T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ROL MODELLERİNDE KURAL MOTORLARININ KULLANILMASI

ÖNDER GÜLER

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN
YRD. DOÇ. DR. YUNUS EMRE SELÇUK

İSTANBUL, 2011

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ROL MODELLERİNDE KURAL MOTORLARININ KULLANILMASI

Önder GÜLER tarafından hazırlanan tez çalışması 22.09.2011 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Yrd.Doç. Dr. Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi _____

Jüri Üyeleri

Prof.Dr.Nadia ERDOĞAN

İstanbul Teknik Üniversitesi _____

Yrd.Doç.Dr.Songül ALBAYRAK

Yıldız Teknik Üniversitesi _____

ÖNSÖZ

Benden yardımlarını, desteğini, sabrını ve bilgisini esirgemeyen değerli hocam Yrd. Doç. Dr. Yunus Emre Selçuk' a ve manevi desteklerini her zaman için hissettiğim aileme en içten teşekkürlerimi sunarım.

Ağustos, 2011

Önder GÜLER

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

NYP	Nesneye Yönelik Programlama
OOP	Object Oriented Programming

ŞEKİL LİSTESİ

Sayfa

ÖZET

ROL MODELLERİNDE KURAL MOTORLARININ KULLANILMASI

Önder Güler

Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Tez Danışmanı: Yrd.Doç. Dr. Yunus Emre Selçuk

Yazılım dünyasında yaygın kabul gören Nesneye Yönelik Programlama (NYP) yaklaşımında, varlıklar ayrı sınıflarla temsil edilmektedirler ve varlıkların programın yaşam döngüsü boyunca sınıflarını değiştirme gibi bir durumları söz konusu değildir. Bu yüzden durağan sistemlerin nesneye yönelik programlama ile modellenmesi uygundur. Fakat günümüz sistemlerinin dinamik bir yapıya sahip olduğu için NYP gerçek hayattaki ihtiyaçları karşılamakta yeterli olamamaktadır. Rol modelleri tam bu noktada devreye girerek gerçek yaşamdaki sistemleri modelleme konusunda bir düzenleme önermiştir. Fakat rol modelleri kullanılarak hazırlanmış bir uygulamada alana özgü kısıtlar veya diğer bir ismiyle kurallar yazılması gerektiği zaman kuralların farklı sınıflara bölünüp dağıtılması da gerekecektir. Kuşkusuz kuralların uygulama içerisindeki bu denli dağılmışlığı yönetimini zorlaştıracaktır. Bu tez çalışmasının amacı da, rol modelleri kullanılarak yazılan programlarda, alana özgü iş kurallarının, iş analistleri tarafından kodlanabilecek kadar basit bir sözdizimi ile ifade edilebilmesi ve kuralların tek bir yerde toplanarak yönetiminin daha efektif bir hale getirilmesidir.

Anahtar Kelimeler: Rol modelleri, kural motorları, Jawiro, Drools

ABSTRACT

USING OF RULE ENGINES WITH ROLE MODELS

Önder Güler

Department of Communications Engineering

MSc. Thesis

Advisor: Assistant Professor Yunus Emre Selçuk

Many programming approach are accepted by the world of software and they have already been used by the lots of software projects being live. One of the most popular programming approaches is Object Oriented Programming (OOP). In OOP, assets are represented by separate classes and these assets can't change their classes throughout their life cycle. Therefore OOP is suitable for modeling of static systems. But today's systems have a dynamical structure that OOP is not enough to meet the needs of these systems. Role models propose a solution to the modeling of dynamic systems. In contrast, when you need to write domain-specific constraints, you will have to divide the rules into distributed classes. Of course in the application it is hard to manage such a complexity. The aim of this thesis is expressing of domain-specific rules such a simple syntax that it can be witten by business analysts and collecting rules in one place to make more effective management.

Key words: Role Models, rule engines, Jawiro, Drools

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Tez çalışması içerisinde, giriş bölümü sonrasında rol modellerinden bahsedilmiş, bir uygulamada kontrol edilmesi gereken anormal rol ilişkileri kavramına değinilmiş ve olası senaryolar hakkında örnekler sunulmuştur. Bu amaçla örnek rol modeli olarak Jawiro[1][2] seçilmiştir. Üçüncü bölümde kural motorlarının tanıtımı yapılmış ve özel olarak tez çalışmasında kullanılan Drools kural motorundan ayrıntılı olarak bahsedilmiştir. Dördüncü bölümde ise, tez çalışmasının asıl amacı olan ve rol modellerinde aykırı rol ilişkilerinin önlenmesini gerekli kuralların bir yerde toplanarak soyutlanmasını sağlayan çözüm önerisi, ayrıntılı olarak açıklanmıştır. Beşinci bölümde, öncesindeki bölümde önerilen çözüm, Jawiro'da geliştirilmiş bir uygulama üzerinden örnek kodlar ve senaryolar yardımıyla izah edilmiştir. Tez çalışmasının gövdesini oluşturan son bölüm olan altıncı bölümde ise, kural motoru kullanılarak ve kural motoru kullanılmadan hazırlanan iki uygulama çalışma hızları açısından karşılaştırılarak performans analizi yapılmış ve sonuçlar değerlendirilmiştir.

1.2 Tezin Amacı

Anormal rol ilişkilerini engellemek için Jawiro rol modeli her ne kadar kendi önlemlerini almışsa bile bu konuda daha sağlam bir altyapıya gereksinim duymaktadır. Gerçek dünya uygulamalarında programcının yükünün azaltılması için, Jawiro'nun anormal rol ilişkilerinin engellenmesi mekanizmasının daha yetenekli bir hale getirilmesi gerekmektedir. Bu aynı zamanda kuralların tek bir yerden çağırılabilmesini ve bu

sayede yönetiminin son derece efektif olmasını sağlayacaktır. Bu mekanizmanın yetenekli hale getirilmesinden kasıt yalnızca kolay yönetimi de değildir, ayrıca sunduğu altyapı ile alan bilgisine gerek duymaksızın, kuralların tetiklenmesini sağlayan, kural motoru ile veri alışverişine girebilen, Jawiro'nun sunmuş olduğu kısıt arayüzünü gerçekleyen bir altyapı sunmasıdır. Birçok yazılım projesindeki diğer bir önemli gereksinim de altyapıyı hazırlayan kişilerle alana özgü kuralları (kısıtları) hazırlayan kişilerin farklı kişiler olması durumudur. Bu durumda da alana özgü kuralları yazacak olan kişiler basit bir sözdiziminden faydalanarak kurallarını yazacaklardır. Elinizdeki tez çalışması işte bu gereksinimleri gerçekleştirme amacını taşımaktadır. Amaca ulaşmak için ise bir kural motorundan yararlanılacaktır. Kural motorları hakkında özet bilgi üçüncü bölümde verilmiştir.

1.3 Bulgular

Kural motorları kullanılarak, rol modellerinde aykırı rol ilişkilerinin önlenmesiyle ilgili eksikliklerin giderilmesi için çeşitli adımlar atılabilir. Tez çalışması ile ortaya konulan yapının sağladığı iyileştirme, kuralların merkezi bir yerden çağırılabilmesinin sağlanması ve bu sayede kuralların yönetiminin etkinliğinin arttırılması olmuştur. Ortaya konulan diğer bir iyileştirme ise, uygulama alanı ile program kodlarının ayrımı olmuştur. Bu çalışma ile yapılan iyileştirmeler, sıradan bir bilgisayar sisteminin çalışma zamanı başarımında hissedilir bir düşme pahasına elde edilmektedir. Bu nedenle arzu edilen kolaylıklara ulaşmak isteyenlerin daha efektif çalışan bir kural motoru kullanmaları önerilir.

BÖLÜM 2

ROL MODELLERİ

Bu bölümde, tez çalışmasında kullanılan temel iki araçtan biri olan rol modelleri hakkında genel bilgi verilecektir.

2.1 Rol Modellerine Giriş

Rol modelleri NYP'ye nesne düzeyinde özelleştirme yeteneği kazandıran bir yaklaşımdır. Bu durumda bir gerçek dünya varlığı, her biri yükümlü olduğu görevlerden birini yürüten bir rol nesnesi olmak üzere, birden fazla nesne ile temsil edilir. Bir nesnenin bir rolü, bu nesnenin diğer nesnelerin ondan beklediği gibi davranabilmesi için gereken bir özellikler kümesi olarak tanımlanabilir. Rol kullanımı temeline dayanan programlama biçimine rol tabanlı programlama (RTP), rollerin tasarımı için bir tarz ve rollerin kullanımı için çeşitli işlevler sunan yazılımlara ise rol modeli adı verilir. Dinamik sistemlerin etkili ve kolay bir biçimde modellenmesi için önerilen yollar arasında rol modelleri; kullanışlı olmaları, NYP kavramları ile iyi uyuşmaları ve problemin çözümü için dolaysız bir yol sunmaları nedeniyle dikkati çekmektedir. Nesne düzeyinde özelleştirmeye dayanan rol modelleri, sınıf düzeyinde özelleştirmeye dayanan NYP'yi doğal bir biçimde genişletir. Nesne düzeyinde özelleştirme, sınıf düzeyinde özelleştirmeyi genişlettiğinden dolayı, rol modelleri tüm NYP dillerinin felsefesine uygundur ve herhangi bir NYP dili ile gerçeklenebilir.

Bir rol modelinin sağlaması gereken temel özellikler, Selçuk tarafından [1] şu şekilde belirlenmiştir:

- Bakış açısı: Bir gerçek dünya varlığının oynadığı rollerin her biri, o varlığa bir bakış açısı oluşturur. Bir nesneye erişim o anda kullanılan rolü ile sınırlıdır. Ancak rol

geçişi sayesinde, mevcut rol içerisinden başka bir rolün üyelerine erişim sağlanılabilir.

- Rol hiyerarşisi: Bir rol (üst rol) başka bir rolü (alt rol) oynayabilmelidir. Bu da rollerin çeşitli hiyerarşik düzenler oluşturacak şekilde birbirleri ile ilişkilendirilebilmeleri anlamına gelecektir.
- Rol kazanımı ve terki: Roller birbirlerinden bağımsız olarak kazanılabilmeli ve terk edilebilmelidir.
- Birden fazla nesne ile gösterim: Bir gerçek dünya nesnesinin sahip olduğu rollerin tümü ile tanımlandığı gerçeği korunmalıdır. Bu amaçla her rol nesnesi sahibinden, kendi rollerinden ve hiyerarşi kökünden haberdar olmalıdır.
- Rol geçişi: Bir varlık herhangi bir rolüne herhangi bir anda geçiş yapabilmelidir.
- Sahibi üzerinden üye erişimi: Bir nesnenin bir rolü, diğer rollerinin üyelerine, üstteki iki özellikten biri sayesinde erişebilmelidir.
- Çakışma engeli: Değişik roller bir çakışma olmaksızın aynı adlı üye metot ve değişkenlere sahip olabilmelidir.
- Sınıf düzeyi kalıtım ile nesne düzeyi kalıtımın birlikte kullanımı: Nesne düzeyi kalıtım sınıf düzeyi kalıtımın yerini almaz. Aksine, nesne düzeyi kalıtım sınıf düzeyi kalıtımı tamamlar. Gerçek dünyada birlikte görülebilen bu iki tür kalıtım ilişkisi rol modellerinde de birlikte kullanılabilmelidir.
- Rol varlığı kontrolü: Varlıklar herhangi bir rol tipini oynayıp oynamadıkları hakkında sorgulanabilmelidir.
- Nitelikli roller: Bir varlık aynı rol tipinin birden fazla örneğine sahip olabilmelidir. Böyle rollere bu çalışmada nitelikli rol adı verilmiştir. Nitelikli roller birbirlerinden bir niteleyici ile ayrılırlar.

