Belge Sorumluluk İş-Birliği Modelleme Yöntemi ile Otomatik Kod Üretimi

Automatic Code Generation with Document Responsibility Collaboration Modelling Method

Tugkan Tuglular
Dept. of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
tugkantuglular@iyte.edu.tr

Onur Leblebici
Univera, Inc.
Izmir, Turkey
onur.leblebici@univera.com.tr

***Özet*—Güncel yazılım geliştirme yaklaşımlarında tasarım için yoğun olarak UML tercih edilmektedir. Ancak iş süreçlerindeki varlıklar ile UML tasarımlarındaki sınıflar arasındaki kavramsal boşluk az değildir. Anılan boşluğu azaltmak için, bu bildiride her iş için hayati önem taşıyan belgelerin kullanılması önerilmektedir. Aynı zamanda belge kavramını merkeze alan Belge Sorumluluk İş-Birliği adını verdiğimiz yeni bir yöntem sunulmaktadır. Bu yöntemde belgeler programlama açısından sınıfların meta-modelleri ve kalıcılık açısından ise ilişkilerin meta-modelleridir. Önerilen Belge Sorumluluk İş-Birliği yöntemi, etki alanlarının belgelerden oluştuğu ve bir etki alanında tanımlanmış bir belgenin, sorumluluğunu yerine getirmek için herhangi bir etki alanındaki başka bir belge ile birlikte çalışabileceği etki alanı kavramını kullanmaktadır. Belge Sorumluluk İş-Birliği yöntemi, analizden tasarıma geçiş aşamasından başlayarak uygulamanın ortaya konması aşamasına kadar olan süreci tanımlamaktadır. Önerilen yöntemi doğrulamak için sipariş yönetimi etki alanından bir örnek verilmiştir.**

Anahtar Kelimeler—model güdümlü geliştirme, alana özgü dil, sorumluluk güdümlü tasarım, sınıf sorumluluk iş-birliği

***Abstract*—UML is highly preferred for design in current software development approaches. However, the conceptual gap between entities in business processes and classes in UML designs is not small. To reduce this gap, this paper proposes using documents that are vital to every business. The proposed new method called Document Responsibility Collaboration puts the concept of documents at the center. In the proposed method, documents are meta-models of classes in terms of programming, and at the same time, they are meta-models of relationships in terms of permanence. The proposed Document Responsibility Collaboration method uses the domain concept in which domains are made up of documents, and a document defined in a domain can work with another document in any domain to fulfill its responsibility. Document Responsibility Collaboration method defines a process, which starts at the transition from analysis to design phase and continues to the code generation phase. An example from the order management domain is provided to validate the recommended method.**

Keywords—model-driven development, domain specific language, responsibility-driven design, class responsibility collaboration

I. GİRİŞ

Selic'in [1] belirttiği gibi, model kullanmanın potansiyel faydaları yazılımda diğer mühendislik disiplinlerinden daha fazladır ve model güdümlü geliştirme (MGG) yöntemleri bu fırsattan yararlanabilir. Fırsat, modellerin bir seviyeden diğerine dönüşümlerinde kolayca kullanılabilen otomasyonda yatmaktadır. MGG'de temel olarak üç model vardır: hesaplamadan bağımsız model (HBM), platformdan bağımsız model (PBM) ve platforma özgü model (PÖM). MGG bağlamında platform bir mimariyi veya uygulamayı ifade eder.

Solms ve Loubser [2] yazılım geliştirmede MGG'nin yaygın olarak benimsenmesini engelleyen açık sorunları listelemiştir:

- PBM'e dahil edilmesi gereken eserler dahil olmak üzere girdi ve çıktıların kesin bir tanımıyla birlikte iyi tanımlanmış, pratik bir tasarım yönteminin olmaması,
- Uygulama mimarilerini tanımlamak için mevcut standartların olmaması,
- Tasarımdan uygulamaya eşleme için gerekli yöntem ve teknolojilerin eksikliği.

Bu sorunları giderecek şekilde Belge Sorumluluk İş-Birliği (BSİ) adlı bir belge tabanlı model güdümlü tasarım yöntemi öneriyoruz. BSİ, temelde belge kavramına sahip bir tasarım şablonu belirtir. BSİ'de belge, XML Şeması Tanımı (XSD) tarafından tanımlanan bir XML dosyası olarak temsil edilen bir etki alanı (*İng. domain*) varlığıdır. BSİ'de kullanılan etki alanı kavramı, etki alanı güdümlü tasarım (*İng. Domain Driven Design*) yaklaşımındaki etki alanı kavramına karşılık gelmemektedir, bunun yerine tasarımcının seçimine göre anlamsal olarak gruplandırılmış varlıkları ifade eder. Bir etki alanı alt etki alanlarından oluşabilir; burada da alt etki alanları belgelerden oluşur ve bir alt etki alanındaki bir belge, sorumluluğunu yerine getirmek için herhangi bir alt etki alanındaki başka bir belgeyle birlikte çalışabilir. Örneğin, bir e-ticaret etki alanının parçası olabilen sipariş yönetimi etki alanı, bu bildiride örnek olarak kullanılmaktadır. Sipariş yönetimi alt etki alanındaki sipariş belgesi, envanter yönetimi alt etki alanındaki stok belgesi ile birlikte çalışır.

