JAVA CARD YAZILIMLARININ MODEL GÜDÜMLÜ GELİŞTİRİLMESİ

Hidayet Burak SARITAŞ Geylani KARDAŞ

Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü, İzmir
burak.saritas@kentkart.com.tr geylani.kardas@ege.edu.tr

Özetçe

Java Card, kullanım alanları ve tercih edilme sıklığı açısından bakıldığında endüstri lideri konumunda bulunan bir akıllı karttır. Akıllı kartlar üzerlerinde mikroişlemci bulunan, veriyi saklayabilme ve işleyebilme yeteneğine sahip; hız, güvenlik, taşınabilirlik özellikleri sayesinde telekomünikasyon, ulaşım, kredi kartı endüstrisi gibi birçok alanda kullanılabilen, entegre cihazlardır. Bu özelliklerine rağmen, akıllı kart yazılımlarını geliştirmek, alt seviye iletişim yapıları, donanımsal sebepler ve Java Card'ın yazılım geliştirme aşamasında kullanıcıya getirdiği bazı kısıtlar nedeniyle, geliştiriciler için sıkıntılı bir hal almaktadır. Bu bildiride, Java Card yazılımlarının otomatik ve daha basit bir şekilde ve hatasız üretilmesini sağlayan model güdümlü bir yazılım geliştirme yöntemi tanıtılmaktadır.

Anahtar sözcükler: *Model güdümlü geliştirme, Java Card, Otomatik kod üretme.*

Model Driven Development of Java Card Software

Abstract

Today Java Card is the most preferred type of smart cards and its application programming interface is the most widely chosen software library for the development of smart card software. Smartcards are portable integrated devices that store and process data. Speed, security and portability properties enable smartcards to have a widespread usage in various fields including telecommunication, transportation and credit card industry. However, the development of Java Card applications is a difficult task due to the hardware and software constraints. Hence, in this study, we introduce a model-based approach which facilitates the Java Card software development by both providing an easy design of smart card systems and automatic generation of the software from the system models.

Keywords: *Model Driven Development, Java Card, Code Generation*

1. Giriş

Akıllı kartlar günümüzde, telekomünikasyondan ulaşıma, kredi kartı endüstrisinden sağlık kuruluşlarına kadar geniş bir yelpazede kullanılmaktadır. Akıllı kartların tercih edilmesindeki başlıca neden taşınabilirliktir. ISO/IEC 7816 [2] fiziksel karakteristikleri ve iletişim altyapıları standartlaştırılan akıllı kartların mikroişlemcisi olması, veriyi işleme ve saklama özelliği bulunması sebebiyle taşınabilirlik ayrı bir önem kazanmaktadır. Standart bir akıllı kart üzerinde Ram Bellek, Rom Bellek, merkezi işlem birimi ve elektronik silinebilir bellek bulunmaktadır [1]. Bu özellikleri sayesinde birçok alanda güvenliği ve otomasyonu sağlayacak uygulamalar akıllı kartlar ile geliştirilebilmektedir.

Artan mobil uygulamalar, günlük işlemlerin elektronik ortama taşınması ve para ödeme araçlarının yaygınlaşması gibi nedenlerle kullanılan özel bilgilerin yüksek güvenlik gereksinimi, akıllı kartlar ile sağlanmaktadır. Kimlik tanımlama, yetkili kişi erişimleri gerektiren durumlarda akıllı kartlar, hafızası ve veri işleme yeteneği sayesinde, hızlı ve güvenli bir iletişim sağlamaktadır. Bu kadar yaygın kullanımı ve gelişmiş özellikleri bulunmasına rağmen akıllı kartlar için yazılım geliştirmek, standart programlamadan daha karmaşık bir yapı ile uğraşmaya neden olmaktadır. Ayrıca az sayıda kişi, akıllı kartlar için yazılım geliştirme sürecinde aktif rol oynamaktadır.

Diğer geliştiriciler gibi, akademik çalışmalarımızda Java Card yazılımlarının dizaynı ve geliştirilmesi aşamasında birçok zorluklar ile karşılaştık [4,5]. Deneyimlerimize dayanarak hazırladığımız bu çalışmamızda, akıllı kartlar için yazılım geliştirme sürecini kolaylaştırmak ve hızlandırmak amacıyla model güdümlü bir yaklaşım tanıtılmaktadır. Çalışmada, yaygınlığı ve endüstride lider konumda bulunması dolayısıyla Java Card [6] üzerinde yoğunlaşılmıştır. Bu amaçla Java Card API yorumlanarak, Java Card yazılımlarını üretmeye yarayacak bir metamodel geliştirilmiştir. Oluşturulan metamodel için bir grafiksel arayüz tasarlanarak, bu arayüz üzerinde kullanıcıların geliştirdiği modeller ile otomatik kod üretebilmeleri sağlanmıştır.

