생한다. 이를 위해 RAM에도 트랜잭션 버퍼를 생성하여 일정량의 로그들을 EEPROM으로 복사하는 메커니즘이 향 후 연구 되어야 할 것이다.

제안한 기법들을 요약하면 개선된 자바카드 시스템 구조는 일반적인 컴퓨팅 시스템 구조와 유사하며, 비휘발성 메모리의 공간 활용도를 높이고 애플릿 실행 속도 또한 개선될 뿐만 아니라 스마트 카드의 생명을 연장시켜주는 이점도 가진다. 스마트 카드 응용 환경이 핸드폰, 가전기기, 홈네트워크 장비 등과 같이 항상 전원을 공급받는 환경으로 점차 발전하고 있음을 고려할 때, 스마트 카드 시스템 개발에서 구조 개선에 대한 새로운 방안을 제시할 수 있을 것이다.

참 고 문 헌

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

[2] Michael Caentsch, "Java Card-From Hype to Reality," *IEEE Concurrency*, pp. 36-43, 1999.

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

[4] Zhiqun Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's guide*, Addison Wesley, 2001.

[5] Java Card Forum, http://www.javacardforum.org.

[6] MCULAND, http://mculand.com/e/sub1/s1main.htm.

[7] SAMSUNG, http://www.samsung.com/Products/Semiconductor.

[8] Sun Microsystems, Inc., *The Java Card™ 2.2.1 Runtime Enviroment Specification*, SUN, 2003.

[9] M. Oestreicher and K. Ksheeradbhi, "Object lifetimes in javacard," *Proc. Usenix Workshop Smart Card Technology*, Usenix Assoc, Berkeley, Calif., pp. 129-137, 1999.

[10] 김영진, 전용성, 전성익, 정교일 "Java Card Platform을 내장한 Smart Card의 구현," 정보과학회지, 제 19권, 제 8호, pp. 33-43, 2001.

[11] X. Leroy, "On-Card Bytecode Verification for Java Card," *E-smart 2001*, LNCS 2140, pp. 150-164, 2001.

[12] Lars R. Clausen, Ulrik Pagh Schultz, Charles Consel, and Gilles Muller, "Java Bytecode Compression for Low-End Embedded Sys-

tems," *ACM Transactions on Programming Languages and Systems*, Vol. 22, No. 3, pp. 471-489, 2000.

[13] Sun Microsystems, Inc., *The Java Card™ 2.2 Runtime Environment Specification*, Sun Microsystems, Inc., 2002.

[14] Sun Microsystems, Inc., *The Java Card™ 2.2 Virtual Machine Specification*, Sun Microsystems, Inc., 2002.

[15] Min-Sik Jin and Min-Soo Jung, "A study on fast JCVM by moving object from EEPROM to RAM," *Proceeding of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Application*, RTCSA'05, pp. 84-88, 2005.

[16] Yoon-sim Yang, Won-Ho Choi, Min-Sik Jin, Cheul-Jun Hwang and Min-Soo Jung, "An Advanced Java Card System Architecture for Smart Card Based on Large RAM Memory," *2006 International Conference on Hybrid Information Technology*, ICHIT2006, Vol. 2, pp. 646-650, 2006.

이 동 욱

2006년 2월 경남대학교 컴퓨터공학부 학사
2006년 3월~현재 경남대학교 컴퓨터공학과 석사과정
관심분야 : Java Technology, Java Card, Home-Network

황 철 준

2003년 경남대학교 컴퓨터공학부 학사
2005년 경남대학교 컴퓨터공학부 석사
2005년~현재 경남대학교 컴퓨터공학부 박사과정
관심분야 : Ubiquitous, Smart Card, Compiler

양 윤 심

2001년 경남대학교 컴퓨터공학과 학사
2003년 경남대학교 컴퓨터공학과 석사
2007년 경남대학교 컴퓨터공학과 박사
2007년 1월~현재 (주)이-머니 금융
관심분야 : Embedded, Java Card, JavaMachine

정 민 수

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