Önerilen BSİ yöntemi, yazılım geliştirme yaşam döngüsünün analizden tasarıma geçiş aşaması ile başlar. BSİ,

978-1-7281-8541-5/20/$31.00 ©2020 IEEE

analizden tasarıma geçiş için bir arayüz tanımlamaktadır. Bu çalışmada kullanıcı gereksinimlerini toplamak için kullanıcı hikayesi yaklaşımı takip edilmiştir. Bir kullanıcı rolü, kullanıcı hikayelerinde ifade edilen yetkinlikleri ile tanımlanmaktadır. Kullanıcı hikayelerindeki roller, BSİ'deki rollerle eşleşir. BSİ'de roller, belgeler üzerinde oluşturma, okuma, güncelleme ve silme işlemlerini gerçekleştirmektedir. Bu işlem talepleri ile ilgili olarak, bir belgenin ilgili sorumlulukları yerine getirmesi beklenmektedir. Örneğin, bir sipariş belgesinin yaratma talebi üzerine sorumluluğu, oluşturma işleminden önce dikkate alınan hesap için ödeme için bekleyen bir sipariş olup olmadığını kontrol etmek olabilir. Görüldüğü gibi, BSİ olay güdümlü bir yöntemdir.

Bu bildiride, BSİ yöntemi ile birlikte NoSQL desteği olan, olaya güdümlü, nesne yönelimli bir uygulama mimarisi kullanmayı ve bir BSİ tasarımından bu mimariye otomatik bir dönüşüm gerçekleştirme önerilmektedir. BSİ, bir belge için olayları, sorumlulukları ve ilgili iş akışını ortaya koymakla beraber, bir sorumluluğu yerine getirmek için gerekli iş mantığını temsil etmek ve sonrasında oluşturmak için bir araç sağlamamaktadır. Otomatik olarak oluşturulan kodda, iş mantığı kısmı geliştirici tarafından tamamlanmak üzere bırakılır. Her ne kadar anılan iş mantığını, örneğin OCL gibi bir dil aracılığıyla temsil etmek mümkün olsa da, böyle bir yaklaşım BSİ yöntemini karmaşıklaştıracak ve anlaşılması ile kullanımını zorlaştıracaktır.

BSİ yöntemini kullanmanın avantajları iki yönlüdür. İlk olarak, bir proje veya bir tasarım süreci; müşteriler, analistler, mimarlar, geliştiriciler ve test uzmanları gibi tüm taraflarca bilinen belge kavramı ile düzenlenir. İkincisi, otomatik kod üretimi sayesinde yüksek kod kalitesi sağlanabilir. BSİ ile klasör / paket yapısı, sınıf yapısı, NoSQL belge yapısı proje içerisinde standart hale gelir. Bu standardizasyon sayesinde tasarım ve uygulama eserleri arasındaki izlenebilirlik kolaylaşır. Gelecekte, BSİ'nin getirdiği API, müşteri kodu ve test komut dosyası avantajlarını ele almayı planlıyoruz.

Bildirinin geri kalan kısmında MGG ve sorumluluk güdümlü tasarıma ilişkin temel bilgiler Bölüm II'de ilgili çalışmalar ile birlikte verilmiştir. Bölüm III'te önerilen BSİ yöntemi açıklanmıştır. İzleyen bölümde sipariş yönetimi etki alanının BSİ tasarımını sunulmuş ve Microsoft .Net Core ve MongoDB kullanan örnek bir mimarideki uygulamaya dönüşümü anlatılmıştır. Son olarak, Bölüm V'te sonuçlar ile gelecek çalışmalar açıklanmıştır.

II. İLGİLİ ÇALIŞMALAR

A. Model güdümlü geliştirme

Schmidt [3] model güdümlü mühendisliği (MGM), etki alanlarını modellerde etkili bir şekilde ifade etmek için kullanılan bir yaklaşım olarak tanımlamaktadır. Model güdümlü geliştirme, MGM'nin özel bir şeklidir [4]. MGG'de amaç, model yoluyla iş sürecini temsil etmek ve model yürütme, model dönüşümü veya kod üretimi gibi tekniklerle otomasyonu kolaylaştırarak bunları yazılım uygulamalarına dönüştürmektir [2].

Nesne Yönetim Grubu (Object Management Group-OMG) MGG'nin uygulanmasını desteklemek için model güdümlü mimariyi ortaya koymuştur [5]. Model güdümlü mimari, soyut bir tasarımın uygulamaya dönüştürülmesi için gerekli üç modeli tanımlamıştır [6]:

- Hesaplama bağımsız model, teknolojik detayları dikkate almadan etki alanı kavramlarını tanımlar.
- Platformdan bağımsız model, HBM'i hesaplama yönleriyle zenginleştirir.
- Platforma özgü model, belirli bir teknoloji platformuna özgü uygulama yönleriyle PBM'i zenginleştirir.

