DSL4JavaCard: Java Card Platformu İçin bir Alana Özgü Dil

Miray Tosun[1,2] ve Geylani Kardaş[2]

[1]OBASE Bilgisayar ve Danışmanlık Tic. A.Ş, 34768, Ümraniye, İstanbul
miray.tosun@obase.com
[2] Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, 35100, Bornova, İzmir
geylani.kardas@ege.edu.tr

Özet. Java Card platformu ve beraberinde gelen uygulama programlama arayüzü akıllı kart uygulamalarının geniş kullanıma sahip Java programlama dili ile nesne yönelimli olarak geliştirilmesine imkan vermektedir. Ancak işlemci ve bellek yönünden oldukça kısıtlı olan bu kartlar için sadece Java dilinin bir alt kümesi kullanılabilmektedir ve geliştiriciler bilgisayar ve akıllı kart arasındaki paket alışverişi için alt seviye bir protokolün detay ve kısıtları ile uğraşmak durumundadırlar. Bu bildiride Java Card uygulamalarının geliştirilmesinde görülen bu zorlukları gidermek amacıyla kart üzeri uygulamaların model güdümlü geliştirilmesine imkan veren DSL4JavaCard isimli bir alana özgü dil tüm bileşenleri ile birlikte tanıtılmaktadır. Dilin soyut sözdiziminin dayandığı üstmodel ve modelleme aşamasında bir takım kısıtların kontrol edilmesini sağlayan somut sözdizimi tanıtıldıktan sonra DSL4JavaCard modellerinden Java Card uygulama kodlarının nasıl otomatik olarak elde edildiği anlatılmıştır. Ayrıca önerilen dilin kullanımı ve değerlendirmesi bir durum çalışması üzerinden yine bu bildiride yer almaktadır. Değerlendirmeler sonucunda DSL4JavaCard'ın ilgili literatürde yer alan önceki çalışmalardan farklı olarak özellikle Java Card güvenlik bileşenlerinin de model güdümlü geliştirilebilmesine imkan verdiği ve işletimsel semantiğinin temel Java Card bileşenlerinin tamamının, kullanıcı tanımlı ve iş uygulamasına özel öğelerin ise önemli bir kısmının otomatik üretilmesini sağladığı görülmüştür.

Anahtar Kelimeler: Alana özgü dil, Model güdümlü geliştirme, Akıllı kart, Java Card, DSL4JavaCard

1. Giriş

Kendine özel işlemcisi ve belleği olan ve verilerin üzerinde güvenli bir şekilde işlenmesini ve saklanmasını sağlayan akıllı kartlar günümüzde başta mobil iletişim ve bankacılık alanı olmak üzere çeşitli sektörlerde yaygın olarak kullanılmaktadır. Donanım özellikleri masaüstü bir bilgisayara kıyasla oldukça kısıtlı kalan bu kartlar üzerinde çalıştırılacak yazılımların geliştirilmesi de yine masaüstü uygulamalara göre oldukça zahmetli ve zaman alıcıdır. Yazılım geliştiricilerin alt seviye bir akıllı kart iletişim protokolüne (ISO/IEC 7816) [1] hakim olması, sınırlı bellek kaynakları

kullanmak amacıyla sadece çok ilkel veri yapılarını oluşturması, onaltılı (ing. "hexadecimal") yapıdaki veri paketlerini hazırlaması ve yine bu paketlerin onaltılı yapıda işlenmesini yazılımsal olarak sağlaması gerekmektedir.

İlk olarak Sun Microsystems tarafından geliştirilen ancak şu an Oracle tarafından geliştirilmesi ve dağıtımı sürdürülen Java Card yazılım çerçevesi ve platformu [2] yazılım geliştiricilere Java programlama dilini kullanarak akıllı kartlar için ISO/IEC 7816 standartına dayalı ve nesne tabanlı olarak yazılım geliştirme imkanını sunmakta ve yukarıda değinilen akıllı kart yazılım geliştirme zorluklarını azaltmaya önemli ölçüde yardımcı olmaktadır. Ancak akıllı kartların bellek kısıtları nedeniyle Java Card çerçevesi Java programlama dilinin çok küçük bir alt kümesini kapsamaktadır ve temel veri tipleri olarak sadece *boolean*, *short* ve *byte* kullanılabilmektedir. Tamsayılar, karakterler ve *String* yapıları Java Card içerisinde kullanılamamaktadır. Bunların dışında yine çok boyutlu diziler, dinamik sınıf yükleme, çöp toplayıcı, nesne serileştirme ve klonlama Java Card'da desteklenmemektedir. Üstelik onaltılı formattaki veri paketlerinin oluşturulması ve işlenmesi zorluğu devam etmektedir. Alana özgü dillerin (ing. "domain-specific language") (DSL) [3,4,5] geliştiricilere sağladığı soyutlama seviyesi, ifade gücü ve kullanım kolaylığı Java Card uygulamalarının geliştirilmesi sırasında karşılaşılan zorlukları gidermeye yardımcı olabilir. Test ve hata ayıklama imkanlarının çok kısıtlı olduğu bu sistemler için özellikle modelleme seviyesinde kontroller akıllı kart yazılımı geliştirmeyi kolaylaştırabilir. Ayrıca DSL'lerin bir çoğunun sunduğu otomatik kod üretimi özelliği de kart uygulamalarının daha hızlı ve etkin geliştirilmesini sağlayabilir. DSL'lerin tüm bu avantajları göz önüne alınarak bu bildiride Java Card uygulamalarının model güdümlü geliştirilmesini ve yazılım kodlarının otomatik üretilmesini sağlayan DSL4JavaCard isimli bir DSL tanıtılmaktadır.

