Journal of Korea Multimedia Society Vol. 12, No. 1, January 2009(pp. 41-49)

다중 트랜잭션 기법을 이용한 자바 카드 가상 기계 성능 향상

노태헌[†], 이동욱[††], 정민수[†††]

요 약

오늘날 대부분의 스마트카드는 자바카드 플랫폼을 채택한 자바카드가 표준안으로 자리매김 하고 있다. 자바카드는 전원이 차단되면 작업 중이던 데이터가 손실되는 문제점을 가지고 있다. 데이터 손실의 해결책으로 트랜잭션 개념이 사용되지만, 한 번에 하나의 트랜잭션 처리만을 허용하므로, 트랜잭션이 필요한 작업들은 이전에 작업 중인 트랜잭션이 끝날 때까지 대기해야 하고, 이는 자바 카드의 성능을 저하시키는 요인이다. 본 논문에서는 자바 카드의 수행 성능을 향상시키기 위한 방안으로 다중 트랜잭션 버퍼에서 트랜잭션을 수행할 수 있도록 이중 잠금 규칙을 규정하고, 이 규칙에 따라 트랜잭션을 처리함으로써 트랜잭션 처리 성능을 개선시키고자 한다. 이러한 연구 결과를 통해 데이터의 안전성을 높이고 트랜잭션이 필요한 데이터의 대기 시간을 줄이고 자바 카드의 수행 속도를 증가 시킬 수 있다.

A Performance Enhancement of Java Card Virtual Machine with Multi-Transaction

Tae-heon Noh[†], Dong-wook Lee[††], Min-soo Jung[†††]

ABSTRACT

Smart Card is currently more popular in mobile communication, and smart card with java card platform becomes a standard choice. Java card has a problem that it gets lost working data when power is off. Transaction is the idea to solve a problem of data loss, but it accepts only one transaction process, and other transaction process need to hold until the current working transaction is finished. This might be a factor to drop the Java card's performance. In this paper, we define a rule of dual-lock which can run transaction at multiple transaction buffer as a method for a better java card performance, and we suggest this rule to improve a capability of transaction process. From this research, we are able to improve the data stability, reduce the java card transaction delay time, and get a higher processing speed of java card.

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

※ 교신저자(Corresponding Author) : 정민수, 주소 : 마산시 월영 2동 449번지(631-701), 전화 : 055)249-2217, FAX : 055)248-2554, E-mail : msjung@kyungnam.ac.kr
접수일 : 2008년 6월 2일, 완료일 : 2008년 10월 20일
[†] 준회원, 경남대학교 컴퓨터공학과 석사 과정
(E-mail : purely33@nate.com)
[††] 준회원, 경남대학교 컴퓨터공학과 석사
(E-mail : smalldo2@nate.com)
[†††] 종신회원, 경남대학교 컴퓨터공학부 교수
※ 본 연구는 2008년 경남대학교 학술연구 장려금 지원으로 수행되었음

1. 서 론

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

스마트카드는 일반적인 신용 카드의 크기와 모양이 매우 비슷하지만 카드 자체에 프로세서와 메모리를 내장하고 있어서 일정 수준의 정보 저장과 처리 능력을 가지는 하나의 작은 컴퓨터라고 할 수 있으며, 이러한 스마트카드의 데이터 저장 및 처리 능력 때문에 다양한 분야에서 활용되고 있다.

특정 목적을 위한 폐쇄형 방식의 스마트카드에 탑재되는 어플리케이션은 특정 하드웨어와 카드 운영체제(COS : Chip Operating System)에 한정적으로 적용 가능하며 이는 주로 어셈블리어나 C언어로 작성되었다. 이것은 스마트카드 어플리케이션 개발에 많은 시간과 비용을 요구하게 되었으며, 일단 카드가 발급된 이후에는 어플리케이션을 업그레이드하거나 추가하는 것이 쉽지 않았다. 이러한 폐쇄형 스마트카드의 문제점을 극복하기 위해 하드웨어에 의존하지 않는 개방형 플랫폼인 자바 카드가 등장하게 되었다.

현재의 자바 카드 기술은 스마트카드 영역을 넘어서 USIM(Universal Subscriber Identity Module) 및 유비쿼터스 환경의 표준 플랫폼으로 자리 매김하고 있다. 무선 환경에서 이동전화 사용자는 SIM 카드로 불리는 손톱만한 크기의 자바 기술이 탑재한 IC 칩을 사용한다. 이동전화에 삽입되는 SIM 카드에는 사용자 및 이동전화 인증을 위한 인증 데이터 및 킷값을 저장할 뿐만 아니라, 모바일 뱅킹 및 티켓팅과 같은 서비스를 지원하기 위한 어플리케이션도 탑재된다. 이러한 자바 카드 기술이 탑재된 SIM 카드는 이미 전 세계 이동전화 사용자들에게 다양한 서비스를 제공하고 있다.

자바 카드는 개방형 플랫폼이라는 점에서 비롯되는 동적인 어플리케이션의 추가 및 삭제, 바이트 코드의 이식성, 플랫폼 독립성, 높은 보안성 등의 장점을 제공하지만, 자바 언어의 특성에서 기인되는 인터프리터 방식으로 인해 C 언어나 어셈블리어로 작성된 스마트카드보다 수행 속도가 느리다[1]. 또한 자바 카드 가상 기계(JCVM : Java Card Virtual Machine)가 애플릿의 바이트 코드를 실행하는 동안 하드웨어의 구조적인 이유로 다음과 같은 문제점을 가지고 있다.