HBM ile gereksinimler modellendikten sonra, bu süreçteki bir sonraki adım, platforma özgü herhangi bir ayrıntı eklemeden etki alanının temel kavramlarını yakalaması gereken PBM'i oluşturmaktır [7]. PÖM ilgili standartların eksikliği nedeniyle, model güdümlü mimari araçları genellikle yalnızca Java EE ve Microsoft.Net [2] gibi belirli referans mimarileri veya platformları üzerindeki eşleşmeleri destekler. BSİ yöntemi, PÖM'e geçerken PBM'den doğrudan işletilebilir .Net koduna dönüşüm sağlar.

Braek ve Melby [8] PBM'ler için asgari gereksinimi şu şekilde tanımlamıştır:

i. Platformdan bağımsız ve dolayısıyla teknolojiden bağımsız
ii. İnsanlar tarafından anlaşılabilir ve bakımı yapılabilir
iii. Uygulama doğrulamasını kolaylaştırmak için analitik
iv. Gerçek dünyayı gerçekçi bir şekilde modelleyebilmesi açısından gerçekçi

Önerilen BSİ yöntemi aşağıda sıralanan yetkinlikleri sayesinde yukarıdaki gereksinimleri karşılar:

(i) platformdan bağımsız unsurlardan oluşan belge kavramı,

(ii) CRC diyagramı veya UML sınıf diyagramı benzeri bir belge ve etki alanı diyagramı,

(iii) PBM'in XSD tabanlı meta modeli ve

(iv) belge akışları yoluyla iş sürecinin gerçekçi kavramsallaştırılması.

Kleppe ve diğ. [5], model dönüşümü şöyle tanımlamıştır:

- Dönüşüm, bir dönüşüm tanımına göre kaynak modelden hedef modelin otomatik olarak üretilmesidir.
- Dönüşüm tanımı, kaynak dildeki bir modelin hedef dildeki bir modele nasıl dönüştürülebileceğini tanımlayan bir dizi dönüşüm kuralıdır.
- Bir dönüşüm kuralı, kaynak dilde bir veya daha fazla yapının hedef dilde bir veya daha fazla yapıya nasıl dönüştürülebileceğinin bir açıklamasıdır.

Mens ve Van Grop [9] bir model dönüşümünün birden çok kaynak model ve / veya birden fazla hedef model için de geçerli olması için bu tanımların genelleştirilmesi gerektiğini öne sürmüştür. Bunun yaygın bir örneği, bir PBM'i alıp onu bir dizi PÖM'e çeviren bir dönüşümdür [9]. Model güdümlü mimariyi kullanacak şekilde bir dönüşüm, aslında belirli bir platform için bir PÖM üretmek üzere bir PBM'in dönüştürülmesi için kurallar ve diğer bilgileri içeren bir şartnamedir [10].

Model dönüşümleri için geliştirilmiş olan alana özgü diller bulunmaktadır [11]: Sorgular - Görünümler - Dönüşümler (QVT) [12], Model Dönüşüm Dili (MTL) [13] ve ATLAS

Dönüşüm Dili (ATL) [14]. Bu alana özgü özel diller olmadan, model dönüşümleri XSLT ile, Java veya .Net gibi üst düzey programlama dilleri kullanılarak da yapılabilir. Önerilen BSİ yöntemi, .Net'te geliştirilen bir model dönüşüm programı kullanmaktadır.

B. Sorumluluk güdümlü tasarım

Wirfs-Brock ve Wilkerson [15], sorumluluk güdümlü tasarımın (*İng. Responsibility Driven Design*) amacını açıklarken bir programı istemci / sunucu modeli ile canlandırmıştır. Onlara göre sorumluluk güdümlü tasarımın amacı, bir istemcinin istekleri üzerine sunucunun hizmetler sunmasıdır. İstemci / sunucu etkileşimi, her iki tarafın da yerine getirmesi gereken bir sözleşme ile belirlenir. Sorumluluk güdümlü tasarım, istemci / sunucu modelinden ilham alır ve aşağıdaki soruları sorarak sözleşmeye odaklanır [15]:

Bu nesne hangi eylemlerden sorumludur?

Bu nesne hangi bilgileri paylaşıyor?

Sorumluluk güdümlü bir tasarım nesne yönelimli programlama kullanılarak hayata geçirilecek ise, sınıflar bilgiyi saklamak ve bir eylem gerçekleştirmek için kodlanmalıdır [16]; burada bilgi ile nesnenin durumu ve eylem ile bir yöntem veya bir dizi yöntem kastedilmektedir.

Sınıf Sorumluluk İş-Birliği (*İng. Class Responsibility Collaboration-CRC*), eş zamanlı olarak iki ayrı araştırma grubu tarafından [15], [17] sunulan nesne yönelimli bir analiz ve tasarım yöntemidir. Bu yöntem sadeliği nedeniyle oldukça popüler olmuştur [18]. CRC yöntemi, CRC kartları olarak adlandırılan özel olarak tasarlanmış kartlar aracılığıyla kullanılır. CRC kartlarının ana bölümleri şunlardır: sınıfın adı, sınıfın sorumlulukları ve sınıfın iş birlikleri. Bazı CRC sürümleri ayrıca rolleri, nitelikleri, metotları ve hatta varsa süper sınıfını içerir. CRC yöntemi, sorumluluk güdümlü tasarımın ayrılmaz bir parçası haline gelmiştir.