Rollerle ilgili olarak diğer bazı önemli hususlar ise, Selçuk tarafından [1] “rollerin ileri düzey özellikleri” adı altında aşağıdaki gibi belirtilmiştir:

- Aykırı rol ilişkilerinin önlenmesi: Kullanıcıların, sistemin yapısına aykırı işler yapmaları olasılığı her zaman vardır. Bu yüzden bu tür ihlallerin önlenmesi bir zorunluluktur.
- Kalıcılık: Kalıcılık yeteneğine sahip bir rol modeli bütün bir rol hiyerarşisini kalıcı bir ortama saklayabilir ve sonradan tekrar kullanmak için kalıcı ortamdan geri yükleyebilir.
- Rol aktarımı: Bir rol mevcut sahibinden bir başka sahibe, alt rollerini kaybetmeden aktarılabilmelidir.
- Nesne düzeyinde çoklu kalıtım: Bir rol örneği farklı sınıfların rol örnekleri tarafından oynanabilmelidir.
- Ad ile üye erişimi: Bir rol hiyerarşisinde bulunan nesnelerden herhangi birisinin istenilen bir üye değişkeni veya metoduna, sahibine doğrudan bir referans olmadan, sadece o üyenin adı belirtilerek erişilebilir.
- Baskın roller: Önceki kural hiyerarşideki bazı katılımcıların baskın kılınması ile değiştirilebilmelidir.
- Rollerin askıya alınması ve askıdan indirilmesi: Geçici bir süre ihtiyaç duyulmayacak rollerin askıya alınabilmesi ve gerek duyulduğunda askıdan indirilerek kullanılabilmesidir.

Rollerin yukarıda verilen özelliklerinin kullanımının örneklerle ayrıntılı olarak anlatılması, bu tez çalışmasının kapsamı dışındadır. Jawiro ile rollerin kullanımı hakkında temel bilgi için Selçuk ve Erdoğan tarafından yapılan diğer çalışmalar incelenebilir [1][2].

2.2 Rol Modellerine Neden İhtiyaç Duyulur?

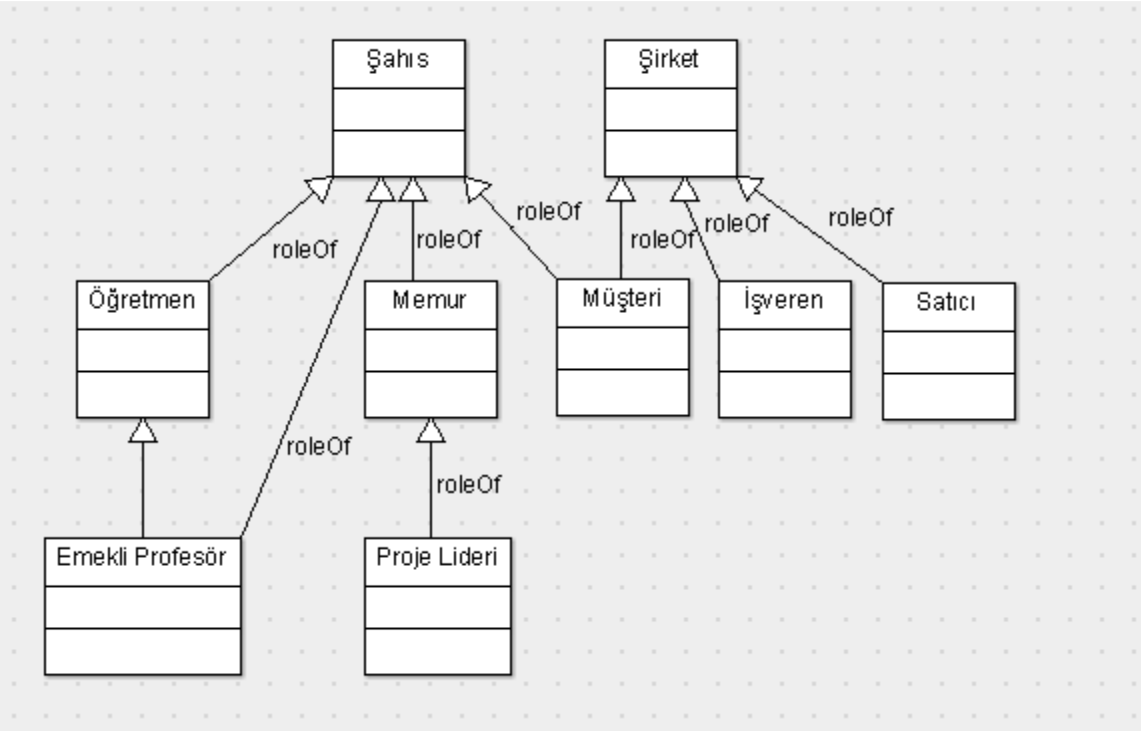
Sorumlulukları zaman içerisinde farklılaşan varlıkları modellemeye çalışan bir kişi, nesneye yönelik programlamanın güçlü ancak nispeten esnek olmayan sınıf yapısından kaynaklanan çeşitli güçlüklerle karşılaşacaktır. Söz konusu zorlukların neler olduğunun saf Nesneye Yönelik Programlama yaklaşımı kullanılarak gösterilmesi, buna ek olarak

çeşitli tasarım kalıpları ve rol modelleri kullanarak değinilen zorlukların ne oranda giderilebileceği, Selçuk ve Erdoğan tarafından yapılan bir çalışmada [2] gösterilmiştir.

Kısaca özetlemek gerekirse, rol modelleri uygulama aktörlerinin tıpkı gerçek dünya aktörlerinin toplum içerisinde sahip olduğu gibi birden fazla role sahip olmasını sağlar. Bu durumda uygulama içerisindeki bir aktör tek bir nesne yerine birden fazla nesne tarafından temsil edilerek, bu aktörün gerçek dünyada karşılaştığı görev değişiklikleri uygulamaya daha kolay yansıtılabilecektir.

2.3 Anormal Rol İlişkilerinin Engellenmesi

Rol ilişkilerindeki aykırılıklar veya bir başka deyişle anormallikler, aynı rol örneğinin aynı anda birden fazla sahibinin bulunması gibi olağandışı ilişkiler ve modellenen sistemin izin vermediği kural dışı ilişkiler olmak üzere iki gruba ayrılabilir. Örneğin bir projenin birden fazla lideri olamaz; bir başka deyişle aynı proje lideri rolünü birden fazla memur oynayamaz. Olağandışı ilişkilerin yanı sıra, modellenen sistemin kurallarına ters düşen, kural dışı ilişkiler de kurulmaya çalışılabilir. Örneğin bir memur rolüne işveren rolü eklenmesi, Şekil 2.3.1'de verilen sisteme ters düşecektir. Bir yazılımı hazırlayanlar ile kullananların farklı kişiler olmasından kaçınılamaz. Kullanıcıların zamanla sistemin yapısına aykırı işler yapmamaları için anormal ilişkilerin önlenmesi bu nedenle bir zorunluluk olarak karşımıza çıkar.



Şekil 2.1 Rol modellerinin kullanımı ile modellemeyi örnekleyen UML şeması

Olağan dışı ilişki türleri, hem rol modellerinin hem de gerçek dünya sistemlerinin temel kavramlarına zıt düşen ilişkilerdir. Dolayısıyla olağan dışı ilişkileri önleyen aşağıdaki kurallar Jawiro rol modeli içine değiştirilemez biçimde kodlanmıştır:

- Bir rol nesnesinin yalnız bir tek sahibi olabilir.
- Askıya alınmış bir rol JAWİRO uygulama programlama arabiriminin komutları ile kullanılamaz. Örneğin başka bir sahibe aktarılamaz.
- Askıya alınan bir role, rol modelinin komutlarıyla ulaşılamaması bir gerekliliktir. Bir sorumluluğun askıya alınmasının nedeni, gerçek dünyanın o anki koşullarında o sorumluluğun kullanılması için bir engelin ortaya çıkmasıdır. Gerçek dünyada engellenen bir sorumluluğun uygulama programında serbestçe çalıştırılabilmesi ise bir tutarsızlık ortaya çıkartacaktır.

İzin verilmeyen kural dışı ilişkilendirmeler ise, modellenen sistemin kurallarına göre belirlenir. Her sistem için farklı kurallar geçerli olabileceğine göre, bir rol modeline programcı tarafından çeşitli kurallar esnek bir biçimde tanıtılabilmeli ve rol modeli de bu kurallara uyulmasını sağlamalıdır.

BÖLÜM 3

KURAL MOTORLARI

Bu bölümde, tez çalışmasında kullanılan temel iki araçtan biri olan kural motorları hakkında genel bilgi verilecektir.

3.1 Kural Motoru Nedir?

Gerçek yaşamdaki birçok uygulama, otomatik olarak sürece müdahale edebilen iş politikaları, prosedürleri ve iş mantığına ihtiyaç duymaktadır. Gerektiğinde yazılım süreci üzerinde kısıtlama da yapabilen iş kuralları, gücünü, bilgiyi uygulama mantığından ayırmasından ve herhangi bir iş mantığında değişime gidileceği vakit uygulamaya ait kaynak kodun yeniden değiştirilip derlenmesine ihtiyaç duymamasından almaktadır.

Birçok uygulama, piyasa ekonomisindeki dinamik değişimlere ayak uydurmak zorundadır. Örneğin sigorta ve bankacılık uygulamaları, tasarım aşamasında kimsenin önceden tahmin edemeyeceği ve planlayamayacağı piyasa ekonomisindeki değişimlere göre kendisini yenilemek zorundadır. Bu durumda çözüm, iş analistlerinin ve yazılım geliştiricilerin organizasyon verileri üzerinde karar mekanizması kurmalarını sağlayan araçlardan olan kural motorlarından faydalanmaktır. Kural motorları, iş kurallarının değerlendirilip yürütülmesini sağlamaktadır. Kural motorlarının görevi, son kullanıcılar tarafından hazırlanan kuralların, uygulamanın çalışma tarzını etkilemeksizin gerçeklenebilmesidir. Kural motorlarını detaylı olarak anlatmadan önce kural(rule), olgu(fact) ve vargı(conclusion) gibi kural motorlarından bahsederken kullanılacak olan önemli kavramlardan söz etmek yerinde olacaktır.

- Olgu: Olgular, kurallara girdi olarak verilen, üzerinde çalışılan alana ait bilgi parçacıklarıdır.
- Vargı: Diğer olguların kurallara verildikten sonra yapılan çıkarsamalar sonucu elde edilen yeni olgulardır.
- Rule: Kurallar, olgular vasıtasıyla yeni olgu veya vargıların elde edilmesini sağlayan, aynı zamanda belirli şartlar sağlandığında belirli olayların yürütülmesini sağlayan if-then-else benzeri yapılardır.

Kural motorları, esasında, iş mantığını hariçte tutmak fikrine dayanan if- then komutlarını yorumlamayı sağlayan karmaşık bir yorumlayıcı olarak düşünülebilir. Burada bahsi geçen if-then deyim(statement) leri kuralların ta kendisidir. Bir kural, şart ve icra edilecek olay olmak üzere iki kısımdan oluşmaktadır. Şart sağlandığında ilgili olay tetiklenip çalışacaktır. if kısmı, miktar >=100 gibi bir şart kısmını then kısmı ise %5 lik bir indirim yap gibi yürütülecek olaydan oluşmaktadır. Kural motoruna verilecek girdiler, kural yürütme kümesi(RuleExecutionSet) olarak adlandırılan kurallar ve bu kuralların üzerinde işlem yapacağı veri nesnelerinden oluşmaktadır. İlgili kuralın çıktısını, kurala verilen girdi oluşturmaktadır. Kural çıktısı, verilen girdinin değiştirilmiş hali olabileceği gibi yeni veri nesneleri veyahut olası yan etkiler olarak isimlendirilen müşteriye mail atma gibi seçenekler de olabilmektedir.

Kural motorları, çok dinamik olarak değişen iş mantığına sahip uygulamalarda ve son kullanıcıların iş kuralları üzerinde rahatça değişiklik yapabileceği uygulamalar ile birlikte kullanılmalıdır. Kural motoru, binlerce olgu üzerinde kolayca ve güvenilir bir şekilde uygulanabilir büyük bir araçtır.

Başlangıçta da bahsedildiği gibi, iş politikasında yapılan herhangi bir değişikliğin, uygulamaya vakit kaybetmeden yansıtılması gereken zaman kritik uygulamalarda, kural motorları yaşamsal bir öneme sahiptir. Uygulamalar, kendisi ile ilgili olan nesneleri, kural motoruna geçerek kural motorlarıyla iletişim kurarlar. Daha sonra uygulama, sonuçları gözlemler ve kullanıcıya sunar veya bunları akış içerisinde kullanmaya devam eder. Kural motoru, hangi kuralın ilk olarak çalıştırılacağına gelen girdi ye bakarak karar verir. Bu yüzden kullanıcının, kurallar arasında herhangi bir akış sırası veya öncelik sırası belirlemesine gerek kalmaz.

