자바카드에서 Post-issuance API에 관한 연구

이정우[0], 전성익
한국전자통신연구원
(jeow7, sijun)@etri.re.kr

A Study on Post-issuance API in Java Card

Jeong-Woo Lee[0], Sung-Ik Jun
IC Card Research Team, Electronics and Telecommunications Research Institute

요 약

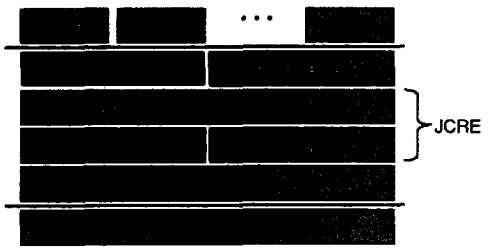
자바카드의 큰 특징 중의 하나는 바로 필요한 응용프로그램이나 Library를 카드가 발급된 후에도 카드 상에 업로드 시켜 이용할 수 있다는 것이다. 하지만 카드를 발급할 때 Mask 형태로 ROM 상에 올려져 있는 API의 경우 수정하는 것이 불가능하며 또한 단순히 새로운 API를 추가한다 하더라도 기존에 이를 사용하던 애플릿의 변경이 불가피 하게 된다. 본 논문에서는 발급 후 API를 추가하는 예를 보이고 이런 문제점을 개선한 모델을 제시하여 보다 유연한 API 개발 환경을 제공하게 한다.

1. 서 론

현재 금융, 통신 시장을 기반으로 스마트 카드의 활용이 급속히 늘어나고 있다. 스마트 카드 자체도 발전을 거듭해 가며 32bit 이상의 처리성능을 가진 CPU를 가지고 더 많은 기능을 수행해 나가고 있는 실정이다. 현재 많은 스마트 카드들은 Java의 장점을 이어받은 JavaCard OS를 바탕으로 개발되고 있다. 여기에 Open Platform을 수용하여 많은 산업체에서 응용될 수 있는 스마트 카드가 만들어지고 있다. 이런 스마트 카드들은 특히 발급 후 응용프로그램을 추가하여 사용할 수 있기 때문에 하나의 카드로 여러가지 응용서비스를 받을 수 있다. Open Platform을 수용한 카드의 경우 카드상의 응용프로그램에 해당하는 애플릿의 자유로운 추가 삭제 기능을 가지고 있기 때문에 카드 사용자는 필요한 서비스만 받아서 사용하는 것이 가능하다. 이렇게 카드상에서 사용되는 응용서비스 프로그램의 경우 읽고 쓰기가 가능한 EEPROM상에 기록되기 때문에 추가 삭제가 가능한 것이다. 하지만 OS와 API를 비롯한 JCRE(Java Card Runtime Environment) 환경은 카드 발급시 Mask 형태로 ROM에 저장되기 때문에 더 이상 수정하는 것이 불가능하다. 단지 새롭게 추가된 Library 부분을 애플릿처럼 카드에 올려 사용할 수 있을 뿐이다. 이것 역시 기존에 사용되던 API의 오류수정이나 성능개선시 이를 사용하는 기존에 존재하던 애플릿이나 API의 경우 예전의 것을 사용할 수 밖에 없다. 본 논문에서는 이러한 단점을 개선할 수 있도록 EEPROM 상에 새로운 기능의 라이브러리 패키지를 추가한 후 이를 애플릿에서 이용하는 예를 보이고 이를 좀 더 확장한 모델을 제시함으로써 ROM API에 대한 단점을 근본적인 해결할 수 있는 방법을 모색해 보고자 한다.

2. 관련 연구

2.1 Java Card



[그림 1] 자바카드의 구조[1]

자바카드는 스마트카드에 자바카드 OS를 운영체제로 사용한 스마트카드를 일컫는 말이다. 자바카드 기술은 자바 프로그램 언어로 쓰여진 프로그램을 스마트 카드 및 기타 리소스 제약 디바이스 상에서 동작할 수 있도록 만든다. 스마트 카드의 작은 메모리 제약 때문에 자바카드 플랫폼은 스마트 카드에 쓰일 수 있도록 객체지향의 능력을 보존하면서 자바 언어 중 선택된 서브셋을 제공한다. 자바카드의 운영환경의 중요한 특성은 스마트 카드 시스템과 응용 사이에 분명한 경계를 제공하는 것이다. 따라서 각종 응용들은 스마트 카드 시스템의 복잡성과 자세한 사항을 알 필요 없이 스마트 카드 시스템 잘 정의된 고차원의 프로그래밍 인터페이스를 통하여 만들어진다.[2]

스마트 카드는 세가지 형태의 메모리를 가지고 있다.