Önerilen BSİ yöntemi, CRC yöntemini genişletmektedir. Özellikle iş süreçlerindeki varlıkların, nesnelere sığmayacak kadar büyük kavramlar olduğu görülmektedir. Bu nedenle, herhangi bir iş süreci için doğal olan ve bu nedenle problem etki alanı ile çözüm etki alanı arasındaki kavramsal boşluğu azaltan belge kavramı kullanılmıştır. Belgeler, belirli bir amaca yönelik olarak çalışan nesne kompozisyonlarıdır. BSİ yönteminde belge kavramı için bir meta-model sunulmuştur.

III. BELGE SORUMLULUK İŞ-BİRLİĞİ MODELLEME YÖNTEMİ

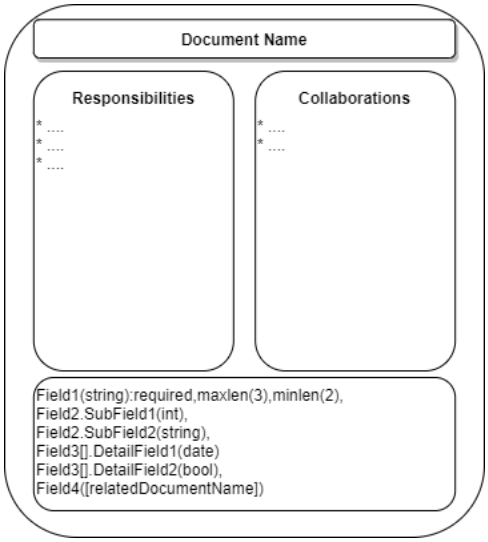
BSİ yöntemi MGG'nin analizden tasarıma geçiş aşamasıyla başlar. Bu nedenle, BSİ'daki ilk adım PBM'i tanımlamaktır. Meservy ve Fenstermacher [7], platforma bağımlı modellemenin, koda nihai dönüşümü destekleyecek kadar kesin olma gerekliliğine ek olarak, birçok farklı alanın anlambilimini yakalayabilmesi için yeterince genel bir temsil gerektirdiğini belirtmiştir. Bu bağlamda BSİ için iki eser sunulmuştur:

i. birçok farklı alanın anlambilimini yakalayacak kadar genel olan BSİ için bir tasarım şablonu,

ii. BSİ için koda dönüşümü destekleyecek kadar kesin olan bir meta-model ve dönüşüm kuralları.

A. BSİ tasarım şablonu

Bir BSİ belgesi için tasarım şablonu Şekil 1'de sunulmaktadır. Bir BSİ belgesi, verileri ve işlemleri, sözleşmelerini ve iş akışlarını saklar. BSİ belge tasarım şablonunda; Ad, Sorumluluklar, İş-Birlikleri ve Alanlar (*İng. Fields*) olarak adlandırılan dört bölüm tanımlanmıştır. BSİ tasarım şablonu Şekil 1'de gösterilmiştir.



Şekil 1. BSİ tasarım şablonu

Ad bölümü bir belgeyi temsil eder. İsimler açık ve belgenin misyonuna uygun olmalıdır. Alan bölümü, geçerlilik kuralları ile birlikte nitelik ve/veya ilgili veriler oluşur. Alanlar sorumluluklarla uyumlu olmalı ve sorumluluklar iş birlikleriyle tutarlı olmalıdır. Ancak, bir alanın sorumluluklarla doğrudan bir bağlantısı olması gerekmez. Örneğin, belgeye Açıklama adlı bir alan eklenebilir ve bu alan herhangi bir sorumlulukla ilgili olmayabilir, yalnızca belgedeki veriler olarak da bulunabilir.

Sorumluluklar, bir belgeye iliştirilen iş işlevlerini temsil eder. Bu işlevsellikler bir yaratma, güncelleme, silme ve saklama işlemlerinden önce veya sonra tetiklenebilir. İki tür sorumluluk tanımlanmıştır. Biri doğrudan bir belge ile ilgilidir ve ikincisi ise belge üzerinde bazı işlemleri gerçekleştirmek için başka bir belgeyi dinler. Örneğin, stok seviyesini düşürmek için sipariş belgesi dinlenmelidir. Daha sonra bir sipariş alındığında, karşılık gelen ürünler stoktan çıkarılmalıdır. Sorumluluğun adı kolay ve açıkça anlaşılabilir bir şekilde yazılmalıdır.

İş-Birlikleri, sorumluluklar ve alanlar kontrol edilerek belirlenir. Örneğin, stok belgesinin sipariş belgesini dinleme sorumluluğu vardır ve bu nedenle stok siparişle iş birliği yapar. İş-Birliği bölümü, belgeler arasındaki etkileşimleri ve bunların bağımlılıklarını gösterir. İş-Birlikleri bölümünde az bağlantı olması düşük bağımlılık açısından, aynı sınıflar arası bağımlılıkların düşük olması gibi, önemlidir.