Çalışmamızda kullandığımız yaklaşım ikinci bölümde anlatılmaktadır. Aynı bölüm altında, tasarladığımız modelimizin ayrıntılarını anlatabilmek amacıyla ele aldığımız örnek Java Card uygulaması hakkında bilgiler verilmektedir. Üçüncü bölümde, Java Card için geliştirdiğimiz platforma özgü model ile ilgili tanımlar, Java Card API, Java Card metamodeli ve kullanıcılar için hazırladığımız görsel arayüz hakkında anlatımlar yer almaktadır. Tasarlanan modeller ve tanımları, grafiksel arayüz ve otomatik kod üretme aşamaları ayrıntılı olarak anlatılmıştır. Daha önce benzer alanlarda sunulan yaklaşımlar, çalışmamızın geleceği ve geliştirmesine devam edilen kısımlar bildirinin son iki bölümünde yer almaktadır.

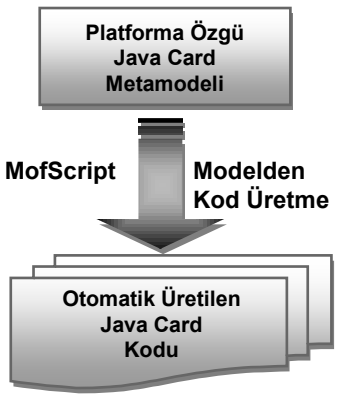
2. Yaklaşım

Bu çalışmamızın amacı, Java Card [6] için platforma özgü bir ortam geliştirip bu ortamı sağlayacak bir metamodel oluşturmak ve sonraki aşamada platform bağımsız bir model geliştirerek dilin özelliklerine olan bağımlılığı kaldırmaktır. Bu nedenle Java Card API incelenerek bir metamodel oluşturulmaya çalışılmıştır.

Java Card metamodeli Eclipse Modelleme Ortamı (Eclipse Modeling Framework - EMF) üzerinde Ecore metamodeli ile tasarlanmıştır [8]. EMF, yeni bir model geliştirebilme ve bu model için gerçek zamanlı kod üretebilmeyi sağlayan araçlar sunmaktadır. EMF platformunun sağladığı araçlar ile geliştirilen Java Card metamodeli, Java Card yazılımları geliştirebilmek için gereken birçok model elamanını içinde barındırarak, geliştiriciler için kullanışlı bir ortam haline getirilmeye çalışılmıştır. Ayrıca Java Card metamodelinin Ecore gösterimi ile tasarlanması, Java Card modeli üzerinde oluşturulan örneklerin doğruluğunu kontrol edebilmeyi sağlamakta ve bu örneklerden kod üretim aşamasını güçlendirmektedir.

Ecore tabanlı hazırlanan Java Card metamodeli için grafiksel bir arayüz hazırlamak amacıyla Graphical Modeling Framework (GMF) kullanılmıştır [9]. GMF, kullanıcılar için, EMF üzerine dayalı kullanışlı araçlar ve gerçek zamanlı bir grafiksel arayüz geliştirme ortamı sunmaktadır. Kullanıcılar geliştirdikleri modeller üzerinde zengin içerikli grafiksel arayüzler oluşturmak için GMF ortamını kullanabilmektedirler. Daha önce tasarladığımız Java Card ecore modeli kullanılıp yeni bir GMF projesi oluşturularak, Java Card yazılımlarını geliştirmek için kullanılabilecek bir arayüzün kodları, model elemanları ve özellikleri tanımlanmaktadır.

Java Card grafiksel arayüzü ile oluşturulacak modellerden kod üretme aşaması MOFScript[14] ile yapılmaktadır. Mofscript, Eclipse üzerinde kulanılabilen ve modelden kod üretmeye yarayan bir araçtır. Kullanıcılar tarafından grafiksel arayüz üzerinde tasarlanan model elemanları, Şekil 1'de tanıtıldığı gibi Mofscript ile yorumlanarak Java Card koduna dönüştürülmektedir



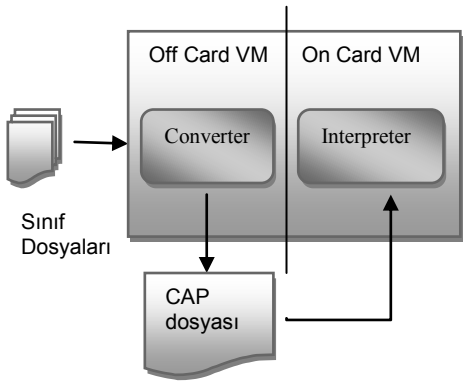
Şekil 1. Yaklaşım diyagramı

2.1 Java Card

Günümüzde akıllı kartlar içinde en çok tercih edilen kart tipi Java Card ve buna istinaden en geniş kullanım alanı olan geliştirme ortamı Java Card platformudur. Java dilinin ayrıcalıkları, geliştiriciler için birçok kolaylıklar getirmektedir. Bu açıdan bakıldığında da geliştirme aşaması zor olan bir dili öğrenmek yerine zaten yaygın kullanımı olan Java dilinin özelliklerinden yararlanmak harcanan zamandan tasarruf sağlamaktadır. Java dili ile daha önceden tasarlanmış bir programı değiştirmek, yeniden bir program geliştirmek veya programları test etme aşaması daha kolaylaşmaktadır.

