Akıllı Kart Yazılımlarının Model Güdümlü Geliştirilmesi

Hidayet Burak Sarıtaş[1], Geylani Kardaş[2]

[1,2] Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, 35100, Bornova, İzmir
[1] burak.saritas@kentkart.com.tr, [2]geylani.kardas@ ege.edu.tr

Özet. Akıllı kartlar kendi içerilerinde gömülü bir işlemci ve bellek bulundurabilen, farklı türlerdeki verileri güvenli bir şekilde saklama ve bu bilgileri kullanabilme özelliğine sahip entegre devrelerdir. Günlük hayatta tanımlama, doğrulama ve veri güvenliği gerektiren birçok ticari işlemi en az seviyede kullanıcı etkileşimi gerektirerek sağlayan akıllı kartlar, boyutları dolayısıyla da kullanım alanlarını genişletmektedirler. Bu özelliklerine rağmen, akıllı kart yazılımlarını geliştirmek, alt seviye iletişim yapıları, donanımsal sebepler ve geliştirme aşamasında kullanıcıya getirdiği bazı kısıtlar nedeniyle, geliştiriciler için sıkıntılı bir hal almaktadır. Bu çalışmada, akıllı kart yazılımlarının otomatik, daha basit bir şekilde ve küçük kod hataları engellenerek üretilmesini sağlayan model güdümlü bir yazılım geliştirme yöntemi tanıtılmaktadır. Akıllı kart standartları göz önünde bulundurularak oluşturulan platform bağımsız bir üstmodelden model dönüşümleri sonrası iki önemli akıllı kart platformu için yazılım modelleri ve otomatik kod üretimi sağlanmaktadır.

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 [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. Ancak 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.

Bu çalışmada, uygulama geliştirme süresince yaşanılan sıkıntıları ve akıllı kartlara özel programlama dili bağımlılığını ortadan kaldırmak amacıyla model güdümlü bir yazılım geliştirme yöntemi tanıtılmaktadır. ISO/IEC 7816 standartları ve akıllı kart uygulaması geliştirirken gereken ortak programlama özellikleri gözönünde bulundurularak genel bir yazılım metamodeli ve model dönüşümlerine dayanan bir akıllı kart uygulama geliştirme sistemi önerilmektedir. Bu sistem içinde metamodelin geliştiriciler için kolay kullanılabilmesi açısından grafiksel bir arayüz tasarlanarak, bu sayede örnek modeller üretilebilmesi sağlanmıştır. Akıllı kart metamodelinden üretilen örnek modeller, hazırladığımız dönüşüm kodları ile popüler akıllı kart platformlarından olan Java Card

[3] ve Basic Card [4] modellerine dönüştürülmekte ve sonrasında bu modellerden otomatik kod üretimi sağlanmaktadır. Java Card ve Basic Card seçimi, akıllı kart sektöründeki yaygınlıkları ve programlama geliştirme ortamları dikkate alınarak yapılmıştır.

Çalısmanın dayandığı yaklaşım ikinci bölümde anlatılmaktadır. Üçüncü bölümde, akıllı kart uygulamaları için geliştirilen modeller, platform bağımsız akıllı kart modelinden Java Card ve Basic Card modellerine dönüşümler ve örnek bir elektronik cüzdan uygulaması tanıtılmaktadır. Dördüncü bölümde, elektronik cüzdan uygulamasının modellenmesi ve otomatik kod üretme kısmı anlatılmaktadır.

2 Yaklaşım

Model Güdümlü Geliştirme (MDD – Model Driven Development) ile yazılım geliştirme süreci modeller üzerinden tanımlanmaya çalışılmıştır. Kullanılan modeller ile yazılım geliştirme sürecine soyut bir bakış açısı getirilmiş ve farklı yazılım geliştirme platformlarının karmaşıklığı azaltılmaya çalışılmıştır. Object Management Group (OMG) tarafından MDD kapsamında önerilen Model Tabanlı Mimari [5] ile geliştirilmesi hedeflenen yazılım sistemlerinin farklı soyutlama seviyelerine ait modelleri ve bu modeller arasında model dönüşümleri tanımlanarak farklı uygulama platformları için bu yazılımların hızlı ve etkin bir biçimde geliştirilmesi hedeflenmektedir. Bu amaçla akıllı kart tiplerinin yapıları incelenip, yazılım geliştiriciler için otomatik kod üretme yöntemleri sağlanabilmesi ve bu yöntemler ile platform bağımsız olarak çalışılabilmesi düşünülmektedir.

