

Java Card Yazılımlarının Model Güdümlü Geliştirilmesi

Hidayet Burak Sarıtaş Geylani Kardeş

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

Özetçe

Java Card, kullanım alanları, dinamik olarak uygulama geliştirilebilmesi 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 çalışmada, 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 multi application 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 ulaşım, kredi kartı endüstrisinden sağlık kuruluşlarına kadar geniş bir yelpazede kullanılmaktadır. Akıllı kartların tercih edilmesindeki başlıca neden taşınabilirliktir. ISO/IEC 7816 [1] ile 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 [2]. 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 [3,4]. Deneyimlerimize dayanarak hazırladığımız bu çalışmamızda, akıllı kartlar için yazılım geliştirme sürecini kolaylaştırmak ve hızlandırmak amacıyla model güdümlü bir yaklaşım tanıtılmaktadır. Çalışmada, yaygınlığı ve dinamik programlanabilen kartlar endüstrisinde lider konumda bulunması dolayısıyla Java Card [5] üzerinde yoğunlaşmıştır. 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.

Çalışmada kullanılan yaklaşım ikinci bölümde anlatılmaktadır. Üçüncü bölümde, Java Card için geliştirilen platforma özgü model ile ilgili tanımlar, Java Card API, Java Card metamodeli, ve kullanıcılar için hazırlanan görsel arayüz hakkında anlatımlar yer almaktadır. Aynı bölüm altında, tasarlanan modelin ayrıntılarını anlatabilmek amacıyla ele alınan örnek Java Card uygulaması hakkında bilgiler verilmektedir. Tasarlanan modeller ve tanımları, grafiksel arayüz ve otomatik kod üretme aşamaları ayrıntılı olarak anlatılmıştır. Daha önce benzer alanlarda sunulan yaklaşımlar, çalışmanın geleceği ve geliştirmesine devam edilen kısımlar son bölümünde yer almaktadır.

2. Yaklaşım

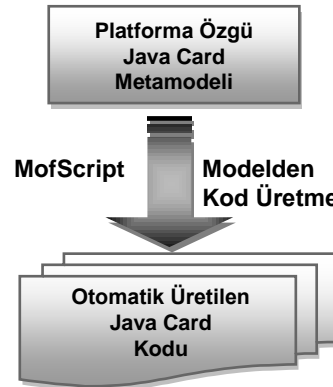
Bu çalışmanın amacı, Java Card [5] için platforma özgü bir ortam geliştirip bu ortamı sağlayacak bir metamodel oluşturmak ve sonraki aşamada platform bağımsız bir model geliştirerek dilin özelliklerine olan bağımlılığı ortadan kaldırmaktır. Bu nedenle Java Card API incelenerek bir metamodel oluşturulmaya çalışılmıştır.

Java Card metamodeli Eclipse Modelleme Ortamı (Eclipse Modeling Framework - EMF) üzerinde Ecore metamodeli ile tasarlanmıştır [6]. 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 elemanını 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 [7]. 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 kullanılacak bir arayüzün kodları, model elemanları ve özellikleri tanımlanmaktadır.

Java Card grafiksel arayüzü ile oluşturulacak modellerden kod üretme aşaması MOFScript[8] ile yapılmaktadır. Mofscript, Eclipse üzerinde kullanılabilen 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



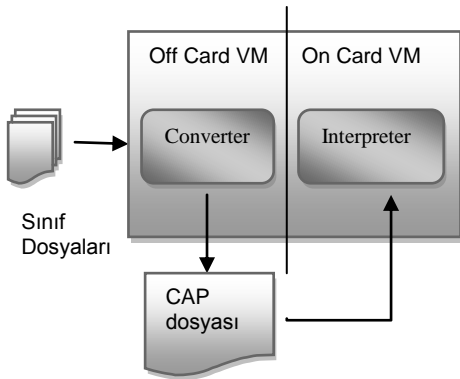
Şekil 1. Yaklaşım diyagramı

2.1 Java Card

Günümüzde, dinamik olarak çoklu uygulamalar geliştirilebilen akıllı kartlar içinde en çok tercih edilen kart tipi Java Card ve buna istinaden en geniş kullanım alanı olan geliştirme ortamı Java Card platformudur.[9] 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 sağlamak 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.

