Rol Modellerinin Yazılım Kalitesine Katkıları

Contributions of Roles to Software Quality

Yunus Emre Selçuk
Bilgisayar Mühendisliği Bölümü
Yıldız Teknik Üniversitesi, İstanbul
yunus@ce.yildiz.edu.tr

Nadia Erdoğan
Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi, İstanbul
nerdogan@itu.edu.tr

Özet

Bir varlığın sahip olduğu bir rol, bu varlığın diğer nesnelerin kendisinden beklediği gibi davranabilmesi için gereken özellikler kümesi olarak tanımlanabilir. Roller uygulandıkları araştırma alanlarına olumlu katkılarda bulunurlar. Bu bildiri rolleri dinamik sistemlerin modellenmesi açısından inceleyerek rollerin yazılım kalite ölçütlerini olumlu yönden etkilediğini göstermeyi amaçlamaktadır. Bu amaçla rollerin yazılım kalitesini en büyük oranda etkileyen özellikleri üzerinde durulmuştur. Bu özelliklerin daha iyi açıklanabilmesi için örnekler vermek üzere, Java programlama diline rol desteği kazandıran bir rol modeli olan JAWIRO seçilmiştir. Bu bildiride iç yazılım kalite ölçütlerinin rollerin kullanımından nasıl etkilendiği ilk olarak teorik açıdan incelenmiş, ardından örnek bir gerçek dünya sistemini tasarlamak üzere önce rol modelleri, ardından diğer geleneksel yaklaşımlar kullanılarak pratik gerçekleme örnekleri verilmiştir.

Abstract

A role of an entity can be defined as the set of properties which are important for an object in order to be able to behave in a certain way, as expected by a set of other objects. Roles make positive contributions to research fields in which they are applied to. This paper evaluates roles in the context of modeling dynamic systems and argues that roles have positive effects on software quality, by focusing on the properties of roles that have the most effect on software quality. Those properties are demonstrated by using JAWIRO, a role model which extends the Java programming language with role support. In this paper, individual software quality criteria are briefly evaluated from a theoretical viewpoint for how they are affected by the usage of roles. This evaluation is followed by more detailed practical design and implementation issues.

1. Giriş

Rol kavramı yalnızca bilgisayar bilimlerinin yanı sıra sosyoloji ve dil bilimi gibi farklı alanlarda da kullanılan yaygın bir kavramdır. Bilgisayar bilimlerinde roller nesne tabanlı [1], bakış (aspect) tabanlı [2] ve etmen tabanlı [3] programlama alanlarında önemli bir yere sahiptir. Bununla birlikte modelleme [4], çözümleme ve tasarım [5], veri tabanları [6], ontolojiler [7], rol tabanlı erişim kontrolü (RBAC) [8] ve bilgisayar destekli işbirlikçi çalışma (CSCW) [9] alanlarında da rollerden yararlanılabilmektedir. Bu bildiride roller dinamik sistemlerin modellenmesi bağlamında ele alınacak ve rollerin yazılım kalitesine olan etkileri incelenecektir.

Dinamik sistemlerde bulunan nesneler, çeşitli yetenek ve sorumlulukların kazanımı ve terki ile sürekli değişir ve evrimleşir. Nesneye yönelik programlamanın (NYP) temel özelliği ise, bir nesne ve ait olduğu sınıf arasındaki ilişkinin kalıcı, durağan ve dışlamalı bir yapıda olmasıdır. NYP'nin bu durağan sınıf yapısı ile dinamik sistemlerin modellenmesi kolay değildir. NYP durağan sınıf yapısına dinamizm katacak yeteneklere sahip olmasına rağmen dinamik sistemlerin modellenmesinde karşılaştığı güçlüklerden tam anlamıyla kurtulamaz. Bu nedenle literatürde dinamik sistemlerin modellenmesi için yollar öneren birçok çalışma bulunmaktadır. Değişik yaklaşımlara örnek olarak prototip tabanlı diller [10], dinamik sınıflandırma [11], özneye yönelik programlama [12] ve tasarım kalıpları [13] verilebilir.

Dinamik sistemlerdeki aynı türden varlıklar, çeşitli yetenek ve sorumlulukların kazanımı ve terki ile birbirlerinden bağımsız şekilde evrimleşerek, türdeşlerinden kısmen farklılaşır. Bu durumda sınıf düzeyi özelleştirme yerine nesne düzeyinde özelleştirme kullanımı daha uygun olacaktır. Geleneksel NYP yaklaşımında ise kalıtım sınıf düzeyindedir, aynı türden olan varlıkların tümü aynı özellik kümesine sahip olur. Bir varlığın çeşitli sorumluluklar kazanıp terk ederek türdeşlerinden farklılaşması ise nesne düzeyinde kalıtım olarak adlandırılır. Rol modelleri bu aşamada dinamik sistemlerin modellenmesi için dolambaçsız bir çözüm önerir: Bu bağlamda bir gerçek dünya varlığı, her biri bu varlığın bir sorumluluğunu simgeleyen birden fazla rol nesnesi ile modellenir. Söz konusu her sorumluluk bir *rol* olarak adlandırılır. Bir varlığın bir rolü, bu varlığın diğer nesnelerin kendisinden beklediği gibi davranabilmesi için gereken özellikler kümesi olarak da