Kısacası; uygulamalarda kural motorlarından faydalanmak aşağıdaki avantajları sağlayacaktır;

- Uygulama alanına(domain) ve uygulamaya özgü politikaları içeren kural motorları kolayca anlaşılabilir olacaktır.
- Kurallar ile diğer programlama dilleri arasında maksimum düzeyde bağımsızlık olacaktır.
- Kurallar uygulama mantığı ile bilgi kısmını birbirinden ayırır.
- Kurallar, uygulamanın kaynak kodu değiştirilmeksizin değiştirilebilir. Böylece kaynak kodun sürekli değiştirilmesine ve her değişiklikte yeniden derlenmesine gerek kalmaz.

Yukarıda sayılan faydaların maliyeti ise; kural dilinin iş analistleri tarafından öğrenilmesi ve uygulama ile kurallar arasında kurulacak arayüz için gerekli olan zamandır. Bununla beraber eğer şirket başka bir kural motoru kullanmaya karar verirse iş analistlerin yeni kural motoruna ait sözdizimini ve yazım biçimini öğrenmesi gerekecektir.

3.2 JSR94

Güncel kullanımda olan kural motorlarına örnek olarak; Drools[3], Fair Isaac Blaze Advisor[4], ILOG JRules[5] ve Jess[6] verilebilir. Birçok kural motoru, uygulama ile kurallar arasında arayüz olarak kendine özel API'sini kullanmaktadır. Fakat bu durumun bazı sakıncaları vardır; örneğin şirket herhangi bir sebeple kullanmakta olduğu kural motorundan vazgeçip başka bir kural motoru kullanmaya karar verirse, uygulama içerisinde kurallar ile iletişime geçmek için yazılan kod parçacığını değiştirmek ve uygulamasını yeniden derlemek zorunda kalacaktır.

JSR94, kural motoru gerçeklemesini Java için standart hale getirmek maksadıyla atılan bir adım olmuştur. Yukarıda isimleri belirtilen dört kural motoru JSR94'ü desteklemektedir.

JSR94 API 'sinin başlıca görevleri şu şekildedir;

- Kullanılacak kural kümesinin(rule set) kaynak kod içerisinde etkin hale getirilmesi veya etkinliğinin kaybettirilmesi.

- Kuralların ayrıştırılması(parsing)
- Kural öteverilerinin(Metadata) izlenmesi
- Kuralların yürütülmesi
- Kural kümesi yürütüldükten sonra sonuçların elde edilmesi
- Sonuçların süzülmesi

JSR94 ün sorumluluğu altında olmayan davranışlar ise şu şekildedir:

- JSR94, kural motorunun kendisi değildir.
- Kuralların yürütülmesi konusunda hiçbir sorumluluğu yoktur.
- Kuralların ifade edildiği dile karışmaz.
- Java EE teknolojileri için bir yerleştirme(deployment) mekanizması sağlamaz.

3.3 Yaygın Kullanımdaki Bazı Kural Motorları:

Bu bölümde yaygın kullanımdaki bazı kural motorları hakkında özet bilgi verilecektir. Tez çalışmasında kullanılan Drools kural motoru üzerinde ise biraz daha ayrıntılı durulacaktır.

3.3.1 Jess[6]

Kurallar Lisp gibi fonksiyonel programlama diline çok yakın bir dille yazılmaktadır. Jess paralı bir üründür, yalnızca akademik kullanımlar için herhangi bir ücret talep edilmemektedir.

3.3.2 Hammurapi[7]

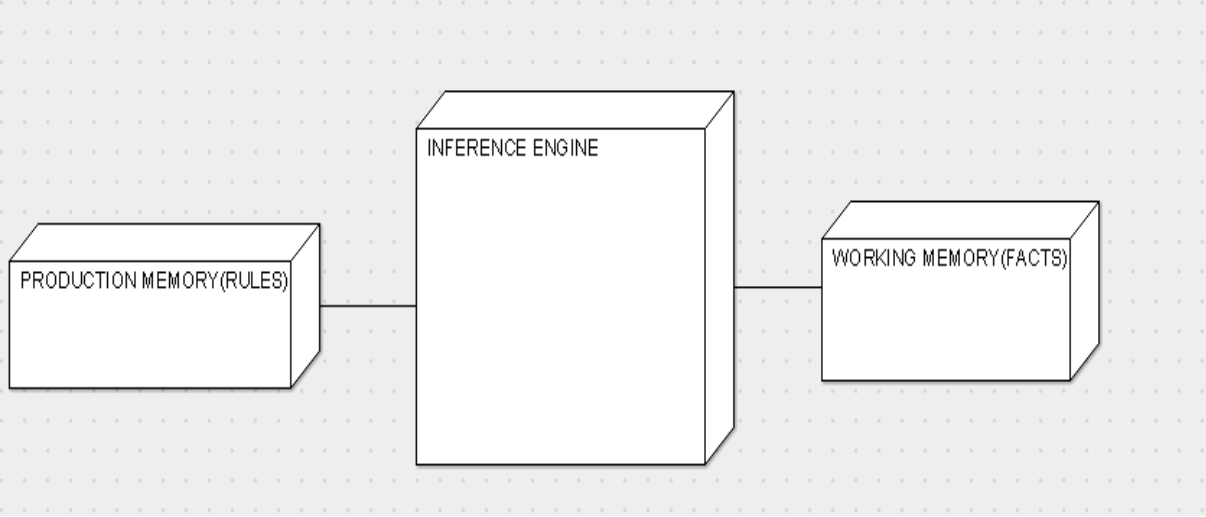
Hammurapi, java geliştiriciler için hazırlanmış bir java kural motorudur. Kurallar java sözdiziminde yazılmaktadır. Kurallar, java dilinde yazıldığı için java derleyicisi tarafından byte kodlara dönüştürülürler.

3.3.3 Fair Isaac Blaze Advisor[4]

Blaze Advisor, Structured Rules Language(SRL) kullanmaktadır. SRL doğal dile yakın kolay bir dildir. İş politikaları, pratikleri ve prosedürlerinin yanı sıra yapay zeka tabanlı kural programlama da yapılabilmektedir. SRL, xml görünüme sahiptir.

3.3.4 Drools[3]

Drools kural motorunda kurallar, Üretim Belleğinde (Production Memory) tutulmaktadır. Olgular ise değiştirilmek üzere Çalışan Bellekte(Working Memory) tutulmaktadır. Olgular ve Kuralların eşleştirilmesi Çıkarsama motorunda(Inference Engine) yapılmaktadır. Bazen aynı olgu için birden fazla kural çalıştırılabilmektedir. Buna Kural çakışması denmektedir. Gündem(Agenda), Çakışma çözüm stratejisi(Conflict Resolution Strategy) kullanarak çakışan kuralların yürütülme sırasını belirlemektedir. Çıkarsama Motoru, etkin bir örüntü işleme algoritması olan Rete algoritmasını [8] gerçekler. Drools kural motorunun genel yapısı aşağıda Şekil 3.1' de görüldüğü gibidir. Drools, İleriye Dönük Zincirleme ile (forward chaining) çalışır; yani olgular kurallara verilir ve son olarak kendisi de bir olgu olan vargı elde edilir.



Şekil 3.1 Drools kural motorunu oluşturan temel yapılar

Çakışma olabileceğini düşündüğümüz kurallardan, önce çalışmasını istediğimiz kurala daha büyük bir çıkıntı (salience) değeri vermemiz gerekir. Kurallarda varsayılan çıkıntı değeri sıfırdır.

3.3.4.1 Drools İç Yapısını Oluşturan Öğeler

Drools kural motoru bünyesinde, kullanıcılar tarafından hazırlanan kuralların yürütülmesi için Bilgi Yapıcı (KnowledgeBuilder), BilgiTabanı (KnowledgeBase), WorkingMemoryEntryPoint, Gündem ögelerini içermektedir.

Bilgi Yapıcı, kaynak durumundaki Drl uzantılı bir dosyayı veya bir Excel dosyasını alarak, BilgiTabanının kullanabileceği bir paket haline getirir. Daha sonra bu paket Bilgi Tabanına eklenir. BilgiTabanı, uygulamaya ait kurallar, prosesler, fonksiyonlar ve tipler (type) gibi bilgi tanımlarının depolandığı birimdir. DurumBilgiliOturum (StatefulKnowledgeSession), BilgiTabanı vasıtasıyla şu şekilde yaratılır.

StatefulKnowledgeSession ksession=kbase.newStatefulKnowledgeSession();

Bir diger Drools ögesi WorkingMemoryEntryPoint dir. WorkingMemoryEntryPoint, olguların, araya eklenmesi(insert), güncellenmesi(update) ve erişimi(retrieve) için metotlar içermektedir. Tez çalışmasında, sadece olguların araya eklenmesinden faydalanıldığı için, açıklamasına yer verilmiştir.

- Araya ekleme(Insertion) :

Çalışan Bellek e olgu hakkında bildirimde bulunmaktır. Bu işlem, ksession.insert(YourObject) ile yapılır. Daha sonra akış içerisinde fireAllRules() komutu verildiğinde kurallar çalıştırılacaktır. Şekil 3.2'de bir olgunun çalışan bellek e eklenmesini sağlayan kod parçası verilmiştir.

```
Cheese stilton = new Cheese("stilton");
FactHandle stiltonHandle = ksession.insert( stilton
);
```

Şekil 3.2 Olgunun Çalışan Belleğe eklenmesi

Son Drools ögemiz, kendisi bir Rete algoritması özelliği olan Gündem dir. Gündem, çakışma çözünürlük stratejisi sayesinde, fireAllRules() sonrasında kuralların çalışma sırası belirlemektedir.

3.3.4.2 Drools Kural Dosyasının Yapısı

Doğal bir dile sahip olan Drools, drl uzantılı bir dosya olan (kural dosyası) içerisinde kural, sorgu, fonksiyon, global değişkenler, aktarımlar(import) ve nitelikler(attribute) içerilmesine izin vermektedir.

Bir Kural dosyasının genel yapısı aşağıdaki gibidir:

- Paket(package): Mantıksal olarak birbirleriyle ilişkili kuralların bir arada tutulduğu yapıdır.
- Aktarım: Aktarım ifadesi, tıpkı java dilindeki aktarım ifadesi gibi çalışmaktadır. Kullanılmak istenen nesnenin tam adresi belirtilmelidir.
- Globals: Kural içerisinde kullanılan global anahtar kelimesi ile global değişkenler tanımlanmaktadır. Global değişkenlerin kullanım amacı, Kurallar ile Uygulama(Application) arasındaki haberleşmeyi sağlamaktır. Global ler Çalışan Belleğe girdi olarak verilmezler.
- Fonksiyonlar(Functions): Fonksiyon kullanımının ana avantajı, mantığın tümünü tek bir yerde tutmaktır. Fonksiyonlar genellikle kuralların sonuç kısımlarında kullanılırlar. Özellikle bu kısmın farklı parametreler için kullanılması gerektiğinde fonksiyonun farklı parametrelerle çağrılması faydalı olacaktır.
- Kural: Esasında kuralların temel sözdizimi aşağıda Şekil 3.3' deki gibidir:

```
rule "name"
attributes
when
  LHS
then
 RHS
End
```

Şekil 3.3 Drools ta bir kural dosyasının genel yapısı

Burada LHS ile belirtilen kısım, şart(condition) kısmıdır. RHS ile belirtilen kısım ise şart sağlandığında yapılacak işlemleri belirtmektedir. Nitelik ise kuralların nasıl davranacağını belirleyen, kural motoruna verilen bir ipucudur. Burada kural ismi tanımlanırken "" karakterlerinin kullanımı zorunlu değildir.

Dilin bazı anahtar sözcüklerinin, değişken ismi olarak kullanılması yasaklanmıştır. Fakat kullanıcı bu anahtar sözcükleri tek tırnak içerisinde kullanırsa herhangi bir sakıncası olamaz. Bu anahtar kelimeler şunlardır:

true, false, accumulate, collect, from, null, over, then, when

- Yorumlar

Tek satır yorumlar için # veya // kullanılmaktadır. Birden fazla satır için /* */ sembolleri yorum amaçlı kullanılmaktadır.