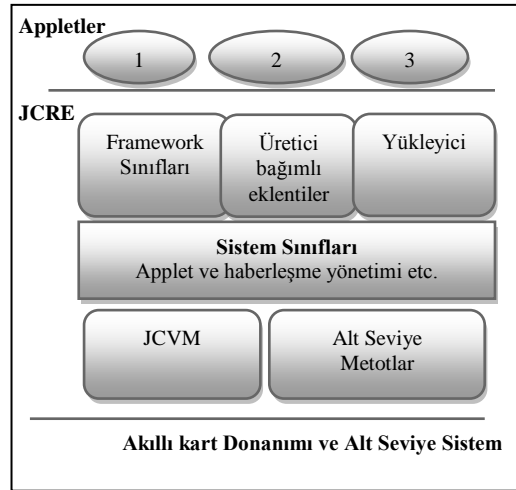
Java Card platformu, Java dilinin sınırlı bir kümesini içermektedir. Bu sebeple Java dilinin bazı özellikleri Java Card platformunda bulunmamaktadır. Java Card Sanal Makinesi (JCVM), Java dilinde kullanıldığı halinden farklı noktalar içermektedir [10]. JCVM, Şekil 2’de gösterildiği gibi, hafıza ve işlemci kısıtlarından dolayı off-card VM (converter) ve on-card VM (interpreter) olarak iki farklı bölüme ayrılmaktadır. Java Card platformunun kısıtları converter tarafından denetlenmektedir.



Şekil 2. Java Card Sanal Makinesi

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 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 ilklendir. JCRE çalışmaya başladığı zaman, tüm applet elemanları, nesnelere oluşturulur ve JCVM çalışmaya başlar.



Şekil 3. Java Card Çalışma Zamanı Ortamı

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 direkt bir iletişim hiçbir zaman yapılmaz. JCRE ilk önce gelen paketleri inceler ve gelen komuta göre uygun metotları çağırır.

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 (Uygulama Programlama Ara Yüzü), Java dilinin kısıtlı bir alt kümesini içermektedir. Bu sebeple bazı veri tipleri ve metotlar Java Card dil özelliklerinde bulunmamaktadır. Aşağıda Java Card dili için, desteklenen ve desteklenmeyen özellikler listesi bulunmaktadır [10].

- **Desteklenenler:**

Küçük ilkel veri tipleri (Boolean, Short, Byte), Java paketleri, Ara yüzler, Sınıflar, Çalışma zamanı hata yakalama mekanizmaları (Exceptions), Nesne tabanlı Java programlama özellikleri (Kalıtım...), Sanal metotlar, Aşırı yükleme (Overloading), Dinamik nesne oluşturma, Faaliyet alanları (Access Scope), Yükleme kuralları, Integer veri tipi (seçimli olarak)

- **Desteklenmeyenler:**

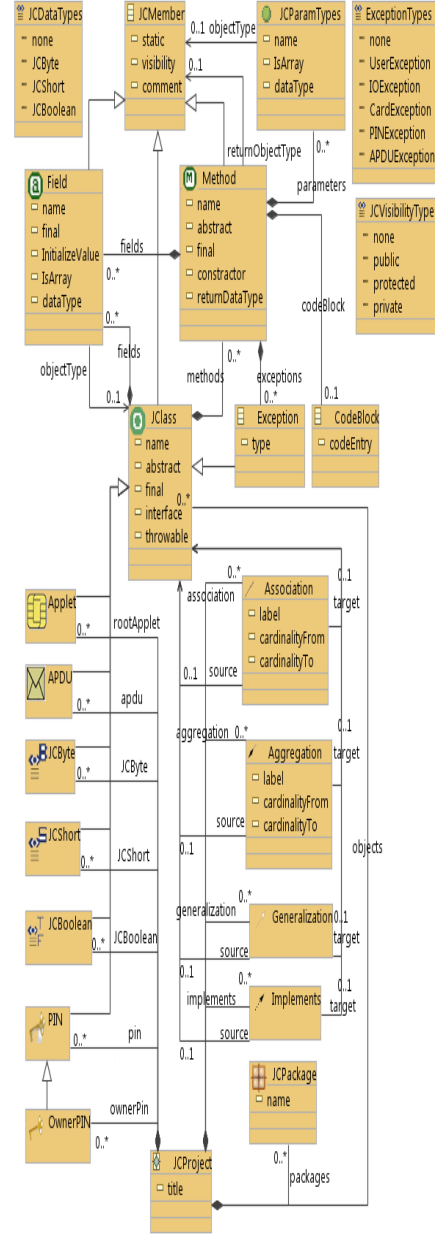
Büyük ilkel veri tipleri (Long, Double, Float, Characters and Strings), Çok boyutlu diziler Dinamik sınıf yükleme, Güvenlik mekanizması, Çöp toplama sistemi (Garbage Collection), Çoklu işlem parçacıkları, Nesnelerin serileştirilmesi, Nesne klonlama.