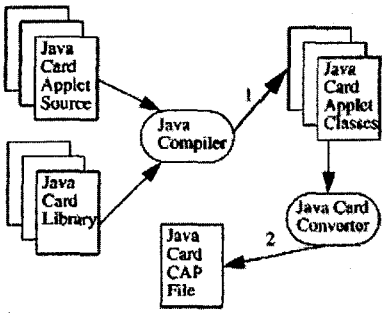
- ROM(Read Only Memory):카드에 OS같은 프로그램이나 영구 저장될 데이터를 저장하는데 사용, 변경 불가
- EEPROM(Electrical Erasable Programmable read-only memory) : 일반PC에 사용되는 하드디스크와 같은

역할, 파워가 꺼진 후에도 데이터가 남아 있어 쓰거나 변경하는 것이 가능

- RAM(random access memory) : 일시적인 데이터들을 저장하거나 변경하는데 쓰이는 임시 작업공간

초기 발급시 JCRE 환경은 Mask Generator를 통하여 Mask 형태로 ROM에 저장되어 더 이상 변경이 불가하게 되며 이후 추가로 저장되어지는 응용 애플릿들은 EEPROM상에 저장되어진다. 자바카드 2.2 버전부터는 추가로 저장된 애플릿의 삭제기능이 추가되었기 때문에 불필요한 응용을 제거할 수 있게 되었다.

응용애플릿은 카드 발급시 기본 애플릿 형태로 저장되어 발급되기도 하지만 추후 필요한 응용서비스에 따라 이러한 기능을 수행하는 애플릿을 추가할 수 있다. 이러한 애플릿은 컴파일 후 컨버터를 거쳐 자바카드상에서 쓰이는 CAP(Converted APplet) 파일 형태로 카드에 저장되어진다. 카드에 저장된 애플릿 사이에는 애플릿 방화벽이 존재하여 각 애플릿 간의 독립성을 보장하고 있다. 애플릿 간의 통신은 공유인터페이스를 통하여 가능하다.

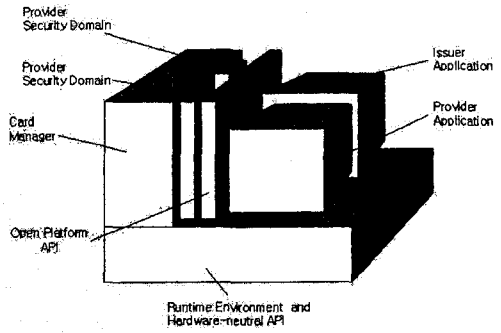


[그림 2] CAP 파일의 생성과정[5]

2.2 Open Platform

OP는 1998년, 다중 응용 프로그램(Multi-application)용 chip OS 및 CPU상에서 동일한 응용 프로그램을 사용한 스마트 카드에 대한 안전한 프로그램 적재 및 설치, 관리를 위해 Visa에 의해 개발된 플랫폼으로 VOP(Visa OP)로 공표되어 사용되다가 현재 GlobalPlatform이라는 기관에 의해 관리되고 있으며 OP로 명칭이 바뀌면서 공개되었다.[3] OP는 출발은 Java Card를 대상으로 하여 개발이 되었으며 현재는 사실상 다중 응용 프로그램용 스마트 카드의 관리에 대한 표준으로 자리 매김하고 있다.[4] 그림 3은 OP의 주요 구성요소를 보여주는 그림으로써 그 주요기능은 다음과 같다.[6]

- Card Manager : 카드 발급자(card issuer)의 카드상의 대리자 역할을 하며 응용 프로그램을 위한 완전한 수행 환경을 관리하며 카드 전체 시스템 및 보안관리 클래스이다.
- SecurityDomains : 응용 프로그램 공급자(Application Provider)의 카드상의 대리자 역할을 하며 특정한 AP에 의해 제공되는 모든 응용 프로그램들에 대한 암호 서비스를 제공하는 일종의 응용 프로그램이다.
- OP API : 응용 프로그램으로 하여금 Card Manager에 의해 감독되는 카드 운용 서비스에 대한 접근을 가능하게 하는 프로그램 인터페이스이며 Card Manager가 카드 내용을 관리하도록 지원하는데, 실시간 수행환경에 독립적으로 정의되어 있다.