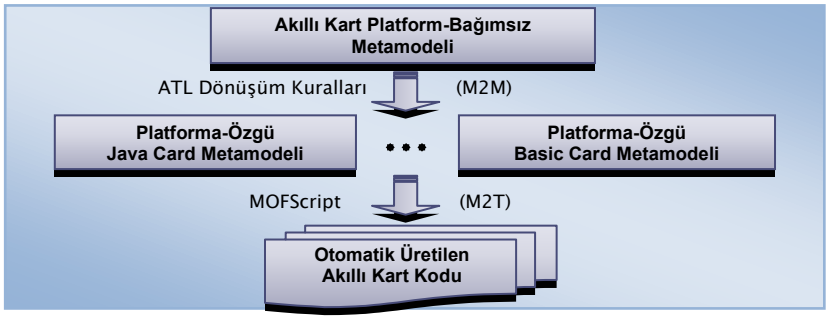
Çalışma kapsamında, akıllı kartlar için ortak özellikler barındıran ve platformdan bağımsız bir şekilde uygulama geliştirebilmeyi sağlayacak bir metamodel ve sistem geliştirilmiştir. Bu sayede geliştiricilerin dile bağlı özelliklerden kurtulup, akıllı kartlar ile ilgili genel özellikleri, ISO/IEC 7816 standartlarını bilmeleri ve programlama mantığını anlamaları yeterli olmaktadır.

Akıllı kart 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 hazırladığımız akıllı kart metamodeli, akıllı kart yazılımları geliştirebilmek için gereken birçok model elamanını içinde barındıran platform bağımsız bir metamodeldir (platform independent metamodel – PIMM). Ecore gösterimi, tasarlanan akıllı kart metamodeli kullanılarak üretilen örnek modellerin doğru ilişkiler ile kurulabilmesini sağlamakta ve platforma özgü modellere dönüşümler kontrollü bir şekilde yapılabilmektedir.

Akıllı kart metamodelinin kullanılabilirliğini artırmak amacıyla Eclipse Graphical Modeling Framework'e (GMF) [7] dayalı grafiksel bir araç tasarlanmıştır. GMF, kullanıcılar için, EMF üzerine dayalı kullanışlı araçlar ve gerçek zamanlı bir grafiksel arayüz geliştirme ortamı sunmaktadır. Kullanıcılar zengin içerikli grafiksel arayüzler içeren GMF ortamını kullanarak kendi modellerini geliştirebilmektedirler.

Akıllı kart metamodeli ile oluşturulan örnek modeller, Atlas Transformation Language (ATL) [8] ile hazırlanan modelden modele (M2M) dönüşüm kuralları uygulanarak platforma (örneğin JavaCard ve BasicCard) özgü modellere (Platform Specific Model – PSM) dönüştürülmektedir. ATL dönüşümleri sonucunda hazırlanan modeller, grafiksel olarak platforma özgü modeller üzerinde tekrar oluşturulabilmekte ve bu örnek modellerden kod üretme aşamasına geçilebilmektedir.

Akıllı kart grafiksel arayüzü ile oluşturulacak modellerden kod üretme (M2T) aşaması MOFScript[9] ile yapılmaktadır. MOFScript, Eclipse üzerinde kulanılabilen ve modelden kod üretmeye yarayan bir araçtır. Üretilen kodlar yüksek oranda doğrudan derlenebilir ve küçük değişiklikler ile çalıştırılabilir durumda olmaktadır. Şekil 1'de akıllı kartlar için tasarlanan modelgüdümlü yazılım geliştirme yaklaşımı gösterilmiştir.