Java Card, Java dilini kullanabilmeye olanak sağlamakta ve Java tabanlı uygulamalar için güvenli ve şifreli bir ortam oluşturmaktadır. Java Card için ana tema, akıllı kartlara yazılım geliştiren kişiler için standart bir geliştirme ortamı sunmaktır. Java Card teknolojisi, kod üretme aşaması için kullanışlı bir ortam sağlamakta ve üretilen kodun farklı akıllı kartlarda çalışabilmesini amaçlamaktadır.

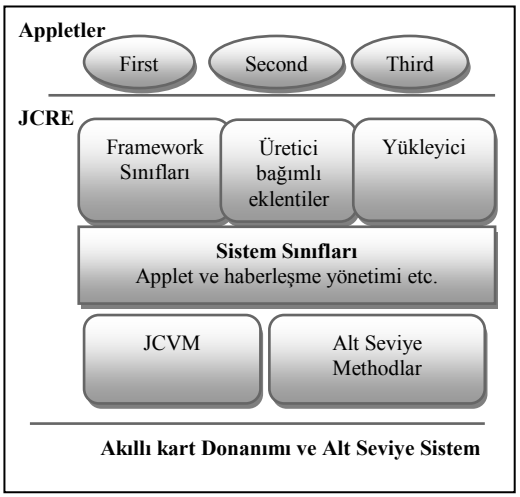
Java Card platformu, Java dilinin sınırlı bir kümesini içermektedir. Bu sebeple Java dilinin bazı özellikleri Java Card platformunda bulunmamaktadır. Java Card Sanal Makinesi (JCVM), Java dilinde kullanıldığı halinden farklı noktalar içermektedir [3]. JCVM, Şekil 2'de gösterildiği gibi, hafıza ve işlemci kısıtlarından dolayı off-card VM (converter) ve on-card VM (interpreter) olarak iki farklı bölüme ayrılmaktadır. Java Card platformunun kısıtları converter tarafından denetlenmektedir. Converter, Java dosyalarının yüklenmesini ve Java Card dilinin semantik kurallarını kontrol etmektedir. Geliştirilen Java Card programı (Applet) akıllı karta yüklenmeden önce, sınıf dosyaları converter tarafından CAP (Converted Applet) dosyasına dönüştürülür. Oluşturulan CAP dosyası akıllı karta yüklenir ve interpreter tarafından çalıştırılır.



Şekil 2. Java Card Sanal Makinesi

Şekil 3'te mimarisi anlatılan Java Card Runtime Environment (JCRE), yüklenen applet dosyaları, iletişim altyapısı ve JCVM üzerinde bir kontrol mekanizması gibi çalışmaktadır. JCRE'nin yaptığı işe bakılarak akıllı kartlar içindeki işletim sistemi gibi davrandığı söylenebilir. Diğer bir açıdan bakıldığında JCRE, akıllı kartların işletim sistemi üzerinde platform bağımsız bir arayüz sunmaktadır. Java

Card üretimi sırasında, JCRE, kullanım hayatı boyunca yalnızca bir defa ilklenir. JCRE çalışmaya başladığı zaman, tüm applet elemanları, objeler oluşturulur ve JCVM çalışmaya başlar.



Şekil 3. JCRE

Üretilen Cap dosyaları Java Card hafıza bölgesinde saklanmaktadır. JCRE'nin, çalışma ortamı üzerinde yüklenen applet dosyalarını kontrol etmek üzere tam bir kontrol mekanizması bulunmaktadır. Başarılı bir applet yükleme işleminde sonra, applet ile direk bir iletişim hiçbir zaman yapılmaz. JCRE ilk önce gelen paketleri inceler ve gelen komuta göre uygun methodları çağırır.

2.2 Cüzdan Uygulaması

Geliştirilen Java Card modelinin özelliklerini ve kısıtlarını ayrıntılı anlatabilmek amacıyla bir elektronik cüzdan uygulaması üzerinde modelin kullanımı gösterilmeye çalışılmıştır. Akıllı kartlarda geliştirilebilen cüzdan uygulaması ile kullanıcıların yanlarında taşıyabileceği küçük boyuttaki kartları kullanması sağlanarak toplu ulaşım, otopark, sinema, telekomünikasyon gibi alanlarda fiziksel olarak taşınan para trafiği azaltılabilmektedir. Yüksek güvenlik seviyesi, kartlar üzerinde grafiksel ve yazılımsal olarak kullanıcı kişiselleştirmesi yapılabilmesi ve takip edilebilmesinin kolay olması gibi nedenlerden dolayı akıllı kartlar ve bu kartlar üzerinde geliştirilen elektronik cüzdan uygulamaları gibi projeler, ticari ve bireysel anlamda daha birçok konuya kaynak olabilmektedir. Çalışmamızda geliştirdiğimiz model ortamını temel özellikleri ile tanıtabilmek ve program zorluğunu azaltmak amacıyla seçilen Java Card cüzdan uygulamasının basit ve anlaşılır olması düşünülmüştür.