B. BSİ dönüşümü

Bir modelden başka bir modele eşleme ve dönüşüm genel olarak şartname ve dönüşüm kuralları ile açıklanır [19]. Önerilen BSİ yöntemi, PBM'i .Net koduna dönüştürür. Dönüşüm için şartname, Şekil 2'de özetlenen BSİ XSD kullanılarak tanımlanmıştır:

Belge / Alanlar: Alanlar, belgelerin nitelikleri ve / veya ilgili verileridir. Tamsayı gibi ilkel tipler olabilir veya karmaşık tipler olarak oluşturulabilirler. İlgili verilerin bir liste olarak gösterilmesi gerekiyorsa, ayrıca bir koleksiyon yapısı da mevcuttur. Örneğin, sipariş belgesinde ilkel sipariş tarihi

alanı, adres karmaşık türü alanı, satır ayrıntıları toplama alanları bulunur. Alanlarda ayrıca doğrulama ek açıklamaları da vardır.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Document" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:element>

    <xs:complexType name="PrimitiveFieldType" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:complexType>

    <xs:simpleType name="FieldType" xmlns="http://www.w3.Permissionorg/2001/XMLSchema">…
    </xs:simpleType>

    <xs:simpleType name="MeasurementType" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:simpleType>

    <xs:complexType name="ValidationAnnotationType" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:complexType>

    <xs:simpleType name="ValidationType" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:simpleType>

    <xs:complexType name="ComplexFieldType" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:complexType>

    <xs:complexType name="DetailsFieldType" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:complexType>

    <xs:simpleType name="DocumentObservationEvents" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:simpleType>

    <xs:simpleType name="DocumentDeleteOption" xmlns="http://www.w3.org/2001/XMLSchema">…
    </xs:simpleType>