DSL4JavaCard'ın sözdizimi Java Card çerçevesini tam olarak destekleyen bir üstmodele bağlı olarak geliştirilmiştir. İçerdiği bakış açıları akıllı kart temel yapıları haricinde güvenlik ve doğrulamaya ait varlıkları ve ilişkileri de desteklemektedir. Bu yönüyle literatürdeki bir çok benzer çalışmadan [6,7,8,9,10] farklılaşmaktadır. Somut sözdizim bir takım akıllı kart modeli kontrollerinin ve doğrulama kurallarının işletilmesiyle desteklenmektedir. Bunlara ek olarak DSL4JavaCard'ın işletimsel semantiği modellenen yazılımlara ait Java Card kodlarının otomatik olarak elde edilmesini sağlamaktadır. Tüm bu dil bileşenleri ve bunların entegrasyonu yukarıda değinilen ve bildirinin bir sonraki bölümünde anlatılacak olan literatürdeki diğer çalışmalar göz önüne alındığında DSL4JavaCard'a akıllı kart yazılımı geliştirme alanı için önerilen ilk DSL olma özelliğini kazandırmaktadır.

Bildirinin geriye kalan kısmında ilk olarak literatürdeki ilgili çalışmalar anlatılmıştır. Üçüncü bölümde DSL4JavaCard'ın soyut ve somut sözdizimi anlatılmaktadır. Dördüncü bölümde otomatik kod üretiminin sağlandığı işletimsel semantik anlatılmıştır. Beşinci bölüm bir durum çalışması üzerinden DSL4JavaCard'ın değerlendirmesini içermektedir. Elde edilen sonuç ve ileriye yönelik çalışmalar altıncı bölümde yer almaktadır.

2. İlgili Çalışmalar

Java Card platformu için model güdümlü yaklaşımın ve/veya Java Card yazılımlarının otomatik olarak üretilmesinin göz önüne alındığı ilk çalışmanın [6] olduğu söylenebilir. Bu çalışmada Java Card uygulamalarının ("Applet") üretilebilmesi ve denetlenebilmesi için hazırlanan bağımsız bir sertifikasyon mekanizması önerilmiştir. Önerilen yöntemin resmi bir tanımlaması ise [11]'te verilmiştir. DSL4JavaCard model kontrolünü ön tanımlı bir üstmodel aracılığı ile gerçekleştirdiğinden [6] ve [11]'ten ayrışmaktadır. Bonnet ve ark. kart üzeri yazılımların model güdümlü olarak kişiselleştirilmesini [12] ve bu çerçevenin akıllı kart konfigürasyonuna adaptasyonunu [13] önermektedir. Bizim çalışmamız ise akıllı kart konfigürasyonu ve kart donanımı üretiminden ziyade akıllı kart yazılımlarının model güdümlü geliştirilmesini desteklemektedir. Öte yandan [14]'te güvenli akıllı kart uygulamalarını UML temelli sistem modelleri üzerinden geliştirmek için model güdümlü bir yöntem önerilmiştir. [8]'de ise bu yöntemin Java Card'lar için bir uygulaması tanıtılmıştır. Ancak model dönüşümlerinin ve kod üretiminin nasıl uygulandığı bu çalışmalarda belli değildir. Ayrıca bu çalışmalar kart uygulamalarının otomatik geliştirilmesinden çok güvenli geliştirilmesi üzerinde durmaktadır.

B Metoduna dayalı olarak Java Card uygulamalarının geliştirilmesini öneren [15]'te Java Card platformuna özel iletişim ve kod ayrıntılarını birleştirmek için kodun elle değiştirilmesi gerekmektedir. [7]'deki çalışma [15]'teki yöntemi otomatik hale getirmeyi amaçlamaktadır ancak çalışma sadece fikirlerden ibarettir ve yazarlarının da belirttiği gibi otomatik kod üretimi için bu yöntemin bir araç desteğine ihtiyacı vardır.

DSL4JavaCard'in öncül bir çalışması olarak kabul edilebilecek [9]'da Java Card yazılımlarının model güdümlü geliştirilmesi için bir üstmodel ve bu üstmodele dayalı kod dönüşümleri tanıtılmıştır. Ancak bu çalışmada sadece temel kart bileşenlerinin model güdümlü geliştirilmesi söz konusudur. Java Card'ların istisna ve güvenlik bileşenlerinin geliştirilmesi göz önüne alınmamıştır.

[16], akıllı kart uygulamalarının model güdümlü mimariye göre platform bağımsız ve platforma özel modellenmesini göz önüne alsa da önerilen yöntem uygulandığında platform bağımlı diye adlandırılan modellerin aslında sadece platform bağımsız modellere ek alanlar katmaktan ibaret olduğu ve model güdümlü mimarinin önerdiği soyutlama seviyelerinin sağlanmadığı görülmektedir. Ayrıca bu çalışma sadece dosya tabanlı akıllı kart uygulamalarının geliştirilmesini kapsamaktadır.