Şekil 1. Akıllı kart yazılımlarının model güdümlü geliştirilmesi yaklaşımı

3 Akıllı Kartlar

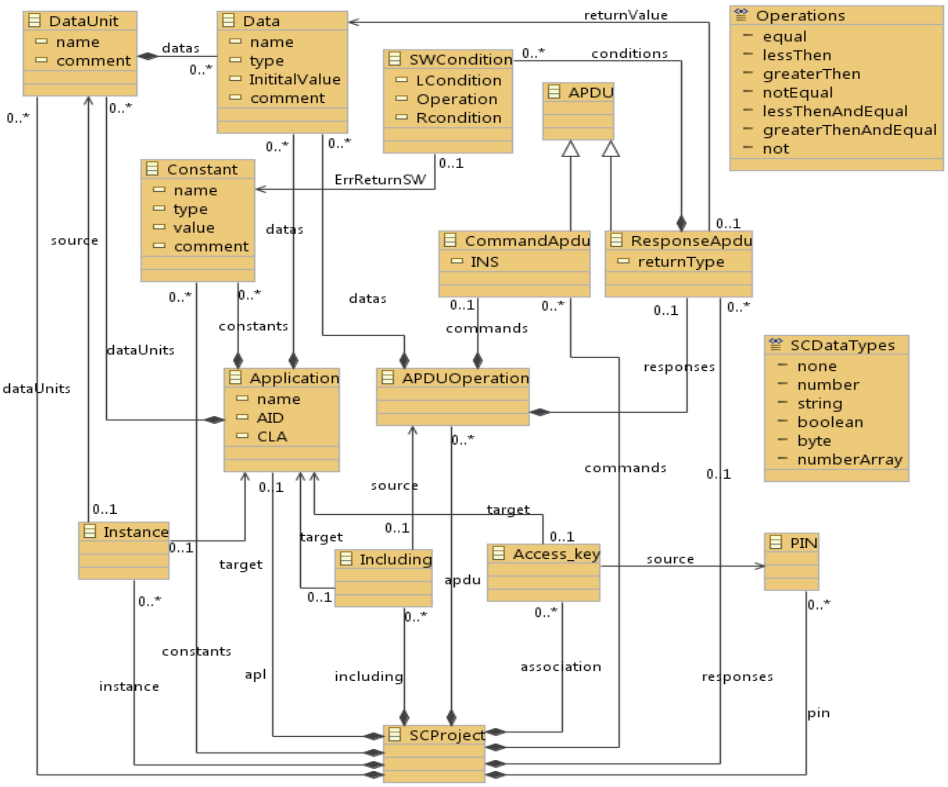
ISO/IEC 7816 standardı ile akıllı kartlar için karakteristik ve temel fonksiyonel özellikler tanımlanmaktadır. Bu sayede akıllı kartlar için standart bir iletişim alt yapısı geliştirilmiştir. Akıllı kartlar ile okuyucu arasında tüm iletişim APDU (Application Protocol Data Unit – Uygulama Protokolü Veri Yapıları) paketleri ile sağlanmaktadır. Terminal programları akıllı karta komut APDU (command APDU) paketleri göndererek karşılığında cevap APDU (response APDU) paketi beklerler. Gelen komut APDU paketine göre, akıllı kart işletim sistemi tarafından daha önceden kart içine yüklenmiş uygulama seçilir. Seçilen akıllı kart uygulaması, gelen APDU paketini işleyerek cevap APDU paketini terminal uygulamasına gönderir.

3.1 Platform-Bağımsız Akıllı Kart Metamodeli

Akıllı kartlar için haberleşme alt yapısı standartlaşmış olsa da, akıllı kart içine yüklenen uygulamalar farklı programlama dillerinde geliştirilmektedir. Bu çalışmada kod üretme amacıyla hedef olarak belirlediğimiz akıllı kartlar için de farklı platformlar üzerinde yazılım geliştirilmektedir. Java Card için Java Card Geliştirme Ortamı (Java Card Development Environment) ve Basic Card için ZC-Basic dili (ZeitControl-Basic Language) kullanılmaktadır.

Akıllı kart metamodeli, farklı akıllı kart platformlarını tek bir çatı altında toplayabilmek için tasarlanmış bir platform bağımsız modeli (PIM) tanımlamaktadır. Bu çalışmada

yalnızca Java Card ve Basic Card platforma özgü modelleri tasarlanmış olsa da, akıllı kart metamodeli daha sonra eklenebilecek platforma özgü modellere de uyum sağlayabilecek şekilde geliştirilmeye çalışılmıştır. Eclipse modelleme ortamı kullanılarak Ecore formatında hazırlanan akıllı kart metamodeli Şekil 2'de gösterilmektedir. Kolay okunabilirliğinin olması ve yer kısıtlarından dolayı metamodel elemanlarının bazı özellikleri bu resim üzerinde gösterilmemektedir.



