객체지향 모델링에 기반한 스마트카드용 응용 프로그램 개발 기법의 제안

김시관+, 오득환+, 박인용+, 강오한°
+금오공과대학교 컴퓨터공학부, °안동대학교 컴퓨터교육과
+{sgkim, dhoh, iypark}@se.kumoh.ac.kr, °ohkang@andong.ac.kr

Study of Smart Card Applications Development Based On Object-Oriented Design

Si-Gwan Kim+, Duck-Hwan Oh+, Inyong Park+, Oh-Han Kang°
+Kumoh National Institute of Technology, °Andong National University

요 약

최근의 스마트카드는 여러 가지 응용 프로그램을 하나의 카드에 적재시켜 사용할 수 있으며 보안 기능이 뛰어난 점 등 여러 가지 장점 때문에 최근 인터넷의 빠른 보급과 더불어 스마트카드의 효용성이 점차 증가하고 있는 추세에 있다. 본 논문에서는 스마트카드의 업계 표준으로 자리잡고 있는 자바 카드를 중심으로 자바 카드의 주요 특성과 응용 프로그램 작성 기법을 객체지향 중심 기법을 채택한 설계 기법을 제안한다. 객체 지향 기법을 채택함으로써 향후 예상되는 기능 확장등에 적절히 대처할 수 있는등의 장점을 가지고 있다.

1. 서 론

MS(Magnetic Stripe)카드의 사용이 점차 보편화되고, 이에 따른 여러 문제점들이 대두됨에 따라 보다 안전하고 다양한 기능을 수행할 수 있는 대체수단이 필요하게 되었다. 현재의 스마트 카드는 1968년 Jergen Dethloff가 마이크로프로세서의 휴대 수단으로써 IC카드에 대한 개념을 제안하고, 이를 1974년 프랑스의 Roland Moreno가 최초 형상화하여 탄생하였다. 1990년대에 접어들면서 스마트 카드 산업은 컴퓨터를 이용한 인터넷 사용의 급증과 정보통신 환경의 변화 등으로 빠르게 성장하고 있다. 스마트 카드의 활용은 통신, 금융, 교통 그리고 최근에는 전자상거래 등 여러 분야에 걸쳐 응용되고 있는 추세다.

자바 언어는 바이트코드를 수행하는 가상기계로 인해 이식성과 보안 응용 프로그램에 적합한 환경을 제공하고 있다. 자바 환경을 스마트 카드에 접목하려는 시도가 미국의 선마이크로시스템즈[1]에 의해 시도되었으며 자원을 많이 소모하는 자바 가상 기계의 한계를 극복한 자바 고유의 기능을 축약된 형식으로 제공하는 자바카드 가상기계(JCVM)를 제공함으로써 스마트카드 환경에서도 자바 언어가 사용가능하게 되었다. 자바 카드 표준은 자바카드가상기계, 응용프로그램인터페이스와 자바카드실행환경(JCRE)을 제공함으로써 자바 언어로써 스마트카드 응용 프로그램의 작성이 가능하게 되었다.

자바카드 환경에서 응용 프로그램을 작성한다는 것은 JCRE, 카드 애플릿, 터미널 프로그램등 여러 가지 요소가 필요하기 때문에 모델링 기법을 사용한다면 시스템을 설계하는데 많은 도움을 받을 수 있게 된다. OMG에 의해서 표준화된 UML 언어는 객체지향 설계에 도구[2]로써 시스템을 이해하기가 수월해 진다. Vanderwalle[3]과 Martin[4]에서는 단순한 응용 프로그램에 대해서 객체지향 기법을 채택하여 설계한 예를 보여주고 있다. 본 논문에서는 애플릿 뿐만 아니라 JCRE등 관련 시스템의 전반적인 설계 기법을 제안한다.

본 연구는 한국과학재단 신진교수연구과제 지원(2002-003-D00321)으로 수행되었음.

본 논문에서는 스마트카드의 업계 표준화가 되고 있는 자바카드를 중심으로 객체지향 기법을 채택하여 자바카드 응용 프로그램과 전반적인 시스템의 동작 상태의 작성 기법을 제안하고자 한다. 본 논문은 2절에서는 자바카드의 전반적인 동작을 설명하고 3절에서는 스마트 카드의 업계 표준인 자바카드를 중심으로 객체 지향 기법을 채택한 설계 기법을 제안하고 4절에서는 결론을 제시한다.

2. 자바카드의 동작

본 절에서는 자바 카드의 기본적인 동작과 여러개의 응용 프로그램(애플릿)이 카드내에 존재할 때 발생되는 객체 공유 기법에 대하여 설명한다.