첫째, 자바 카드는 자체적으로 전원이 공급되지 않는 구조로 전원이 잠시 공급되는 동안 애플릿을 실행하는 구조이다. 애플릿을 실행하던 중 예기치 않게 전원이 차단되거나 카드가 카드 리더기(CAD : Card Acceptance Device)에서 빠질 경우 데이터의 손실을 막기 위해 비휘발성 메모리인 EEPROM에 데이터를 항상 저장하고 갱신한다[2,3].

둘째, 자바 카드가 가지는 메모리 자원과 그 메모리를 다루는 자바 카드 가상 기계의 기능 제한적인 특성으로 인하여 데이터를 추가하거나 갱신하기 위한 작업을 순차적으로 한 번에 하나의 데이터 처리만을 수행한다.

본 논문에서는 자바 카드가 가지는 하드웨어와 소프트웨어 기능상의 문제점들을 해결하기 위해서 최근 개발되고 있는 향상된 하드웨어 자원을 가지는 스마트카드에 기존의 자바 카드 가상 기계가 새로운 하드웨어 환경에 최적화 될 수 있도록 데이터의 무결성을 유지하는데 사용되는 버퍼 영역을 확장하고, 확장한 버퍼 영역을 두 단으로 분리하여 메모리 구조를 변경한다. 분리된 버퍼 영역을 활용하여 데이터를 기록하기 위한 대기 시간을 줄임으로써 자바 카드의 수행 성능을 향상 시키고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 자바 카드, 자바 카드에서 생성되는 객체, 트랜잭션 버퍼와 방식에 대해 기술하고, 3장에서는 기존 트랜잭션 방식의 문제점에 대해 기술하고, 다중 트랜잭션 알고리즘 설계 및 다중 트랜잭션 버퍼를 이용하는 트랜잭션 기법을 구현한다. 4장에서는 실험 결과 및 실험 평가하며, 마지막으로 5장에서는 결론 및 기대효과를 제시한다.

2. 관련연구

2.1 자바 카드

자바 카드란 스마트카드의 한 종류로써 개방형 플랫폼에 속한다. 자바 카드 기술은 지금까지의 스마트카드들이 하드웨어 환경에 따라 개별적인 어플리케이션을 사용하고, 이러한 어플리케이션 개발이 각각의 스마트카드 하드웨어에 맞추어져 이루어지던 것이 자

바의 플랫폼 독립적인 실행 특성을 도입함으로써, 스마트카드 어플리케이션 개발에 있어서 가장 효과적인 방법으로 자리 잡게 되었다. 스마트카드에 도입된 자바 카드 기술은 다음과 같은 장점들을 가진다[4,5].

자바 카드 애플릿은 스마트카드 하드웨어 플랫폼에 독립적이므로, 자바 카드 가상 기계가 탑재되어 있다면 자바 카드 애플릿의 구동이 가능하다. 또한 자바 언어 자체의 보안적인 특성의 많은 부분을 수용하고 있고, 자바 카드 내부의 독립적인 애플릿들은 방화벽에 의해 보호되므로, 높은 보안성을 가진다. 그리고 어플리케이션을 개발하는데 필요한 공통적인 API를 제공함으로써 보다 효율적인 개발환경을 제공하므로 애플릿 개발 시간과 비용이 절감된다. 폐쇄형 스마트카드는 카드가 한번 발급되면 애플릿을 추가하거나 업데이트가 어렵지만, 자바 카드는 카드가 발급된 후에도 추가, 삭제할 수 있는 사후 발급(Post Issuance)기능을 가지며, 다중 어플리케이션을 탑재할 수 있다[2,5,6].

자바 카드는 스마트카드라는 자원 제약적인 하드웨어 환경을 기반으로 둔 자바 기술이며, 제한된 메모리 문제를 해결하기 위해 스마트카드에서 동작하는 자바 카드 가상 기계의 크기가 작아야 한다. 자바 카드 가상 기계는 그림 1과 같이 크게 두 부분으로 나누어진다. 이것은 제한된 메모리 자원을 갖는 자바 카드의 특성을 고려하여 자원을 많이 사용하는 부분(온-카드)과 적게 사용하는 부분(오프-카드)으로 구분한 것이다[7-12].

2.2 자바 카드에서 생성되는 객체

임시 객체는 자바 카드 API를 호출하여 생성되는 객체로서 한번의 카드 세션 동안만 유지될 필요가 있는 객체를 말하고, 이 객체의 데이터는 RAM에 저장된다. 영속 객체는 카드 세션 동안 객체가 가지는 상태 정보를 유지하는 객체로서, 카드 세션이 끝나더라도 데이터는 EEPROM에 저장되어 있다. 그리고 카드에 전원이 차단되거나 카드 리더기에서 카드가 빠지는 경우, 영속 객체는 그 상태와 데이터들을 지속적으로 유지하고, 트랜잭션 과정을 통해 변경되는 데이터를 트랜잭션 버퍼에 저장한다.