Şekil 2. Akıllı Kart Metamodeli

Application model elemanı metamodelin anahtar öğesidir. Esas olarak bir akıllı kart programını temsil etmektedir. Application elemanı, akıllı kart uygulaması geliştirirken bir başlangıç noktası olarak görülmekte ve her akıllı kart örnek modeli içine eklenmektedir. Java Card metamodelindeki *Applet* elemanı ve Basic Card metamodelindeki *File Definition* elemanı, *Application* model elemanına karşılık gelmektedir.

Akıllı kart metamodelindeki bir diğer önemli eleman *APDU* öğesidir. Akıllı kart standartlarında da belirtildiği üzere kart ile terminal program arasındaki haberleşme *APDU* paketleri ile yapılmaktadır. Benzer şekilde bu işlemleri karşılamak amacıyla *APDU* model elemanı oluşturulmuştur. *APDU* model elemanı *commandAPDU* ve *responseAPDU* model elemanları için bir meta eleman olarak tanımlanmıştır. Bu

şekilde tasarlanan *APDU* modeli ile uygulama geliştirme aşamasının anlaşılır olması ve platforma özgü modellere dönüşümlerin kolaylaştırılması amaçlanmıştır.

commandAPDU model elemanı için belirtilen *INS* (Instruction Byte(s)) öğesi ile, program içinde çalıştırılacak komutların kontrolü sağlanmaktdır. Girilen *INS* değerine göre gelen *APDU* paketleri işlenerek istenilen *commandAPDU* öğresi çalıştırılmaktadır. Gelen *APDU* paketine bir cevap döneceği durumlarda veya cevap hazırlama aşaması işlemleri için *responseAPDU* elemanı kullanılmaktadır. Gelen APDU paketinin işlenmesi sırasında, *responseAPDU* tarafından durum komutları (Status Words-SW) gönderilmektedir. Herhangi bir koşul oluşması durumunda gönderilecek SW değerleri için *SWCondition* model elemanı tanımlanmıştır. *responseAPDU* içinde tanımlanabilen *SWCondition* elemanı ile istenilmeyen bir koşul oluşması durumunda sabit olarak tanımlanmış SW değerlerinin geri dönüşü sağlanmaktadır. Bir akıllı kart uygulaması geliştirirken çalıştırılması gereken işlemler için tanımlanan *commandAPDU* ve *responseAPDU* elemanları *APDUOperation* ismi altında oluşturulan metamodel öğesi içinde tanımlanabilmektedir. Bu sayede akıllı kart *Application* elemanı ile ilişkilendirilen bir *APDUOperation* elemanı, içerisinde, *commandAPDU* ve *responseAPDU* elemanlarını içermektedir. *APDUOperation* elemanı ile, komut veya işlemler çalıştırılırken gerekecek veri tipi veya değişken tanımlaması yapılabilmektedir. *Application* elemanı ile *APDUOperation* elemanlarını ilişkilendirmek amacıyla *Including* ilişki oku tanımlanmıştır. Bu amaçla eklenen her *APDUOperation* elemanı bir *Including* ilişki oku ile *Application* öğesine bağlanmalıdır.

Geliştirilen akıllı kart uygulamalarına yetkili erişimi sağlamak amacıyla metamodel içerisinde *PIN* elemanı tanımlanmıştır. Okuyucu ile akıllı kart arasında bir bağlantı açılmadan önce *PIN* değerinin doğrulaması yapılmaktadır. Okuyucu *PIN* değerini içeren bir *APDU* paketini gönderir ve akıllı kart içindeki uygulama bu değerin kontrolünü yapar. Örnek model oluşturulması sırasında eklenen *PIN* elemanı ile, doğrulama ve PIN değerini ilkleme komutları otomatik kod üretme aşaması çalıştırıldıktan sonra üretilen koda eklenmektedir. Doğrulama yapılmadığı sürece diğer komutlar çalıştırılamayacaktır. Akıllı kart modeline *PIN* elemanı ekleyebilmek amacıyla *Access_key* ilişkisi tanımlanmıştır. Geliştirilen uygulama içerisine *PIN* özelliklerini eklemek için *Application* ve *PIN* elemanları arasında *Access_key* ilişkisi tanımlanmalıdır.