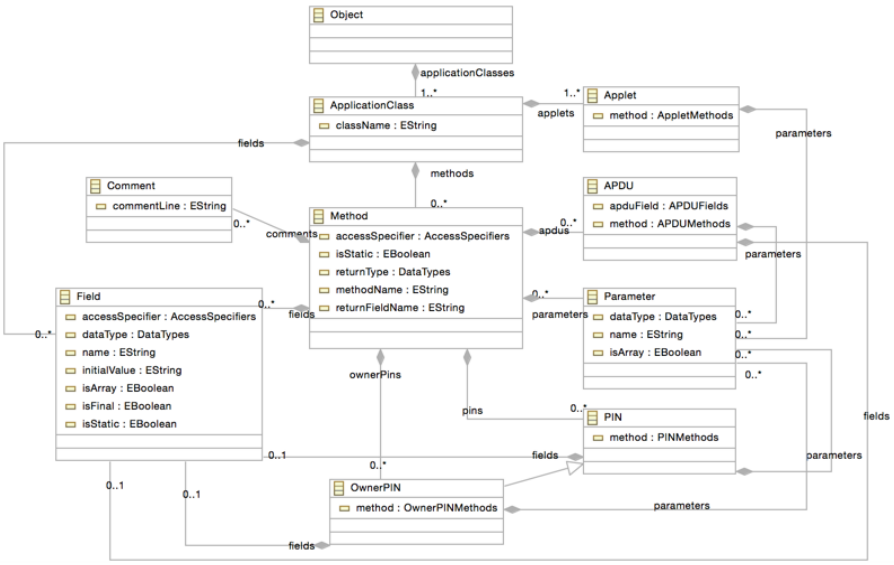
Akıllı kart yazılımlarının model güdümlü geliştirilmesi ve otomatik elde edilmesi kapsamında en güncel çalışmanın [10]'da gerçekleştirildiği görülmektedir. Bu çalışmada model güdümlü mimariye dayalı olarak kart üzeri yazılımların platform bağımsız olarak modellenmesi ve tanımlanan modelden modele ve modelden metne dönüşümler sonucunda bu modellerin aralarında Java Card'in da bulunduğu çeşitli akıllı kart işletim ortamlarında çalışacak şekilde kodlarının üretilmesi sağlanmıştır. Genel bir model güdümlü mimari sunan bu çalışmadaki Java Card üstmodeli de DSL4JavaCard'in içerdiği bakış açılarına sahip değildir ve sadece temel akıllı kart bileşenlerini desteklemektedir. PIN oluşturma haricindeki akıllı kart güvenlik ve şifreleme fonksiyonları [10]'da kapsam dışındadır.

3. DSL4JavaCard Sözdizimi

DSL4JavaCard'ın soyut sözdizimi geliştirdiğimiz bir üstmodele dayanmaktadır. Eclipse Ecore [17] tabanlı olarak geliştirilen üstmodel Java Card platformunun kart üzeri (ing. "on-card") bileşenlerini ve bunların birbirleri ile ilişkilerini tanımlamaktadır. Çok sayıda üstvarlık ve ilişkilerden oluşan üstmodelin daha rahat anlaşılması ve somut sözdizimi oluşturmanın kolaylaştırılması amacıyla üstmodel 3 adet bakış açısı (ing. "viewpoint") içermektedir. Java Card Çekirdek, İstisna ve Güvenlik adı verilen bu bakış açıları takip eden alt bölümlerde anlatılmaktadır. Son alt bölümde ise geliştiricilerin DSL4JavaCard sözdizimini kullanarak akıllı kart uygulama modelini oluştururken gerekli görsel ortamı sağlayan ve modelleme sırasında model elemanları üzerinde statik semantik kontrollerin ve doğrulamaların gerçekleştirilebildiği dil somut sözdizimi anlatılmıştır.

3.1 Çekirdek Bakış Açısı

Çekirdek bakış açısında (ing. "core viewpoint") herhangi bir Java Card uygulamasının geliştirilmesi sırasında ihtiyaç duyulan temel bileşenler ve bunların ilişkileri yer almaktadır. Şekil 1'de DSL4JavaCard üstmodelinin bu bakış açısına ait Ecore diyagramının bir kısmı verilmiştir. Yer kısıtları nedeniyle sadece önemli üstvarlık ilişkileri bu diyagramda gösterilebilmiştir.



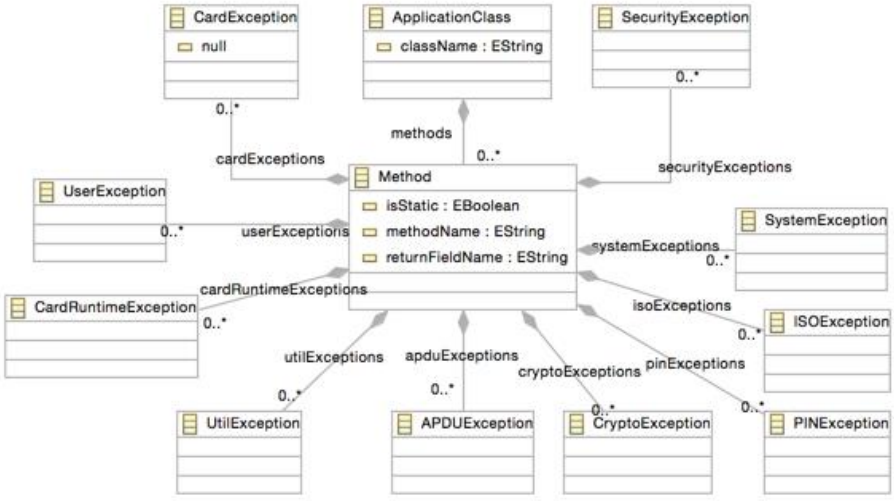
Şekil-1. DSL4JavaCard çekirdek bakış açısı kısmi üstmodeli