</xs:schema>
```

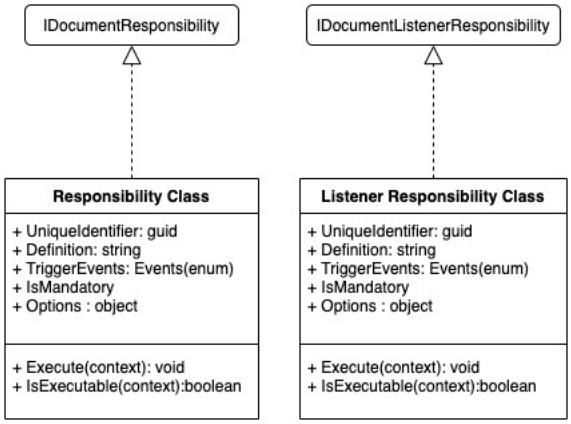
Şekil 2. BSİ XSD özeti

Belge / Sorumluluklar: BSİ, bir sorumluluk için aşağıdaki özellikleri tanımlar:

- Bir sorumluluk sadece bir belgeye aittir, bir belgenin birden fazla sorumluluğu olabilir.
- Her sorumluluk zorunlu olup olmadığını belirtmelidir.
- Her sorumluluk bir veya daha fazla tetikleyici olay belirtmelidir. Mevcut tetikleyici olaylar:
  - Oluşturmadan önce
  - Oluşturduktan sonra
  - Güncellemeden önce
  - Güncellemeden sonra
  - Silmeden önce
  - Silmeden sonra
  - Başarılı saklama
  - Başarısız saklama

Olayları tetikleme ile ilgili işletme sırası önemliyse, sorumluluk zincir endeksi (*İng. chain index*) tanımlamalıdır. Belge ile ilişkilendirilmiş olan sorumlulukların bazıları diğerlerinden önce, sonra veya en son çalışmasının istenildiği durumlar olabilmektedir. Bu ihtiyacın oluştuğu durumlarda önceliği belirtmek için zincir endeksi bilgisi kullanılmaktadır. Sorumlulukları birbirinden bağımsız ama sıralamalarının fark yaratabileceği özel durumlarda sisteme sıralamayı belirtmek için zincir endeksi alanı kullanılır. Örneğin, yeni bir sipariş oluşturulduğunda çalışacak olan stok denetimi ve depo çıkış isimli iki sorumluluk olduğunu varsayalım. Depo çıkış işlemi yapmadan önce stok denetimi yapılması gerektiği düşünülürse, uygulamaya bu bilgiyi vermek için zincir endeksi alanı kullanılır.

Sorumluluk sınıfı için PÖM Şekil 3'te verilmiştir. Dönüşüm kuralları yardımıyla PÖM, BSİ Belge Modeli ve BSİ Sorumluluk Modeli kullanılarak oluşturulur. XSD üzerinde verilen elemanları kaynak koda çevirmek için yorumlayıcı tasarım şablonu (*İng. interpreter design pattern*) ile XML üzerindeki (XSD içinde tanımlanan) her bir elemana karşılık gelen kod oluşturma sınıfları aracılığı ile belgelere ait XML dosyaları otomatik olarak kaynak kodlara çevrilmektedir. BSİ tarafından kullanılan dönüşüm kurallarına örnek olarak Şekil 4'te sipariş belgesine BSİ dönüşüm kurallarının uygulanması gösterilmiştir ve sonuçta .Net kodu otomatik olarak ortaya çıkmaktadır.



Şekil 3. BSİ sorumluluk sınıfı PÖM

`<Document Name="Order" Subdomain="DARC.SalesManagement">`	namespace DARC.SalesManagement
`<Document Name="Order" Subdomain="DARC.SalesManagement">`	[Document("Order")] public class Order : DocumentBase
`<PrimitiveField Name="OrderDate" Type="DateTime" DefaultValue="@"> <Validation Type="Required" Value="true" /> </PrimitiveField>`	[Required] [DateTimeMember(DefaultValue = 0)] public DateTime OrderDate { get; set; }
`<DetailsField Name="Lines" Type="OrderLine"> … <Validation Type="Required" Value="true" /> </DetailsField>`	[Required] [DocumentDetailMember] public DocumentDetailElementCollection<OrderLine> Lines { get; set; }
`<PrimitiveField Name="OrderedBy" Type="String"> <Validation Type="Required" Value="true" /> </PrimitiveField>`	[Required] [StringMember] public string OrderedBy { get; set; }
`<PrimitiveField Name="Status" Type="EOrderStatus" DefaultValue="WaitingForPayment"> <Validation Type="Required" Value="true" /> </PrimitiveField>`	[Required] [EnumMember(DefaultValue = EOrderStatus.WaitingForPayment)] public EOrderStatus Status { get; set; }
`<PrimitiveField Name="TotalPrice" Type="decimal"> </PrimitiveField>`	[DecimalMember] public decimal TotalPrice { get; set; }
`<ComplexField Name="DeliveryAddress" Type="Address"> … <Validation Type="Required" Value="true" /> </ComplexField>`	[Required] [ComplexTypeMember] public Address DeliveryAddress { get; set; }
`<DetailsField Name="Lines" Type="OrderLine"> … </DetailsField>`	public class OrderLine : DocumentDetailElementBase
`<PrimitiveField Name="ProductIdentifier" Type="Relation" RelatedDocument="Product"> <Validation Type="Required" Value="true" /> </PrimitiveField>`	[Required] [RelationMember(nameof(Product))] public string ProductIdentifier { get; set; }
`<PrimitiveField Name="Quantity" Type="Double"> <Validation Type="Required" Value="true" /> <Validation Type="MinValue" Value="0" /> </PrimitiveField>`	[MinValue(0)] [Required] [DoubleMember] public double Quantity { get; set; }
`<PrimitiveField Name="Price" Type="Decimal"> <Validation Type="Required" Value="true" /> <Validation Type="MinValue" Value="0" /> </PrimitiveField>`	[MinValue(0)] [Required] [DecimalMember] public decimal Price { get; set; }
`<DetailsField Name="Discounts" Type="OrderLineDiscount"> <PrimitiveField Name="Ratio" Type="Double"> </PrimitiveField> </DetailsField>`	public class OrderLineDiscount : DocumentDetailElementBase { [DoubleMember] public double Ratio { get; set; } }