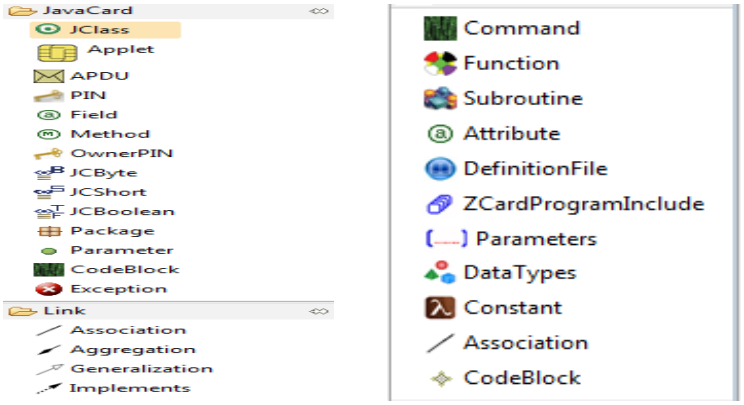
Constant ve *Data* elemanları veri tipi tanımlamaları yapmak amacıyla kullanılmaktadır. *SCDataTypes* model elemanı sayesinde beş adet veri tipi tanımlanabilmektedir. Bunlar *number*, *numberArray*, *string*, *boolean*, *byte* veri tipleridir. Örnek model içerisinde yapılan *Constant* ve *Data* tanımlamalarına göre, tüm veri tipleri, dönüşüm yapılacak model içindeki uygun veri tiplerine çevrilmektedir. Kullanıcı tanımlı veri tipleri ise *DataUnit* elemanı ile tanımlanabilmektedir. Bu sayede farklı veri tiplerini bir araya toplayarak yeni bir veri tipi oluşturulabilmektedir. Uygulama içerisinde yeni bir veri tipi kullanabilmek amacıyla tanımlanan *Instance* ilişkisi, *Application* ve *DataUnit* elemanları arasında kurulmalıdır.

3.2 Platforma Özgü Modeller (Java Card PSM – Basic Card PSM)

Platform bağımsız seviyede akıllı kart metamodeli ile modellenebilen uygulamalarda platforma özgü seviyede değişiklikler yapabilmek için, Java Card ve Basic Card

metamodelleri tanımlanmıştır. Java Card ve Basic Card için geliştirilen metamodeller ve grafiksel modelleme kullanıcı arayüzü ile akıllı kart uygulamalarına dile özgü seviyede de müdahale edilebilmesi sağlanmıştır.

Java Card ve Basic Card için geliştiricilerin uygulama modelleyebilmelerini sağlayan metamodel elemanlarının palet görüntüsü Eclipse GMF ortamında Şekil 3'teki gibi tanımlanmıştır. Java Card için önerdiğimiz platforma özgü metamodel, bu metamodele dayanan örnek modellerde kullanılan elemanlar ve Java Card akıllı kartları için kod üretme aşaması ile ilgili bilgi [10]'da ayrıntılı olarak verilmektedir. Bu sebeple platforma özgü metamodel açıklamaları bu bildiride sınırlı tutulmuştur.



Şekil 3. Java Card ve Basic Card Modelleme Elemanları

Java Card metamodelinin anahtar öğesi Applet model elemanıdır. Applet, akıllı kart tarafında bulunan Java Card programını temsil etmektedir. Java Card programında *process* metodu tarafından işlenen APDU paketlerini kullanacak işlemler için *Method* ve *APDU* elemanları tanımlanmıştır. OwnerPIN ve PIN, Java Card programında kullanılabilecek şifre işlemleri için tanımlanmış model elemanlarıdır. Bunların dışında Java Card diline özgü veri tipi tanımlamaların yapılabildiği, sınıflar arasında ilişkilerin kurulabildiği elemanlarda palet üzerinde yer almaktadır.