Üstmodelin anahtar öğesi her bir Java Card uygulamasını temsil eden *Applet* model elemanıdır. Her bir Applet, okuyucu tarafından gelen isteklere uygun cevapları geri döndürür. Uygulama Protokolü Veri Birimi (ing. "Application Protocol Data Unit") (*APDU*) ise ISO / IEC 7816 ile standartları belirlenmiş, akıllı kart ile okuyucu

arasındaki iletişimi sağlayan paket yapısını üstmodelde temsil etmektedir. Okuyucu tarafından bir APDU komutu gönderilir ve akıllı kart tarafından bu komuta cevap olarak yeni bir APDU paketi geri gönderilir. *PIN* ve bunu uygulayan *OwnerPIN* üstvarlıkları yine temel Java Card programlarında bulunan ve kart sahibinin kimlik doğrulamasını sağlayan model elemanlarıdır. *Object* üstvarlığı Java programlama dilinden gelen ve kart uygulamalarında temel nesnelerin oluşturulmasını sağlayan model elemanıdır. *ApplicationClass* Java Card uygulamasına özel *Method*, *Field* ve *Applet* bileşenlerini barındırmaktadır. Bir Java Card uygulamasında temel unsurlar o uygulamaya ait sınıfın oluşturulması ve gerekli ise sabitlerin belirlenmesi ve metotlara ayrıştırılması ile oluşur. Üstmodel içerisinde *ApplicationClass* bunu temsil etmektedir. *Comment*, *Parameter* ve *Field* üstvarlıkları *Method* üstvarlığına bağlı ve Method içerisinde kullanılan bileşenlerdir. Buna göre Comment üstvarlığının örnekleri metot içerisinde kullanıcıya not yazabileceği bir alan sağlamaktadır. Bununla birlikte *Parameter* üstvarlığı metotların girdilerini belirtebilmek amacı ile üstmodelde yer almaktadır. Son olarak metot içerisinde tanımlanmak istenen değişkenler Field bileşenleri ile oluşturulmaktadır.

3.2 İstisna Bakış Açısı

İstisna bakış açısı (ing. "exception viewpoint") Java Card API içerisinde tanımlanmış ve yaygın olarak kullanılan istisna sınıflarının ve ilişkilerinin DSL4JavaCard bünyesinde modellenmesini sağlamaktadır. Şekil 2'de bu bakış açısına ait önemli üstvarlıkları ve ilişkileri içeren kısmi üstmodel verilmiştir.



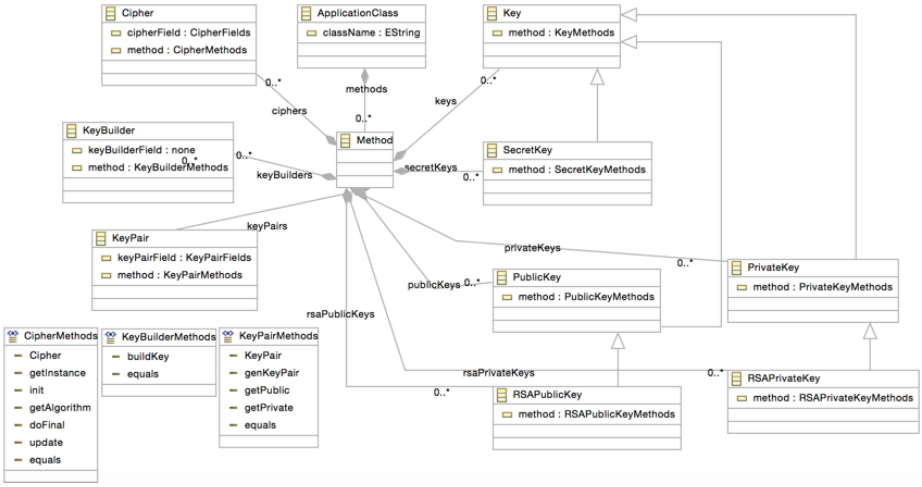
Şekil-2. DSL4JavaCard istisna bakış açısı kısmi üstmodeli

Exception bileşenleri üstmodel içerisinde Method üstvarlığı bünyesinde tanımlanabildiği gibi aynı zamanda metot oluşumu sırasında metodun istisna fırlatmasını da sağlayacak şekilde geliştiriciye sunulmuştur. Bu yapı ile kod

oluşturma sırasında farklı alternatifler ile modelde gerçeğe yakınlık sağlanması da amaçlanmıştır. En çok kullanılan istisna çeşitleri *APDUException*, *CardException*, *PINException* ve *ISOException*'dır. *APDUException*, APDU alışverişi sırasında karşılaşılan istisnalar için modelde yer almaktadır. *CardException* ise akıllı kart ile iletişimdeki sorunları modellemektedir. *PINException*, *OwnerPIN* üstmodel elemanının kimlik doğrulama işlemleri sırasında oluşan istisnaların yakalanmasını sağlamaktadır. *ISOException*, yine APDU alışverişi sırasında karşılaşılan istisnalar için kullanılmaktadır ve ISO 7816-4 cevap durumunu istisna kodu olarak içermektedir. Son olarak *CryptoException* ise güvenlik bileşenlerinde şifreleme işlemleri sırasında çıkabilecek olan istisna durumları için modelde yer almaktadır.

3.3 Güvenlik Bakış Açısı

Temel akıllı kart modelinin oluşturulmasına ek olarak DSL4JavaCard sözdizimi kart üzeri şifreleme ve güvenlik fonksiyonlarının sağlanması amacıyla kullanılabilecek üstvarlıkları ve bunların ilişkilerini de içermektedir. Söz konusu güvenlik bakış açısına (ing. "security viewpoint") ait önemli üstvarlıkları ve ilişkileri içeren kısmi üstmodel Şekil 3'te verilmiştir.