영속 객체가 가지는 필드의 추가나 갱신은 원자성(Atomic)을 지녀야 한다[3]. 즉, 영속 객체가 가지는 데이터를 수정하던 중에 카드에 전원이 차단된 후 전원이 다시 재 인가되면 이전의 값으로 복구되어야 한다는 것이다. 전원이 차단되었을 경우 이전 값으로 복구하기 위해 변경되기 이전의 데이터를 저장하는데 사용되는 영역이 바로 트랜잭션 버퍼이다.

2.3 트랜잭션 버퍼

트랜잭션은 카드 내부에 영속 객체가 생성되거나 영속 객체가 가지는 값이 변경이 되어서 EEPROM에 값을 저장하는 경우, 데이터를 기록하는 과정 중에 발생할 수 있는 프로그램 상의 오류나 전원 차단 등으로부터 자바 카드의 데이터를 변경 전의 상태로 복구할 수 있게 해주는 기능이다.

자바 카드에서는 트랜잭션을 위한 영역으로 EEPROM의 가장 하단부에 트랜잭션 버퍼라는 영역이 존재한다.

트랜잭션 버퍼는 일반적으로 2K 바이트 정도의 크기를 가지며, EEPROM에 저장된 영속 객체, 객체 관리를 위한 테이블정보, 레퍼런스 등이 추가나 변경될 경우 이전 EEPROM의 값을 유지하기 위해 사용된다[3].

그림 2는 트랜잭션 버퍼의 구조를 나타내며, 트랜잭션 버퍼는 로그 엔트리(Log Entry)들로 구성된다.

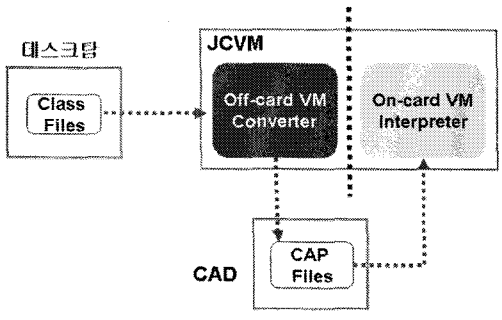


그림 1. 자바 카드 가상 기계의 실행 환경

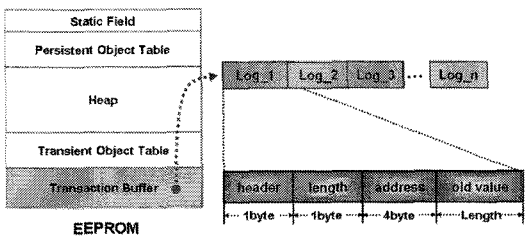


그림 2. 트랜잭션 버퍼의 구조

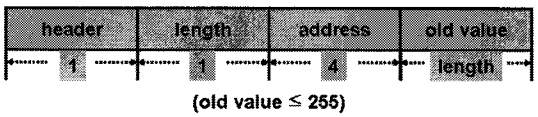


그림 3. 로그엔트리의 구조

트랜잭션 버퍼를 구성하는 로그 엔트리는 그림 3과 같이 4개의 필드로 구성되며, 커밋의 상태를 나타내는 1바이트 길이의 Header 필드, 이전에 기록되어 있던 old value의 길이를 나타내는 1바이트의 Length 필드, EEPROM에 저장된 위치를 나타내는 4바이트 길이의 address 필드와 실제 데이터 값을 나타내는 old value 필드로 구성된다.

2.4 트랜잭션 방식

스마트카드는 자체적으로 전원 공급을 할 수 없는 구조로 갑자기 전원이 차단되는 경우가 존재한다. 자바 카드는 데이터 무결성을 보장하기 위해서 단위 트랜잭션(Atomic Transaction)에 대한 커밋(Commit)과 롤백(Rollback)이 가능한 트랜잭션 개념을 제공한다. 그러므로 자바 카드는 트랜잭션을 이용하여 객체의 데이터를 하나의 단위로 갱신하는 것이 가능하다. 자바 카드 API는 트랜잭션 기능을 사용할 수 있도록 JCSystem 패키지에서 beginTransaction(), commitTransaction(), abortTransaction() 메소드를 제공하고 있다[3,14].

그림 4는 트랜잭션을 수행하는 메소드를 나타낸다. 기본적으로 beginTransaction()을 호출함으로써 트랜잭션이 시작된다. 그리고 트랜잭션이 정상적으로 수행되었다면 commitTransaction()을 호출하여 트랜잭션을 완료시킨다. 만약, 트랜잭션 중에 abortTransaction()이 호출되면 트랜잭션이 정상적으로 완료되지 않은 경우이므로 트랜잭션을 시작하기 전의 상태로 복구되게 된다[8].