Tablo 1'de ayrıntıları verilen elektronik cüzdan uygulaması, sabitlerin tanımlandığı PurseDeclarations isimli bir arayüz, standart olarak her Java Card uygulamasında bulunması gereken process ve install methodlarını içeren bir Purse isimli Applet ve cüzdandan para düşme, bakiye öğrenme gibi methodların bulunduğu Purse isimli bir sınıf içermektedir. Purse Applet'i içerisinde, Purse sınıfından bir nesne tanımlanmakta ve PurseDeclarations arayüzü implement edilmektedir [7].

2.2.1 Tanımlar:

- **Purse.java**

stateOfPurse: Cüzdan uygulamasının durum bilgisi

Olası Durum Bilgileri:
STATE_DEBIT_ALLOWED
STATE_CREDIT_ALLOWED
STATE_ADMIN_ALLOWED

purseBalance : Cüzdanda yüklü para miktarı

pursePINs : Debit, Credit, UpdatePIN methodlarını güvenli hale getirmek için kullanılan şifreler

debitPINValue : Debit işlemi için başlangıçta tutulan PIN değeri

creditPINValue : Credit işlemi için başlangıçta tutulan PIN değeri

adminPINValue : Yetkili işlemleri için başlangıçta tutulan PIN değeri

Tablo-1: Cüzdan Uygulaması Dosyaları

Kaynak dosyalar	Methodlar	Açıklama
Purse *Java Source*	Verify UpdatePIN Debit Credit getBalance getMaxBalance Discount	PurseApplet tarafından kullanılan basit elektronik cüzdan komutlarını içermektedir.
Purse Declarati-ons *Java Source*	-	Applet sınıfı tarafından iletişim aşamasında kullanılan APDU paketlerinin CLA ve INS byte'larının tanımlarının yapıldığı ve sabitlerin tanımlandığı dosyadır.
Purse Applet *Java Source*	Install Process Receive	Standart Applet tanımlamalarının yapıldığı, Java Card API'sinin kullanımını gösteren bir applet dosyasıdır.

3. Platforma Özgü Java Card Modeli

Java Card API incelenerek EMF üzerinde hazırlanan metamodel ve GMF ile hazırlanan Java Card model elemanları, kullanıcılara, hazırladıkları modeli görsel bir editör eşliğinde ifade edebilmeleri için bir tasarlama ortamı sunmaktadır. Hazırlanan metamodel on-card Java bileşenleri ve ilişkilerini kapsamaktadır.

Bu bölümde sırasıyla Java Card API hakkında bilgiler, Java Card API yorumlanarak geliştirilen Java Card Metamodeli ve bu metamodel ile tasarlanan görsel geliştirme ortamı tanıtılacaktır.

3.1 Java Card API

Java Card API, Java dilinin kısıtlı bir alt kümesini içermektedir. Bu sebeple bazı veri tipleri ve methodlar Java Card dil özelliklerinde bulunmamaktadır. Aşağıda Java Card dili için, desteklenen ve desteklenmeyen özellikler listesi bulunmaktadır [3].

- Desteklenenler:

Small Primitive types, Boolean, Short, Byte, Java Packages, Interfaces, Classes, Exceptions, Java object oriented features (Inheritance...), Virtual Methods, Overloading, Dynamic Object Creation, Access Scope, Binding Rules, Int Data Type (optional)

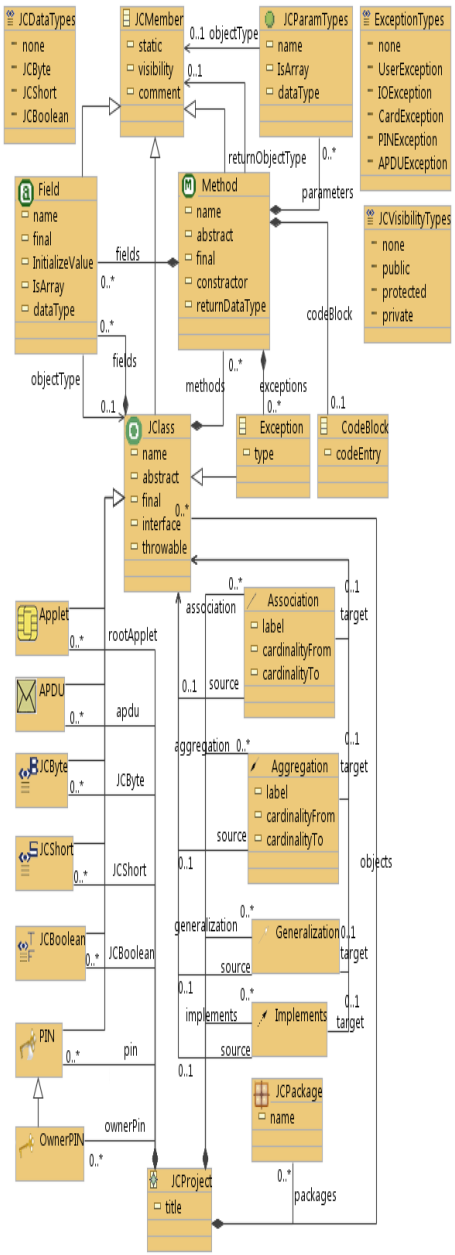
- Desteklenmeyenler:

Large primitive types, Long, Double, Float, Characters and Strings, Multidimensional, Arrays, Dynamic Class Loading, Security Manager, Garbage Collection, Threads, Object Serialization, Object Cloning.