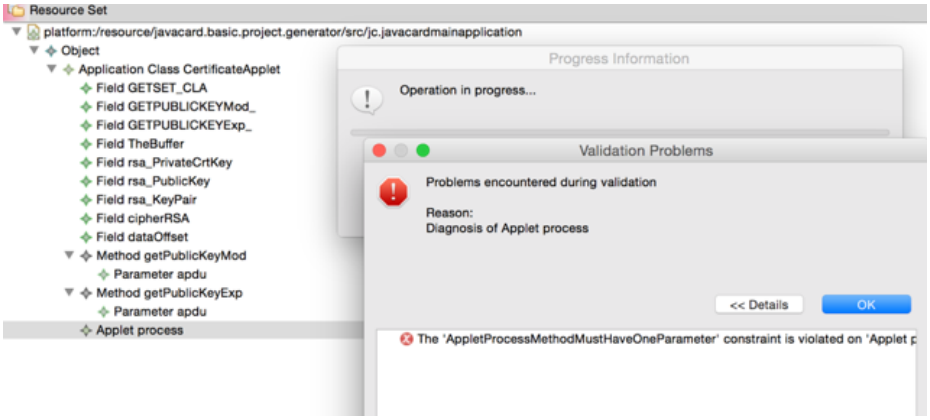


Şekil-3. DSL4JavaCard güvenlik bakış açısı kısmi üstmodeli

Java Card güvenlik bileşenleri yine kart yazılımına ait metotlar içerisinde tanımlanacak şekilde üstmodelde yer almaktadır. Örneğin *KeyPair* model elemanı anahtar çiftlerini oluşturmak için bir kapı görevi üstlenmiştir. İçerisindeki *getPublic* ve *getPrivate* metotları *PublicKey* ve *PrivateKey* döndürmektedir. *Cipher*, şifreleme (ing. "encryption") ve şifre çözme (ing. "decryption") gibi temel şifreleme işlemleri için kullanılan model elemanıdır. *KeyBuilder* ise model içerisinde anahtarların oluşmasını sağlar. Şekil 3'te görülen anahtar model elemanlarının (*Key*, *SecretKey*, *PublicKey*, *PrivateKey*, *RSAPublicKey*, *RSAPrivateKey*) oluşturulması *KeyBuilder*'ın *buildKey* servisinin çağrımı üzerinden sağlanmaktadır.

3.4 Somut Sözdizim

Önceki alt bölümlerde anlatılan DSL4JavaCard üstmodeline dayalı olarak akıllı kart yazılım modellerinin oluşturulabilmesi amacıyla geliştiricilere bir somut sözdizim sunulmuştur. Dilin statik semantiğinin sağlanması ve özellikle modelleme sırasında Java Card model kısıtlarının kontrol edilmesi ve bir takım doğrulama işlemlerinin yapılması amacıyla Eclipse EMF modelleri ile çalışabilen Xpand'e [18] dayalı bir modelleme ortamı hazırlanmıştır. Kısıt kontrolleri için Nesne Kısıt Dili (ing. "Object Constaint Language") (OCL) kullanılmıştır. Örneğin Şekil 4'te geliştirilen Java Card modelinin DSL4JavaCard kurallarına göre doğrulanması süreci sırasında oluşan bir hatanın (çağrılan metodun tanımlanması için eksik parametre girişi) geliştiriciye gösterildiği ekran görüntüsü verilmiştir.



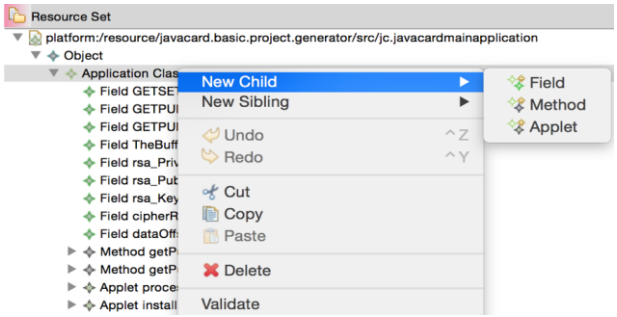
Şekil-4. Oluşturulan bir Java Card modelinin DSL4JavaCard kurallarına göre doğrulanması

Sonrasında doğru kod üretiminin gerçekleştirilmesi amacıyla modelleme sırasında (bir başka deyişle DSL4JavaCard somut sözdizimi kullanıldığı sırada) mevcut statik semantik kontrollerin sağlanması amacıyla yazılmış olan kurallardan bir örnek Şekil 5'te verilmiştir. Kullanıcı ApplicationClass örneğine bir isim verilmediyse şekilde yer alan kurallar içerisinde ilk sıradaki kural hata verecek şekilde oluşturulmuştur. Bir sonrakinde ise ApplicationClass içerisinde Applet model elemanına bağlı olan *process* ve *install* metotların çağırılması gerekliliği kontrol edilmektedir. Çünkü her Java Card uygulamasında bu çağrımlara ihtiyaç vardır.

```
class ApplicationClass
{
    attribute className : String[?];
    property fields : Field[*] { ordered composes };
    property methods : Method[*] { ordered composes };
    property applets : Applet[+] { ordered composes };
    invariant ApplicationNameSizeMustGreaterThenZero('ApplicatioClass Must Have Name To Identify The Class'):
    className.size()>0;
    invariant WarningAplicationClassFieldsCanUseWithAccessSpecifier('Warning : You can use fields with Access Specifier'):
    self.fields->forAll(f | f.accessSpecifier <> AccessSpecifiers::none);
    invariant ApplicationClassMustHaveAppletProcessMethods('All Java Card Application Uses Applet process Method'):
    self.applets->size()>0 and self.applets->forAll(e | e.method <> AppletMethods::process);
    invariant ApplicationClassMustHaveAppletProcessMethods('All Java Card Application Uses Applet install Method'):
    self.applets->size()>0 and self.applets->forAll(e | e.method <> AppletMethods::install);
}
```

