자바카드에서의 멀티 트랜잭션 처리

정임영[0], 전성익, 정교일
한국전자통신연구원
(imyoung[0], sijun, kyoil)@etri.re.kr

A Multi-Transaction Management in Java Card

Im-Young Jung[0] Sung-Ik Jun Kyo-il Chung
Electronics and Telecommunications Research Institute

요 약

자바 스마트 카드에서의 트랜잭션 관리는 현재, 중첩된 트랜잭션 처리와 두개 이상의 동시적인 트랜잭션 처리를 고려하지 않고 있다. 그러나, 자바카드는 현재 카드상의 응용 애플릿의 멀티 셀렉션이 가능하고 멀티 통신 채널을 통한 다중 통신이 가능해졌다. 단일 트랜잭션만을 허용함으로 제약이 있었던 두개 이상의 애플릿에 걸친 트랜잭션 처리나 한 번의 통신으로 트랜잭션의 수명을 정하고 있었던 부분에서의 개선요구는 꾸준히 있어왔다. 동시에 처리할 수 있는 트랜잭션의 수를 늘리는 것은 카드에서 메모리와 처리능력을 꾸준히 높이고 있는 현 상황에서는 당연한 요구이다. 본 논의에서는 현재 ETRI에서 개발 중인 차세대 IC카드에의 멀티 트랜잭션 처리로 스마트카드에서의 기존 트랜잭션 처리를 확장해보고자 한다.

1. 서론

오직 한번에 하나의 트랜잭션 처리만을 허용할 경우, 카드상의 여러 응용 애플릿을 선택하고 카드가 여러 클라이언트와의 통신이 동시에 가능하도록 논리적으로 여러 개의 통신채널을 열 때 발생되는 트랜잭션 처리 요구는 순서를 정해서 먼저의 것이 끝나면 다음 것을 수행하도록 해야만 하는 제약이 있다. 또한 한 트랜잭션의 수명은 대개가 너무 짧고 길어야 애플릿이 선택되어 이 선택이 해제될 때까지이다. 현재 자바카드의 경우 멀티 통신 채널과 여러 애플릿을 동시에 사용할 수 있도록 스마트 카드 자체의 자원을 늘리고 개선시킴으로서 충분히 두 단의 트랜잭션 처리나 동시에 최대 두 개정도의 트랜잭션 병행 처리가 가능한 환경이 되었다[1][2][3]. 그러나, 아직까지는 오직 단위 트랜잭션[1]으로 한번에 한 트랜잭션 처리만 가능하도록 되어있다. 이는 SUN사에서 배포된 Java Card 2.1에서 Java Card 2.2로의 개선된 버전이 나오면서도 일관되게 유지된 부분이다[1][2][3][4][5][6]. 따라서, 본 논의는 스마트 카드에서 최고 두 단의 중첩도를 가지는 트랜잭션 처리와 동시에 최고 두 개까지의 트랜잭션 병행처리를 제안한다. 모델을 세운 것은 현재 본 팀[2]이 개발 중인 차세대 IC Card(NGIC card-Next Generation Integrated Circuit card)이므로, 먼저 NGIC 카드를 소개하고 본 논의를 전개한다.

[1] 중첩되지 않은 트랜잭션으로 하나의 트랜잭션 요구를 단일 처리하는 것을 말한다.

[2] 한국전자통신연구원(ETRI) 정보보호연구본부 IC카드연구팀

2. 차세대 IC card(NGIC Card-Next Generation Integrated Circuit card)

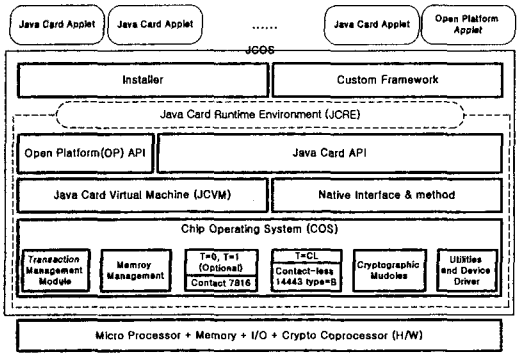

그림 1. 차세대 IC Card 구조

차세대 IC카드는 SUN사와의 독점 라이센스 계약으로 현재 ETRI에서 개발되고 있는 Java Smart Card로서 ISO 7816, 14443 표준을 따른다. 그림 1에서 보듯이 응용 애플릿이 올라가는 부분아래 Java Card Virtual Machine(JCVM)과 Chip Operating System(COS) 부분을 합쳐 JCOS라 불리는 부분이 있고 이 아래 하드웨어 부분으로 NGIC 카드는 구성된다. 그리고, JCVM, Java Card API, 또, 주변의 이들을 지지하는 서비스들로 이루어진 Java Card Runtime Environment(JCRE)를 따로 명명하기도 한다. 시간이 걸리는 암호처리 부분을 하드웨어로 구현해서 효율을 높였는데, RSA, ECC, ECDSA, DSA 등의 다양한 암호

모듈을 수용한다. 본 논의에 관계된 트랜잭션 관리 부분을 담당하는 곳은 Transaction Management Module(TMM)의 트랜잭션 매니저 부분이다.