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) [11] 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'ye bağlı olarak türetilen Java Card Metamodeli

Metamodelin anahtar ögesi *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*” metotlarını otomatik olarak tanımlanır.

OwnerPIN ve *PIN*, Java Card programında kullanılacak 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 kısmındaki elemanlar, Java Card API içerisindeki temel parametre tipleri, alt alanlar, metotlar 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 objesini temsil etmektedir. Java Card dilinin kısıtlamaları dikkate alındığında, Java dilinde bir sınıf içerisinde bulunan metot 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 Cüzdan Uygulaması

Geliştirilen Java Card modelinin özelliklerini ve kısıtlarını ayrıntılı anlatılabilmek 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, [12]’de yer alan örnekten alınmıştır. Uygulama, sabitlerin tanımlandığı *PurseDeclarations* isimli bir arayüz, standart olarak her Java Card uygulamasında bulunması gereken *process* ve *install* metotlarını içeren bir *Purse* isimli *Applet* ve cüzdandan para düşme, bakiye öğrenme gibi metotların bulunduğu *Purse* isimli bir sınıf içermektedir. *Purse Applet*’i içerisinde, *Purse* sınıfından bir nesne tanımlanmakta ve *PurseDeclarations* arayüzü implement edilmektedir.

3.3.1 Tanımlar:

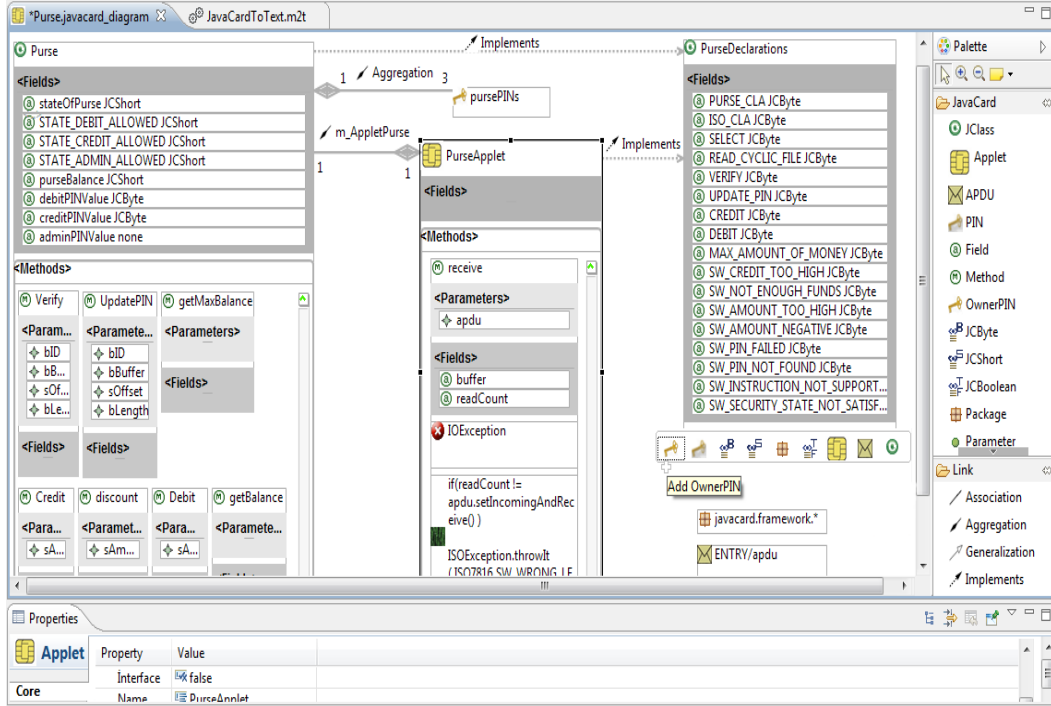
• Purse.java

stateOfPurse: Cüzdan uygulamasının durum bilgisi

Olası Durum Bilgileri:

STATE_DEBIT_ALLOWED

STATE_CREDIT_ALLOWED



Şekil-5: Java Card modelleme araçları ile tasarlanan Elektronik Cüzdan modeli

STATE_ADMIN_ALLOWED

purseBalance : Cüzdana yüklü para miktarı
 pursePINs : Debit, Credit, UpdatePIN metotlarını 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

3.4 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 Grafikselle Modelleme (GMF) araçları kullanılarak tanımlanmıştır. Bu düşünce

kapsamında, Java Card geliştirme ortamı için Ecore metamodeli 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, metot 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 metotlar içerisine eklenebilmektedir.

Şekil 5'te Java Card modelleme araçları ile, klasik bir cüzdan uygulaması için hazırlanmış model bileşenlerinin ekran görüntüsü bulunmaktadır.