Şekil-5. Bir statik semantik kontrolü örneği

Şekil 6'da ise geliştiricilere sunulan görsel modelleme ortamına ait bir ekran görüntüsü yer almaktadır. DSL4JavaCard'ın sözdizimine uygun bir şekilde bu modelleme ortamında kart yazılımına ait model oluşturulabilir. Modellemenin büyük bir kısmı model elemanlarına sağ tıklamak ve gelen menülerden o elemana ait içerebileceği yeni model elemanlarını oluşturmak ve/veya özellik değerlerini belirlemekten ibarettir. Böylelikle geliştiricinin özellikle kapsülleme veya DSL4JavaCard üstmodelindeki varlık ilişkilerine uymayan model kurgularına izin verilmemektedir. Yine modelleme ortamının bir diğer avantajı ise geliştiriciler kes-kopyala ya da oluşturulanı sürükle bırak yöntemi ile daha önceden oluşturulmuş olan model bileşenlerini istedikleri gibi taşıyabilirler. Ama taşıma, kopyalama ya da yapıştırma sırasında yine DSL4JavaCard model doğrulama kuralları otomatik olarak kontrol edilmektedir.



Şekil-6. DSL4JavaCard modelleme ortamına ait bir ekran görüntüsü

4. İşletimsel Semantik

Bir önceki bölümde anlatılan DSL4JavaCard sözdizime göre oluşturulan modellerin Java Card platformunda işletilebilmesi amacıyla DSL4JavaCard modellerini akıllı kartlar içerisine yüklenecek Java dosyalarına dönüştürmek gerekmektedir. Bu amaçla bir takım modelden metne dönüşüm kuralları bu çalışmada yine Xpand dili [18] kullanılarak hazırlanmıştır. Xpand EMF tabanlı modellerden kod üretilmesini sağlamaktadır. DSL4JavaCard modelleri de Ecore'a dayalı olduğundan Xpand'de yazdığımız dönüşüm kuralları bu modeller üzerinde çalışabilmektedir. Hazırlanan dönüşüm kurallarından bir kesit Şekil 7'de verilmiştir. Bir DSL4JavaCard modeli üzerinde bu kurallar işletildiğinde her bir model elemanına karşılık uygun Java kodlarının çıktı dosyasına yazılması sağlanmaktadır. Örnek kodda mavi renkli kısımlar kurala göre dosyaya yazılması gereken Java kodlarını gösterirken geriye kalan kodlar ise Xpand sözdizimi ile yazılmış dönüşüm kurallarını belirtmektedir.

Şekil 7'deki kurallar incelendiğinde Java Card Applet örneğinin girdi modeli üzerinde gezinme sonucunda bulunmasının ve modelde kurgulanmış ilişkilere göre de özellik ve metotlarının belirlenerek bunlara uygun Java Card kodlarının çıktı Java dosyasına yazılmasının sağlandığı görülmektedir. Dönüşüm kuralları şablon kod oluşturmanın ötesinde bir takım Java Card metodunun gövdesinin neredeyse

tamamını üretebilmektedir. Örneğin gelen APDU paketindeki komuta göre çalışması gereken metoda uygun diğer parametreler ile yönlendirmeyi sağlayan akıllı kart "process" metoduna ait "switch-case" bloğunun tamamı modelden koda dönüşümler sonucunda üretilebilmektedir.

```
«FILE className + ".java"»
public class «className» extends Applet {
«FOREACH methods AS m ITERATOR metIt»
    «FOREACH m.keyPairs AS k ITERATOR keyIt»
        «IF keyIt.counter1 == 1»
            «LET "KeyPair _" + m.methodName + k.method + ";" AS keyPairVar»
                «keyPairVar»«ENDLET»«ENDIF»«ENDFOREACH»«ENDFOREACH»
  «EXPAND Impl FOREACH fields»
   public static void install(byte[] bArray, short bOffset, byte bLength) {
   //install method automatically generated
     new «className»(bArray,bOffset,bLength);
 }
  protected «className»(byte[] bArray, short bOffset, byte bLength){
  //constructor method automatically generated
  register();
  }
  «EXPAND Meth FOREACH methods»
  «FOREACH applets AS app»
 «IF app.method.toString() != "install".toString()»
 «IF app.method.toString() == "process".toString()»
 public void «app.method»(«FOREACH app.parameters AS par SEPARATOR ','»«par.dataType» «par.name»«ENDFOREACH»){
 byte[] buffer = apdu.getBuffer();
 switch(buffer[ISO7816.OFFSET_INS])
 {//in the process method all functions are called
    «FOREACH methods AS m ITERATOR iter»
     case «iter.counter1» : «m.methodName»(«FOREACH m.parameters AS mpar SEPARATOR ','»«mpar.name»«ENDFOREACH»);
     return;
    «ENDFOREACH»
    default: ISOException.throwIt (ISO7816.SW_INS_NOT_SUPPORTED);
 }}«ELSEIF app.method.toString() == "select".toString()»
 public boolean «app.method»(){//  Perform any applet-specific session initialization.
 return true;}
 «ELSEIF app.method.toString() == "deselect".toString()»
 public void «app.method»(){
 // Perform appropriate cleanup.
 }«ELSE»«ENDIF»«ENDIF»«ENDFOREACH»
}«ENDFILE»
```