3. 트랜잭션 프로토콜

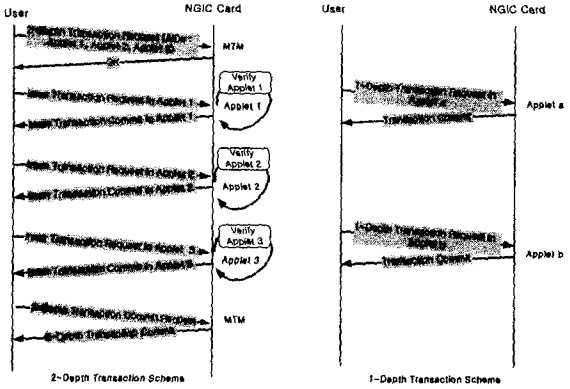

그림 2. 자바카드의 멀티 트랜잭션 프로토콜

두 단의 중첩 트랜잭션을 실행하기 위해서 이를 특별히 관리하는 애플릿으로 Muti-Transaction Manager(MTM)를 둔다. 이는 카드에 이미 다운로드 되어있어 언제든지 선택 가능하다고 본다. 카드의 클라이언트가 중첩 트랜잭션을 요구하는 것은 두 단의 중첩 트랜잭션의 바깥단 부분은 MTM을 선택하는 것이고 이 안쪽 단에 포함이 되는 단위 트랜잭션들이 수행이 될 applet명단을 전해줌으로 이후 이 applet들이 선택되어 트랜잭션이 수행이 될 때는 중첩 트랜잭션이 수행 중이라는 것을 알 수 있다. 중첩 트랜잭션의 내부에 속하게 되는 단위 트랜잭션의 최대 수는 트랜잭션 버퍼의 크기에 따라 조절이 가능하다. 이렇게 카드의 외부에서의 요청으로 중첩 트랜잭션 요구가 올 수 있지만 JCRE에서의 필요에 의해서도 중첩 트랜잭션을 호출할 수 있다. 그리고, MTM에게 중첩 트랜잭션을 커밋한다는 요구가 오면 이 MTM을 선택해지 하는 것으로 이 트랜잭션은 커밋된다. 물론 중첩 트랜잭션을 구성하는 각각의 단위 트랜잭션이 다 커밋이 되어야 전체를 커밋할 수 있다. 아니면 중첩 트랜잭션 전체처리를 무효화 시킨다.

멀티 트랜잭션 처리는 그림 2에서 보듯이 이미 두 단의 중첩 트랜잭션이 수행 중일 때 이와 상관없는 단위 트랜잭션이 병행해서 수행 가능하다는 것을 보여준다. 최대 두 개의 트랜잭션이 수행 가능한 것은 각 트랜잭션이 번호로서 구분이 되고 트랜잭션 버퍼가 링크로 연결된 리스트 형태로 운영되기 때문이다. 트랜잭션 버퍼에 관한 것은 다음 장에서 구체적으로 논한다.

트랜잭션의 로깅 기법은 기존 자바카드 트랜잭션을 대상으로 제안된 Old-Value Logging이나 New-Value Logging[7], 어떤 것이든 본 논의의 트랜잭션 프로토콜에는 영향을 주지 않는다.

4. 트랜잭션 버퍼

그림 3과 4는 각각 스마트카드의 트랜잭션 버퍼의 운영과 트랜잭션 버퍼를 구성하는 EEPROM의 페이지 구성을 나타낸다.