3.2 Geliştirilen Java Card Metamodeli

Hazırlanan metamodelin, Eclipse EMF kullanılarak görsel bir diyagramı oluşturulmuştur. Şekil 4'te gösterilen Java Card metamodelinin elemanlarının ayrıntılı açıklamaları bu bölümde anlatılacaktır.

Gerçek anlamda, üretilen model, Model Tabanlı Mimari (MDA) [10] bakış açısına göre Java Card için oluşturulmuş platforma özgü metamodel olarak adlandırılabilir. Şekil 4'te gösterilen Java Card metamodelinin bazı elemanları, sınırlı alan dolayısı ile ihmal edilmiştir. Metamodel üzerindeki önemli ana elemanlar aşağıda anlatıldığı gibidir.



Şekil 4. Java Card API incelenerek yorumladığımız Java Card Metamodeli

Metamodelin anahtar öğesi *Applet* model elemanıdır. Applet, akıllı kart tarafında bulunan Java Card programını temsil etmektedir. Okuyucu tarafından gelen isteklere, uygun cevapları geri döndürür.
Application Protocol Data Unit (APDU), ISO / IEC 7816 ile standartları belirlenmiş, akıllı kart ile okuyucu arasındaki iletişimi sağlayan paket yapısıdır. Java Card teknolojisinde, okuyucu tarafından bir *APDU* komutu gönderilir ve akıllı kart tarafından bu komuta cevaben yeni bir *APDU* paketi geri gönderilir. Standart bir Java Card programında bulunması gereken applet dosyasının karşılığı bu model elemanı ile karşılanmaktadır. Oluşturulan Applet elemanı, üretilecek kod içinde bulunması gereken "*process*" ve "*install*" methodlarını otomatik olarak tanımlanır.

OwnerPIN ve *PIN*, Java Card programında kullanılabilecek şifre işlemleri için tanımlanmış model elemanlarıdır. *PIN*, akıllı kart programına yetkili erişimleri sağlamak amacıyla şifre tanımlarının yapılabildiği bir arayüz sağlar. Java Card API, *PIN* arayüzünün hazırda kullanılması için aynı isimde bir sınıf sağlamaktadır. Bu yüzden *OwnerPIN*, model elemanı olarak, tasarlanan Java Card metamodeli üzerinde gösterilmektedir. Okuyucu ile akıllı kart üzerindeki applet arasında bir bağlantı açılmadan önce *PIN* değerinin doğrulaması yapılmaktadır. Okuyucu *PIN* değerini içeren *APDU* paketini gönderir ve applet *PIN* değerinin kontrolünü yapar. *PIN* değeri doğru girilmediği sürece bağlantı kurulamaz. *PIN* ve *OwnerPIN* model elemanları ile herhangi bir sınıf veya Applet elemanları arasında bir ilişki tanımlanabilmektedir. Başlangıç değeri, ismi gibi ayarlar, özellikler bölmesinde yapılabilmektedir.

Java Card metamodelinin üst kısımındaki elemanlar, Java Card API içerisindeki temel parametre tipleri, alt alanlar, methodlar gibi öğeleri içermektedir.

Java Card sınıfları arasındaki ilişkiler de (*Association, Aggregation, Generalization, Implements*) metamodel içerisinde tanımlanabilmektedir.

Metamodelde bulunan *JClass* elemanı, Java Card programı içerisinde tanımlanacak sınıflar için temel bir sınıf obejesini temsil etmektedir. Java Card dilinin kısıtlamaları dikkate alındığında, Java dilinde bir sınıf içerisinde bulunan method ve özelliklerin büyük bir bölümü *JClass* elemanı ile tanımlanamamaktadır.
Byte, Boolean ve Short, Java Card API içerisinde tanımlı olan veri tipleridir. Bunların dışındaki veri tipleri için kısıtlar eklenerek Java Card metamodeli üzerinde destek verilmemiştir.

3.3 Java Card Programlarının Modellenmesi

Model tabanlı yaklaşımlar doğal olarak, tasarımı yapılan modeller için bir kullanıcı arayüzü gerektirmektedir. Gerçekleştirilen arayüzler kullanıcılara görsel olarak kolay kullanım ve doğrulama ortamı sunmaktadır. Bu amaçla, çalışmamızda elde ettiğimiz Java Card için tasarlanan model tabanlı geliştirme ortamını desteklemek amacıyla görsel modelleme araçları geliştirilmiştir. Daha önce anlatıldığı gibi, modelleme araçları Eclipse platformu üzerinde Grafiksel Modelleme (GMF) araçları kullanılarak tanımlanmıştır. Bu düşünce kapsamında, Java Card geliştirme ortamı için Ecore metemodeli tanımlandı, ana model elemanlarını ve ilişkilerini belirtmek amacıyla grafiksel gösterimler hazırlandı ve kod üretme amaçlı, tasarlanan grafiksel elemanlar ile Java Card bileşenleri ilişkilendirildi.