Şekil-7. DSL4JavaCard modellerinden Java Card dosyalarını otomatik oluşturan dönüşüm kurallarından bir örnek

Java Card geliştiricilerinin yukarıda anlatılan kod dönüşümü iç yapısını bilmesine gerek yoktur. Geliştiricilerin tek yapması gereken bir önceki bölümde anlatılan modelleme ortamını ve sözdizimi kullanarak DSL4JavaCard model örneklerini oluşturmaktır. Hazırlanan model örnekleri statik semantik kontrollerden ve doğrulamadan geçtikten sonra yukarıda anlatılan modelden koda dönüşüm işlemine girdi olarak alınmakta ve Java Card ortamında hatasız olarak derlenip çalıştırılabilecek Java dosyaları çıktı olarak elde edilebilmektedir.

5. Durum Çalışması ve Değerlendirme

DSL4JavaCard'ın Java Card uygulamalarını geliştirmede kullanılmasını değerlendirmek amacıyla bir durum çalışması gerçekleştirilmiştir. Çalışma uygulamalarda şifreleme amacıyla kullanılabilecek bir anahtar ikilisini akıllı kart içerisinde oluşturan ve bu anahtar çiftinden genel anahtar yapısında olanını (ing. "Public Key") akıllı kart ile ana bilgisayar (ing. "host") arasındaki bağlantı üzerinden güvenli bir şekilde istenildiğinde ana bilgisayara yollayabilen bir Java Card uygulamasının DSL4JavaCard kullanılarak geliştirilmesini kapsamaktadır. *CertificateApplet* adlı bu uygulamaya ait yazılım modelinin DSL4JavaCard sözdizimi

kullanılarak oluşturulması Şekil 8'de sol tarafta gösterilmiştir. Oluşturulan modelden dilin işletimsel semantiği kullanılarak otomatik olarak üretilen Java Card kodlarından bir parça ise yine aynı şekilde sağ tarafta verilmiştir. Uygulama için gerekli Java Card şifreleme ve güvenlik bileşenlerinin DSL4JavaCard sözdizimi ile modellenebilidği görülmektedir. Yine model içerisinde *Applet* bileşenine ait *process* ve *install* metotlarının yanı sıra geliştiricinin kendi oluşturabildiği *getPublicKeyExp*, vb. metotlar da yer almaktadır. Oluşturulan kodlar içerisinde de *PublicKey* ve ona ait metot çağırımlarının üretilebildiği; bunun yanı sıra, *APDU* kullanımına ait metotların da modele bağlı olarak üretildiği gözlenmektedir. Geliştirilen model üzerinde gezinme ve otomatik kod üretimi Intel Core i5 2,6 GHz, 8GB RAM ve 256 GB SSD sabit diske sahip bir masaüstü bilgisayarda 1 sn.'den daha kısa sürede tamamlanmıştır.

Şekil-8. *CertificateApplet* uygulamasına ait DSL4JavaCard modeli ve bu modelden otomatik üretilen Java Card kodları

DSL4JavaCard'ın söz konusu uygulamanın geliştirilmesindeki üretkenliğini değerlendirmek amacıyla durum çalışması sonucunda sadece otomatik kod dönüşümü ile elde edilen ve üzerinde henüz herhangi bir ekleme, çıkarma, vb. değişiklik yapılmamış Java Card kodları ile aynı uygulamanın (CertificateApplet) (https://community.oracle.com/thread/2195878) adresinde yer alan ve tam sürüm kaynak koduna herkesin erişebildiği Java Card kodlarının bir karşılaştırması gerçekleştirilmiştir. Belirtilen adresteki kodlar ilgili uygulama için gerekli tüm kodlardır ve doğrudan derlenip akıllı kart üzerinde çalıştırılabilecek özelliktedirler. Kod satır sayısı (ing. "Line of Code) (LOC) bazında karşılaştırıldığında DSL4JavaCard kullanılarak otomatik üretilen kodların tam sürüm uygulamanın %77'sini oluşturduğu; bir başka deyişle tüm programın %77'sinin sadece DSL4JavaCard modellemesi ile elde edilebildiği görülmüştür. LOC ölçümünün yanı

sıra üretilen kodun etkinliğini gözlemlemek amacıyla otomatik kod üretimi sonrasında akıllı kart uygulamasının hangi kısımlarının tam olarak üretilebildiği veya üretilemediği incelenmiştir. İnceleme sonucu Tablo 1'de verilmiştir. Buna göre temel Java Card bileşenlerinin tamamının, kullanıcı tanımlı ve iş uygulamasına özel öğelerin ise önemli bir kısmının DSL4JavaCard kullanılarak otomatik bir şekilde üretilebildiği görülmektedir.