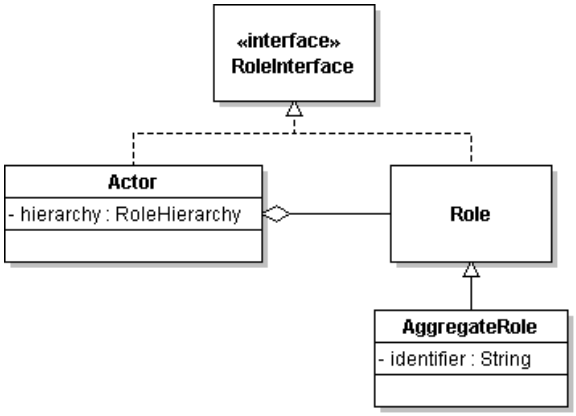
tanımlanabilir [1]. Bir rol modeli ise roller için bir tasarım ve kullanım biçimi tanımlar.

2. Rollerin Özellikleri ve Gerçeklenmeleri

Rollerin özellikleri dinamik sistemlerin modellenmesi ile ilgili gereksinimlere göre şekillenmiştir. Mevcut rol modellerinin roller ile ilgili önerdikleri özellikler değerlendirildiğinde, nesne düzeyinde özelleşmeyi mümkün kılan en genel yetenekler rollerin temel özellikleri olarak adlandırılabilir. Rollerle yapılabilecek işlerden söz konusu genel yeteneklerin dışında kalan diğer yetenekler ise rollerin ileri düzey özellikleri olarak adlandırılabilir. Rollerin özelliklerinin ayrıntılı tanımları için Selçuk ve Erdoğan'ın diğer çalışmalarından bazıları incelenebilir [4, 15, 16]. Bu bölümde, ortak noktaları Internet üzerinden erişilebilmeleri olan bazı rol modelleri, rollerin çeşitli özelliklerinin gerçeklenmesi açısından incelenecektir.

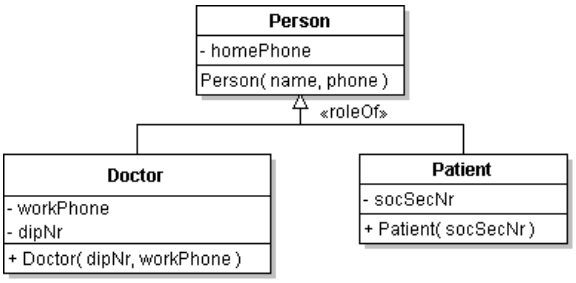
2.1. JAWIRO

JAWIRO, Selçuk ve Erdoğan tarafından, rollerin temel ve ileri düzey özelliklerini mümkün olduğunca kısıtlamalardan uzak bir biçimde gerçeklemek amacıyla oluşturulmuş bir rol modelidir [4, 15, 16]. JAWIRO rollerin ilişkisel hiyerarşisini modellemek için ağaç gösterimi kullanılarak, sahipliği ilişkilerinin daha anlamlı bir biçimde modellenmesi hedeflenmiştir. JAWIRO'nun rollerin kullanımı ile ilgili arayüzü Şekil 1'de verilmiştir. JAWIRO'da rol nesneleri `Role` sınıfı ile, bir rol hiyerarşisinin kökü ise `Actor` sınıfı ile modellenir. Bu iki sınıfın benzer davranışları olabileceği için, `Actor` ve `Role` sınıfları `RoleInterface` adlı ortak bir arayüzü gerçekler. Nitelikli rolleri (aynı rol türünden birden fazlasının oynanılabileceği roller) modelleyen `AggregateRole` sınıfı, `Role` sınıfından kalıtımla türetilmiştir. Rol modelinin omurgasını ise, her `Actor` örneğinin bu türden bir üyeye sahip olduğu `RoleHierarchy` sınıfı oluşturmaktadır.



Şekil 1: JAWIRO Kullanıcı Arayüzü

Şekil 2'de, rollerin temel ve ileri düzey özelliklerinden bazılarının JAWIRO ile kullanılması örneklerine taban oluşturmak üzere seçilmiş basit bir rol hiyerarşisi görülebilir. Modellenecek gerçek dünya varlığı olan insan nesneleri, `Person` adında ve `Actor` sınıfından kalıtımla türetilen bir sınıf ile simgelenmektedir ve bir rol hiyerarşisinin kökü olarak rol kazanımı ve terki ile nesne düzeyinde özelleşme yeteneğine sahip olacaktır. Kişilerin rolleri ise `Role` sınıfından kalıtımla türetilen sınıflar ile modellenir. Şekil 2'deki `Doctor` ve `Patient` sınıfları bu şekilde oluşturulmuştur. Şekil 2'de verilen hiyerarşi ile çalışma örnekleri veren kod ise Şekil 3'te görülebilir.