영속 객체가 생성되거나 영속 객체의 데이터가 변

```
//begin a transaction
JCSystem.beginTransaction();
...
//update of persistent data
...
//commit a transaction
JCSystem.commitTransaction();
```

```
//begin a transaction
JCSystem.beginTransaction();
...
//update of persistent data
...
//abort a transaction
JCSystem.abortTransaction();
```

그림 4. 트랜잭션을 위한 메소드

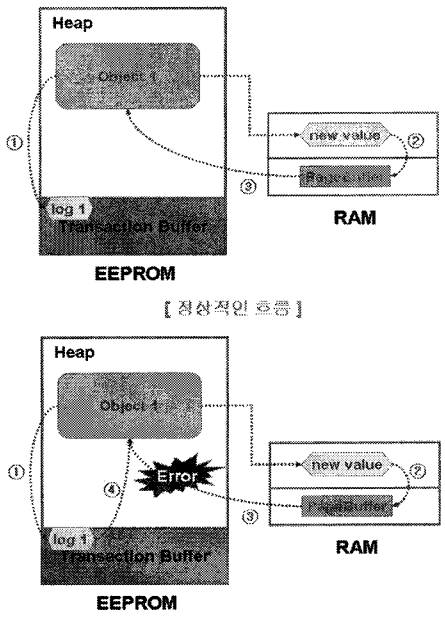


그림 5. 데이터 수정에 따른 트랜잭션의 흐름

경 될 경우 트랜잭션은 변경되기 전의 데이터 길이, 주소, 기록되어있던 값(Old Value)을 트랜잭션 버퍼에 먼저 복사한 후 EEPROM의 실제 위치에 변경될 데이터를 저장하고 트랜잭션을 완료시킨다.

만일, 데이터를 갱신하기 위해 트랜잭션 처리를 하고 있을 때 프로그램 상의 오류로 abortTransaction()이 호출되면 트랜잭션 버퍼에 저장된 이전의 데이터로 바로 복구시키고, 전원 차단으로 인해 트랜잭션이 중지되었을 경우에는 전원이 재 인가될 때 그림 5와 같이 트랜잭션 버퍼에 저장된 데이터를 이용하여 이전 값으로 복구 시킨다[3,6,8].

3. 효율적인 다중 트랜잭션 알고리즘

3.1 기존 트랜잭션 방식의 문제점

기존의 자바 카드에서는 그림 6과 같이 한번에 하나의 트랜잭션 처리만을 허용한다. 스마트카드가 가지는 하드웨어 자원인 메모리의 용량이 적었기 때문에 자바 카드 개발초기에 메모리 자원을 최소한으로 사용하기 위해서 트랜잭션 버퍼를 하나만 할당했기 때문이다.

만일, 트랜잭션 처리를 위해 beginTransaction()

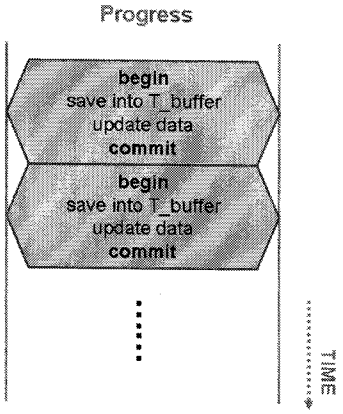


그림 6. 단일 트랜잭션 스케줄

을 호출하였을 때 이미 다른 트랜잭션이 처리되고 있다면 JCRE는 TransactionException을 호출시킴으로써 다중 트랜잭션 처리를 막고 있다.

자바 카드가 EEPROM을 사용하는 것은 전원이 차단되어도 데이터를 유지하기 위한 목적이었지만, 이는 자바 카드의 수행 속도를 저하시키는 원인이 되었다. 자바 카드의 수행속도가 저하 되는 이유는 RAM에 비해 쓰기 속도가 1000배 정도 느린 특성을 가지는 EEPROM에 변경되는 영속 객체의 모든 데이터를 저장하기 때문이다[3,4].

기존의 자바 카드에서는 단일 트랜잭션 기법을 사용함에 있어서 표 1과 같이 이진 잠금(Binary Lock) 규칙을 통해 다중 트랜잭션 처리를 막고 있다. 하나의 트랜잭션이 수행되면 트랜잭션을 요구하는 다른 작업이 접근하지 못하도록 잠금(Lock)상태로 만들고, 트랜잭션이 완료되면 잠금 해제(Unlock)를 하여 트랜잭션을 수행 가능한 상태로 만든다.

자바 카드의 속도 측면에서 보면, 연속적이지 않은 트랜잭션을 처리하기 위해서는 크게 문제되지 않지만, 만약 갱신해야 할 데이터가 많을 경우에는 이전의 트랜잭션 처리가 완료될 때까지 트랜잭션이 필요한 데이터들은 트랜잭션을 위한 대기 시간이 길어지게 되므로 자바 카드의 수행속도를 느리게 만드는 원인으로 작용하고 있다.

표 1. 이진 잠금(Binary Lock) 규칙

- 트랜잭션은 하나의 Progress로 처리된다.
- 트랜잭션이 수행되면 트랜잭션을 잠금(Lock) 상태로 설정한다.
- 트랜잭션이 완료되면 트랜잭션을 잠금 해제(Unlock) 상태로 설정하여 트랜잭션 가능 상태로 만든다.
- 하나의 트랜잭션 수행 중에 다른 트랜잭션을 요구하는 작업은 접근하지 못한다.