Tablo 1. DSL4JavaCard kullanılarak Java Card kod bileşenlerinin üretilmesine ait değerlendirme sonuçları

Java Card kod bileşeni	**Otomatik kod üretiminin değerlendirilmesi**
İşleme (ing. "process"), yükleme (ing. "install") gibi temel Java Card applet metotları	Tamamı üretilebiliyor.
Kullanıcı tanımlı ve uygulamaya özel metotlar	Metot tanımlarının tamamı üretilebiliyor. Ancak kod üretimi sonrarında gövdelerinde önemli ölçüde eklentiye ihtiyaç duyuluyor.
Sabit veri tipleri	Tamamı üretilebiliyor.
Sınıf özellikleri ve metot değişkenleri	Tamamı üretilebiliyor.
İstisna nesneleri	Tamamı üretilebiliyor. Seçenekler ile istisna nesneleri farklı alternatiflerle oluşturulabiliyor. 20 farklı istisna sınıfını destekliyor.
Anahtarlama, şifreleme ve yetkilendirme gibi güvenlik bileşenleri	Tüm güvenlik metotları ve özellik tanımları otomatik üretilebiliyor. Ancak metot içerisinde kod üretimi sonrasında eklentilere ihtiyaç duyuluyor.
Açıklama alanları	Tamamı üretilebiliyor.

6. Sonuç ve İleriye Yönelik Çalışmalar

DSL4JavaCard akıllı kart yazılımlarını Java Card platformunda daha hızlı, daha etkin ve model güdümlü bir şekilde geliştirmeye yönelik bileşenler içeren bütünleşik bir DSL'dir[1]. Akıllı kart yazılım geliştiricileri bu dilin somut sözdizimini kullanarak sistem yazılım modellerini kart ortamının kısıtlarına uygun bir şekilde hazırlayabilmekte; hazırlanan bu modeller de otomatik modelden metne dönüşüme tabi tutularak Java Card platformunda işletilebilir yazılım kodları elde edilebilmektedir. Gerçekleştirilen durum çalışması otomatik üretilen kodların gereken tüm kodun oldukça büyük bir miktarının sadece modelleme ile elde edilmesine imkan verdiğini, yine üretilen kodun niteliksel olarak da sistem ihtiyaçlarını önümle ölçüde karşıladığını göstermiştir.

İleriye yönelik planlanan ilk çalışma değişik boyutlarda karmaşıklığa sahip kart yazılımlarının geliştirilmesinde DSL4JavaCard'in kullanımının daha yapısal bir şekilde değerlendirilmesi ve yeni değerlendirme sonuçlarına göre dilde

[1] Bildiride yer kısıtları nedeniyle tamamı verilemeyen DSL4JavaCard'in Ecore üstmodeli ve Xpand tabanlı somut sözdizim ve tüm kod üretim kuralları durum çalışmasına ait örnek akıllı kart modeli ve otomatik elde edilen Java Card kodları ile birlikte http://mas.ube.ege.edu.tr/downloads/dsl4javacard_bundle.rar adresinden temin edilebilinir.

iyileştirmelerin sağlanmasıdır. Bir diğer çalışma akıllı kart dışında kalan (ing. "off-card") ancak akıllı kart uygulaması için gerekli ana bilgisayar yazılımlarının hazırlanması için ek bakış açılarının ve dil yapılarının DSL4JavaCard'a eklenmesidir.

Kaynaklar

1. ISO/IEC 7816 Standards. ISO/IEC 7816 Standards family for Identification cards - Integrated circuit cards. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_tc_browse.htm?commid=45144 (son erişim: Haziran 2015)
2. Oracle Co. Java Card Technology. http://www.oracle.com/technetwork/java/embedded/javacard/ (son erişim: Haziran 2015)
3. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices, 35 (6): 26-36 (2000)
4. Mernik, M., Heering, J., Sloane, A.: When and how to develop domain-specific languages. ACM Computing Surveys, 37(4): 316-344 (2005)
5. Fowler, M.: Domain-specific Languages. Addison-Wesley Professional (2011)
6. Coglio, A. Code generation for high-assurance Java Card applets. In: 3rd NSA Conference on High Confidence Software and Systems. pp. 85-93 (2003)
7. Gomes, B.E.G., Moreira, A.M., Deharbe D. Developing Java Card Applications with B. Electronic Notes in Theoretical Computer Science, 184: 81-96 (2007)
8. 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 (2009)
9. Sarıtaş, H. B., Kardaş, G.: Java card yazılımlarının model güdümlü geliştirilmesi. Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi, 4:19-28 (2011)
10. Saritas, H. B., Kardas, G.: A model driven architecture for the development of smart card software. Computer Languages, Systems & Structures, 40(2): 53-72 (2014)
11. Coglio, A., Green, CA: Constructive Approach to Correctness, Exemplified by a Generator for Certified Java Card Applets. Lecture Notes in Computer Science, 4171:57-63 (2008)
12. Bonnet, S., Potonniee, O., Marvie, R., Geib, J-M.: A Model-Driven Approach for Smart Card Configuration. Lecture Notes in Computer Science, 3286: 416-435 (2004)
13. Bonnet, S., Marvie R., Geib J-M.: Putting Concern-Oriented Modeling into Practice. In: 2nd Nordic Workshop on UML, Modeling, Methods and Tools, Turku, Finland (2004)
14. Moebius, N., Stenzel, K., Grandy, H., Reif, W.: SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications. In: 4th International Conference on Availability, Reliability and Security, pp. 841-846 (2009)
15. Tatibouet, B., Requet, A., Voisinet, J.C., Hammad, A.: Java Card code generation from B specifications. Lecture Notes in Computer Science 2885: 306-318 (2003)
16. Nikseresht, A., Ziarati, K.: MDA Based Framework for the Development of Smart Card Based Application. In: 2011 International MultiConference of Engineers and Computer Scientist, pp. 1-6 (2011)
17. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)
18. Eclipse. Xpand: a statically-typed template language. https://eclipse.org/modeling/m2t/?project=xpand (son erişim: Haziran 2015)