Şekil 2: Örnek Rol Hiyerarşisi

```
package test;
import jawiro.*;
//Nesne tanımları
Person ahmet, mehmet;
Doctor docAhmet;
Patient patMehmet;
//Rol hiyerarşilerinin oluşturulması:
//1.Köklerin oluşturulması.
ahmet = new Person( "Ahmet Ata",
   "212-7778899");
mehmet = new Person( "Mehmet Mert",
   "216-1112233");
//2.Rollerin oluşturulması.
docAhmet = new RoleDoctor(
   "212-4445566", "123456");
patMehmet = new RolePatient("56789");
//3.Rollerin eklenmesi
ahmet.addRole(docAhmet);
mehmet.addRole(patMehmet);
//Rol varlığı kontrolü
if(ahmet.canSwitch
   ("test.RoleDoctor") )
   //Rol geçişi
   ((RoleDoctor)ahmet.as
      ("test.RoleDoctor")).
      addPatient(patMehmet);
//Hasta iyileşip rolünü terk ediyor
patMehmet.resign();
```

Şekil 3: JAWIRO ile Rollerin Temel Özelliklerinin Kullanımı

Şekil 3'te verilen örnek kodda, rollerin kullanım biçimine özellikle dikkat edilmelidir. Roller birbirinden bağımsız olarak çalışma anında kazanılıp terk edilebileceği için, varlığın gerçekten o anda istenen role sahip olup olmadığını sorgulamak yerinde bir karar olacaktır. Bu amaçla `canSwitch` metodu kullanılmış ve parametre olarak rolün tam sınıf adı verilmiştir. İstenen rolün oynandığı bulunursa, o role geçiş yapmak için ise `as` metodu kullanılır. Rol geçişi programcıya ilgili rol nesnesini gösteren bir işaretçi sağlar ve Java'nın tip dönüşümü (type casting) yeteneği ile kullanılabilir. Bununla birlikte, bir rolü kullanmak için tek yol rol geçişi değildir. `RoleInterface` arayüzünde tanımlı `executeMethod( String methodName, Object... params )` metodu kullanılarak, adı ve parametreleri verilen metot çalıştırılabilir. *Ad ile üye erişimi* adı verilen bu özellik sayesinde roller tipten bağımsız olarak kullanılabilir.

Ad ile üye erişimi özelliğinin, hiyerarşi katılımcılarından birden fazlasının aynı adlı üyeye sahip olması durumunda bir çakışma ve/veya rastlansallık ortaya çıkmaması için, bazı roller *baskın rol* olarak işaretlenebilir. Bu amaçla `RoleInterface` arayüzü `dominateSearch ( boolean dominate )` adlı bir metoda sahiptir. Eğer hiçbir rol baskın olarak işaretlenmemişse, bu durumda en çok özelleşmiş, bir başka deyişle hiyerarşinin en derinindeki rolün üyesine erişim sağlanır. Bununla birlikte, hiyerarşinin kökünün de baskın olması durumunda JAWIRO önceliği köke verecektir.

Modellenen gerçek dünya sisteminin gerektirdiği durumlarda, bir rol türü farklı türlerden sahipler tarafından oynanabilir. JAWIRO bu gereksinimi karşılamak üzere bir rol örneğinin farklı sınıfların rol örnekleri tarafından oynanabilmesi, bir başka deyişle *nesne düzeyinde çoklu kalıtım* özelliğini destekler. Bu durumun herhangi bir belirsizliğe yol açmaması için gerekli önlemler alınmıştır. Belli bir rol örneği aynı anda yalnız bir sahibe ait olabileceği için mantıksal bir belirsizlik zaten söz konusu değildir. Tip belirsizliği ise sadece derleme anında geçerlidir; çalışma anında ad ile üye erişimi özelliği kullanılarak, herhangi bir role tipten bağımsız olarak erişilebilir.

2.2. EpsilonJ

Java diline rol desteği kazandıran bir rol modeli olan EpsilonJ [17], rollerin bağlam (context) özelliğini destekler. Bağlam kavramı bir birlikte çalışma ortamı gibi ele alınır ve aynı roller gibi yeniden kullanılabilen bir programlama yapısıdır. Örneğin bir doktor ve bir hasta rolü, hastane bağlamı içerisinde kullanılır. Bağlamlar nesnelerden ayrı olarak tertiplenen yeniden kullanım birimleri olarak düşünülebilir. EpsilonJ'de her rolün tanımı, ait olduğu bağlam içerisinde yapılır, bu nedenle ancak aynı bağlamdaki roller birbirleri ile etkileşebilir. Şekil 4'te örnek bir bağlam tanımlıdır.

```
context Company {
   static role Employer {
      int salary = 100;
      void pay() {
         Employee.getPaid(salary);
      }
   }
   role Employee {
      int save;
      void getPaid(int salary) {
         save += salary;
      }
   }
}
```