`<ComplexField Name="DeliveryAddress" Type="Address"> <PrimitiveField Name="Description" Type="String"> </PrimitiveField> <ComplexField Name="Location" Type="GeoLocation"> … </ComplexField> <Validation Type="Required" Value="true" /> </ComplexField>`	public class Address : ComplexTypeElementBase { [StringMember] public string Description { get; set; } [ComplexTypeMember] public GeoLocation Location { get; set; } }
`<ComplexField Name="Location" Type="GeoLocation"> <PrimitiveField Name="Latitude" Type="Long"> </PrimitiveField> <PrimitiveField Name="Longitude" Type="Long"> </PrimitiveField> </ComplexField>`	public class GeoLocation : ComplexTypeElementBase { [LongMember] public long Latitute { get; set; } [LongMember] public long Longitute { get; set; } }

Şekil 4. BSİ belge sınıfı PÖM

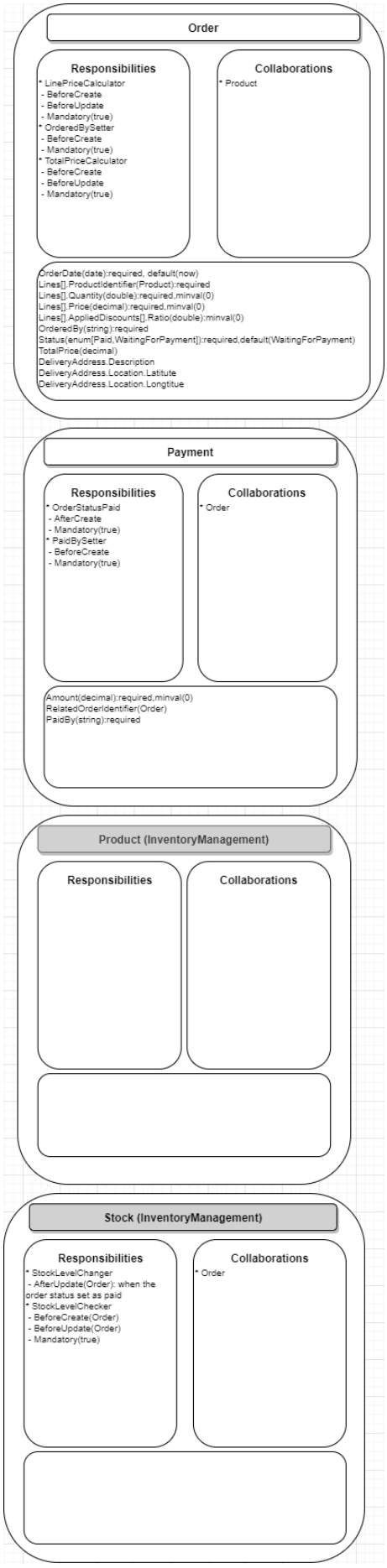
IV. ÖRNEK ÇALIŞMA

Bu bölümde yaklaşımımızı göstermek için bir e-ticaret işi ile ilgili bir örnek çalışma sunulmuştur. Örnek çalışmada yalnızca satış yönetimi ve envanter yönetimi etki alanlarıyla ilgili kullanıcı hikayelerini kullanılmış ve dolayısıyla bu iki etki alanın belgeleri BSİ belge şablonuna göre modellenmiştir. En son adımda, sipariş belgesinin .Net koduna dönüştürülmesi açıklanmıştır. Sayfa sınırından dolayı bir sipariş iş süreci için sınırlı sayıda kullanıcı hikayesi seçilmiştir:

- Yönetici kullanıcısı olarak, ürünü müşteri müşterilerinin satın alabilmesi için tanımlamalıyım.
- Yönetici kullanıcısı olarak ürünü isim, fiyat, vergi oranıyla tanımlamalıyım.
- Yönetici kullanıcısı olarak, ürün kategorilerini tanımlamam ve bu kategorilere ürün eklemeliyim, böylece kullanıcı ürünleri kategorilere göre arayabilir.
- Yönetici kullanıcısı olarak, müşterilerin dikkatini çekebilmek için ürünler için indirimler tanımlamalıyım.
- Yönetici kullanıcısı olarak, stokları ve yeni stok siparişlerini kontrol edebilmem için ürünlerin stok seviyelerini ayarlamalıyım.
- Müşteri kullanıcısı olarak, ürünleri alışveriş sepetime ekleyebilmem için listelemeliyim.
- Müşteri kullanıcısı olarak, alışveriş arabama izin verilen miktarda ürünü eklemeliyim, böylece satın alabilirim.
- Müşteri kullanıcısı olarak, toplam tutar ve indirimler için alışveriş özetimi görüntülemeliyim.
- Müşteri kullanıcısı olarak, alışveriş sepetimi ödemeliyim, böylece mallarımı satın alırım.

Müşteri kullanıcıları ürün satın almak istediklerinde, sipariş belgesi bu işlem için kullanılır. Şekil 5'te verilen satış yönetimi etki alanın içinde çok temel bir sipariş modeli oluşturulmuştur. Bir etki alanı bir dizi uyumlu belgeden oluşmaktadır. Uyum, belgeleri bir etki alanına gruplarken önerdiğimiz ölçüttür. Kullanıcı öykülerinde tanımlanan gereksinimleri karşılamak için tamamlayıcı bir envanter yönetimi etki alanı da gereklidir. Envanter yönetimi etki alanı, sipariş belgesinin birlikte çalıştığı belgeleri de içerir.

Sipariş belgesinin bu modelde tanımlanan sorumluluklarını inceleyelim. Siparişin bilgilerinin saklanması en temel sorumluluktur. Aynı zamanda, yeni bir sipariş kaydı oluşturulduğunda, ürünün stok seviyesini güncelleyen bir sorumluluk Stok modelinde görülebilir. Örneğin, bir müşteri yeni bir sipariş kaydı oluşturmak istediğinde, LinePriceCalculator sorumluluğu ile satın alınacak ürünler için fiyat hesaplamaları yapılır, kimliği doğrulanmış kullanıcı, bu siparişi OrderedBySetter aracılığıyla yapan kişi ve siparişin toplam fiyat bilgileri olarak ayarlanır. Sipariş TotalPriceCalculator üzerinden hesaplanır ve belgede ayarlanır. Ardından, Stok modelinde belirtilen StockLevelChecker sorumluluğu, alınan ürünlerin stok seviyelerini kontrol eder ve yetersiz stoklar varsa, kullanıcı uyarılır.



Şekil 5. Satış yönetimi etki alanı

Müşteri bir siparişi güncellemek istediğinde, bu sefer LinePriceCalculator, TotalPriceCalculator sorumlulukları devreye girer ve böylece yeni eklenen, kaldırılan veya değiştirilen ürünler varsa, bunlarla ilgili hesaplamalar tekrar yapılır. Daha sonra Stok modelinde tanımlanan StockLevelChecker tekrar etkinleştirilir ve satın alınacak ürünlerin stok seviyelerini kontrol edilir. Son olarak, müşterinin satın alma işlemini tamamlamak için Ödeme belgesi aracılığıyla ödeme yapması durumunda, Ödeme modelindeki PaidBySettter, kimliği doğrulanan kullanıcıyı ödeme yapan olarak ayarlar ve OrderStatusPaid sorumluluğuyla ödenen müşteri tarafından verilen sipariş durumunu günceller.