Geliştiriciler, modelleme ile Java Card sistemlerini hazırlamak amacıyla görsel editör ortamını kullanabilmektedirler. Modelleme ortamının ana elemanları ve aralarındaki ilişkileri tanımlamak için kullanılan elemanlar palet üzerinde gösterilmektedir. Geliştiriciler kendi modellerini oluşturmak için istedikleri model elemanını ve ilişki okunu kullanabilmektedirler. Ayrıca, modelleme ortamı yanlış bir ilişki kurulmasını engellemek ve tasarlanan modellerin tutarlı olması sağlamak için bazı kısıtlar getirmektedir. Örneğin, method ve alt alanlar sadece applet veya sınıfların içinde tanımlanabilmekte, ilişki okları sadece belli sınıflar ve veri tipleri arasında oluşturulmakta ve *Exception* model elemanı ile kod blokları sadece methodlar içerisine eklenebilmektedir.

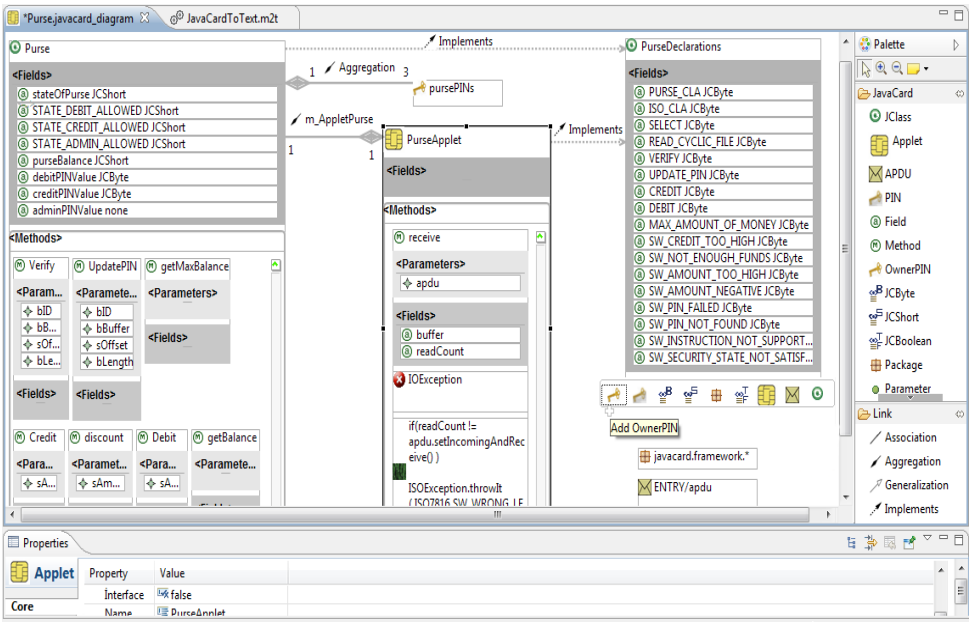
Şekil 6'da Java Card modelleme araçları ile, klasik bir cüzdan uygulaması için hazırlanmış model bileşenlerinin ekran görüntüsü bulunmaktadır.

Geliştiriciler model elemanlarını paletten tasarım alanına doğru kolayca tut-bırak yöntemiyle ekleyebilmekte ve model elemanlarının tüm özelliklerini Şekil 5'teki gibi benzer alanlar ile Eclipse editör ortamında düzenleyebilmektedirler. Ayrıca bazı özellikler grafiksel alan üzerinde değiştirilebilmektedir.

Modelleme ortamında varolan veya yeni eklenmiş objelerin dizisinin oluşturulabilmesi için, *Aggregation* ilişkisi kullanılmaktadır. Bu sayede herhangi bir sınıf içerisine, istenilen objeyi birden fazla içeren veri tipi eklenebilir. Böyle bir kullanım Şekil 6'daki Purse sınıfı ile pursePINs objesi arasında örneklendirilmektedir. Bu ilişki ile Purse sınıfı içerisine, pursePINs isminde üç adet OwnerPIN objesi içeren bir dizi eklenmektedir. Sınıflar içerisinde oluşturulan diziler, Java Card kısıtlarından dolayı sadece tek boyutlu olabilmektedir. Bu yüzden, modelleme ortamında birden fazla boyutlu diziler oluşturma imkanı bulunmamaktadır.

Property	Value
Abstract	false
Comment	
Final	false
İnterface	false
Name	PurseApplet
Static	false
Throwable	false
Visibility	public

Şekil-5: Model elemanlarının özelliklerinin düzenlenebildiği sekme



Şekil-6: Java Card modelleme araçları ile tasarlanan Elektronik Cüzdan modeli

Şekil 6'da tasarlanan cüzdan uygulaması modelinde, istenilen kodları oluşturmak amacıyla, gerekli method ve alt alanlar paletten tasarım alanına eklenmiştir. Anlaşılırlığı ve basitliği korumak amacıyla, Java Card programında gerekli olan paket alış verişini sağlayan methodlar ve ayrıntıları modelleme ortamında gösterilmemektedir.