Basic Card metamodelinde bulunan *ZCardProgram* elemanı Basic Card için bir kaynak programı temsil etmektedir. Uygulama içerisinde prosedür tanımları *Command*, *Subroutine* ve *Function* elemanları ile yapılmaktadır. Bu elemanların Basic Card uygulamasında ayrı ayrı kullanıldıkları yerler ve çeşitli özellikleri bulunmaktadır. Tanımlanabilecek veri tipleri, sabitler ve elemanlar arasında kurulabilecek ilişkiler palet üzerinde görülmektedir. Hem Java Card hem de Basic Card modelleme ortamında geliştiricilerin o anda aklına gelebilecek kod ayrıntılarını yazabilecekleri CodeBlock elemanı tanımlanmıştır. Bu alana eklenecek kod blokları, üretilen kod içerisine kullanıcı tanımlı alanlar olarak eklenecektir.

3.3 Akıllı Kart Uygulamaları için Modelden Modele Dönüşüm (M2M)

Akıllı kart modelleme ortamı üzerinde oluşturulan örnek modeller ATL dönüşüm kuralları ile platforma özgü örnek modellere dönüştürülmektedir. Grafiksel modelleme arayüzü ile oluşturulan örnek modeller için otomatik olarak oluşturulan Ecore formatı, ATL dönüşüm kurallarına girdi olarak verilmektedir. Akıllı kart metamodelinden

üretilen örnek modellerde bulunan elemanların dönüşüm sonucunda Java Card ve Basic Card modellerinde karşılık geldikleri elemanların bir kısmı Tablo 1'de verilmiştir.

Tablo 1. Akıllı Kart Model elemanları ve çeşitli PIM elemanları arasındaki eşleşmeler

Akıllı Kart Metamodeli Platform Bağımsız Metamodeli	Java Card Metamodeli	Basic Card Metamodeli
SCProject	JCProject	ZCardProgram
Application	Applet	DefinitionFile
APDUOperation (command – response)	Method	Command
PIN	PIN	Attribute
Constant	Field	Constant

Yer kısıtları dolayısıyla tüm eleman dönüşümlerinin verilemediği tabloda APDUOperation elemanı ile ilgili birkaç ayrıntıdan bahsetmek gerekir. Bu elemandan platforma özgü modele dönüşüm ile oluşturulacak *Method* ve *Command* elemanlarının tipi, geri dönüş değeri, aldığı parametre ve içlerinde tanımlanan veri tipleri belirlenebilmektedir. Bunun dışında *Constant* elemanının da, karşılık geldiği metamodellerdeki elemanların kısıtları dikkate alınarak dönüşümü yapılmaktadır. Model dönüşümleri için geliştirilen ATL kurallarının bazıları Tablo 2'de gösterilmektedir.

Tablo 2. Platforma özgü modellere dönüşüm için yazılan bazı ATL kuralları

SmartCard2JavaCard.atl

```
rule SmartCard2JavaCard{
    from
        smartCard : MM!SCProject
    to
        javaCard : MM1!JCProject(
          title <- smartCard.title,
          rootApplet <- Set{MM !
Application.allInstances()},
          ownerPin <-
Set{MM!PIN.allInstances()},
           aggregation <-
Set{MM!Access_key.allInstances()})}
```

```
helper context MM ! Application def :
getAssociations():
    MM ! Application = MM !
Including.allInstances()
    -> select(includes | includes.target
= self);
```

```
rule Application2Applet{
    from
        appl : MM!Application
    to
        app_let : MM1 ! Applet(
         name    <- appl.name,
         fields<-
Sequence{appl.constants, appl.datas},
         methods<-
appl.getAssociations())}
```

SmartCard2BasicCard.atl

```
rule SmartCard2BasicCard{
    from
        smartCard : MM!SCProject
    to
        javaCard : MM1!ZCardProgram(
       name <- smartCard.title,
        commands <- Set{MM !
APDUOperation.allInstances()->
select(a|a.oclIsKindOf(MM ! CommandApdu)
= false)},
       attributes<-
Set{MM!PIN.allInstances()->
select(a|a.oclIsKindOf(MM !
APDUOperation) = false)},
       defFile     <-
Set{MM!Application.allInstances()})}
```

```
rule Application2DEF {
        from
         cons : MM!Application
        to
         defs : MM1!DefinitionFile (
name <- cons.name,
constants<- Sequence{cons.constants}
)}
```

```
rule Constant2Constant {
        from
         cons : MM!Constant
        to
         defs : MM1!Constant (
        name <- cons.name,
        InitialValue     <- cons.value
)}
```

Bu kurallar ile oluşturulan paltforma özgü örnek modeller tekrar eklenti yapmak veya düzeltmek amacıyla grafiksel ortamda açılabilmektedir. Görsel modelleme ortamı üzerinde dile özgü eklentiler yapılabilmekte ve üretilecek kod oranı artırılabilmektedir.