3.3.4.3 Kural Detayları

Kural, bazı şartlar sağlandığında bazı işlemlerin yapılması anlamına gelmektedir. Eğer bir kural dosyası içerisinde aynı isimli iki tane kural ile karşılaşılırsa, hata üretilir. İç içe kural kavramı Drools'da yoktur. Kural dosyası içerisinde, kurallar yazılırken bazı kullanım desenleri söz konusudur. Bunlar; Sol Taraf Koşul Elemanları(Left Hand Side Conditional Elements) olarak adlandırılmaktadır, Çoklu Kısıtlama(Multi Restriction), Kodiçi Değerlendirme Kısıtı(Inline Eval Constraints). Sol taraf koşul elemanlarının içerisinde en çok görülen kulanımlar, aşağıdaki deyime benzer kullanımlardır.

```
Person( likes : favouriteCheese )
```

Bu örnekte, herhangi bir değişkene olgu üyesi set edilmektedir. Burada likes bir değişken, favouriteCheese ise herhangi bir Person objesinin üyesi durumundadır. Bir diğer sol taraf koşul elemanı da; çoklu kısıtlama desenidir; çoklu kısıtlama deseni, oturum vasıtasıyla, çalışan bellek e girilen objenin herhangi bir alanı üzerinde kısıtlama yapabilmek amacıyla kullanılmaktadır. Şekil 3.4'te bazı örnek kullanımlar görülmektedir.

```
Person( age > 30 && < 40 )
Person( age ( (> 30 && < 40) ||
              (> 20 && < 25)) )
Person( age > 30 && < 40 || location == "london" )
```

Şekil 3.4 Çoklu kısıtlama deseni kullanımları

Burada bahsedilecek olan son sol taraf koşul elemanı eval fonksiyonudur. Eval fonksiyonu evaluation kelimesinin kısaltılmış halidir. Bu fonksiyonun yaptığı işlem; kendisine verilen ifadeyi mantıksal açıdan değerlendirip true veya false şeklinde çıktılar üretmektir. Şekil 3.5 de kullanımı gösterilen eval fonksiyonunun görevi, tüm erkeklerin bayanlardan iki yaş büyük olduğu çiftleri bulmaktır.

```
Person( girlAge : age, sex = "F" )
Person( eval( age == girlAge + 2 ), sex = 'M' )
```

Şekil 3.5 Eval fonksiyonun kullanımı

Kural dosyalarında, sol taraf koşul elemanları haricinde belirli koşullar gerçekleştiğinde kural içerisinde yapılması gereken işlemlerin tanımlandığı sağ taraf kısmı bulunmaktadır. Bu kısımda şart belirtmemek gerekiyor. Çünkü bu kuralın atomikliğini bozan bir davranış olur. Burada asıl amaç, çalışan belleğe (Working memory) olgu girme veya mevcut bir olgunun güncellenmesidir.

BÖLÜM 4

ÖNERİLEN YAKLAŞIM

Jawiro rol modeline göre hazırlanmış herhangi bir yazılım için, uygulama alanı dikkate alınarak uygulamaya yeni bir rol ekleneceği, herhangi bir rol askıya alınacağı, aktöre ait bir rol tamamen ortadan kaldırılacağı zaman uygulamaya has kısıtlar yazılması gerekmektedir. Bu tez çalışmasında önerilen yaklaşım; bu kısıtların bir Drools dosyası içerisinde hazırlanması ve kullanıcının ConstraintDroolsManager olarak isimlendirilen bir yardımcı sınıf vasıtasıyla Drools dosyasında, Drools sözdiziminde hazırlanmış olan kurallara erişip kullanabilmesidir. ConstraintDroolsManager, ConstraintStrategy arayüzünü gerçeklemektedir. ConstraintStrategy arayüzü, Jawiro'da hâlihazırda tanımlanmış olup, rol tabanlı yaklaşımla geliştirilmiş olan bir uygulama için genel bir kısıt şablonu sunmaktadır. ConstraintStrategy arayüzünün gerçeklenmek üzere sunduğu başlıca onaylama metotları; approveAddRole, approveSuspend, approveResume ve approveResign metotlarıdır. Tanımı Şekil 4,1'de verilmiş olan ConstraintStrategy arayüzünün metotlarının amaçları şu şekildedir:

- approveAddRole: Üzerinde çalışılan uygulama alanı dikkate alınarak, herhangi bir rolün Aktöre eklenmesinin uygun olup olmadığının kararı bu metot içerisinden verilmektedir.
- approveSuspend: Üzerinde çalışılan uygulama alanı dikkate alınarak, herhangi bir rolün geçici bir süreliğine aktörden alınmasının uygun olup olmadığının kararı bu metot içerisinden verilmektedir.

- approveResume: Üzerinde çalışılan uygulama alanı dikkate alınarak, daha önce geçici bir süreliğine aktörden alınmış olan rolün tekrar sahibine iade edilip edilemeyeceğinin kararı bu metot içerisinden verilmektedir.
- approveResign: Üzerinde çalışılan uygulama alanı dikkate alınarak, herhangi bir rolün tamamen aktörden alınmasının uygun olup olmadığının kararı bu metot içerisinden verilmektedir.
- ConstraintDroolsManager sınıfı ConstraintStrategy arayüzünü gerçeklediği için, yukarıda değinilen anlarda devreye girerek, izin verilmeyen rol ilişkilerinin kurulmasını önleyebilir. İzin verilen ve/veya verilmeyen ilişkiler hakkındaki kurallar ise, bir veya daha fazla Drools kural dosyası içinde tanımlanır. Kuralların ihtiyaç duyacağı bilgiler, Drools oturumuna ConstraintDroolsManager sınıfının onaylama metotlarından aktarılır. ConstraintDroolsManager sınıfı hakkında ayrıntılı bilgi bir sonraki alt bölümde yer almaktadır.

ConstraintDroolsManager sınıfının gerçeklediği ConstraintStrategy arayüzünün genel görünümü ise Şekil 4.1'de gösterilmiştir.

```
package Jawiro;
import java.io.Serializable;
public abstract interface ConstraintStrategy extends
Serializable {
  public abstract boolean approveAddRole(String paramString1,
String paramString2);
  public abstract boolean approveResign(String paramString1,
String paramString2);
  public abstract boolean approveSuspend(String paramString1,
String paramString2);
  public abstract boolean approveResume(String paramString1,
String paramString2);
  public abstract void setActor(Actor paramActor);
}
```

Şekil 4.1 ConstraintStrategy arayüzü

4.1 ConstraintDroolsManager Sınıfı

Önceki bölümünde bahsedildiği üzere, ConstraintDroolsManager sınıfında ConstraintStrategy arayüzüne ait dört metot gerçeklenmiştir. Buradaki her bir metodun gerçeklenmesi java sözdizimine uygun olarak yazılacak Jawiro kodu yerine Drools kural dosyasında hazırlanmıştır. Drools kural dosyası içerisinde yazılan bu kurallar öncelikli olarak derlenerek BilgiTabanının kullanabileceği bir paket haline

getirilir. Hazırlanan bu paketin BilgiYapıcıya gösterimi ve daha sonrasında istemci taraf(ConstraintDroolsManager sınıfı metotları) ve Kural motoru arasında, bu kuralların yürütülmesi amacıyla oturum değişkeninin tanımlanması gibi her bir metot içerisinde yazılması zorunlu işlemler, ConstraintDroolsManager sınıfının kurucu metodu içerisinde bir kez yazılarak bu sorumluluk yerine getirilmiştir. Ayrıca alana özgü aktör yine yapıcı metot içerisinde üye değişkenlere atanmaktadır. Burada BilgiTabanının tanımlanması işlemi, readKnowledgeBase() adlı metot içerisinde yazılmıştır. Son olarak da uygulama ile kural motoru arasında bir oturum kurmaya yönelik ksession değişkeni tanımlanmıştır.

Örnek kod parçası şekil 4.2 de görüldüğü gibidir.

```
public ConstraintDroolsManager(Actor myActor){
try{
this.myActor = myActor;
kbase = readKnowledgeBase();
ksession = kbase.newStatefulKnowledgeSession();
logger = KnowledgeRuntimeLoggerFactory.newFileLogger(
ksession, "test");
}
      catch(Exception e){
         e.printStackTrace();}
      }
```

Şekil 4.2 Drools ve uygulama arasındaki bağlantının sağlanması için gereken kodun, yapıcı metot içerisinde bir kez yazılması

Yukarıdaki yapıcı metot içerisinde yazılan kod parçası dikkatle incelenecek olursa, yapıcı metoda alana özgü herhangi bir parametre geçirilmediği görülecektir. Kuşkusuz bu kullanım, altyapıya uygulama bağımsız bir özellik kazandırmaktadır. Böylece altyapıyı kullanarak yeni bir uygulama geliştirmek isteyen kişilerin kendi aktör değişkenlerini ConstraintDroolsManager sınıfının yapıcı metoduna parametre olarak geçmesi yeterli olacaktır.

ConstraintDroolsManager sınıfının üye alanları olarak, actor sınıfından üye alanının yanı sıra, Drools kural motoruna özgü üye alanları da bildirilmiştir. Bunlar, temel üye alanları olmakla birlikte sınıf içerisinde yardımcı üye alanları da tanımlanmıştır. Bu üye alanları, kural dosyası ile uygulama arasındaki veri paylaşımını sağlayacak olan, aktör nesnesine eklenen her bir rolü saklayan ve kural dosyasına ileten liste nesnesi ile ihtiyaç hasıl olduğunda true veya false mantıksal değerlerini ana uygulama ve Kural

Motoru arasında taşıyacak olan RoleCheck nesnesidir. ConstraintDroolsManager sınıfının üye alanları Şekil 4.3 de görülmektedir.

```
public StatefulKnowledgeSession ksession;
public KnowledgeRuntimeLogger logger;
public LinkedList<String> list;
public KnowledgeBase kbase;
public RoleCheck roleCheck;
private Actor myActor;
```

Şekil 4.3 ConstraintDroolsManager sınıfının üye alanları görülmektedir.

Uygulama tarafında yazılımı geliştiren kişiler, ConstraintDroolsManager sıfından üretilen nesneye, o anda kullandıkları aktör nesnesini geçmek zorundadır. Uygulama tarafında yazılımı geliştiren kişiler herhangi bir rolü tanımladıkları aktör nesnesine ekleyecekleri zaman bu rol, ConstraintDroolsManager sınıfının üye alanlarından birisi olan liste değişkeninde saklanır. Kural motoru ve uygulama arasında ortak bir alan olarak kullanılan liste değişkeni vasıtasıyla rollerin eklenme sırası kolay bir şekilde yönetilebilmekte, bazı rollerin bir rolden sonra eklenmesi engellenmekte veya gerektiğinde son eklenen rol tespit edilerek kural motoruna bildirilmektedir. İşte tam bu noktada rol listesi, kural motoru ve uygulama arasında iletişimi sağlayan bir araca dönüşmektedir.