3.2 다중 트랜잭션 알고리즘 설계

자바 카드에서는 카드의 전원 차단이나 오류에 대해 데이터의 무결성을 보장하기 위해서 EEPROM에 저장된 영속 객체, 객체 관리 테이블, 레퍼런스 등이 변경될 때마다 변경되는 데이터를 트랜잭션 버퍼에 저장한다[13]. 하지만 표 2와 같이 이로 인해 EEPROM의 70% 이상에 달하는 쓰기 수행이 트랜잭션 버퍼에서 발생하게 되었고, 자바 카드의 수행 속도를 저하시키는 요인이 되었다[9]. 이 문제점의 해결책으로 본 논문에서는 다중 트랜잭션 버퍼를 이용하여 트랜잭션의 처리 시간을 감소시킴으로써 자바 카드의 수행 성능을 향상 시키고자 한다.

기존의 자바 카드에서는 하나의 트랜잭션 버퍼를 사용하지만 본 논문에서 제안하는 다중 트랜잭션 버퍼를 이용하는 트랜잭션 기법은 트랜잭션을 위한 영역인 트랜잭션 버퍼가 두 개 필요하다. 일반적으로 자바 카드에서 트랜잭션 버퍼의 크기는 2K 바이트를 할당하지만, 본 논문에서는 두 개의 트랜잭션 버퍼를 할당하기 위해 전체 트랜잭션 버퍼의 크기를 3Kbytes로 할당하고, 그것을 1.5K 바이트씩 두 개로 분리하였다. 트랜잭션 버퍼를 두 개로 분리하더라도 기존 2K 바이트 크기의 트랜잭션 버퍼에 기록하던 로그 데이터를 1.5K 바이트씩 분리된 트랜잭션 버퍼에 나누어서 기록하기 때문에 기존보다 더 많은 로그 데이터를 저장할 수 있다.

그림 7에서 보는 바와 같이 확장한 트랜잭션 버퍼를 Transaction Buffer_0, Transaction Buffer_1 으로 나누었다. 트랜잭션 버퍼를 두 단으로 나눈 이유

표 2. EEPROM의 각 영역별 기록 횟수

EEPROM Area	The number of EEPROM write
Static Field	1,681
Persistent Object Table	161
Heap	1,275
Transient Object Table	39
Transaction buffer	10,121
Total	13227

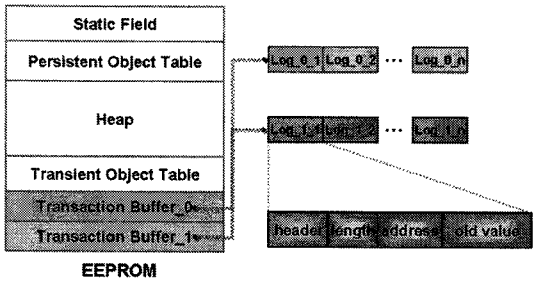


그림 7. 트랜잭션 버퍼의 확장

는 하나의 트랜잭션 버퍼에 너무 많은 기록 작업을 수행하면 스마트카드의 수명을 단축시키는 요인이 되므로, 이것을 두 개의 영역으로 나누어 데이터를 분산하여 기록하려는 의도도 있지만, 가장 중요한 이유는 다중 트랜잭션 버퍼를 이용하여 트랜잭션을 빠르게 처리하기 위해서이다. 분리된 각각의 트랜잭션 버퍼에는 동일한 구조와 형태를 가지는 로그 엔트리들이 저장되게 된다.

그림 8은 다중 트랜잭션 버퍼를 이용하는 트랜잭션의 스케줄을 나타낸다. 스케줄에 따라 트랜잭션을 수행하기 위해서는 beginTransaction()이 호출될 때 변경될 데이터의 길이, 주소, 값을 매개 변수로 받아와야 하고, 트랜잭션을 제어할 수 있는 규칙이 필요하다. 본 논문의 트랜잭션 스케줄은 표 3의 이중 잠금(Double Lock) 규칙에 의해 제어된다.

제안된 트랜잭션 스케줄과 트랜잭션 규칙을 따르게 되면 연속적으로 일어나는 트랜잭션의 수행 시에 이중 잠금 규칙을 통해 트랜잭션이 제어되고, 트랜잭션 수행 중에 다른 트랜잭션이 요구 되더라도 트랜잭

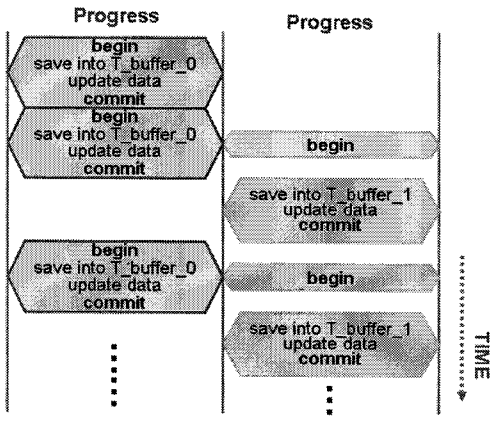


그림 8. 다중 트랜잭션 스케줄

표 3. 이중 잠금(Double Lock) 규칙

- 트랜잭션은 두 개의 Progress로 처리된다.
- 트랜잭션_0과 트랜잭션_1 모두 잠금 해제 상태일 때 트랜잭션_all은 잠금 해제 상태이다.
- 트랜잭션_0만 잠금 상태일 때 트랜잭션_all은 잠금 해제 상태이다.
- 하나의 트랜잭션이 수행되면 트랜잭션_0을 잠금 상태로 설정한다.
- 트랜잭션_0이 수행 중일 때, 다른 트랜잭션이 요청되면 트랜잭션_1을 잠금 상태로 설정 한다.
- 트랜잭션_0과 트랜잭션_1이 모두 잠금 상태이면 트랜잭션_all을 잠금 상태이다.
- 트랜잭션_0을 먼저 수행완료 한 후, 트랜잭션_1을 수행한다.
- 트랜잭션_0과 트랜잭션_1이 모두 완료되면 트랜잭션_all을 잠금 해제한다.