4.1 EEPROM

보통의 스마트 카드는 영구메모리를 EEPROM이나 플래쉬메모리로 주로 쓴다. NGIC Card는 EEPROM을 사용한다. EEPROM은 RAM에 비해 비휘발성인 특징과 함께 쓰기에 시간이 많이 걸리는 단점이 있다[8]. 그리고, 쓰기, 읽기에서 한 번에 최대로 접근할 수 있는 양으로 페이지를 언급하는데 이 크기는 메모리 제조사마다 다르고 4B, 8B, 16B, 32B, 64B 등의 순으로 달라질 수 있다. 이 페이지크기로 심하게 제약을 받는 것은 쓰기 때인데, 두 페이지 사이에 걸쳐서 쓰려면 두 페이지 접근사이에 기다려야 하는 블록시간이 존재한다. 이는 같은 페이지 내에서도 한 지점에의 쓰기 이후 갭시간이 15usec정도 이상이 되면 또 나타나는 부분이다.[9][10] 블록시간과 갭시간도 메모리의 제조사마다 수치적으로는 차이가 있을 수 있으나 블록시간이 거의 10msec내외로서 스마트카드 개발자는 상당히 신경을 써야 하는 부분이다[9][10]. 이런 EEPROM의 특성으로 본 논의는 EEPROM에 놓이게 되는 트랜잭션 버퍼 부분도 몇 개의 페이지로 구성이 된다고 보고, 페이지 단위로의 트랜잭션 버퍼 관리를 제안한다.

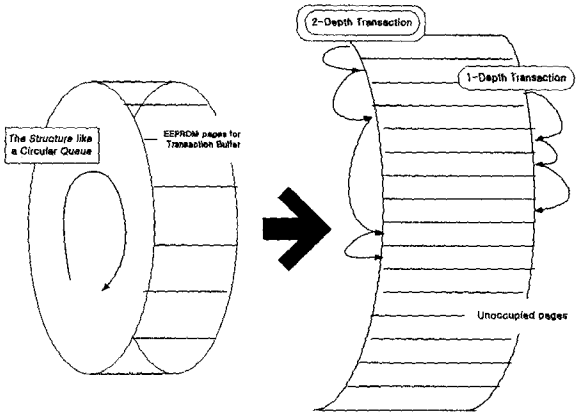

그림 3. 트랜잭션 버퍼의 구조

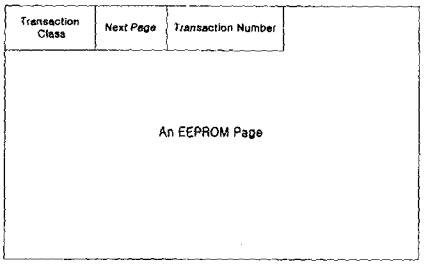

그림 4. 트랜잭션 버퍼를 구성하는 EEPROM 페이지

4.2 트랜잭션 버퍼의 운영

트랜잭션 버퍼는 그림 3에서와 같이 원형 큐형태로 운영이 된다. 그러나 큐처럼 반드시 먼저 쓰인 부분이 나중에 쓰이게 되는 것은 아니고, 다만 쓰이는 방향이 시계방향인 한 방향으로 운영이 된다

는 것이다. 앞 절에서 소개되었듯이 트랜잭션 버퍼는 여러 개의 페이지로 구성이 되고 이들 페이지를 링크로 엮어 만든 리스트가 현재 진행되고 있는 트랜잭션의 로그들을 기록하는 부분이 된다. 이 리스트가 최대 두개 존재하게 함으로써 최대 두개의 트랜잭션이 동시에 진행이 될 수 있는 것이다. 두 단의 중첩 트랜잭션의 경우도 하나의 페이지 리스트로 표현이 되어 그 속에 속하는 단위 트랜잭션의 로그들은 이 리스트 안에 차례로 기록되게 된다. 동시에 진행되는 두 개의 트랜잭션을 구분하기 위해서는 트랜잭션 마다 이를 식별할 수 있는 일련 번호를 준다. 이는 단조증가 수로 구분을 하는데 중첩 트랜잭션과 단위 트랜잭션 각각을 위한 하나씩의 수 리스트가 있다. 이전에 끝난 트랜잭션이 사용했던 수 다음의 것을 새로운 트랜잭션에 부여하게 된다. 그런데, 이 수치들도 최대치 이후는 다시 최소치를 가지게 되므로 트랜잭션 중 전원차단의 경우 전원차단 이전에 진행 중이던 트랜잭션 로그 리스트를 찾기 위해 트랜잭션 버퍼를 구성하는 페이지 수보다 2이상 많은 트랜잭션 구분 수를 준비한다[3]. 그리고, 이런 트랜잭션 로그 리스트의 정보를 담기 위해 각 트랜잭션 버퍼를 구성하는 페이지들은 헤더필드 부분을 그림 4에서와 같이 갖게 된다. TRANSACTION CLASS 부분은 중첩 트랜잭션인지 아닌지와 이 페이지가 나타내는 것이 트랜잭션의 끝인지에 따라 3가지 상태-1-depth transaction, 2-depth transaction, transaction end-를 가지게 되고 0값도 가질 수 있는데, 이는 현재 쓰이지 않고 있는 페이지란 것을 나타낸다. 그리고, 되도록 고른 EEPROM의 사용을 위해 처음 쓰이는 페이지나 가장 오래 안쓰인 페이지를 골라서 각 트랜잭션의 로그들을 위해 할당을 하게 된다. 이를 위해 트랜잭션이 끝나면 쓰인 페이지 들을 회수하게 되는데 이때 회수한 페이지들도 리스트로 엮어서 이들이 사용된 선후 관계를 나타내게 된다.