3.4 Akıllı Kartlar için Örnek Cüzdan Uygulaması

Geliştirilen akıllı kart modelinin özelliklerini ve kısıtlarını örnek üzerinde anlatabilmek amacıyla bir elektronik cüzdan uygulaması ile 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 akıllı kart cüzdan uygulamasının basit ve anlaşılır olması düşünülmüştür.

4 Akıllı Kart Uygulamalarının Grafiksel Modellenmesi

Üçüncü bölümde ayrıntıları anlatılan akıllı kart metamodeli ile Eclipse GMF üzerinde akıllı kart uygulamaları geliştirmek amacıyla grafiksel bir arayüz tanımlanmıştır. Modelleme ortamı ve metamodel öğeleri görsel olarak Şekil 4 üzerinde gösterilmektedir. Akıllı kart uygulamasına eklenebilecek metamodel elemanları modelleme ortamının sağ tarafında görülmektedir. Metamodel elemanları arasında kurulabilecek ilişki tipleri için, aynı alan üzerinde *Link* sekmesi bulunmaktadır. Yeni oluşturulan elemanlar ve ilişki tipleri arasındaki kurallar modelleme ortamı tarafından denetlenmektedir. Bu sayede elemanlar ve ilişkiler arasındaki yanlış tanımlamalar editör üzerinde engellenmektedir.

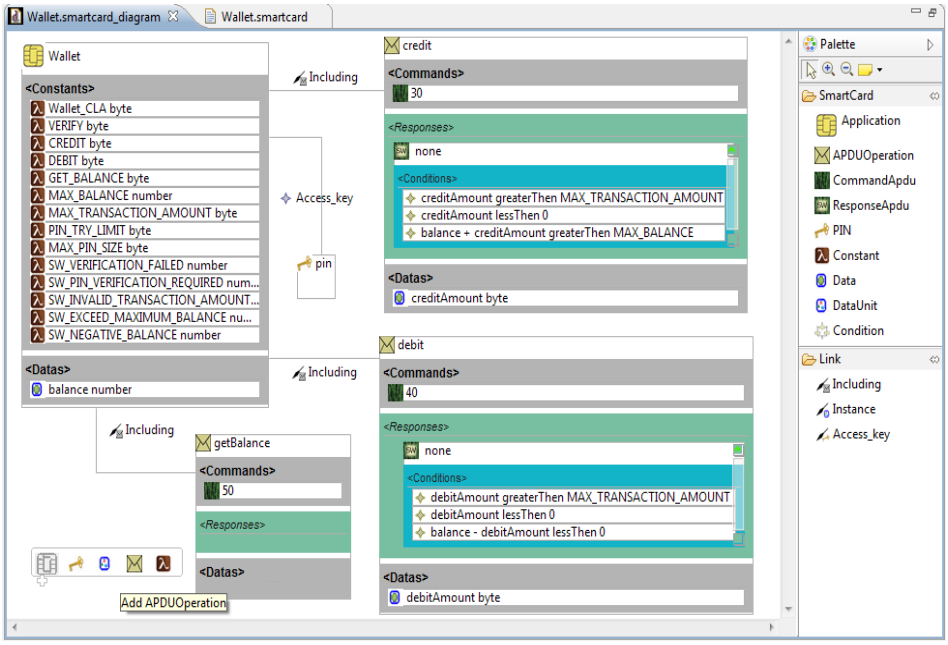
Bu kontrollerden bazıları şu şekilde tanımlanmıştır: *Command* ve *response APDU* elemanları sadece *APDUOperation* elemanı içerisinde tanımlanabilmektedir. İlişki okları sadece ilgili elemanlar arasında oluşturulabilmektedir. *Constant* elemanı *Application* içinde ya da *DataUnit* içinde tanımlanabilir. *Constant* ve *Data* elemanları ilgili olmadıkları bir alanda kullanılamazlar. Bir adet *Application* elemanı tanımlanabilir.