Uygulama ve kural motoru arasındaki iletişimi sağlayan diğer bir nesnede, yine aynı şekilde ConstraintDroolsManager sınıfının içerisinde kullanılan ve RoleCheck sınıfından üretilen nesnedir. Şekil 4.1' den de anlaşılacağı üzere, ConstraintStrategy arayüzünün rol kısıtlamalarıyla ilgili tüm metotları, client kısma boolean bir değer(olumlu veya olumsuz) dönmektedir. Örneğin; client tarafındaki yazılımcı aktöre herhangi bir aykırı rol eklemeye çalıştığı zaman ConstraintStrategy bu işlemin olumsuz bir şekilde sonuçlanması için false değeri döndürecektir. RoleCheck nesnesinin görevi uygulama ve kural motoru arasında mantıksal ifadelerin taşınabilmesidir zira Drools ve uygulama arasında liste türü değişkenler taşınabilmesine rağmen boolean değişkenler taşınamamaktadır. Ancak bu sorun kural motoru ve uygulama arasında kullanılacak olan mantıksal ifadenin RoleCheck sınıfının üye alanı yapılmasıyla aşılmıştır. Şekil 4.4'te, yukarıdaki paragrafta bahsi geçen RoleCheck ve Liste nesnelerinin ConstraintDroolsManager sınıfı içerisindeki kullanımı approveAddRole metodunun kaynak kodunda görülmektedir. Kod içerisinde ksession.insert ifadesi ile uygulama

aktör nesnesi ve rolecheck nesnesi çalışma hafızasına aktarılmaktadır. ksession.fireAllRules() ifadesi ise, kural dosyasında o an için hangi kural uygunsa, kural motoru tarafından o kuralın çalıştırılmasını sağlayan komuttur. logger.close() ifadesi ise, tıpkı bir veritabanı üzerinde yürütülen herhangi bir işlemden sonra, kurulan veritabanı bağlantısını kapatma işlemine benzetilebilir.

```
public boolean approveAddRole(String parentClassName, String
childClassName) {
      roleList.add(childClassName);
      roleCheck = new RoleCheck();
      roleCheck.setArg0(parentClassName);
      roleCheck.setArg1(childClassName);
      roleCheck.setCheck(true);
      roleCheck.setMethodName("add");
      ksession.setGlobal("roleListGlobal", roleList);
      ksession.insert(myActor);
      ksession.insert(roleCheck);
      ksession.fireAllRules();
      logger.close();
      return roleCheck.getCheck();
}
```

Şekil 4.4 ConstraintDroolsManager içerisindeki approveAddRole Metodu

ConstraintDroolsManager sınıfı içerisinde gerçeklenmesi sağlanan bir diğer metot da approveSuspend metodudur. approveSuspend metodu, aktör nesnesinin o an için ertelenmesini istediğimiz rolleri için kullanılmaktadır. Bu metot ile yapılmak istenen, kural dosyası içerisinde, kullanıcılar tarafından alana özgü hazırlanan ve herhangi bir rolün o anda askıya alınabilir olup olmadığının sonucunu bildiren kuralın çıktısını uygulamaya iletmektir. Şekil 4.5'de approveSuspend metodu görülmektedir.

```
public boolean approveSuspend(String parentClassName, String
childClassName) {roleCheck = new RoleCheck();
roleCheck.setCheck(true);
roleCheck.setArg0(parentClassName);
roleCheck.setArg1(childClassName);
roleCheck.setMethodName("suspend");ksession.insert(myActor);
ksession.insert(roleCheck);
ksession.fireAllRules();logger.close();
return roleCheck.getCheck();}
```

Şekil 4.5 ConstraintDroolsManager içerisindeki approveSuspend Metodu

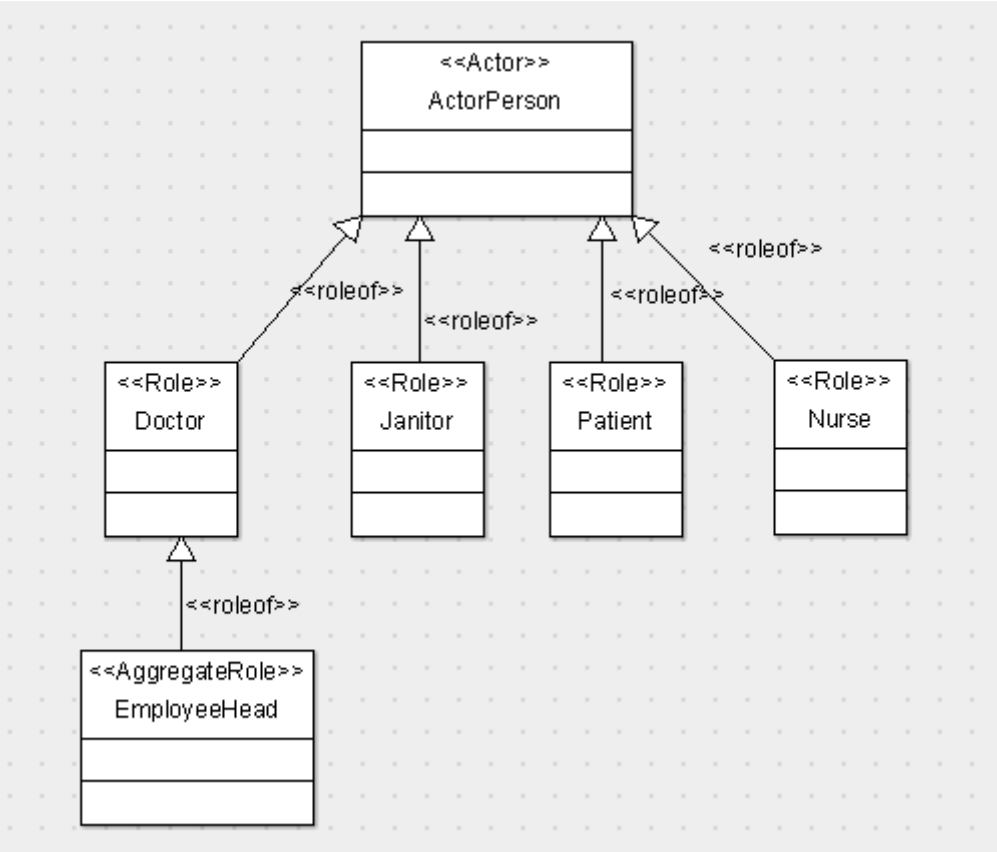
BÖLÜM 5

ÖRNEK UYGULAMA

Bu bölümde önerilen yaklaşım kullanılarak, örnek bir rol hiyerarşisinde izin verilmeyen kural dışı ilişkilerin nasıl önlenebileceği anlatılacaktır. Önleme eylemlerinin başarılı olduğunun gösterilmesi için çeşitli birim sınamaları (unit test) hazırlanacaktır. Birim sınama gereci olarak jUnit seçilmekle birlikte, hazırlanan birim sınamaları başka sınama gereçlerine de taşınabilir.

5.1 Modellenen Uygulama

İçinde insan öğesi olan sistemler, çoğu zaman dinamik sistemleri oluşturur. İnsan doğasının zamanla değişen ve evrimleşen dinamik yapısı, bu özelliğini içinde bulunduğu sisteme de yansıtır. Bir insan herhangi bir anda, çeşitli ortak alanlara ait olmaları nedeniyle birbirleriyle ilgili olan, çeşitli yükümlülükleri yerine getirir. Bu tür ortamların her biri, roller için örnek uygulama alanları oluşturur. Tez çalışması kapsamında örnek bir de uygulama geliştirmek için, böylesi bir alan olan hastane ortamı seçilmiştir. Örnek uygulama olarak, basit bir hastane konsol uygulaması hazırlanmıştır. Gerçeklenen sistemdeki sınıf ve nesne düzeyi kalıtım ilişkileri, Şekil 5.1'deki UML şemasında görülebilir.



Şekil 5. 1 Örnek uygulamadaki sınıf ve nesne düzeyi kalıtım ilişkileri

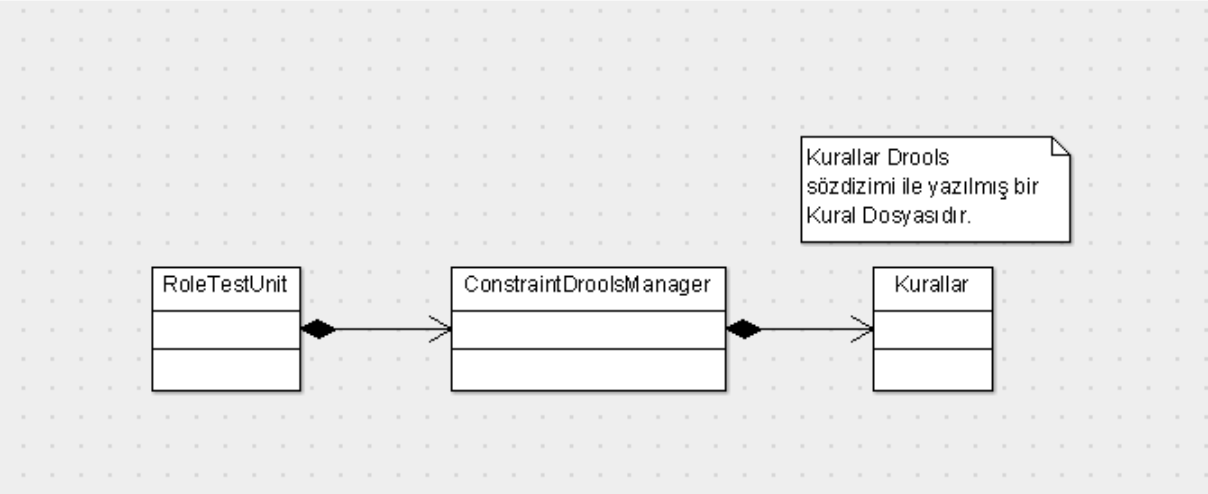
Uygulama UML şemasının detaylarını kısaca özetlemek gerekirse; ActorPerson sınıfı Jawiro.Actor sınıfından, adı Role ile başlayan sınıflar ise Jawiro.Role sınıfından kalıtımla türetilmiştir. RoleDoctor sınıfı doktor rolünü, RoleNurse sınıfı hemşire rolünü, RoleJanitor sınıfı ise hastane hizmetlisi (odacı) rolünü modeller. RoleEmployeeHead sınıfı ise örnek uygulamamızda özelleşmiş bir rol olan başhekim rolüne karşılık gelmektedir: Başhekim olabilmek için bir kişinin aynı zamanda bir doktor olması gerekmektedir.

5.2 Modellenen Uygulamanın Detayları

Tezimizin asıl amacı, Jawiro rol tabanlı modelleme yöntemi kullanılarak hazırlanmış herhangi bir uygulamada aykırı rol ilişkilerinin ve doğal olmayan rol atama, durdurma ve iptali gibi operasyonların önlenmesinin kural motorları vasıtasıyla sağlanması olduğu için uygulamanın, gerçek bir otomasyonun sağlaması gereken tüm işlevleri sağlamasının gerekli olmadığı düşünülmüştür. Bu sebeple konsol uygulaması hazırlanmış ve gerekli testlerin standartları sağlaması ve sonuçların rahatça

izlenebilmesi amacıyla, örnek uygulama JUnit kütüphanesi kullanılarak gerçeklenmiştir. Junit kütüphanesinin kullanımının örneklerle anlatılması, bu tez çalışmasının kapsamı dışındadır. İlgilenenler Tahchiev ve arkadaşlarının çalışmasını[9] veya Junit web sayfasını[10] inceleyebilir.

Uygulamanın domain kısmı, RoleTestUnit isimli bir java dosyası içerisinde gerçeklenmiştir. Uygulama ile sürekli etkileşim içerisinde olacak olan ve alana özel kuralların Drools sözdizim kurallarına göre kodlandığı kural dosyası ise Kurallar.drl dosyasıdır. Örnek uygulamada kullandığımız bir diğer çok önemli sınıfımızda, Kurallar.drl dosyası ile RoleTestUnit Java sınıfı arasındaki etkileşimi sağlayan bölüm 4.1'de ayrıntılı olarak ele aldığımız ConstraintDroolsManager sınıfıdır. Bir kısıt yöneticisi olarak davranan ConstraintDroolsManager sınıfı, kullanıcıya başka bir sözdizim kuralına bağlı kalınarak gerçek yaşamdaki projelerde bir başka kişi yada kişiler tarafından hazırlanacak olan bir kural dosyası içerisinde kodlanmış kuralları kullanma fırsatı sunar. Genel olarak bu yapıdan bahsedecek olursak, RoleTestUnit sınıfında uygulamanın alana özel aktör ve rolleri tanımlanmakta, daha sonra bu aktör ve roller ConstraintDroolsManager sınıfına ilgili kuralların çalıştırılabilmesi amacıyla verilmektedir. Bu sınıflar arasındaki etkileşim Şekil 5.2'de görüldüğü gibidir.



Şekil 5. 2 Hastane uygulaması sınıfları arasındaki ilişki

Şekil 5.2'den de anlaşılacağı üzere RoleTestUnit sınıfı ConstraintDroolsManager sınıfını, ConstraintDroolsManager sınıfı da Kurallar kural dosyasını kullanmaktadır.