Şekil 4: EpsilonJ ile Bir Bağlam ve Rollerinin Tanımı

Şekil 4'teki `static` tanımı, Java'daki tanımından farklı olarak, bir bağlamda o türden sadece bir rol nesnesinin bulunabileceğini gösterir. Rolleri ve nesneleri birbirleri ile bağlamak için `bind` ve `newBind` metotları kullanılır. Bu metotlardan ilki hem statik olan hem de olmayan roller için kullanılabilirken, ikincisi sadece statik olmayan roller için kullanılabilir. Her iki metot da bir nesneyi bir role bağlar ancak `newBind` metodu nesneyi role bağlamadan önce o bağlamda yeni bir rol örneği oluşturur [17]. Dolayısıyla EpsilonJ'de `static` olan rollerin JAWIRO'da `Role` sınıfı ile modellenen normal rollere, EpsilonJ'de statik olmayan rollerin ise JAWIRO'daki nitelikli rollere karşılık geldiği söylenebilir [4, 15, 16]. Şekil 5'te ise, örnek bağlam ve rollerin kullanımı görülebilir.

```
class Person {
   int money;
}
Person tanaka = new Person();
Person sasaki = new Person();
Company todai = new Company();
todai.Employer.bind(sasaki);
todai.Employee.newBind(tanaka);
sasaki.(todai.Employer).pay();
tanaka.(todai.Employee).getPaid();
```

Şekil 5: EpsilonJ ile Bağlam ve Rollerin Kullanımı

2.3. RoleX/BRAIN

BRAIN [18] çerçeve programının bir parçası olan RoleX [19] etkileşim sistemi, Java etmenlerinin dinamik olarak rol kazanımı ve terk etmesini, etmenlerin sekizli kodunu (bytecode) çalışma anında değiştirerek sağlar. Etmenler birbirleri ile eylemler ve olaylar sayesinde etkileşir. RoleX'te eylemler, etmenlere roller tarafından kazandırılan yeteneklerin gerçeklemeleridir; olaylar ise eylemlerin yürütülmesi ile ortaya çıkar. Roller bir rol havuzunda saklanır ve etmenler bu havuzdan gereksinim duydukları rolleri talep eder. Her rol bir rol tanımlayıcısı ile ilişkilidir. Bir rol tanımlayıcısı, bu rol ile ilgili genel bilgilerin yanı sıra, rolün eylemleri ile

ilişkili işlem tanımlayıcıları da içerir. Bir rol tanımlayıcısının kısmi örneği Şekil 6'da görülebilir.

```
<role xmlns=
   "http://polaris.ing.unimo.it/schem
   a/RoleDescriptionSchema"
   xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance"
   xsi:SchemaLocation=
   "http://polaris.ing.unimo.it/schem
   a/RoleDescriptionSchema" >
   <GenericRoleDescription>
      <description>Flight Booker
      Role</description>
      <roleName>
            flight_booker
      </roleName>
      <keyword>
            flight reservation
      </keyword>
      <keyword>travel
      airplane</keyword>
      <version>1</version>
      <OperationDescription>
         <name>Book</name>
         <aim>book a flight</aim>
         <keyword>flight</keyword>
         <version>1.0</version>
         <methodName>book_flight</met
         hodName>
         <returnType>
            <className>java.lang.Boole
         an</className>
         </returnType>
         <parameter>
            <className>java.lang.Strin
         g</className>
         </parameter>
         <parameter>
            <className>java.util.Calen
         dar</className>
         </parameter>
      </OperationDescription>
      <EventDescriptor>
         <name>DeletedEvent</name>
         <aim>Inform that thereserved
         flight has been deleted
         </aim>
         <className>examples.tabican.
         DeleteEvent</className>
         . . .
         <ReceivingEvent>true</Receiv
         ingEvent>
      </EventDescriptor>
      . . .
   </GenericRoleDescription>
</role>
```

Şekil 6: RoleX'te Bir Rol Tanımlayıcısı [19]

3. Rol Kullanımının Yazılım Kalitesine Etkileri

Bu bölümde rollerin yazılım kalitesine etkileri incelenecektir. Bu amaçla önce bireysel kalite ölçütlerine rollerin etkileri teorik olarak tartışılacak, ardından pratik tasarım ve gerçekleme konuları üzerinde durulacaktır. Bu bölümde ağırlık pratik gerçekleme örneklerine verilmiştir, teorik konulardaki daha ayrıntılı bilgi için Selçuk ve Erdoğan'ın bir başka çalışması incelenebilir [16].