3.4 Java Card Programları için Otomatik Kod Üretme

Grafiksel olarak Java Card program geliştirme ortamı tasarlanmasına rağmen, gerçek hayatta geliştirilen uygulamalarda bu yeterli olmamakta ve kaçınılmaz olarak sistem geliştiriciler kod yazmak zorunda kalmaktadırlar. Tasarlanan sistem, kod üretme sırasında geliştiricilere önayak olmakta ve basit işlemlerin birçoğunda geliştiriciye kolaylık sağlamaktadır.

Şekil 7'de bir kısmı gösterilen, modellerden kod üretme amacıyla geliştirilen dönüşüm kodları MOFScript dili ile yazılmıştır. MOFScript, Eclipse içinde bir araç olarak kullanılabilmekte ve geliştirilen modellerden herhangi bir anda kod üretilebilmektedir.

Grafiksel arayüz üzerinde tasarlanan modellerin çıktısı, MOFScript dili [14] için bir girdi olarak algılanmakta ve Java Card için yazılan modelden koda dönüşüm kodları ile Java Card programı elde edilmektedir. Geliştiriciler tasarlamak istedikleri Java Card programının görsel modellemesini tamamladıktan sonra, model elemanlarının ve ilişkilerinin *Ecore* formatında tutulduğu dosya MofScript tarafından dönüşüm kodlarını oluşturmak için kullanılmaktadır.

```
texttransformation JavaCard (in ec:"org.javacard.com") {
  ec.JCProject::main () {
    var fileName:String = ""
    self.rootApplet->forEach(c:ec.JClass) {
        fileName = c.name + ".java"
    }
    file("F:\\" + fileName)
    '
    import javacard.framework.APDU;
    import javacard.framework.ISO7816;
    import javacard.framework.Applet;
    import javacard.framework.ISOException;'
self.packages->forEach(p:ec.JCPackage) {
    '
    import ' p.name ';'
}
```

Şekil-7: Örnek MOFScript kodu

```
package com.applets.purse;

import javacard.framework.*;

public class PurseApplet extends Applet implements PurseDeclarations
{
    private Purse m_AppletPurse;

    public static void install(byte[] buffer, short offset, byte length){
        new PurseApplet();
    }
    public PurseApplet(){
        m_AppletPurse = new Purse();
        register();
    }
    public void process(APDU apdu) throws ISOException{
        byte[] buffer = apdu.getBuffer();

      switch (buffer[ISO7816.OFFSET_INS]) {
      case (byte) 0x00:
       break;
      default:
       ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
      }
    }
    public short receive(APDU apdu)
    throws ISOException
    {
        byte[] buffer = apdu.getBuffer();
        short readCount = (short) (buffer[ISO7816.OFFSET_LC] & 0x00FF);
        if(readCount != apdu.setIncomingAndReceive() )
            ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
        return readCount;
    }
}
```

Şekil-8: Üretilen PurseApplet sınıfının kodu

MOFScript dili ile hazırlanan Java Card için modelden koda dönüşüm kodları (Şekil 9), elektronik cüzdan uygulaması için hazırlanan model dosyasını girdi olarak alarak (Şekil 6), PurseAplet sınıfı için Şekil 8'deki gibi bir kod üretmektedir.

4. İlgili Çalışmalar

Bu alanda daha önce yapılan çalışmalarda, akıllı kartlar için kod üretme, kişiselleştime ve üretim aşamalarını hızlandırmak ve kolaylaştırmak amacıyla model güdümlü yaklaşımlar kullanılmıştır. Geliştirilen yöntemler içerisinde, kodlanmış akıllı kart programlarının doğruluğunun test edilmesi, UML diyagramları ile güvenli akıllı kart programlarının oluşturulması gibi amaçlar bulunmaktadır. Akıllı kartlar için üretim aşamasından sonra kişiselleştirme ve ilgili uygulamaların yüklenmesi gibi işlemler yapılmaktadır. Bu üretim ve konfigürasyon aşamasını hızlandırmak ve birleştirmek amacıyla model tabanlı yaklaşım kullanan bir yöntem [11]'de verilmiştir.

Başka bir çalışmada, MetaSlang dili kullanılarak, Java Card applet kodlarının üretilmesi ve üretilen kodun doğruluğunun denetlenmesi ile ilgili bir yöntem anlatılmaktadır.[12] Bu dil, Java Card programlamaya yakın, görsel olmayan bir yapıda geliştirmeye olanak vermektedir. Geliştiricilerin bu konuda bilgili olmaları beklenmektedir.