5.2.1 RoleTestUnit Sınıfı

RoleTestUnit sınıfı, Java JUnit standartlarına uygun ve kural motorumuzun testini sağlıklı bir şekilde yapabilmek için hazırlanmış bir sınıftır. RoleTestUnit sınıfında hastane uygulaması alanına özgü aktörler ve roller ilklendirilmiş ve her bir kısıt testinin sonuçlarını görebilmek amacıyla test metotları yazılmıştır. Her bir metodun doğru çalışıp çalışmadığı, açılan JUnit sekmesinde metot isimlerine karşılık gelen renklerden anlaşılmaktadır. Eğer bir metot ismine karşılık yeşil denetim pulu belirdiyse bu işaret metodumuzun test ten istediğimiz şekilde geçtiğini ve doğru çalıştığını göstermektedir. Buna karşılık eğer metodumuzun isminin karşısında kırmızı denetim pulu belirdiyse, bu da metodumuzun, testi geçemediğinin göstergesidir. RoleTestUnit sınıfını yakından inceleyecek olursak; genel olarak 3 kısımdan oluştuğu görülecektir. Birinci kısım, uygulamaya özgü alanların ilklendiği Şekil 5.3'deki setUp metodunun bulunduğu kısımdır: Bu metot içerisinde yapılan işlem; Zeynep ve Kemal adında iki tane aktör, daha sonra test metotları içerisinde kullanılacak olan alana özgü hemşire, doktor, hasta, başhekim rolleri ve son olarak da kural motoru ile uygulama arasındaki etkileşimi sağlayan ConstraintDroolsManager nesnesini tanımlamaktır.

```
protected void setUp() throws Exception {
 super.setUp();
       personZeynep = new ActorPerson("Zeynep");
       personKemal = new ActorPerson("Kemal");
       personX = new ActorPerson("X");
       employer = new RoleEmployer();
       yedekDoktor = new RoleDoctor(employer);
       doctor = new RoleDoctor(employer);
       nurse = new RoleNurse(employer);
       patient = new RolePatient();
       janitor = new RoleJanitor(employer);
       roleEmployeeHead = new RoleEmployeeHead(personX);
 constraintDroolsManager=new
 ConstraintDroolsManager(doctor,personZeynep);
 personZeynep.setConstraintManager(constraintDroolsManager);}
```

Şekil 5. 3 Uygulamaya özgü alanların ilklendiği setUp metodu

RoleTestUnit sınıfındaki ikinci kısım, tearDown metodudur. tearDown metodu ile yapılan işlem, setUp metodu içerisinde ilklemeleri gerçekleştirilen alanların, içlerinin boşaltılmasıdır. Şekil 5.4'de ilgili kod parçası görülmektedir.

```
protected void tearDown() throws Exception {
        super.tearDown();
        personZeynep = null;
        employer = null;
        yedekDoktor = null;
        doctor = null;
        nurse = null;
        patient = null;
        janitor = null;
        roleEmployeeHead = null;
        constraintDroolsManager = null;
        personKemal = null;

}
```

Şekil 5. 4 Uygulamaya özgü alanların içlerinin boşaltıldığı tearDown metodu

RoleTestUnit sınıfındaki üçüncü kısım, Drools kural dosyası içerisinde hazırlanan uygulamaya özgü kısıtları temsil eden kuralların doğru çalışıp çalışmadığını test etmeye yönelik olarak hazırlamış olduğumuz test metotlarımızdır. Uygulamamızda, başhekim rolünün bir aktöre atanması ve o an için herhangi bir aktörün doktor rolünün askıya alınıp alınamayacağının tespitine yönelik test metotları yazıldı. Fakat test metotlarımızın işletilmeden önce JUnit kütüphanesinin bir gereği olarak TestSuit e eklenmesi gerekmektedir. Bu durum Şekil 5.5'de gösterilmiştir.

```
public static Test suite(){
        TestSuite suite = new TestSuite();
        suite.addTest(new
RoleTestUnit("cannotAddDoctorRoleToNurseTest"));
        suite.addTest(new
RoleTestUnit("addEmployeeHeadRoleTest"));
        suite.addTest(new
RoleTestUnit("suspendDoctorRoleTest"));
        return suite;}
```

Şekil 5. 5 Test metotlarının TestSuit e eklenmesi

Test metotlarımızı ayrıntısıyla inceleyecek olursak; ilk test metodumuz, sistemde herhangi bir aktöre başhekim rolünü atayıp atayamayacağımızı bize bildiren addEmployeeHeadRoleTest metodudur. Şekil 5.6 metodumuzun kaynak kodunu göstermektedir. Uygulamamızda Zeynep adı ile tanımlanan aktöre öncelikle doktor rolü verilmekte ve sonrasında addEmployeeHeadRoleTest metodu içerisinde aynı isimdeki aktöre başhekimlik rolü verilip verilemeyeceği anlaşılmaya çalışılmaktadır. Uygulamamızın temel mantığına göre, daha öncesinde doktor rolüne sahip herhangi bir aktör istenildiği takdirde başhekimlik rolüne de sahip olabilmektedir. Bu yüzden metot çalıştırıldığı zaman bize sonuç olarak olumlu bir yanıt dönmesini beklemekteyiz. Yani

JUnit bize yeşil denetim pulunu gösterecektir. Zaten kod içerisinde yazılmış olan assertTrue ifadesi bu varsayımı ifade etmektedir.

```
public void addEmployeeHeadRoleTest(){
     personZeynep.addRole(doctor);
     personZeynep.addRole(roleEmployeeHead);

assertTrue(personZeynep.canSwitch("demoApp.RoleEmployeeHead")
);}
```

Şekil 5. 6 RoleTestUnit içerisindeki addEmployeeHeadRoleTest metodu

İkinci test metodumuz, hemşire rolüne sahip bir aktörün doktor rolüne sahip olup olamayacağını test etmeyi sağlayan Şekil 5.7'de kodu gösterilen cannotAddNurseRoleToDoctorTest metodudur. Hastane uygulamamızın bir gereği olarak, hemşire rolüne sahip bir aktör sonradan doktor rolüne sahip olamaz. Bu yüzden assertFalse(personZeynep.addRole(doctor)) iadesi kullanılmıştır. Bu ifade parantez içerisindeki işlemin yanlış bir işlem olmasını beklediğimizi anlatmaktadır.

```
public void cannotAddDoctorRoleToNurseTest(){
    personZeynep.addRole(nurse);
    assertFalse(personZeynep.addRole(doctor));
}
```

Şekil 5. 7 RoleTestUnit içerisindeki cannotAddDoctorRoleToNurseTest metodu

Üçüncü test metodumuz, Şekil 5.8 de görülen suspendDoctorRoleTest test metodudur. Zeynep isimli, hem doktor hem de başhekim rollerini üstlenmiş olan aktörün, sahibi olduğu doktor rolünü erteleyip erteleyemeyeceğini test etmeyi sağlayan suspendDoctorRoleTest metodudur. Hastane uygulamasının bir kuralı olarak, sistemde kendisini yedekleyebilecek bir doktor rolüne sahip bir aktör yoksa kişi doktor rolünü terk edemez. Yazdığımız test metodunda, Kemal isminde bir aktör tanımlanmış ve onun sistemdeki yedek doktor olması sağlanmıştır. Böylece Zeynep isimli aktör, üzerindeki doktor rolünü geçici olarak askıya alabilecek ve eğer varsa başhekimlik rolünü de Kemal ismindeki yedek doktora devredecektir. Test metodumuzun içerisinde Kemal isimli aktörü yedek doktor olarak belirlediğimiz için; assertTrue(personZeynep.suspendRole("demoApp.RoleDoctor")) ifadesi olumlu sonuç dönecektir. Bu test metot gövdesi içerisinde Zeynep isimli aktörün başhekimlik rolünü

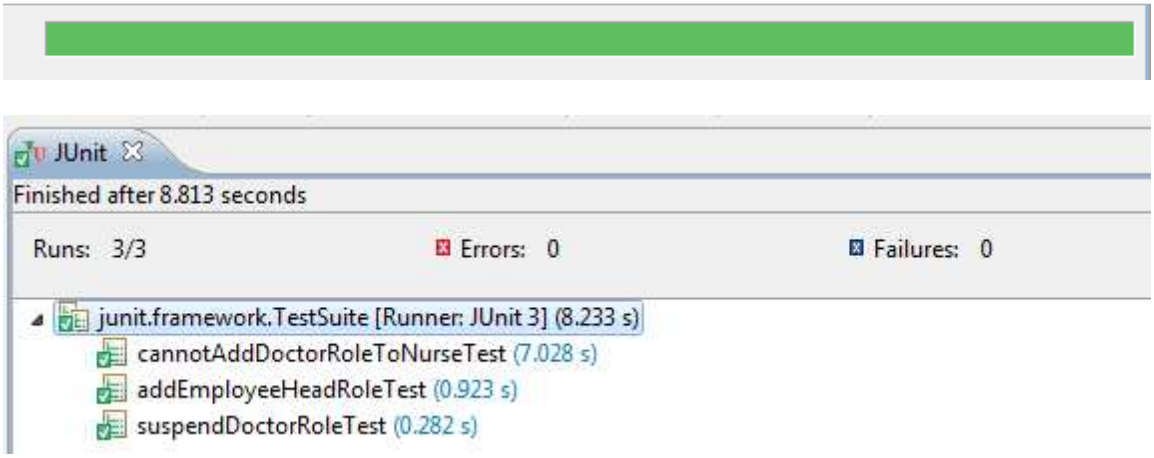
nasıl devredeceği kodlanmamıştır. İlgili kodlama programcının hazırlayacağı düşünülen kural dosyası içerisinde kodlanmıştır.

```
public void suspendDoctorRoleTest(){
        personZeynep.addRole(doctor);
        personZeynep.addRole(roleEmployeeHead);
        personKemal.addRole(yedekDoktor);
        doctor.setSubstitute(yedekDoktor);

assertTrue(personZeynep.suspendRole("demoApp.RoleDoctor"));}
```

Şekil 5. 8 RoleTestUnit içerisindeki suspendDoctorRoleTest metodu

Yukarıdaki üç metodun çalıştırılması sonucu jUnit ekranında ortaya çıkan görüntü Şekil 5.9' da görüldüğü gibidir. jUnit ekranında görülen yeşil denetim pulu, tüm test metotlarının sınamadan beklediğimiz şekilde başarı ile geçtiğini göstermektedir. Yeşil denetim pulunun hemen altında ise her bir test metodu için ayrıntılı sınama sonucu görülmektedir. Eğer denetim pulu kırmızı olsaydı, bu sonuç test metotlarımızın başarılı olmadığının bir göstergesi olacaktı.



Şekil 5. 9 RoleTestUnit sınıfının çalıştırılması sonucu ortaya çıkan ekran görüntüsü.

5.2.2 Kurallar Dosyası

Kurallar dosyası, uygulamayı yazacak olan programcıların alana özgü kısıt ve kuralları Drools sözdizim kurallarına göre kodlayacakları kısımdır. Bizim örnek uygulamamız olan hastane uygulamasında, hastaneye özgü kısıtlar bu dosya içerisine, her bir kural, alana özgü bir kısıtı sağlayacak şekilde kodlanmıştır. Kural dosyası içerisinde Jawiro ya ait nesne tipinden değişken tanımlayabilmek için Jawiro paketi kural dosyasına import anahtar sözcüğü ile dahil edilmiştir. Aynı zamanda uygulama ve kural dosyası arasındaki haberleşmeyi sağlayacak olan RoleCheck ve roleListGlobal sınıflarıda kural

dosyasına import anahtar sözcüğü ile dahil edilmişlerdir. İlk kuralımız Şekil 5.10'daki AddRoleAfterDoctorRole_Rule kuralıdır. Bu kural ile sağlanmak istenen kısıt, kuralın isminden de anlaşılacağı üzere doktor rolüne sahip bir aktöre, hemşire ve odacı rollerinin eklenmesini engellemektir. Bunun sağlanabilmesi için en son eklenen rolü bilmeye ihtiyacımız vardır. Bu ihtiyaç, uygulama ile kural dosyası arasındaki veri alışverişini sağlayan roleListGlobal liste elemanı vasıtasıyla karşılanmıştır. roleListGlobal adlı listenin son elemanı kontrol edilerek, uygulama tarafında en son hangi rol eklendiği tespit edilebilir.

```
import Jawiro.*;
import com.sample.java.RoleCheck;
global java.util.LinkedList roleListGlobal;

rule "AddRoleAfterDoctorRole_Rule"
when
a : demoApp.ActorPerson()
r : Role Check()
eval(r.getMethodName().equals("add")&&
(a.canSwitch("demoApp.RoleDoctor") &&
roleListGlobal.getLast().toString().equals("demoApp.RoleNurse
"))|| (a.canSwitch("demoApp.RoleDoctor") &&
roleListGlobal.getLast().toString().equals("demoApp.RoleJanit
or"))
then
r.setCheck(false);
System.out.println("Doctor ve
"+roleListGlobal.getLast().toString().substring(12)+" rolleri
bir arada ustlenilemez.");
System.out.println(roleListGlobal.getLast().toString().substr
ing(12)+" Rolu Terk Edilecek...!");
End
```