트랜잭션 중간에 문제가 생겨서 이를 트랜잭션을 무효화 시킬 경우는 중첩 트랜잭션의 경우는 단위 트랜잭션 별로 부분 커밋을 허용하지는 않는다. 이에 대한 구체적인 알고리즘은 로깅 방법에 따라 달라지는 부분이나, 전원이 갑자기 끊기는 경우에 대한 트랜잭션 처리의 복구부분은 복구를 위한 모든 정보는 EEPROM의 트랜잭션 버퍼에서 각 페이지의 헤더로 확인이 가능하기 때문에 보장이 되고, 이 정보들은 트랜잭션 수행중에는 RAM에 테이블로 만들어 이들에 대한 빠른 접근이 가능하도록 한다.

5. 결론 및 향후 과제

이상에서 자바카드에서 기존에는 언급되지 않았던 멀티 트랜잭션 처리를 제안하였다. 그 간 카드의 자원상의 제약만 생각해서 단순한 기능처리만을 고려했었다면, 멀티 응용 애플릿 서비스를 언급하고 있고 또, 이는 커진 카드의 메모리 용량 처리능력에 힘입은 것으로 볼 때, 현재는 이를 뒷받침 할 수 있는 카드 운영체제의 역할도 확장시켜 볼 필요가 있다.

이후는 현재 두개까지의 트랜잭션과 두 단의 중첩 트랜잭션 만을 허용이 되는 부분을 좀 더 확장하는 부분과 효율적인 로깅방법의 연계 부분, 트랜잭션의 각 페이지의 헤더필드부분을 줄일 수 있는 방법, 또, 트랜잭션과 관련해서 EEPROM의 쓰기 시간을 줄일 수 있는 부분에 대한 연구가 이어질 것이다.

6. 참고문헌

[1] Sun Microsystems, Inc.: "*Java Card™ 2.2 Application Programming Interface*", Beta Release, Feb, 2002.

[2] Sun Microsystems, Inc.: "*Java Card™ 2.2 Runtime Environment Specification*", Jun, 2002

[3] Sun Microsystems, Inc.: "*Java Card™ 2.2 Virtual Machine Specification*", Beta Release, Feb, 2002.

[4] Sun MicroSystems, Inc.: " *Java Card™ 2.1 Application Programming Interface*" , Final Revision, June 7, 1999.

[5] Sun Microsystems, Inc.: " *Java Card™ 2.1 Runtime Environment Specification*" , Final Revision, June 7, 1999.

[6] Sun Microsystems, Inc.: " *Java Card™ 2.1 Virtual Machine Specification*" , Final Revision, June 7, 1999.

[7] Marcus Oestreicher: " *Transactions in Java Card*" , In 15th Annual Computer Security Applications Conference (ACSAC'99), pages 291-298. IEEE, 1999.

[8] Zhiqun Chen: " *Java Card Technology for Smart Cards*" , Addison-Wesley, June 2000

[9] Atmel: " *AT28BV256 Specification*" , http://www.atmel.com, 1999.

[10] Atmel: " *AT28LV010 Specification*" , http://www.atmel.com, 1998.

[11] Axel Meckenstock, Detlef Zimmer and Rainer Unland, " *Controlling Cooperation and Recovery in Nested Transactions*" , CADLAB Report 12/94, 1994.

[12] Marek Rusinkiewicz and Amit Sheth, " *Specification and execution of transactional workflows*" , In W. Kim, editor, Modern Database Systems : The Object Model, Interoperability, and Beyond. ACM Press, Cambridge, Messachussetts, 1994.

[3] 가장 최근에 수행이 되던 트랜잭션 구분 수는 오름차순으로 된 수 중의 논리적으로 가장 큰 수가 될 것인데 이 끝수를 찾기 위해 트랜잭션 버퍼를 구성하는 페이지 수보다 2이상 많은 구분 수를 준비한다. 트랜잭션 버퍼의 페이지들을 그들의 링크에 따라 검색하다보면 연속된 트랜잭션 번호들을 발견할 수 있을 것이고 이 연속된 번호들 중 가장 뒷 번호가-늘, 트랜잭션 번호할당은 연속번호로 하게되어 있기때문에 이 번호 다음은 연속이 되지않고 다름 번호로 뛰게되는 부분이 있을 것이다.- 전원이 나가기 전에 수행되던 트랜잭션 번호이다.