2.1 자바카드의 기본 개념 및 동작

자바카드 언어는 기본적으로 자바 언어의 특성 중 일부만을 지원한다. 예를 들어 boolean, byte, short 같은 비교적 작은 크기의 자료형만 지원하며 garbage collection, thread와 같은 기능은 지원하지 않는다. 기존의 JVM(Java Virtual Machine)과는 달리 JCVM(Java Card Virtual Machine)은 자바 카드 바이트코드 해석기가 카드 자체에 존재하는 on-card 부분과 자바 카드 변환기가 PC 혹은 WS에서 실행되는 off-card 부분의 두 부분으로 구성되어 있다.

변환기는 자바 패키지를 구성하는 클래스 파일들을 입력받아 2진 형태의 CAP 파일 형태로 출력을 한다. 이 CAP 파일이 카드내의 바이트코드 해석기에 의해서 처리가 된다. 자바카드 가상기계의 구조와 패키지의 변환과정이 다음 그림에 나타나 있다.

JCRE는 다중 응용프로그램(multi-application) 환경을 지원

한다. 즉, 여러 개의 애플릿이 하나의 자바카드에 함께 존재할 수 있으며, 애플릿은 여러 개의 인스턴스를 가질 수 있다. 예를 들어 wallet 애플릿이 U.S. 달러와 영국 파운드에 대해 각각 생성될 수 있다. JCRE는 자바카드내에서 실행되는 자바카드 시스템의 중요한 구성 요소로서 카드의 자원 관리, 통신, 애플릿 실행, 보안등을 담당하는 일종의 운영체제라고 할 수 있다.

JCVM은 바이트코드의 실행, 메모리 할당, 객체 관리, 보안 등을 담당하고 있다. 네이티브 메쏘드는 저수준의 통신, 메모리 관리, 암호화등을 담당하고 있다. 시스템 클래스는 JCRE의 운영체제에 해당된다. 시스템 클래스는 트랜잭션의 관리, 호스트 응용프로그램과 자바카드 애플릿간의 통신, 애플릿의 생성, 셀렉션 및 디셀렉션을 담당하고 있다. 태스크를 완료하기 위해서는 일반적으로 시스템 클래스는 네이티브 메소드를 호출하게 된다. 인스톨러는 카드 소지자에게 카드가 발행된 이후 필요한 소프트웨어와 애플릿을 안전하게 다운로드하기 위해 사용이 된다. 인스톨러는 온카드 인스톨러와 오프카드 인스톨러가 상호작용하여 다운로드가 진행된다.

2.2 애플릿 파이어월과 객체 공유

하나의 자바카드에는 여러 개의 애플릿(applet)이 동시에 존재할 수 있으며, 추가적으로 애플릿들을 다운로드시켜 사용할 수 있다. 이와같이 자바카드 플랫폼은 다중 응용 프로그램 환경을 지원하기 때문에 종종 중요한 데이터를 서로 공유하는 경우가 발생한다. 응용 프로그램들 간에 서로 협업할 수 있기 위해 자바카드에서는 애플릿 파이어월을 통하여 객체 고립화(Object Isolation) 메커니즘을 사용한다.

애플릿 고립은 애플릿 파이어월을 통해서 이루어진다. 이것은 하나의 애플릿을 정해진 공간에 고립시킴으로써 다른 애플릿에 의해 중요 데이터가 빠져 나가는 것을 방지하며 해킹에 대한 보호가 가능하다. 서로 다른 패키지에 존재하는 애플릿은 직접적으로 객체에 대한 레퍼런스를 가질 수 없다. 그러므로, 애플릿의 오동작 혹은 악의를 가진 애플릿은 다른 애플릿이나 JCRE의 동작에 영향을 미칠 수가 없게 된다. 애플릿 파이어월은 자바카드 객체 시스템(object system)을 컨텍스트(Context)라고 불리는 분리되어 보호되는 객체 공간으로 나누게 된다. 파이어월은 서로 다른 컨텍스트 사이의 경계이다. 그림 1은 파이어월과 객체 고립을 보여주는 예이다.

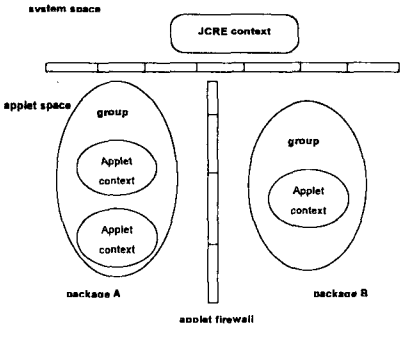

그림 1 자바카드 시스템에서의 파이어월

자바카드 가상기계내에서는 JCRE 컨텍스트나 애플릿의 그룹 컨텍스트들 중에 항상 하나의 활성컨텍스트(*active context*)만 존재한다. 새로운 객체가 생성되면 소유하는(owning) 컨텍스트가 할당되고 현재의 활성컨텍스트가 된다. 따라서, 객체가 인스턴스화 되면 활성 컨텍스트내에 존재하는 활성화 상태의 애플릿에 의해 객체가 소유된다. 한 애플릿내에서 정적으로 생성된 데이터들은 동일한 패키지내의 다른 애플릿에 의해 접근이 가능하다.