UML diyagramları kullandırılarak, platform bağımsız bir model oluşturulup, bu modelden kod üretimi sağlayan bir yöntemde [13], akıllı kart kodlarının üretilmesi sağlanmaktadır. Bu çalışmada, Java Card dili tanımlamaları yerine UML tanımlamaları kullanılmış ve bu sayede geliştiricilerin Java Card kodu üretebileceği anlatılmıştır. Biz çalışmamızda, platform bağımsız modelde, Java Card dil öğelerine yakın model elemanları kullanarak, orta düzey geliştiriciler için yeni bir dil öğrenmelerine gerek kalmadan tasarım yapabilmelerini ve yeni başlayanlar için kolay kullanılır bir geliştirme ortamı sunmayı amaçladık. Benzer konuda yapılan birçok çalışmada, görsel olarak tasarlanabilen, tut-bırak-düzenle tarzı uygulamalar bulunmamaktadır. Bu çalışmamızda geliştiricilerin, Java Card programlamanın zorluklarından uzaklaşıp görsel öğeler kullanarak hatasız kod üretebilmelerini sağlamaya çalıştık. Bu amaçla geliştirilen tüm model üretme, üretilen modelden kod üretme aşamaları, yaygın bir editör olan Eclipse üzerinde gerçeklenebilmektedir.

6. Sonuç

Java Card yazılımlarını model güdümlü geliştirmek için tasarlanmaya başlanan projemizin önemli bir bölümü platforma özgü model tasarımı ile gerçekleştirilmeye çalışıldı. Geliştirmesi devam eden çalışmamızda, platform bağımsız model ile dile özgü kullanımlar ve tanımlamalar model geliştirme aşamasında soyutlanmaya çalışılmaktadır. Fakat isteyen geliştiriciler için platforma özgü modelleme seviyesinde düzenlemelere olanak verilerek, geliştiricilerin platform bağımsız

model üzerinde tasarlayamadıkları kısımlara doğrudan erişebilmeleri sağlanacaktır. Java Card dışında farklı tipteki akıllı kartlar için de platforma özgü modeller tasarlanmaya devam etmektedir. Bu sayede tek bir platform bağımsız model ile birden fazla tipteki akıllı kartlar için otomatik kod üretilmesi sağlanabilecektir. Platform bağımsız modellerden, platforma özgü modellere dönüşüm kurallarının, AtlanMod Transformation Language (ATL) [15] ile geliştirilmesi planlanmakadır.

Gelişen yazılım dünyasında, model tabanlı yaklaşımlar büyük önem kazanmaya başlamıştır. Alt seviye hatalar, dil bağımlılıkları ve küçük değişiklikler için bile zaman alan uğraşlarda bulunmak geliştiriciler için büyük enerji kaybına yol açmaktadır. Bu nedenle benzer ilgi alanındaki ürünler için ortak geliştirme ortamları tasarlamanın büyük faydaları olacaktır. Çalışmamızda kullandığımız yöntemin ve yaklaşımın akıllı kart yazılımlarının geliştirilme aşamasına kolaylık getirip yarar sağlayacağını, bundan sonraki değişik yöndeki düşüncelere de kaynak teşkil edeceğini düşünmekte ve bu yöndeki uygulamalarımızı geliştirmeye devam etmekteyiz.

Kaynakça

[1] **Rankl, W., Effing, W.**: Smart Card Handbook. John Wiley & Sons, West Sussex (2000)

[2] **ISO/IEC 7816** Standards family for Identification cards - Integrated circuit cards,http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45144

[3] **Chen, Zhiqun,** Java Card Technology for Smart Cards: Architecture and Programmer's Guide, Addison-Wesley, September 18, 2000

[4] **Kardas, G. and Tunali, E. T.** "*Design and Implementation of a Smart Card Based Healthcare Information System*", 2006

[5] **Kardas, G., Celikel, E.:** A Smart Card Mediated Mobile Platform for Secure E-Mail Communication. In: 4th International Conference on Information Technology: New Generations, pp. 925-926. IEEE Computer Society Press, New York (2007)

[6] Sun Microsystems: Java Card Technology, http://java.sun.com/javacard/

[7] Sm@rtCafé Toolkit, Giesecke & Devrient GmbH, 2008

[8] Eclipse Modeling Framework Project (EMF) , http://www.eclipse.org/modeling/emf/

[9] Eclipse Graphical Modeling Framework (GMF) , http://www.eclipse.org/modeling/gmf/

[10] Object Management Group Model Driven Architecture, http://www.omg.org/mda/

[11] **Bonnet, S., Potonniee, O., Marvie, R., Geib, J-M.**: A Model-Driven Approach for Smart Card Configuration. In: Karsai, G. Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 416--435. Springer, Heidelberg (2004)

[12] **Coglio, A.**: Code generation for high-assurance Java Card applets. In: 3rd NSA Conference on High Confidence Software and Systems, pp. 85--93 (2003)

[13] **Moebius, N., Stenzel, K., Grandy, H., Reif, W.**: Model-Driven Code Generation for Secure Smart Card Applications. In: 20th Australian Software Engineering Conference, pp. 44--53. IEEE Computer Society Press, New York (2009)

[14] **Oldevik, J., Neple, T., Gronmo, R., Aagedal, J., Berre, AJ.**: Toward Standardised Model to Text Transformations. In: Hartman A., Kreische D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 239--253. Springer, Heidelberg (2005)

[15] **Jouault, F., Kurtev, I.**: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128--138. Springer, Heidelberg (2006)