Sipariş güncelleme süreci, sipariş belgesindeki güncelleme sorumluluğunu tekrar tetikler ve satır fiyatı ile toplam fiyat hesaplamaları ile stok seviyesi denetimleri bir kez daha yapılır. Ardından, durumu (İng. *status*) ödendi olan sipariş belgesi için stok belgesinde belirtilen StockLevelChanger sorumluluğu etkinleştirilir ve sipariş edilen ürünlerin stok seviyeleri güncellenir.

Bu iki etki alanındaki belgelerin tasarımından sonra .Net kodunu elde etmek için dönüştürme kuralları uygulanır. Şekil 4'te sipariş belgesine BSİ dönüşüm kurallarının uygulanması gösterilmişti ve sonuçta ortaya çıkan .Net kodu da EK bölümünde sunulmuştur. Sipariş belgesi kodu işletildiğinde ve yeni bir sipariş örneği (*İng. instance*) oluşturulduğunda, bu sipariş NoSQL yönelimli olarak saklanacaktır.

V. Sonuç

Bu bildiride Belge Sorumluluk İş-Birliği adı verilen yeni bir yöntem sunulmuştur. BSİ'de belgeler üzerinde oluşturma, okuma, güncelleme ve silme (*İng. Create, Read, Update, Delete-CRUD*) işlemleri gerçekleştirebilir. Bu işlem talepleri ile ilgili olarak, bir belgenin ilgili sorumlulukları yerine getirmesi beklenmektedir. Ayrıca, belgeler bir iş hedefine ulaşmak için iş birliği yapar. Önerilen BSİ yöntemi, etki alanlarının belgelerden oluştuğu ve bir etki alanındaki bir belgenin, sorumluluğunu yerine getirmek için herhangi bir etki alanındaki başka bir belge ile birlikte çalışabileceği etki alan kavramı kullanır. BSİ yöntemi analizden tasarıma geçiş aşamasından başlar ve uygulama kodu otomatik olarak oluşturulana kadar sürer.

BSİ yöntemini kullanmanın avantajları şöyle özetlenebilir: Belge kavramına sahip bir proje için düzenli tasarım süreci ortaya çıkar. Otomatik kod üretimi ile kod kalitesi ve standardizasyonu sağlanır. BSİ ile tasarım eserleri, klasör / paket yapısı, sınıf yapısı, NoSQL belge yapısı proje içerisinde standartlaştırılmış olur. Bu standardizasyon sayesinde tasarım ve uygulama eserleri arasında izlenebilirlik kolaylaşır.

Gelecekte, BSİ yöntemini yetkilendirme, API, müşteri kodu ve test komut dosyası özellikleri ile genişletmeyi planlıyoruz.

References

[1] B. Selic, "The pragmatics of model-driven development," *IEEE software*, vol. 20, no. 5, pp. 19–25, 2003.

[2] F. Solms and D. Loubser, "Generating MDA's platform independent model using URDAD," *Knowledge-Based Systems*, vol. 22, no. 3, pp. 174–185, 2009.

[3] D. C. Schmidt, "Model-driven engineering," *COMPUTER-IEEE COMPUTER SOCIETY-*, vol. 39, no. 2, p. 25, 2006.

[4] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, "Model-driven development using UML 2.0: promises and pitfalls," *Computer*, vol. 39, no. 2, pp. 59–66, 2006.

[5] A. G. Kleppe, J. Warmer, J. B. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[6] R. H. Steinegger, P. Giessler, B. Hippchen, and S. Abeck, "Overview of a domain-driven design approach to build microservice-based applications," The Thrid Int. Conf. on Advances and Trends in Software Engineering, 2017.

[7] T. O. Meservy and K. D. Fenstermacher, "Transforming software development: an MDA road map," *Computer*, vol. 38, no. 9, pp. 52–58, 2005.

[8] R. Bræk and G. Melby, "Model-Driven Software Development, chapter Model-Driven Service Engineering," 2005.

[9] T. Mens and P. Van Gorp, "A taxonomy of model transformation," *Electronic notes in theoretical computer science*, vol. 152, pp. 125–142, 2006.

[10] J. Miller and J. Mukerji, Eds., "MDA Guide Version 1.01." Jun. 12, 2003, [Online]. Available: https://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf.

[11] E. Syriani, J. Gray, and H. Vangheluwe, "Modeling a model transformation language," in *Domain Engineering*, Springer, 2013, pp. 211–237.

[12] TOM Group, "Query/view/transformation," *Specification, Object Management Group (OMG)*, 2003.

[13] D. Vojtisek and J.-M. Jézéquel, "MTL and Umlaut NG-Engine and framework for model transformation," 2004.

[14] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, "ATL: a QVT-like transformation language," 2006, pp. 719–720.

[15] R. Wirfs-Brock and B. Wilkerson, "Object-oriented design: a responsibility-driven approach," *ACM sigplan notices*, vol. 24, no. 10, pp. 71–75, 1989.

[16] A. Mayes, "The responsibility driven object-oriented design method advocated by Wirfs-Brock, Wilkerson and Weiner," 1992.

[17] K. Beck and W. Cunningham, "A laboratory for teaching object oriented thinking," *ACM Sigplan Notices*, vol. 24, no. 10, pp. 1–6, 1989.

[18] M. West, "Having fun with objects, using Wirfs-Brock CRCs," 1993.

[19] Y. Singh and M. Sood, "Models and Transformations in MDA," 2009, pp. 253–258.