3.1. Rollerin Yazılım Kalite Ölçütlerine Katkıları

Yazılım projelerindeki yüksek başarısızlık oranından dolayı [20], yazılım ile ilgili herhangi bir öneri bir de yazılım kalitesi açısından incelenmelidir. Bu bölümde, rol modellerinin kullanımının yazılım kalite ölçütlerine etkileri kısaca özetlenecek ve yazılımın iç kalite özellikleri [21] üzerinde durulacaktır.

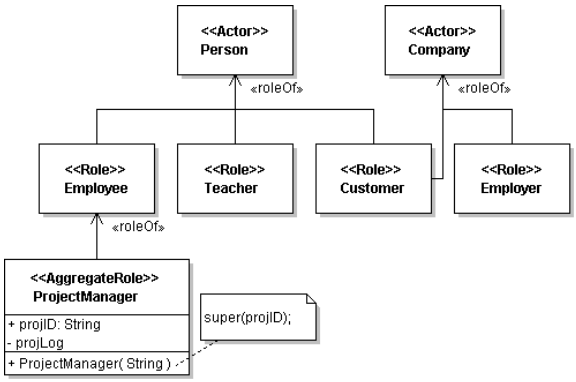
Yu and Ramaswamy [22], değişik bağlaşım türlerini inceleyerek bunları üç kategoride toplamıştır: Parametre bağlaşımı, kalıtım bağlaşımı ve genel bağlaşım. Parametre bağlaşımı bir sınıfın bir metodunda parametre olarak tanımlanmış başka bir sınıfın, herhangi bir metodunu çağırması şeklinde tanımlanmıştır. Parametre bağlaşımının en düşük düzeyde bağlaşıma sebep olduğu değerlendirilmiştir. Kalıtım bağlaşımı ise bir sınıf başka bir sınıftan kalıtım yolu ile türetildiğinde ortaya çıkar ve parametre bağlaşımından daha yüksek düzey bir bağlaşım oluşturur. Genel bağlaşım ise en yüksek bağlaşıma sebep olur ve iki sınıfın aynı ortak değişkene ulaşması şeklinde tanımlanmıştır. Bu tanımlardan yola çıkarak, rollerin ve dolayısıyla nesne düzeyi kalıtım ilişkisinin kullanılmasının, sınıf düzeyinde olan kalıtım bağlaşımını parametre bağlaşımına çevirdiği, bu nedenle de bağlaşımı azalttığı söylenebilir.

Rollerin temel özelliklerinin kullanımı ilgi alanlarının ayrılmasını kolaylaştırarak [16] kaliteye yeniden kullanılabilirliğin artması [23] gibi olumlu katkılarda bulunsa da, diğer yandan da yazılımın tiplere olan bağımlılığını arttırarak istenmeyen bir olumsuzluk sunmaktadır. Bu yan etki ilk anda gözden kaçsa da, rol modelleri genellikle bir rolü kullanmadan önce o rolün tipine geçiş yapmayı gerektirir. Bu gereksinim Şekil 3'te açıkça görülebilir. Tiplere olan bağımlılığın bu gereksiz artımı, yeniden kullanımı azaltacaktır. Ancak Bölüm 2.1'de tartışıldığı üzere, ad ile üye erişimi ve baskın roller özelliklerinin kullanımı ile nesnelere tipten bağımsız erişim sağlanabilir ve aşırı tip bağımlılığının kaliteye olumsuz etkilerinden kaçınılabilir [16].

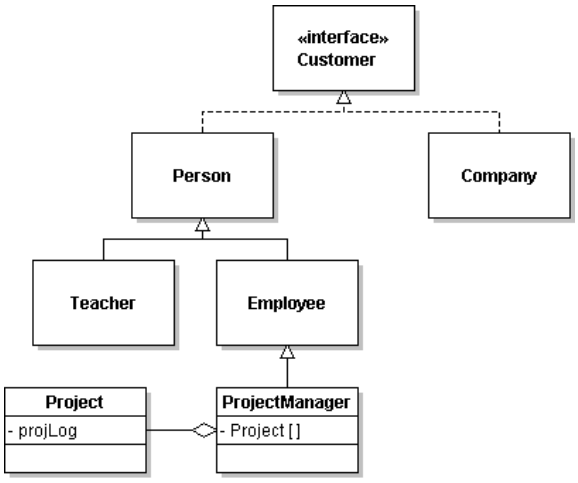
3.2. Tasarım ve Gerçekleme Konuları

Tasarım ve gerçekleme konuları üzerindeki tartışmalara temel oluşturacak rol hiyerarşisi Şekil 7'de görülebilir. Şekil 7'de rol hiyerarşisi kullanılarak modellenen sistemin, Java dilinin yalın olarak kullanılması ile yapılabilecek bir gerçeklemesi Şekil 8'de görülebilir. İlk

gerçek dünya varlığı olan kişinin modellenmesi görece kolaydır. Bu durumda rol modeli kullanılmamasının getirdiği tek ek yük, bir proje yöneticisinin yürüttüğü projelerin bir parça-bütün ilişkisi içerisinde yönetilmesi olacaktır. Halbuki JAWIRO programcının üzerinden bu ek yükü nitelikli roller sayesinde kaldırabilir: Belli bir nitelikli role geçiş yapmayı sağlayan `as(String className, String identifier)` metodu ile istenilen herhangi bir projeye ulaşılabilir. Belli bir tipteki nitelikli rollerin tümüne erişim sağlayan bir `Iterator` nesnesi döndüren `asList(String className)` komutu ile de, tüm projelere ulaşılabilir.