객체에 대한 접근이 있을 경우, 자바 언어의 접근 제어 규칙이 적용되고 또한 객체의 소유 컨텍스트와 현재의 활성화된 객체를 비교해서 일치하지 않으면 거부되며, SecurityException 예외가 발생한다.

애플릿 파이어월은 애플릿의 행동을 자신의 컨텍스트내에 한정시킨다. 따라서, 애플릿이 다른 컨텍스트내의 애플릿이나 JCRE에 의해 소유되는 객체에 대한 접근을 할 수 없다. 그러나, 애플릿 간에 협업이 필요한 경우 자바카드는 JCRE 특권(privileges), JCRE 엔트리 포인트 객체(entry point objects), 글로벌 배열(Global arrays), 공유 인터페이스(Shareable interfaces)와 같은 기법을 제공한다.

공유되는 interface는 javacard.framework.Shareable 인터페이스를 상속 받아야 한다. 이 인터페이스는 RMI의 remote 인터페이스와 유사하다. shareable 인터페이스는 다른 애플릿에서 접근이 가능한 메소드의 집합을 정의한다. Shareable Interface Object는 Shareable 인터페이스를 구현한 클래스의 객체를 shareable interface object(SIO)라고 한다. 자신의 컨텍스트내에서는 SIO는 일반적인 객체이지만 다른 컨텍스트에 대해서는 shareable 인터페이스에 정의된 메소드만 접근이 가능하다. 그 외의 메소드와 필드들은 파이어월에 의해서 접근이 거부된다.

앞의 그림은 Shareable Interface Object를 사용하는 경우의 컨텍스트 스위치에 대한 그림이다. 다른 컨텍스트에 존재하는 SIO를 사용하기 위해서는 JCRE를 통해서 객체에 대한 참조를 얻어와야 한다. 따라서, 컨텍스트 스위치 과정 중에 JCRE를 반드시 거쳐가야만 한다.

3. 자바 카드 시스템 모델링

2절에서 설명한 자바 카드 시스템을 토대로 자바 카드 시스템의 모델링을 UML을 사용한 Use Case 다이어그램은 그림 2와 같다.

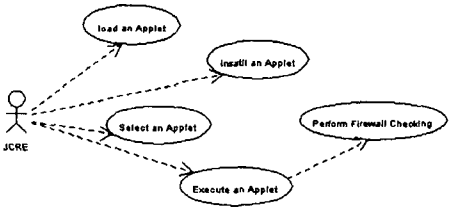

그림 2 자바카드의 Use Case Diagram

위와 같은 use case 다이어그램을 사용하여 자바 카드 구현에 필요한 객체들은 그림 3과 같이 나타낼 수 있다. JCRE 객체는 Firewall 객체, Interpreter 객체, Installer 객체, ApduIO 객체로 구성되어 있으며 자바카드 시스템의 운영체제에 해당되므로 한 개의 인스턴스만 존재한다. 컨텍스트 객체는 한 개 이상의 인스턴스가 존재하며 애플릿 객체는 자바카드에 적재되는 자바 카드 응용 프로그램의 수에 따라 0개 이상의 인스턴스가 존재한다.