4.2 Akıllı Kart Uygulamaları için Modelden Kod Üretme

Akıllı kart örnek modelinden, platforma özgü modellere Bölüm 3.3'te anlatılan dönüşümler ile oluşan hedef modellerden kod üretmek amacıyla MofScript dili kullanılarak M2T kodları hazırlanmıştır. MOFScript, Eclipse içinde bir araç olarak kullanılabilmekte ve geliştirilen modellerden herhangi bir anda kod üretilebilmektedir. Platforma özgü Java Card modelinden kod üretilmesi ve akıllı kart koduna dönüşüm için yazılan bazı MOFScript kodları [10]'da ayrıntılı olarak anlatılmıştır.

Grafiksel arayüz üzerinde tasarlanan modellerin çıktısı, MOFScript dili için bir girdi olarak algılanmakta ve Java Card ya da Basic Card için yazılan modelden koda dönüşüm kodları ile akıllı kart programı elde edilmektedir. Modellemesi yapılan cüzdan

uygulamasının Java Card platformuna özgü modeline dönüşüm işlemleri tamamlandıktan sonra, MOFScript kodlarının çalıştırılması ile elde edilen koddan örnek bir bölüm Şekil 5'te gösterilmektedir.



Şekil 4. Platform-Bağımsız Akıllı kart modelleme araçları ile tasarlanan Cüzdan Uygulaması

```
package bank.purse;

  import javacard.framework.*;
  import javacardx.framework.*;

  public class Wallet extends Applet {
    public  static  final byte Wallet_CLA = 0xB0; // code of CLA byte in the command APDU header
    public  static  final byte VERIFY = 0x20; //
    public  static  final byte CREDIT = 0x30; //
    public  static  final byte DEBIT = 0x40; //
    public  static  final byte GET_BALANCE = 0x50; //
    public  static  final short MAX_BALANCE = 0x7FFF; // maximum balance
    public  static  final byte MAX_TRANSACTION_AMOUNT = 127; //    maximum transaction amount
    public  static  final byte PIN_TRY_LIMIT = 0x03; // maximum number of incorrect tries
    public  static  final byte MAX_PIN_SIZE = 0x08; //
    public  static  final short SW_VERIFICATION_FAILED = 0x6300; //
    public  static  final short SW_PIN_VERIFICATION_REQUIRED = 0x6301; //
    public  static  final short SW_INVALID_TRANSACTION_AMOUNT = 0x6A83; //
    public  static  final short SW_EXCEED_MAXIMUM_BALANCE = 0x6A84; //
    public  static  final short SW_NEGATIVE_BALANCE = 0x6A85; //
    OwnerPIN pin;
    short balance;

    public static void install(byte[] bArray, short bOffset, byte bLength) {
     new Wallet();
    }

    public Wallet() {
        pin = new OwnerPIN((byte)(5),(byte)(8));
      pin.update(/*write your pin value variable here*/, (short)(0), (byte)(/*write your size of pin here in bytes*/));
      register();
    }
```

Şekil 5. Cüzdan uygulaması için üretilen akıllı kart kodunun bir kısmı

5 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. 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, akıllı kart programlamanın zorluklarından uzaklaşıp görsel öğeler kullanarak oluşturabilecekleri tutarlı örnek modellerden 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

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 farklı bir bakış açısı getireceğine inanmaktayız. Bu çerçevede yapılan çalışmamızın devamında otomatik üretilen kodların verimlilik testleri, gerçek koda oranla ne kadar üretilebildiğinin araştırılması ve geliştirilmesi yer almaktadır.

Kaynakça

1. **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
2. **Rankl, W., Effing, W.**: Smart Card Handbook. John Wiley & Sons, West Sussex (2000).
3. Sun Microsystems: Java Card Technology, http://java.sun.com/javacard/
4. ZeitControl Card Systems GmbH: Basic Card, http://www.basiccard.com/
5. Object Management Group Model Driven Architecture, http://www.omg.org/mda/
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. **Jouault, F., Kurtev, I.**: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128--138. Springer, Heidelberg (2006)
9. **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)
10. **Sarıtaş, H. B., Kardas, G.,** Java Card Yazılımlarının Model Güdümlü Geliştirilmesi, Turkish Workshop on Model Driven Software Development, Bilkent Üniversitesi, Mayıs 22
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)