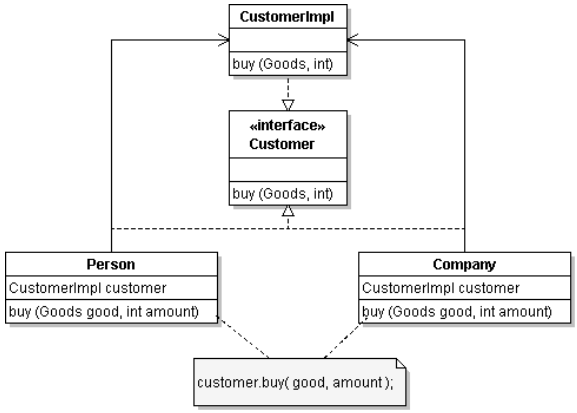


Şekil 7: JAWIRO kullanarak Rol Hiyerarşisi ile Gerçekleme

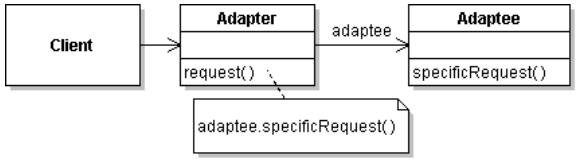


Şekil 8: Java'nın Yalın Kullanımı ile Gerçekleme

Ancak modellenmesi gereken diğer gerçek dünya varlığı olan şirketin tasarıma eklenmesi ile işler karmaşıklaşacaktır. Hem bir kişinin hem de bir şirketin bir müşteri olabilmesini mümkün kılmak için, `Person` ve `Company` sınıfları ortak bir arayüz olan `Customer` arayüzünü gerçeklemelidir. Bu ilk bakışta büyük bir zorluk gibi gözükmese de, bu tür ortak arayüzlerden onlarcasının olabileceği bir gerçek dünya sisteminde, tasarım modeli bu çok sayıda arayüzü gerçekleyen daha da fazla sayıda sınıf bulunmasını gerektirecektir. Ortak arayüz her sınıfta ayrı ayrı gerçeklenebilir, ancak bu durumda yeniden kullanılabilirlik ve uyum olumsuz etkilenecektir. Daha iyi bir çözüm *Object Adapter* [24] tasarım kalıbını kullanarak elde edilebilir. Böyle bir çözümün Şekil 8'de önerilen modelin `Person` ve `Company` sınıfları ile `Customer` arayüzünü nasıl etkileyeceği Şekil 9'da gösterilmiştir. Şekil 10 ise *Object Adapter* tasarım kalıbını hatırlatmak amacıyla verilmiştir [24].

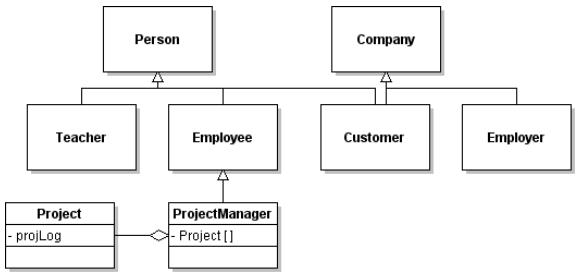


Şekil 9: "Object Adapter" Tasarım Kalıbı ile Çözüm

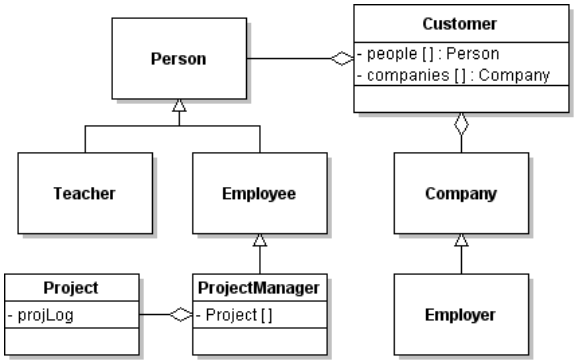


Şekil 10: "Object Adapter" Tasarım Kalıbı

Şekil 4'te verilen örnek rol hiyerarşisi C++ ile modellendiğinde, Şekil 11'de de görülebileceği gibi, sınıf düzeyinde çoklu kalıtım kullanılabilir. Ancak sınıf düzeyi çoklu kalıtım *diamond inheritance* gibi sorunlara yol açabilir [25].