션 버퍼의 위치가 다르기 때문에 순차적으로 대기 중인 트랜잭션 데이터들을 빠르게 처리할 수 있게 된다. 이와 같이 본 논문의 트랜잭션 스케줄은 일반적인 PC 환경의 어플리케이션에서 사용할 수 있는 공유 자원과는 달리 자바 카드 애플릿은 방화벽을 통해 여러 애플릿들이 분리되고, 자기 자신만의 컨텍스트 안에 있는 객체에만 접근할 수 있으며, 독립된 메모리 공간을 사용하기 때문에 다른 애플릿에 의해 객체 데이터를 변경할 수 없으므로 그림 8과 같이 수행하는 것이 가능하다.

3.3 다중 트랜잭션 버퍼를 이용하는 트랜잭션 기법 구현

3.3.1 구현 환경

본 논문에서 설계된 다중 트랜잭션 버퍼를 이용한 트랜잭션 기법을 구현하기 위해 그림 9와 같이 16비트의 마이크로컨트롤러와 160KB 의 ROM, 32KB 의 EEPROM, 6KB 의 RAMdmf 가지는 스마트카드 개발 보드인 S3CC9P9 모델을 이용한다.

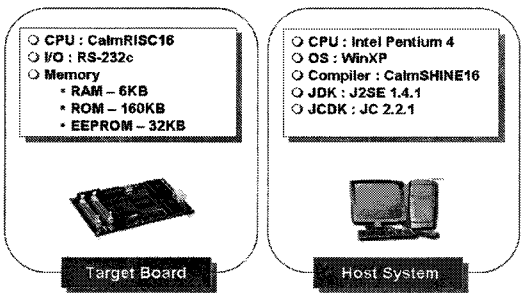


그림 9. 구현을 위한 호스트 시스템과 개발 보드

그림 9에서 보는 바와 같이 호스트 시스템 상에 설치된 CalmSHINE16 개발 도구를 이용하여 다중 트랜잭션 버퍼를 이용하는 트랜잭션 기법을 구현하고, 자바 카드를 타겟 보드 상의 ROM과 EEPROM에 탑재할 수 있는 형태인 이미지 파일로 변환한 후, 시리얼(RS-232C) 통신을 통해 이미지 파일을 타겟 보드에 적재시킨다. 그리고 CalmSHINE16 개발 도구를 사용하여 자바 카드 프로그램의 테스트 및 오류 수정을 한다.

3.3.2 트랜잭션 기법의 구현

그림 10은 다중 트랜잭션 버퍼를 이용하는 트랜잭션 기법에 관한 알고리즘을 나타낸다. 본 논문의 알고리즘은 이중 잠금 규칙에 의해서 제어된다.

자바 카드의 트랜잭션 수행 성능을 향상시키기 위하여 Transaction Buffer_0과 Transaction Buffer_1의 두 개의 트랜잭션 버퍼를 사용한다. 그리고 각각의 트랜잭션 버퍼에서 트랜잭션을 수행할 수 있도록 구현하였다.

연속적이지 않은 트랜잭션을 수행하기 위해서는 이진 잠금 규칙을 따르고, 기본적으로

Transaction Buffer_0에 트랜잭션을 수행한다. 그리고 트랜잭션이 진행 중이라는 것을 나타내기 위해 in_progress_0과 in_progress_1을 static 변수로 선언하여 트랜잭션 진행 상태를 나타낸다.

Transaction Buffer_0에서 트랜잭션을 수행하게 되면 in_progress_0을 true로 설정(Lock)한 후 트랜잭션을 수행하게 되는데, 이때 다른 객체가 트랜잭션 처리를 요구할 경우 in_progress_0의 상태 값이 true라면 Transaction Buffer_1에서 트랜잭션을 수행하게 된다. 하지만 먼저 Transaction Buffer_0에서 작업 중이던 트랜잭션이 완료되고 난 뒤 Transaction Buffer_1에서 다른 객체의 트랜잭션을 수행하게 되는데, Transaction Buffer_0에서 트랜잭션 수행 중에 다른 객체가 요구한 트랜잭션 작업이 존재한다는 것을 나타내기 위해 in_progress_1의 값을 beginTransaction()이 호출되었을 때 true로 설정해주어야 하고, 변경될 데이터의 길이, 주소, 값을 매개 변수로 받아와야 한다. 트랜잭션이 완료되면 in_progress_0과 in_progress_1의 상태 값을 false으로 설정(Unlock)하여 트랜잭션 가능 상태로 만들어 준다.

만일, Transaction Buffer_0과 Transaction Buffer_1에서 모두 트랜잭션을 수행 중이라면 트랜잭션 이중 잠금 규칙에 따라 트랜잭션을 요구하는 어떤 작업도 트랜잭션을 수행하지 못하도록 한다.