Şekil 5. 10 AddRoleAfterDoctorRole_Rule Kuralı

İkinci kuralımız, Şekil 5.11'deki AddRoleAfterNurseRole_Rule kuralıdır. Bu kural da bir önceki kuralımızla aynı mantıkta çalışarak hemşire rolü eklenmiş bir aktöre, başhekim, doktor veya odacı rollerinin eklenmesini engellemektedir. Yine bir önceki kuraldaki gibi aktöre en son eklenen rol, liste değişkenine(roleListGlobal) en son eklenen elemana bakılarak anlaşılmaktadır.

```
rule "AddRoleAfterNurseRole_Rule"
when
a : demoApp.ActorPerson()
r : RoleCheck()
eval(r.getMethodName().equals("add")&&
(a.canSwitch("demoApp.RoleNurse") &&
roleListGlobal.getLast().toString().equals("demoApp.RoleEmplo
yeeHead"))
|| (a.canSwitch("demoApp.RoleNurse") &&
roleListGlobal.getLast().toString().equals("demoApp.RoleDocto
r"))
         || (a.canSwitch("demoApp.RoleNurse") &&
roleListGlobal.getLast().toString().equals("demoApp.RoleJanit
or")))
Then
r.setCheck(false);
System.out.println("Nurse ve
"+roleListGlobal.getLast().toString().substring(12)+" rolleri
bir arada ustlenilemez.");
System.out.println(roleListGlobal.getLast().toString().substr
ing(12)+" Rolu Terk Edilecek...!");
```

Şekil 5. 11 AddRoleAfterNurseRole_Rule Kuralı

AddRoleAfterDoctorRole_Rule, AddRoleAfterNurseRole_Rule kurallarında uygulanan yöntemin aynısı AddRoleAfterJanitorRole_Rule kuralında da uygulandığı için burada açıklamasına girilmesine gerek duyulmamıştır.

Bir sonraki kuralımız, yalnızca doktor rolüne sahip kişilerin başhekimlik rolünü üstlenebilmesini sağlayan AddEmployeeHeadRoleOnlyAfterDoctorRole_Rule kuralıdır. Şekil 5.12'de ilgili kural kodu görülmektedir. Bu kuralımızda da eğer başhekimlik rolüne sahip olmak isteyen aktör, önkoşul olarak doktor rolüne sahip değilse, bu isteği geri çevrilecektir.

```
rule "AddEmployeeHeadRoleOnlyAfterDoctorRole_Rule"
when
a : demoApp.ActorPerson()
r:RoleCheck()
eval(r.getMethodName().equals("add")&&roleListGlobal.getLast(
).toString().equals("demoApp.RoleEmployeeHead") &&
!(a.canSwitch("demoApp.RoleDoctor")) )
 then
 r.setCheck(false);
 System.out.println("Bashekim rolunun eklenebilmesi
icinkisinin doktor olmasi gerekiyor..");end
```

Şekil 5. 12 AddEmployeeHeadRoleOnlyAfterDoctorRole_Rule kural kodu

Sonraki kuralımız, aktörlerin sahip olduğu rollerden askıya alınmak istenenlerin askıya alınıp alınamayacağına karar veren iki kuralımızdan birisi olan cannotSuspendRole_Rule kuralıdır. Kuralın isminden de anlaşılacağı üzere, bu kural belli şartların sağlanması

durumunda rollerin askıya alınmasını engelleyici bir görev üstlenmiştir. Bu şartlar şekil 5.13'deki koddan da anlaşılacağı üzere, eğer bir aktör doktor rolünü üstlenmiş ve acil durumlarda o aktörün bu rolünü üstlenecek herhangi bir yedeği yoksa aktörün sahibi olduğu doktor rolü askıya alınamayacaktır. Buradan çıkarılacak diğer bir sonuç, alan kuralı gereği diğer roller sorunsuz olarak askıya alınabilecektir.

```
rule "cannotSuspendRole_Rule"
when
a : demoApp.ActorPerson()
r : RoleCheck()
eval(r.getMethodName().equals("suspend")&&
r.getArg1().equals("demoApp.RoleDoctor")&&((demoApp.RoleDocto
r)a.as("demoApp.RoleDoctor")).getSubstitute() == null)
   then
    r.setCheck(false);
System.out.println("Bu rolu suspend edemezsiniz..!");
end
```

Şekil 5. 13 cannotSuspendRole_Rule kuralı

Rol askıya alma ile ilgili hazırlanan bir diğer kuralımızda, canSuspendRole_Rule kuralıdır. Şekil 5.14'de kodlaması verilen bu kural ile yapılmak istenen, yedeği olan bir doktorun, doktorluk rolünün askıya alınmasını sağlamaktır. Fakat bu kural da farklı olarak roleTransfer(ActorPerson) fonksiyonu kullanılmıştır. Şekil 5.15'de kodlaması verilen fonksiyon, doktor rolü askıya alınmak istenen kişinin, başhekimlik rolüne sahip olup olmadığına bakıyor ve eğer kişi başhekimlik rolüne sahip ise öncelikli olarak sahip olduğu başhekimlik, yedeği durumundaki aktöre aktarılmaktadır. Kural dosyası içerisinde, fonksiyon kullanılmasının temel amacı aynı işlemin birden fazla kez yapılmasını gerektirecek durumlarda bu işlemin bir kez fonksiyon tarafından yaptırılıp diğer kurallar tarafından kullanılmasını sağlamaktır.

```
rule "canSuspendRole_Rule"
      when
      a : demoApp.ActorPerson()
      r : RoleCheck()
eval(r.getMethodName().equals("suspend") &&
r.getArg1().equals("demoApp.RoleDoctor")&&((demoApp.RoleDocto
r)a.as("demoApp.RoleDoctor")).getSubstitute() != null)
then
      roleTransfer(a);
    System.out.println("Suspend İşlemi Başarılı..!");
end
```

Şekil 5. 14 canSuspendRole_Rule kuralı

```
function boolean roleTransfer(demoApp.ActorPerson a){
RoleEmployeeHead headRole;
RoleDoctor substitute;
RoleDoctor doctor =
(demoApp.RoleDoctor)a.as("demoApp.RoleDoctor");
substitute = doctor.getSubstitute();
//System.out.println("Rol Transfer Islemine Giris..!");
if( doctor.canSwitch("demoApp.RoleEmployeeHead") ) {
headRole =
(RoleEmployeeHead)doctor.as("demoApp.RoleEmployeeHead");
headRole.transfer( substitute.getActor() );
System.out.println("Transfer Islemi Basarili..!");}
return true;}
```

Şekil 5. 15 roleTransfer Metodu

BÖLÜM 6

BAŞARIM ÖLÇÜMLERİ

Intel Core 2 Duo 2 Ghz işlemcili, 1 Gb lık RAM belleğe sahip bilgisayar üzerinde yapılan test ile approveAddRole ve suspendRole metotları, kurallar önce Bölüm 6.1'de gösterildiği gibi yalnızca java kodu yazılarak, sonra da Bölüm 6.2'de anlatıldığı üzere Drools kural motorunda yazılarak test edilmiştir. Test etme yöntemi olarak koda başlarken ve kodu bitirirken System.currentTimeMillis() komutu kullanılmış ve ilk değer, son değerden çıkarılarak bu işlemin yapılabilmesi için geçen zaman hesaplanmıştır. Ölçüm yapılırken iki değere dikkat edilmiştir; bu iki değer Drools un ilk yüklemesi için geçen zaman(firstRun) ve yükleme gerçekleştikten sonra kuralların ortalama çalışma zamanıdır(timeofRunning). Bu iki değer TestResults adlı sınıfın iki üyesinde tutulmuştur. Şekil 6.1'de TestResults sınıfının yapısı görülmektedir.

```
public class TestResults {
private long firstRun;
private long timeofRunning;

public long getFirstRun() {
return firstRun;
}
public void setFirstRun(long firstRun) {
this.firstRun = firstRun;
}
public long getTimeofRunning() {
return timeofRunning;
}
public void setTimeofRunning(long timeofRunning) {
this.timeofRunning = timeofRunning;
}}
```

Şekil 6. 1 Performans hesaplama sonuçlarının tutulduğu TestResults sınıfı

6.1 Drools Kural Motoru Kullanılmadan Yapılan Ölçüm

Kurallar, ConstraintManager sınıfı içerisinde Şekil 6.2'deki gibi düz java kodu ile yazılmıştır. Burada bütün approve metotlarının gösterilmesinden ziyade yalnızca approveSuspend metodu örneklenerek durum izah edilmeye çalışılmıştır.

```
public class ConstraintManager implements ConstraintStrategy
{
public boolean approveSuspend( String parentClassName,
String childClassName ) {
RoleEmployeeHead headRole;
RoleDoctor substitute;
RoleDoctor doctor =
(demoApp.RoleDoctor)myActor.as("demoApp.RoleDoctor");
if( childClassName.compareTo("demoApp.RoleDoctor") == 0 ) {
substitute = doctor.getSubstitute();
if( substitute == null )
return false;
while( doctor.canSwitch("demoApp.RoleEmployeeHead") ) {
headRole = (RoleEmployeeHead)doctor.as
("demoApp.RoleEmployeeHead");
headRole.transfer( substitute.getActor() );
}
}
return true;
}}
```

Şekil 6. 2 Saf java kodu ile yazılmış approveSuspend Metodu

Burada hazırlanan ConstraintManager sınıfı ikinci aşamada, Şekil 6.3'deki gibi TestWithoutDrools sınıfı içerisinde kullanılmıştır.

```
public class TestWithoutDrools{

    public void suspendDoctorRoleTest(){
    RoleEmployer employer = new RoleEmployer();
    RoleDoctor doctor = new RoleDoctor(employer) ;
    RoleDoctor yedekDoktor = new RoleDoctor(employer);
    ActorPerson personX= new ActorPerson("X");
    RoleEmployeeHead roleEmployeeHead = new
RoleEmployeeHead(personX);
    ActorPerson personZeynep = new ActorPerson("Zeynep");
    ConstraintManager constraintManager = new
ConstraintManager();
        personZeynep.setConstraintManager(constraintManager);
        constraintManager.setActor(personZeynep);
        personZeynep.addRole(doctor);
        personZeynep.addRole(roleEmployeeHead);
        ActorPerson  personKemal = new ActorPerson("Kemal");
        personKemal.addRole(yedekDoktor);
        doctor.setSubstitute(yedekDoktor);
        personZeynep.suspendRole("demoApp.RoleDoctor");
    }}
```

Şekil 6. 3 ConstraintManager ın TestWithoutDrools içerisinde kullanımı

Bu testimiz deki son aşamada, TestWithoutDrools sınıfını asıl ölçümü yapacağımız test sınıfı içerisinde kullanmak olacaktır. Şekil 6.4'de görüleceği üzere ölçüme başlamadan önce ve sonra System.currentTimeMillis() deyimi arada geçen zamanı ölçmek amacıyla kullanılmıştır. İlk döngüdeki geçen değer, programın ilk yüklemesi için geçen zamandır. Sonraki döngülerin ortalama değeri ise, programın çalışma süresini vermektedir. Drools kullanılarak hazırlanan performans ölçüm mekanizması da aynı mantıkla hazırlanmıştır.

```
public static void main(String[] args) {
 TestResults tr = new TestResults();
 long sonuc,sum = 0;
 int i=0;
 for(i = 0;i<5;i++){
long start1 = System.currentTimeMillis();
 TestWithoutDrools twd = new TestWithoutDrools();
twd.addEmployeeHeadRoleTest();
 twd.cannotAddDoctorRoleToNurseTest();
 twd.suspendDoctorRoleTest();
 long end1 = System.currentTimeMillis();
 sonuc = end1-start1;
 if(i==0){
 tr.setFirstRun(sonuc);
 }else {
  sum += sonuc;
 }
 }
 tr.setTimeofRunning(sum/i);
 System.out.println("First RunTime:"+tr.getFirstRun());
 System.out.println("Average RunTime
:"+tr.getTimeofRunning());
}
```