Şekil 11: C++ ile Gerçekleme



Şekil 12: Dilden Bağımsız Bir Gerçekleme

Sınıf düzeyi çoklu kalıtım gibi belli bir dile bağlı özellikleri kullanmadan elde edilebilecek bir başka çözüm ise Şekil 12'de görülebilir. Bu durumda `Customer` sınıfı bir müşteri gibi davranabilecek `Person` ve `Company` örneklerinin bir birikimi halini alacaktır. Bu yaklaşımın zayıf yönü de çoklu arayüz gerçeklemesi ve sınıf düzeyi çoklu kalıtımın olumsuz yönlerine benzer: Hazırlanacak model ortak arayüz veya ortak üst sınıf sayısı kadar ek birikim sınıfının kullanımını gerektirecektir. Ayrıca bir birikim sınıfı içerdiği nesnelerin yönetimini sağlayacak ek koda da ihtiyaç duymaktadır. Son olarak, söz konusu birikimli sınıfta yer alabilecek yeni bir tip ortaya çıkarsa, bu birikimli sınıfın içyapısında da değişiklikler yapmak zorunda kalınacaktır.

4. Sonuç

Nesneye yönelik programlamanın tüm yeteneklerine rağmen, dinamik sistemlerin modellenmesi kolay bir problem değildir. Dinamik sistemlerin ve gerçek dünya sistemlerinin modellenmesinde karşılaşılan sorunları çözmek üzere çeşitli tasarım kalıpları oluşturulmuştur [24]. Bununla birlikte, çeşitli araştırmacılar tarafından öne sürülen kalite ölçütlerine göre [21, 26, 27] 'iyi' yazılım üretmek ve literatürdeki tasarım kalıplarına hakim olmak, ancak zaman içerisinde kazanılan deneyim ile mümkün olabilir. Rol modelleri bu noktada kullanıcılara bir dinamik sistemi modelleyen 'iyi' bir yazılım üretmek için doğal ve kolay bir yol sunar. Rol modellerinin kazandırdığı nesne düzeyinde özelleşme yeteneğinin sınıf düzeyinde özelleşmeyi genişletmesi nedeni ile, rol modelleri tüm NYP dillerinin felsefesine uygundur ve herhangi bir NYP dili ile gerçeklenebilir.

Rol modellerinin kazandırdığı nesne düzeyi özelleşme yeteneği sayesinde, NYP'nin durağan sınıf yapısı ile dinamik sistemlerin modellenmesinde karşılaşılan güçlüklerin çözümü için bir yol elde edilmiştir. Bu yolun alternatifi, ilk paragrafta tartışıldığı ve Bölüm 3.2'de pratik örneklerle kanıtlanmaya çalışıldığı üzere, alışılagelmiş yol olan nesneye yönelik tasarımda uzmanlaşmaktır. Bu alternatif yol alışıldık olmakla birlikte, zor ve uzun bir yoldur.

Rol modelleri dinamik sistemlerin modellemesi için kolaylıklar sunmakla birlikte, yazılım kalite ölçütlerine ilgi alanlarının ayrılması sayesinde yeniden kullanılabilirliğin artması gibi olumlu etkiler de yapmaktadır. Ancak özellikle kuvvetli tiplemeli dillerde, rollerin kullanımı yazılımın tip bağımlılığını arttırarak yeniden kullanımı olumsuz etkileyebilir. Bir rol modeli bu olumsuz etkiyi giderecek yollar da sunmalıdır. Şaşırtıcı bir şekilde ve bilebildiğimiz kadarıyla, JAWIRO [16] ve RoleX [19] istisna olmak üzere, tip bağımlılığının azaltılmasına yönelik yeteneklere sahip olan rol modelleri enderdir. Literatürdeki çeşitli rol modelleri hakkında daha ayrıntılı bilgi ve özellik karşılaştırmaları için Selçuk ve Erdoğan'ın bir çalışması [4] incelenebilir. Söz konusu çalışmada çeşitli rol modelleri incelenerek bunların özellikleri karşılaştırılmıştır. JAWIRO rol modeline ise http://www.yunusemreselcuk.com/jawiro/index.html adresinden erişilebilir.

Sonuç olarak bu bildiride rollerin kullanımının yazılım kalitesine olumlu katkıları bulunduğu, teorik tartışma ve pratik örneklerle gösterilmiştir. Bununla birlikte, orta ve büyük ölçekli uygulamalarda rol modellerinin kullanılarak, rollerin olumlu katkılarının deneysel olarak da gösterilmesi gerekmektedir.

6. Kaynaklar

[1] Kristensen, B. B. and Osterbye, K., "Roles: Conceptual Abstraction Theory and Practical Language Issues", *Theory and Practice of Object Systems (TAPOS), Special Issue on Subjectivity in Object-Oriented Systems,* 3:143–160, 1996.