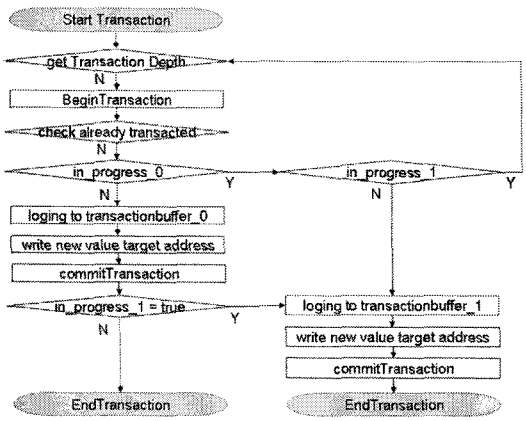


그림 10. 제안된 트랜잭션 기법의 알고리즘

4. 실험 결과 및 실험 평가

본 논문에서 구현된 다중 트랜잭션 버퍼를 이용하는 트랜잭션 기법을 적용시킨 자바 카드와 기존 자바 카드의 테스트 공정성을 위하여 자바 카드 패키지에서 제공하는 Sun사의 기본 샘플들을 이용하여 애플릿의 설치부터 메소드 실행까지의 시간을 비교한다.

이를 위한 실험 환경은 그림 11과 같이 OpenICE-C1605 애뮬레이터 장비와 연결된 S3CC9P9 스마트 카드 개발 보드와 스마트카드 리더기를 이용하여 실험하였다. 실험 결과인 애플릿 수행 시간의 측정 방법은 데스크 탑에서 카드 리더기를 통해 자바 카드에 APDU를 전송하고 응답 받는 시간을 측정한 것이다.

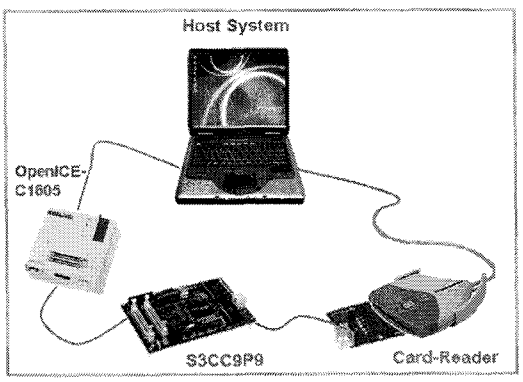


그림 11. 실험 환경

본 논문에서 제시한 다중 트랜잭션 버퍼를 이용하는 트랜잭션 기법에 대한 실험 결과를 기존의 트랜잭션 기법을 사용하는 자바 카드에서의 애플릿 설치부터 메소드 실행까지의 수행 시간과 비교하였다. 그리고 측정에 이용된 시간 단위는 ms(1/1000s)를 사용한다.

표 4에 제시된 결과에서 보는 바와 같이 본 논문에서 제시한 다중 트랜잭션 버퍼를 이용하여 트랜잭션을 처리하는 방법이 자바 카드에 프로그램을 적재하고 실행하는데 약 33% 정도 기존 방식보다 효과적인 것으로 나타났다.

본 논문에서 제시한 다중 트랜잭션 버퍼를 이용하여 트랜잭션을 처리하는 방법을 통한 자바 카드의 실험 결과를 살펴보면 기존의 방법보다 응용 프로그램의 수행 시간이 감소하였고, 그에 따라 자바 카드의 전체적인 수행 성능이 향상되었음을 알 수 있다.

이는 연속적으로 트랜잭션을 제어하고 처리할 수 있도록 트랜잭션 규칙을 규정하고, 트랜잭션 버퍼를 두 개로 나누어 각각의 트랜잭션 버퍼에서 트랜잭션을 수행할 수 있도록 변경하였기 때문에 트랜잭션을 대기하는 시간이 감소함으로써 자바 카드의 성능이 향상된 것이다. 그리고 테스트 샘플에 있는 메소드들이 트랜잭션을 이용하여 데이터를 갱신하는 빈도가 많으면 많아질수록 더욱 효과 적일 것이다.

본 논문에서 제안한 알고리즘의 단점으로는 트랜잭션 버퍼의 크기가 증가하고 두 개로 분리되었기 때문에 카드에 전원이 차단되고 재인가 될 때 데이터를 복구하는 과정에서 기존의 자바 카드보다 시간이 더 소비하게 된다는 문제점이 있다.

표 4. JavaLoyalty 애플릿의 읽기와 기록 속도(단위 : ms)

방식 / 애플릿	기존 방식	제안 방식	감소 비율
ChannelDemo	76,140	50,740	33%
JavaLoyalty	72,703	49,532	32%
JavaPurse	232,109	150,314	35%
ObjDelete	159,420	96,014	40%
PackageA	95,530	56,345	38%
PackageB	74,859	48,405	35%
PackageC	32,734	24,922	24%
PhotoCard	64,608	43,031	33%
RMIDemo	57,328	39,609	33%
평균	96,159	62,101	33%

5. 결론 및 기대효과

스마트카드 개발 초기 환경과 달리 메모리 용량의 증가 및 지속적인 전원 공급과 같은 하드웨어와 환경적인 요인의 변화에 따라 스마트카드도 개선되어야 한다. 만약 자바 카드 및 SIM카드 사용자가 인증 어플리케이션을 다운로드 받고 설치하기 위해 많은 시간을 소비한다면 어플리케이션 설치를 중단해 버리는 일이 빈번히 발생할 것이다. 자바 카드가 EEPROM을 사용하는 것은 전원이 차단되어도 데이터를 유지하기 위한 목적이었지만, 이는 자바 카드의 수행 속도를 저하시키는 원인이 되었다.