[그림 3] Open Platform의 주요 구성[6]

이렇게 OP를 수용한 자바카드는 다양한 애플릿들을 카드 안에 저장함으로써 여러가지 기능을 가진 스마트 카드로 쓰일 수 있게 된다. 특히 Security Domain 등의 사용으로 보안적인 측면에서 훨씬 우수한 기능을 가지게 된다. 특히 Card Manager에 의한 응용애플릿의 관리로 발급 후 애플릿의 추가 삭제가 Java Card의 경우보다 훨씬 수월하게 되어 있다. 추가적으로 확장하려는 API의 경우 애플릿의 경우와 마찬가지로 CAP파일 형태로 카드상에 로딩시켜 사용할 수 있다. 이는 자바카드 시스템 개발의 측면에서 보면 새로운 API의 추가가 필요한 경우 카드를 다시 발급하지 않고 재사용하게 하는 방법이 되며 애플릿 개발자의 경우도 마찬가지로 비슷한 기능을 가진 함수를 애플릿마다 구현하여 사용하기 보다는 라이브러리 형태로 묶어 새로운 API형태로 카드에 저장하여 사용하게 되면 훨씬 효율적으로 애플릿을 개발할 수 있는 환경을 제공 받게 된다. 다음은 간단한 API를 추가하여 사용하는 예를 통해 좀 더 확장된 모델을 제시할 것이다.

3. Library Package 구현

먼저 발급 후 추가되는 API의 형태로 test 패키지를 구현한 뒤 애플릿 옵션을 사용하지 않고 CAP파일 형태로 변환하여 OP기능이 탑재된 Gemplus사의 GemXpresso RAD211 카드에 로딩시킨다. testlib 패키지의 add() Method는 입력값을 2 증가시켜 리턴하는 기능을 수행한다. 아래 애플릿의 getBalance() Method내에서는 add()를 불러와 그 결과를 리턴하게 된다.

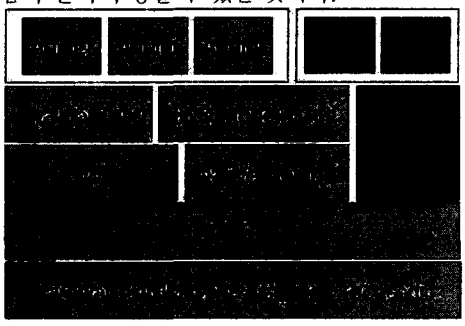
```
// Testlib 클래스를 호출하여 사용
package com.gemplus.examples.oppurse;
import testlib.*;
.....
balance = (short)(lib.add((byte)balance));
// get balance() method 내
```

이렇게 만들어진 애플릿을 역시 CAP 파일로 만들어서 카드 상에 다운로드 시킨다. 아래에서 이렇게 만들어진 애플릿을 실행하여 get balance 명령을 주었을 때 2씩 증가된 결과가 나오는 모습을 보여주고 있다.

```
================== get balance
CMD: get balance
-> 00 30 00 00 02
<- 00 13 90 00
================== get balance //결과가 2씩 증가
CMD: get balance
-> 00 30 00 00 02
<- 00 15 90 00
```

4. API Agent

위에 보여진 것처럼 이미 발급된 카드에 새롭게 추가되는 API의 경우 애플릿처럼 CAP 파일 형태로 로딩시켜 사용하면 된다. 그러나 기존에 사용되던 API의 경우 ROM 안에 저장되어 있기 때문에 변경이 불가능하다. 이를 개선하기 위한 방법으로 변경된 API를 로딩시킨 다음, 이후 추가되는 애플릿의 경우 변경된 API를 import시켜 사용하면 변경된 API를 사용할 수 있게 된다. 그러나 이 경우 역시 변경되기 이전에 올려진 애플릿이나 API에서는 기존 API를 사용할 수 밖에 없다. 이런 점을 보안하기 위해 전체적으로 API를 관리하고 기존의 API를 새로운 API로 경로를 변경시켜 주는 모듈이 필요할 것이다. 이런 역할을 하기 위해 API Agent라는 이름의 모듈이 추가된 모델을 다음과 같이 구성할 수 있을 것이다.