[2] Hannemann, J., Murphy, G. C. and Kizcales G., "Role-Based Refactoring of Crosscutting Concerns", *AOSD'05*, Chicago, Illinois, USA, 135–146, 2005.

[3] Thomas G., and Williams, A. B., "Roles in the Context of Multiagent Task Relationships", *AAAI'05 Fall Symposium Series, Roles: An Interdisciplinary Perspective Subtopic*, Arlington, Virginia, USA, 2005.

[4] Selçuk, Y. E. and Erdoğan, N., "Using Roles with JAWIRO", *AAAI'05 Fall Symposium Series, Roles: An Interdisciplinary Perspective Subtopic,* Arlington, Virginia, USA, 2005.

[5] Reenskaug, T., Wold, P., Lehne, O. A., *Working with Objects: The OOram Software Engineering Method,* Prentice-Hall, 1995.

[6] Albano, E., Bergamini, R., Ghelli, G. and Orsini, R., "An Object Data Model with Roles", *Proc. Int'l. Conf. Very Large Data Bases*, Saratoga, Calif., USA, 39–51, 1993.

[7] Sunagawa, E., Kozaki, K., Kitamura, Y. and Mizoguchi, R., "Role Organization Model in Hozo", *EKAW'06 (LNAI vol. 4248,* Springer, 2006.

[8] Sandhu, R. S., Coyne, E. J., Feinstein, H. L. and Youman, C. E., "Role-Based Access Control Models", *IEEE Computer*, 20(2): 38–47, 1996.

[9] Tripathi, A., Ahmed, T., Kumar, R. and Jaman, S., "Design of a Policy-Driven Middleware for Secure

Distributed Collaboration", *Proc. 22nd Int'l Conf. on Distributed Computing System (ICDCS)*, Vienna, Austria, 393–400, 2002.

[10] Ungar, D. and Smith, R. B., "Self: The power of simplicity", *Proc. ACM Conf. on Object Oriented Programming Systems, Languages and Applications,* Orlando, Florida, 214–242, 1987.

[11] Drossopoulou, S., Damiani, F., Dezani-Ciancaglini, M. and Giannini, P., "More dynamic object re-classification: Fickle", *ACM Transactions On Programming Languages and Systems,* 24(2):153–191, 2002.

[12] Wong, R. K, Chau, H. L. and Lochovsky, F. H., "A Data Model and Semantics of Objects with Dynamic Roles", *IEEE Int'l Conf. on Data Engineering*, Birmingham, UK, 402–411, 1997.

[13] Fowler, M., "Dealing with Roles", *Pattern Languages of Programming (PLOP '97),* Illinois, USA, 1997.

[14] Zendler, A. M., "Foundation of the Taxonomic Object System", *Information and Software Technology,* 40:475–492, 1998.

[15] Selçuk, Y. E. ve Erdoğan, N., "Rol Modelleri ve Nesneye Yönelik Programlamaya Katkıları", *Ulusal Yazılım Mimarisi Kongresi,* 2006.

[16] Selçuk, Y. E. and Erdoğan, N., "A Role Model for Description of Agent Behavior and Coordination", *Lecture Notes in Computer Science,*3963:29–49, 2006.

[17] Tamai, T., Ubayashi, N. and Ichiyama, R., "An Adaptive Object Model with Dynamic Role Binding", *Proc. Int'l. Conf. on Software Engineering (ICSE '05),* 2005.

[18] Cabri, G., Leonardi, L. and Zambonelli, F., "BRAIN: A Framework for Flexible Role-Based Interactions in Multiagent Systems", *Proceedings of the 2003 Conference on Cooperative Information Systems (CoopIS),* 2003.

[19] Cabri, G., Ferrari, L., Leonardi, L., "Injecting Roles in Java Agents Through Run-Time Bytecode Manipulation", *IBM Systems Journal,* 44(1):185–208, 2005.

[20] Peckham, J. (ed.), *Practicing Software Engineering in the 21st Century*, Idea Group Inc., 2003.

[21] McConnell, S., *Code Complete,* Microsoft Press, 557–560, 1993.

[22] Yu, L-G. and Ramaswamy, S., "Component Dependency in Object-Oriented Software", *Journal of Computer Science and Technology,* 22(3):379–386, 2007.

[23] Ossher, H. and Tarr, P., "Using Multidimensional Separation of Concerns to (Re)shape Evolving Software", *Communications of the ACM,* 44:43–50, 2001.

[24] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns Elements of Reusable Object Oriented Software*, AddisonWesley, 1994.

[25] Sintes, T., "Java Diamonds Are Forever; How Does Java Solve The Multiple-Inheritance Diamond Problem". *JavaWorld'01*, San Fransisco, USA, 2001.

[26] Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,* Prentice-Hall, 2004.

[27] Stumpf, R. V. and Teague, L. C., Object-Oriented System Analysis and Design with UML, Prentice-Hall, 2004.