Şekil 6. 4 Kural motoru olmaksızın geçen süreyi ölçen kod parçası

6.2 Drools Kural Motoru Kullanılarak Yapılan Ölçüm

Buradaki ölçüm kural motorunun çalışma hızını ölçmeye yönelik olacaktır. Burada yine kural motoru olmadan yapılan ölçümde olduğu gibi ölçüm sırasında birden fazla java sınıfından faydalanılmıştır. İlk sınıfımız kural motoru ile iletişim kurarak kuralların çalıştırılmasını sağlayan Şekil 6.5'teki ConstraintDroolsManager sınıfıdır.

```
public boolean approveSuspend(String parentClassName, String
childClassName) {roleCheck = new RoleCheck();
roleCheck.setCheck(true);
roleCheck.setArg0(parentClassName);
roleCheck.setArg1(childClassName);
roleCheck.setMethodName("suspend");
ksession.insert(myActor);
ksession.insert(roleCheck);
ksession.fireAllRules();
logger.close();return roleCheck.getCheck();}
```

Şekil 6. 5 ConstraintDroolsManager sınıfı içerisindeki approveSuspend metodu

İkinci sınıfımız, ConstraintDroolsManager sınıfını kullanarak hazırlanmış olan Şekil 6.6'daki DroolsTest sınıfıdır. Burada ölçüm yapmak istediğimiz kuralla ilgili alana özel kod yazılmıştır.

```
public void suspendDoctorRoleTest(){
    RoleEmployer employer = new RoleEmployer();
    RoleDoctor doctor = new RoleDoctor(employer) ;
    RoleDoctor yedekDoktor = new RoleDoctor(employer);
    ActorPerson personX= new ActorPerson("X");
    RoleEmployeeHead roleEmployeeHead = new
RoleEmployeeHead(personX);
    ActorPerson personZeynep = new ActorPerson("Zeynep");
    ConstraintDroolsManager constraintDroolsManager = new
ConstraintDroolsManager(personZeynep,"C:\\Users\\iski\\Deskto
p\\MyUsbFiles\\tezEclipse\\NewDroolProjects\\TezSon3\\src\\ma
in\\rules\\Kurallar.drl");

personZeynep.setConstraintManager(constraintDroolsManager);
        personZeynep.addRole(doctor);
        personZeynep.addRole(roleEmployeeHead);
        ActorPerson  personKemal = new ActorPerson("Kemal");
        personKemal.addRole(yedekDoktor);
    doctor.setSubstitute(yedekDoktor);
        personZeynep.suspendRole("demoApp.RoleDoctor");
}
```

Şekil 6. 6 DroolsTest sınıfı içerisindeki suspendDoctorRoleTest metodu

Son olarak da, Şekil 6.7'de DroolsTest sınıfını kullanan ve TestResults sıfının ölçümle ilgili alanlarının doldurulmasını sağlayan PerformanceTesting sınıfı görülmektedir.

```
public static void main(String[] args) {
 TestResults tr = new TestResults();
 long sonuc,sum = 0;
 int i=0;
 for(i = 0;i<5;i++){
 long start1 = System.currentTimeMillis();
 DroolsTest dt = new DroolsTest();
 dt.addEmployeeHeadRoleTest();
 dt.cannotAddDoctorRoleToNurseTest();
 dt.suspendDoctorRoleTest();
 long end1 = System.currentTimeMillis();
 sonuc = end1-start1;
 if(i==0){
 tr.setFirstRun(sonuc);
 }else {
  sum += sonuc;
 }
 }
 tr.setTimeofRunning(sum/i);
 System.out.println("First RunTime:"+tr.getFirstRun());
 System.out.println("Average RunTime
:"+tr.getTimeofRunning());}
```

Şekil 6. 7 Kural motoru kullanılarak yapılan test sonucunu veren kod parçası

6.3 Performans Testi Sonuçları

Sonuç olarak Drools kural motoru kullanılarak yapılan test sonucunda ilk yükleme için geçen süre 3800 ms iken, ortalama çalışma süresi 421 ms olarak tespit edilmiştir. Drools kural motoru kullanılmadan yapılan test sonucunda ise, ilk yükleme için geçen süre 100 ms ve ortalama çalışma süresi ise 1 ms olarak tespit edilmiştir. Burada her ne kadar Drools kural motoru kullanılarak yazılan uygulama, Drools kural motoru kullanılmadan hazırlanan uygulamaya göre daha az avantajlı görünse de, günümüzdeki işlemci ve ram performanslarının yüksekliği düşünülecek olursa milisaniyeler mertebesindeki değerlerin herhangi bir anlamının olmadığı görülecektir. Bunun yanı sıra, Drools un sözdiziminin basitliği nedeniyle iş mantığının yazım sürecini çok ciddi bir oranda düşüreceği göz önünde bulundurulursa, bu şekildeki bir kullanımın daha büyük uygulamalarda çok avantajlı olacağı aşikârdır.

BÖLÜM 7

SONUÇ VE ÖNERİLER

Dinamik sistemlerin etkin bir biçimde modellenmesi için önerilen çözümlerden biri de rol modelleridir. Rol modelleri dinamizm sağlama hedefine ulaşmaya çalışırken diğer bazı noktaları gözden kaçırabilmekte ya da en azından yeterince pekiştirememektedir. Aykırı rol ilişkilerinin önlenmesi bu eksik kalan noktalardan biridir. Kural motorları kullanılarak, bu eksiğin giderilmesi için çeşitli adımlar atılabilir. Bu çalışmada da rol modelleri arasından Jawiro ve kural motorları arasından da Drools seçilerek, Jawiro'nun çok basit olarak sunduğu aykırı rol ilişkilerini önleme mekanizması iyileştirilmeye çalışılmıştır. Bu amaçla gerçeklenen sistemin yaptığı temel işlem, uygulama alanına yönelik program kodu ile uygulama kurallarını birbirinden ayırmaktır. Bu sayede rol modelleri kullanılarak gerçeklenen bir yazılım projesinde, yazılım geliştiriciler ve iş analistleri birbirlerini engellemeden çalışabileceklerdir. Çeşitli roller arasında ne tür ilişkilere izin verileceği veya verilmeyeceğinin iş analistleri tarafından ve yazılım geliştirme ekibinin etkinliklerinden bağımsız olarak kodlanabilmesi, tez kapsamında gerçeklenen sistem ile mümkün olmuştur.

Tez çalışması ile ortaya konulan yapının sağladığı diğer iyileştirme, kuralların merkezi bir yerden çağırılabilmesinin sağlanması ve bu sayede kuralların yönetiminin etkinliğinin arttırılmasıdır.

Bu çalışma ile yapılan iyileştirmeler, sıradan bir bilgisayar sisteminin çalışma zamanı başarımında hissedilir bir düşme pahasına elde edilmektedir. Bu nedenle arzu edilen kolaylıklara ulaşmak isteyenlerin daha hafif sıklet bir kural motoru kullanmaları önerilir. Bir başka deyişle, tez çalışmasında önerilen yaklaşımın mevcut haliyle yanıt süresinden

(response time) çok, doğruluğun (accuracy) önemli olduğu uygulamalarda kullanılması tavsiye edilir.

KAYNAKLAR

[1] Selçuk,Y., (2006). "Nesneye Yönelik Programlamaya Rol Desteğinin Kazandırılması", Doktora Tezi, İTÜ Fen Bilimleri Enstitüsü, İstanbul.

[2] Selcuk,Y.E. ve Erdogan N., (2011). "Role models-implementation-based approaches to using roles" Software-Practice & Experience, 41:1-22.

[3] Kural Motoru, http://www.jboss.org/drools, 04 Tem 2011.

[4] Kural Motorları, http://www.w3.org/2004/12/rules-ws/paper/55/, 04 Tem 2011.

[5] KuralMotoru, http://logic.stanford.edu/POEM/externalpapers/iRules/WPJRules50Strengths.pdf, 04 Tem 2011.

[6] Kural Motoru, www.jessrules.com/,01 Haz 2011

[7] Kural Motoru, http://www.hammurapi.biz,05 Tem 2011.

[8] Rete Algoritması, http://www.cis.temple.edu/~ingargio/cis587/readings/rete.html, 05Tem 2011.

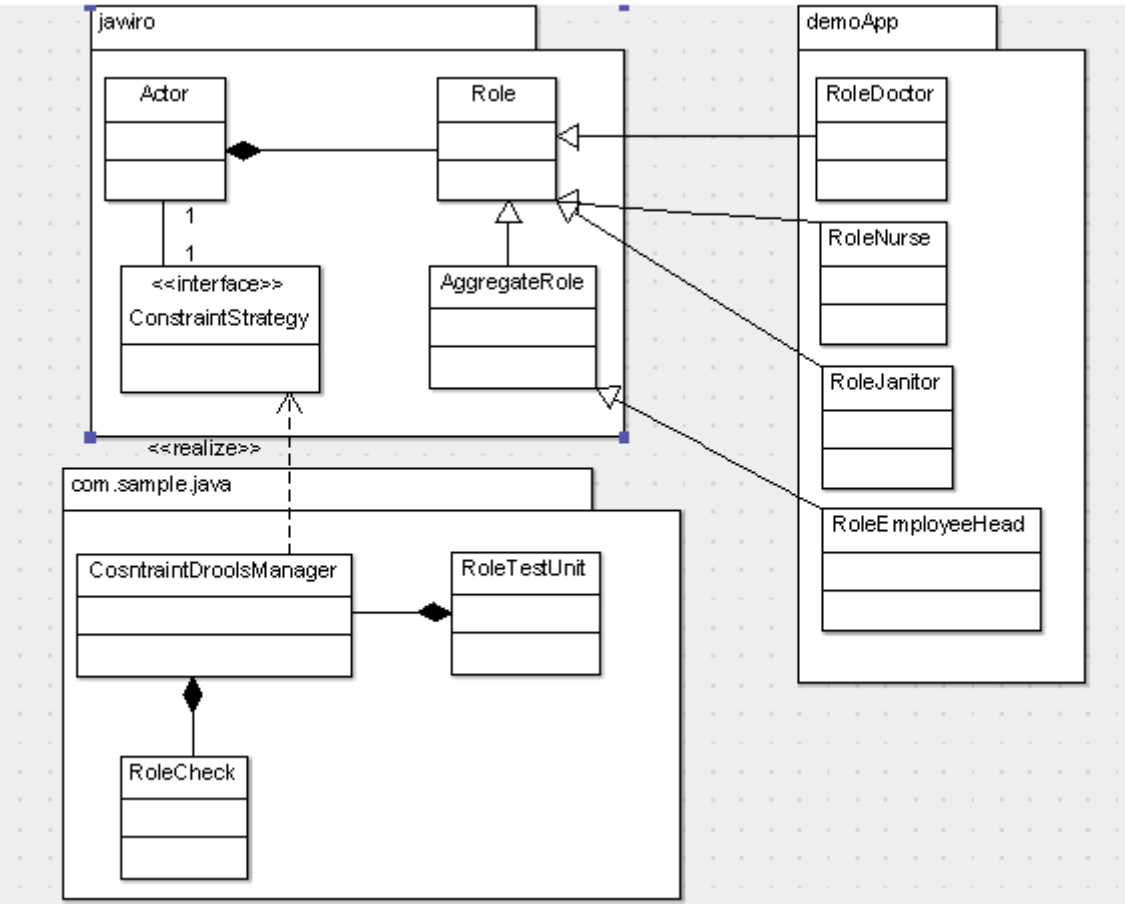
[9] Tachiev, Leme, Massol ve Gregory., (2010). Junit in Action, İkinci Basım, Manning Publications., Stamford.

[10] Java Birim Sınama Gereci, www.junit.org, 04 Tem 2011

EK-A

UYGULAMA MODELİ

Tez metninde değinilen sınıfları gösteren UML şeması aşağıdaki şekilde görüldüğü gibidir. Tez çalışması kapsamında gerçeklenen sınıflar com.sample.java paketinde yer almaktadır. Rol ilişkisi kurallarının incelendiği örnek uygulama alanı demoApp paketinde yer almakta, kullanılan Jawiro rol modeli ise adı ile aynı olan pakette bulunmaktadır.



ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı	:Önder GÜLER
Doğum Tarihi ve Yeri	:01/01/1982 – Zile/TOKAT
Yabancı Dili	:İngilizce
E-posta	:onder.guler@yahoo.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Y. Lisans	Bilgisayar Mühendisliği	Yıldız Teknik Üniversitesi	2011
Lisans	Bilgisayar Mühendisliği	Ege Üniversitesi	2007
Lise	Bilgisayar	ITO-ATL	2000

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2008	İski Genel Müdürlüğü	Uzman Oracle Developer