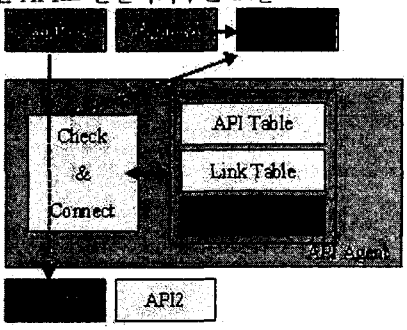
[그림 4] API Agent가 사용된 모델

여기에서는 OP의 Card Manager가 애플릿을 관리하는 것과 비슷하게 API에 대해서 API Agent를 따로 두어 사용하게 된다. 이 API Agent의 주요기능은 API 추가, 삭제 및 변경에 관한 내용을 감지하고 새로운 경로를 설정해 주는 것이다. 그림 4는 API Agent가 추가된 카드의 모델을 나타내고 있다.

기본적으로 SUN에서 제공하는 기본적인 API들에 대한 변경은 금지되어 있다. 따라서 실제로 이런 식으로 접근하여 사용하는 API들은 Java Card API의 연장선에서 사용하는 API들이 될 것이다. 암호API나 확장된 API와 같은 경우가 그런 예가 될 수 있다. API Agent 역시 이런 API들을 관리하는데 사용하게 된다. 또한 본 논문에서 제시하는 모델은 JCRE내의 API를 사용할 수 있도록 SUN에서 라이센스를 받은 Java Card 개발자나 산업용 API 개발자에게 한정되어 사용될 수 있다.

API Agent는 다음과 같은 부분이 필요하게 된다.

- Card 상에 사용되는 API package에 대한 정보
- Import, export 관련된 linking 정보
- 변경된 API package를 체크하는 모듈
- 변경된 API로 연결시켜주는 모듈



[그림 5] API Agent의 동작

기존의 문제점을 수정한 API package가 카드에 올려지면 API Agent는 API package 정보에서 변경된 package를 찾아 변경여부를 기록하게 된다. 애플릿(Applet1)에서 API(API1)를 사용하려고 할 때 API Manager는 변경된 부분을 사용하는지 체크하게 된다. 변경된 부분이 사용되려고 할 때 API Agent는 linking 정보를 변경하여 새롭게 수정된 API(API1')로 연결시켜주게 된다. 이로써 불가능했던 ROM상의 API를 수정하여 사용하는 것이 가능하게 되며 기존에 이 API를 사용하며 존재하던 API나 애플릿들 역시 따로 변경할 필요가 없게 된다. 또한 이후 새롭게 추가되는 애플릿(Applet2)은 API Agent의 관여없이 새로운 API(API1')를 직접 import 하여 사용한다. 이처럼 API Agent는 카드발급 후 시스템의 성능개선을 위한 파일패치와 같은 역할을 대신하게 되는 것이다.

5. 결론

지금까지 카드 발급 후 API를 추가하거나 수정하여 사용하기 위한 방법과 문제점을 생각해보고 이를 보완하고 좀더 확장된, API Agent가 추가된 모델을 제시해 보았다. 이는 API의 단순한 추가뿐만 아니라 ROM API의 수정을 가능케 하여 좀 더 유연한 API 개발환경을 제공하게 할 것이다. 또한 카드의 재사용을 높이는 결과를 가져올 것이다. 앞으로 좀 더 구체적인 API Agent의 구조와 함께 직접 API를 엑세스 하지 않고 ROM이 아닌 EEPROM 상에 적재되어 동작하기에 발생할 수 있는 속도 저하의 문제는 좀 더 연구해 볼 필요가 있을 것이다.

[참고 문헌]

[1] Joshua Susser and Judy Schwabe, "The Java Card™ Virtual Machine",JavaOne Conference, 1999.

[2] Zhiqun Chen, Java Card Technology for Smart Cards, Addison-Wesley, p.15~16, 2000.

[3] http://www.globalplatform.org/specifications2.asp.

[4] Michael Baentsch, et al., "Java Card-From Hype to Reality", IEEE Concurrency, pp.36-43, Oct.-Dec., 1999.

[5] Attali and T. Jensen (Eds.) Java Card 2000, pp. 114-120, LNCS 2041, 2001.

[6] GlobalPlatform, Inc., *Open Platform : Card Specification 2.0.1* , April 7, 2000.