Tablo-1: Cüzdan Uygulaması Dosyaları

Kaynak dosyalar	Metotlar	Açıklama
Purse Java Source	Verify UpdatePIN Debit Credit getBalance getMaxBalance Discount	PurseApplet tarafından kullanılan basit elektronik cüzdan komutlarını içermektedir.
Purse Declarat i-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.
Kaynak dosyalar	Metotlar	Açıklama
Purse Java Source	Verify UpdatePIN Debit Credit getBalance getMaxBalance Discount	PurseApplet tarafından kullanılan basit elektronik cüzdan komutlarını içermektedir.
Purse Declarat i-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.

Geliştiriciler model elemanlarını paletten tasarım alanına doğru kolayca tut-bırak yöntemiyle ekleyebilmekte ve model elemanlarının tüm özelliklerini Şekil 6'daki gibi benzer alanlar ile Eclipse editör ortamında düzenleyebilmektedirler.

Ayrıca bazı özellikler grafiksel alan üzerinde değiştirilebilmektedir.

Modelleme ortamında varolan veya yeni eklenmiş nesnelerin dizisinin oluşturulabilmesi için, *Aggregation* ilişkisi kullanılmaktadır. Bu sayede herhangi bir sınıf içerisine, istenilen nesneyi birden fazla içeren veri tipi eklenebilir. Böyle bir kullanım Şekil 5'teki Purse sınıfı ile pursePINs nesnesi arasında örneklendirilmektedir. Bu ilişki ile Purse sınıfı içerisine, pursePINs isminde üç adet OwnerPIN nesnesi 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	<input checked="" type="checkbox"/> false
Comment	<input type="checkbox"/>
Final	<input checked="" type="checkbox"/> false
Interface	<input checked="" type="checkbox"/> false
Name	<input type="checkbox"/> PurseApplet
Static	<input checked="" type="checkbox"/> false
Throwable	<input checked="" type="checkbox"/> false
Visibility	<input type="checkbox"/> public

Şekil-6: Model elemanlarının özelliklerinin düzenlenebildiği sekme

Şekil 5'te tasarlanan cüzdan uygulaması modelinde, istenilen kodları oluşturmak amacıyla, gerekli metot ve alt alanlar paletten tasarım alanına eklenmiştir. Anlaşılabilirliği ve basitliği korumak amacıyla, Java Card programında gerekli olan paket alışverişini sağlayan metotlar ve ayrıntıları modelleme ortamında gösterilmemektedir.

3.5 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 önyak 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 kullanılabilir ve geliştirilen modellerden herhangi bir anda kod üretilebilmektedir.