본 논문에서는 자바 카드에서 트랜잭션을 빠르게 처리하기 위해 다중 트랜잭션 버퍼를 이용할 수 있는 트랜잭션 기법을 연구하였다. 이는 자바 카드에서 데이터 추가나 삭제, 변경되는 모든 데이터를 EEPROM의 트랜잭션 버퍼에 기록하고 있고, 그 데이터 기록 빈도가 전체 EEPROM의 70% 이상 트랜잭션 버퍼에서 발생하는데 비해 그 기록 방법이 단일 트랜잭션 기법만을 이용함으로써 자바 카드의 수행 속도를 저하시키는데 영향을 주고 있다는 점을 개선하기 위한 것이다. 그리고 다중 트랜잭션 버퍼에서 트랜잭션을 수행할 수 있도록 이중 잠금(Double Lock) 규칙을 규정하고, 이 규칙에 따라 트랜잭션을 처리함으로써 트랜잭션 처리 성능을 개선시킬 수 있었고, 또한 자바 카드의 수행 성능을 향상 시킬 수 있었다. 이러한 연구 결과를 통해 다목적으로 사용되는 스마트카드 환경에 보다 빠르고 안정적인 자바 카드 기술을 제공할 수 있고, 적은 하드웨어 자원을 기반으로 하고 트랜잭션을 제공하는 시스템 환경에서 데이터의 안전성을 높이고, 응용 프로그램을 수행하는 속도를 개선할 수 있는 방안을 제시할 수 있을 것이다.

참 고 문 헌

[1] 김영진, 전용성, 전성익, 정교일, "Java Card Platform을 내장한 Smart Card의 구현," 정보과학회논문지, Vol.19, No.8, pp. 33-43, 2001.

[2] Sun Microsystems, Inc., *The Java Card™*

2.2.1 Runtime Environment(JCRE) Specification, SUN, 2003.

[3] Zhiqun Chen, *Java Card Technology for Smart Cards*, Addison Wesley, 2000.

[4] Wolfgang Rankl and Wolfgang Effing, *Smart Card Handbook*, John Wiley & Sons, 2003.

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

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

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

[8] Sun Microsystems, Inc., *The Java Card™ 2.2 Runtime Environment Specification*, SUN, 2003.

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

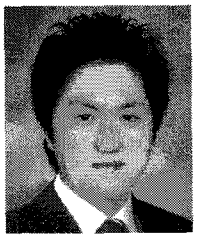
[10] Yoon-sim Yang, Won-Ho Choi, Min-Sik Jin, Cheul-Jun Hwang, and Min-Soo Jung, "An Advanced Java Card System Architecture for Smart Card Based on Large RAM Memory," *2006 International Conference on Hybrid Information Technology*, ICHIT2006, Vol.2, pp. 646-650, 2006.

[11] 이동욱, 황철준, 양윤심, 정민수, "전원 공급이 지속적인 대용량 스마트카드를 위한 JCVM 시스템 구조 개선," 멀티미디어학회논문지, Vol. 10, No.8, pp. 1029-1038, 2007.

[12] 오세원, 최원호, 정민수, "선반입 LRU-OBL 버퍼 기법을 적용한 자바 카드 프로그램 적재 및 실행 속도 개선에 관한 연구," 멀티미디어학회논문지, Vol.10, No.9, pp. 1197-1208, 2007.

[13] M. Oestreicher and K. Ksheeradbhi, "object lifetimes in javacard," *Proc. Usenix Work-shop Smart Card Technology*, Usenix Assoc, Berkeley, Calif., pp. 129-137, 1999.

[14] SUN Korea Developer Network, http://kr.sun.com/developers, 2003.

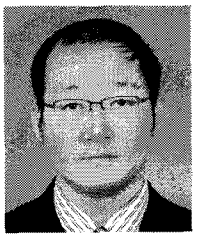


노 태 헌

2007년 2월 경남대학교 컴퓨터공학과 학사

2007년 3월~현재 경남대학교 컴퓨터공학과 석사 과정

관심분야 : Java Technology, Java Card, Embedded



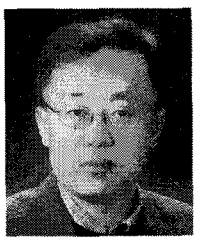
이 동 욱

2006년 2월 경남대학교 컴퓨터공학과 학사

2008년 2월 경남대학교 컴퓨터공학과 석사

2008년 3월~현재 JRSC

관심분야 : Java Technology, Java Card, Home-Network



정 민 수

1986년 서울대학교 컴퓨터공학과 학사

1988년 한국과학기술원 전산학과 석사

1994년 한국과학기술원 전산학과 박사

1990년~현재 경남대학교 컴퓨터공학부 교수

관심분야 : Java Technology, Java Machine, Home-Networking