다음은 공유 인터페이스를 사용하여 서로 다른 애플릿(응용 프로그

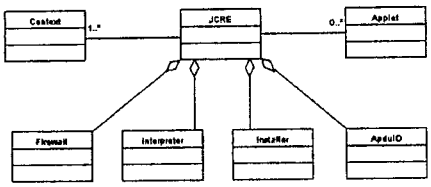

그림 3 자바카드의 클래스 다이어그램

램)에서 객체를 서로 공유하는 기법을 예를 통하여 객체 설계 기법을 제안한다. 예제 응용 프로그램은 WalletApp와 AirMilesApp 2개의 애플릿으로 구성되어 있는데 일부 코드가 그림 4에 나타나 있다. WalletApp 애플릿은 사용자의 전자 지갑에 해당되는 것으로 은행등에서 관리하고 있다고 가정할 수 있으며 이 사용자는 AirMilesApp을 통하여 탑승하는 비행 거리를 항공사가 관리하는 애플릿에 해당된다. 사용자가 항공권을 구매하게 되면 항공사가 보너스 마일리지를 사용자에게 제공하기로 계약했다면 서로 다른 컨텍스트하에 존재하는 WalletApp과 AirMilesApp가 전자 현금과 마일리지를 상호 작용 및 연동해야 하므로 객체 공유의 문제가 발생하는데 이를 공유 인터페이

```
public class AirMilesApp extends Applet implements
AirMilesInterface {
  short miles;
  public void grantMiles(short amount) {
    miles = (short)(miles+amount);
  }
}
```

```
import javacard.framework.*;
public class WalletApp extends Applet {
  short balance;
  byte[] air_miles_aid_bytes = SERVER_AID_BYTES;
  private void debit(short amount) {
    ....
  }
  private void requestMiles(short amount) {
    AID air_miles_aid = JCSystem.lookupAID(air_miles_aid_bytes,
      (short)(0), (byte)air_miles_aid_bytes.length);
    AirMilesInterface sio = (AirMilesInterface)
      JCSystem.getShareableInterfaceObject(
        air_miles_aid, SECRET_CODE);
    sio.grantMiles(amount);
  }
}
```

```
import javacard.framework.Shareable;
public interface AirMilesInterface extends Shareable {
  public void grantMiles(short amount);
}
```

그림 4 공유 인터페이스를 사용한 애플릿 예

스를 적용하여 해결할 수 있다. 이 설계 과정은 다음과 같다.

먼저 AirMilesApp 애플릿에서 공유 인터페이스를 정의한다. 어떤 객체가 서로 다른 컨텍스트의 애플릿이 공유를 허용해 주기 위해서는 AirMilesApp 애플릿은 javacard.framework.Shareable을 상속시켜주는 AirMilesInterface interface를 정의한다. 이 공유 인터페이스에 정의된 여러 메소드들은 다른 애플릿에서도 접근이 가능하게 된다. 애플릿은 공유 인터페이스를 구현하는 AirMilesApp 를 정의하며 필요한 필드등을 정의하는데 이 때의 필드는 파이어월의 규칙에 의해서 공유가 불가능하게 된다. UML을 사용한 클래스 다이어그램은 그림 5와 같다. 이 때의 공유 메소드는 grantMiles 이며 다른 애플릿에서도 접근이 가능하다.

다음으로 WalletApp 애플릿은 공유 인터페이스를 획득하고 AirMilesApp 애플릿의 서비스(메소드)를 요구한다. WalletApp

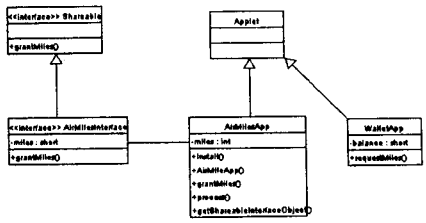

그림 5 예제 애플릿의 클래스 다이어그램

애플릿의 메소드를 접근하기 위해서 JCRE에 설치된 애플릿의 AID들의 리스트 중 원하는 클라이언트 애플릿의 AID를 JCSystem.AID 메소드를 사용하여 구하고 WalletAPP 애플릿은 공유 인터페이스의 참조는 먼저 AirMilesInterface의 참조를 구하고 JCSystem.getAppletShareableInterfaceObject 메소드를 통해서 알아낸다. 이 호출에 의해 JCRE는 AirMilesApp 애플릿의 Applet.getShareableInterfaceObject 메소드를 호출한다. AirMilesApp 애플릿은 WalletApp 애플릿으로의 접근이 가능한지 검사한다. UML로 설계한 순서도가 그림 6에 나타나 있다.

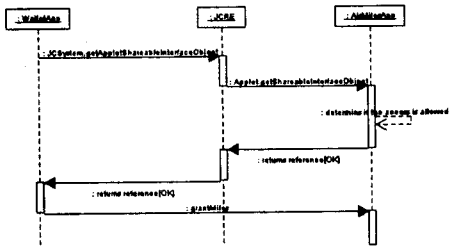

그림 6 예제 애플릿의 순서도

4. 결 론

본 논문에서는 스마트카드의 업계 표준으로 자리잡고 있는 자바 카드를 중심으로 자바 카드의 주요 특성과 응용 프로그램 작성 기법을 객체지향 중심 기법을 채택한 설계 기법을 제안하였다. 스마트카드용 응용 프로그램 개발시 객체 지향 기법을 적용함으로써 유지 보수 및 확정성이 우수한 장점이 있으며 향후 더 일반적인 경우에 있어서의 애플릿의 객체 지향 설계 기법에 대한 연구 및 응용 프로그램 설계 환경 개발에도 객체 지향 개념을 도입할 예정이다.

참고 문헌

[1] Sun Microsystems Inc., *Java Card Platform Specifications*, 2002

[2] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

[3] J. Vanderwalle, E. Vetillard, Developing Smart Card-Based Applications with Java Card, In *ERSADS '99*, April 1999.

[4] H. Martin, Using test hypotheses to build a UML model of object-oriented smart card applications, *Proceedings of ICSSEA'99*, 1999.