Grafiksel arayüz üzerinde tasarlanan modellerin çıktısı, MOFScript dili [8] 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.

```
self.rootApplet->forEach(c:ec.JClass) {
  '
  public class ' c.name ' extends Applet { '
  '
  public static void install(byte[] bArray,
    short bOffset, byte bLength) { '
  'new ' c.name' ();'
  '}'
  'public ' c.name ' () { '
  self.aggregation->forEach(c1:ec.Aggregation) {
  if ((c1.source.name = c.name) ) {
  self.objects->forEach(c2:ec.JClass) {
  if (c2.name = c1.target.name) {
  c1.label ' = new ' c1.target.name' ();
  '
  }
  }
  }
  'register();'
  '}'
  'public void process(APDU apdu)
  throws IOException{
  '}
```

Şekil-7: Örnek MOFScript kodu

MOFScript dili ile hazırlanan Java Card için modelden koda dönüşüm kodları, elektronik cüzdan uygulaması için hazırlanan model dosyasını girdi olarak alarak (Şekil 5), PurseApplet sınıfı için Şekil 8’deki gibi bir kod üretmektedir.

```
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 IOException{
    byte[] buffer = apdu.getBuffer();

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

Şekil-8: Üretilen PurseApplet sınıfının kodu

4.İlgili Çalışmalar

Bu alanda daha önce yapılan çalışmalarda, akıllı kartlar için kod üretme, kişiselleştirme 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 [13]’de verilmiştir.

Başka bir çalışmada, MetaSlang dili kullanılarak, Java Card applet kodlarının üretilmesi ve üretilen kodun doğruluğunun denetlenmesi ile ilgili bir yöntem anlatılmaktadır.[14] Bu dil, Java Card programlamaya yakın, görsel olmayan bir yapıda geliştirmeye olanak vermektedir. Geliştiricilerin bu konuda bilgili olmaları beklenmektedir.

UML diyagramları kullanılarak, platform bağımsız bir model oluşturulup, bu modelden kod

üretimi sağlayan bir yöntemde [15], akıllı kart kodlarının üretilmesi sağlanmaktadır. Bu çalışmada, Java Card dili tanımlamaları yerine UML tanımlamaları kullanılmış ve bu sayede geliştiricilerin Java Card kodu üretebileceği anlatılmıştır. 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 kullanılabilir 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 gerçekleştirilmektedir.

5. 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, AtlasMod Transformation Language (ATL) [16] ile geliştirilmesi planlanmaktadır.

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 tasarlanmanın 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] **ISO/IEC 7816** Standards family for Identification cards - Integrated circuit cards, http://www.iso.org/iso/iso_catalogue/catalogue_e_tc/catalogue_tc_browse.htm?commid=45144
- [2] **Rankl, W., Effing, W.:** Smart Card Handbook. John Wiley & Sons, West Sussex (2000)
- [3] **Kardas, G. and Tunali, E. T.** (2006) "Design and Implementation of a Smart Card Based Healthcare Information System", Computer Methods and Programs in Biomedicine, Elsevier, Vol. 81, Issue 1, pp. 66-78.
- [4] **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)
- [5] Sun Microsystems: Java Card Technology, <http://java.sun.com/javacard/>
- [6] Eclipse Modeling Framework Project (EMF) , <http://www.eclipse.org/modeling/emf/>
- [7] Eclipse Graphical Modeling Framework (GMF) , <http://www.eclipse.org/modeling/gmf/>
- [8] **Oldevik, J., Neple, T., Gronmo, R., Aagedal, J., Berre, A.J.:** Toward Standardised Model to Text Transformations. In: Hartman A., Kreische D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 239--253. Springer, Heidelberg (2005)
- [9] "**Embedded Star Newsletter**, 2004, "Java Card Technology Platform Reaches 750 Million Deployments", <http://www.embeddedstar.com/press/content/2004/1/embedded17020.html>"

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

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

[12] **Sm@rtCafé Professional Toolkit**, Giesecke & Devrient GmbH, 2008, http://www.gdai.com/portal/payment/page-_pageid=42,142959&_dad=portal&_schema=PORTAL.htm

[13] **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)

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

[15] **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)

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