T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ROL MODELLERİNDE BAĞLAM KULLANIMI

MEHMET PEKMEZCİ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
YRD. DOÇ. DR. YUNUS EMRE SELÇUK

İSTANBUL, 2012

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ROL MODELLERİNDE BAĞLAM KULLANIMI

Mehmet PEKMEZCİ tarafından hazırlanan tez çalışması 11/10/2012 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik ve Haberleşme Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Yrd. Doç. Dr. Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Yrd. Doç. Dr. Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi _____

Prof. Dr. Selim AKYOKUŞ

Doğuş Üniversitesi _____

Yrd. Doç. Dr. Utku KALAY

Yıldız Teknik Üniversitesi _____

ÖNSÖZ

Bu çalışma rol modellerinde bağlam kullanımını sağlayabilmek adına gerçekleştirilmiştir. Yrd. Doç. Dr. Yunus Emre Selçuk'un daha önceki çalışmalarında sunmuş olduğu JAWIRO (Extending Java With Roles) anaçatısı, bu tez çalışmasının ilham kaynağı olmuştur.

Literatür taramalarında ön plana çıkan, JAWIRO altyapısında sunulmamış olan bağlam desteğini sağlamış olan Object Teams/Java altyapısından esinlenilerek, sadece Java tabanlı bir altyapıda çözüm sunulabilmesi hedeflenmiştir.

Çözüm için birçok farklı altyapı, birçok farklı senaryo üretilmiş, bazen deneme yanılma yöntemiyle, bazen önceki çalışmalardan edinilen tecrübelerle çalışma son halini almıştır.

Tüm yüksek lisans ve tez çalışmam boyunca desteği, yönlendirmesi ve pozitif yaklaşımı ile tüm süreç boyunca destek olan danışmanım Yrd. Doç. Dr. Yunus Emre Selçuk'a, özellikle tez yazım sürecinde desteklerini esirgemeyip, sabırla destek olan aileme ve arkadaşlarıma teşekkür ederim.

Ağustos, 2012

Mehmet PEKMEZCİ

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

DOM	Document Object Model
JAWIRO	Extending Java With Roles
JCORE	Java Contextual Role Enablement
JVM	Java Virtual Machine
OOP	Object Oriented Programming (Nesne Yönelimli Programlama)
OT/J	Object Teams/Java
XML	Extensible Markup Language

ŞEKİL LİSTESİ

Sayfa

ÖZET

ROL MODELLERİNDE BAĞLAM KULLANIMI

Mehmet PEKMEZCİ

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

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

Rol tabanlı programlama üzerine son 10 yıl içerisinde çeşitli yaklaşımlar, standartlar ve bu standartları gerçekleyen çeşitli altyapılar sunulmuştur. Günümüzde halen güncelliğini koruyan bu konu üzerinde sunulmuş olan altyapılar incelendiğinde göze çarpan en büyük eksiklik, yaygın şekilde kullanılan, saf nesneye yönelimli programlama altyapısı ile tam bir gerçekleştirim sunulamamasıdır.

Sunulan gerçekleştirim çalışmalarından Object Teams/Java altyapısı belirlenen standartlara büyük ölçüde uyum sağlamış ve çeşitli yetenekleri ile programcıya uygun altyapıyı sunabilmiş olmasına rağmen, Java programlama diline yapmış olduğu yapısal eklemeler nedeniyle hem farklı bir dil üzerinde programlama gerçekleştirmek, hem farklı bir mantığı oturtmak, hem de farklı bir geliştirme ve derleme ortamı kullanmak gibi zorunlulukları beraberinde getirmektedir.

Buna alternatif olarak Yunus Emre SELÇUK tarafından sunulmuş olan JAWIRO altyapısı ise tamamen Java programlama dili ile gerçekleştirim sağlamış olmasına rağmen, bağlam desteğini sağlayamamaktadır.

Bu çalışma içerisinde, sadece Java programlama dili temeline dayanan, hem bağlam desteğini sağlayabilmek, hem de sunulan gerçekleştirimi (anaçatıyı) kullanan programcıya çeşitli esneklikler ve kullanım kolaylıkları sağlayabilecek, hem de tanımlanmış olan rol modelleri standartlarına olabildiğince fazla uyum sağlayabilmek amaçlanmıştır.

Önceki paragraflarda açıklanmış olan sebeple gerçekleştirilen çalışma içerisinde sunulan altyapı –JCORE (Java COntextual Role Enablement) - bağlam, rol ve aktör kavramları üzerine kurulmuştur. Tüm geliştirmelerin Java ortamında gerçekleştirilmesi ve standart Java uygulaması şeklinde derlenmesi hedeflenmiş, dolayısıyla bu üç farklı kavram da standart Java sınıfları şeklinde gerçekleştirilecek bir altyapı sunulmuştur. Ancak rol modellemelerinin gerektirdiği dinamik altyapıyı sağlamak üzere kullanıcı tarafından geliştirilmiş ve derlenmiş olan Java sınıfları, JCORE altyapısı tarafından JVM'e yüklenmeden hemen önce değiştirilerek kullanıcının XML üzerinde konfigure etmiş olduğu rol modeline uygun Java sınıf dosyaları oluşturulup derlenip orjinalleri yerine kullanılmaktadır. Bu işlemler programcıyı etkilemeyecek şekilde, orjinal geliştirme sürecine yeni bir gereksinim eklemeden gerçekleştirilmektedir. Sonuçta mevcut çalışmalar ile ilgili olarak daha önce değerlendirilen zorunlulukların giderildiği esnek bir altyapı ortaya çıkmıştır.

Anahtar Kelimeler: rol tabanlı programlama, bağlam desteği, java rol modelleri

ABSTRACT

IMPLEMENTATION OF CONTEXT ON ROLE BASED PROGRAMMING

Mehmet PEKMEZCİ

Department of Computer Engineering

MSc. Thesis

Advisor: Assist. Prof. Dr. Yunus Emre Selçuk

For the last 10 years, several approaches, standards and various implementations for those standards have been proposed on role based programming. Today, when all the implementations for this topic, which still preserves attention, it can be noticed that there is no commonly used pure object oriented implementation.

Although one of the proposed solutions, Object Teams/Java framework provides a high level of compatibility with the defined standards and provides a proper framework along with several different capabilities; due to the structural additions to the Java programming language, it brings obligations for programming on a different programming language, use a different logic on programming and using a customized development and compilation environment.

Another alternative implementation, JAWIRO, proposed by Yunus Emre SELÇUK, is totally based on pure Java programming language. However, contexts are not supported in this framework.

Within this thesis, the aim is to provide context support with a flexible and user friendly framework which is fully based on pure Java and as much compatible as possible with defined role modeling standards.

Due to the reasons explained the previous paragraphs, JCORE (Java COntextual Role Enablement) framework has been proposed within the scope of this thesis study. JCORE is based on context, role and actor entities. Due to the aim of pure Java infrastructure, this framework enables the programmers to program all these three

entities can as standard Java classes. However, due to provide the dynamic infrastructure that role modeling require, all relevant Java classes, that have been developed by the programmer, will be replaced with the modified versions before they have been loaded into the JVM, with respect to the configuration defined on the JCORE XML configuration files. This operation is totally transparent to the programmer. As a result, a flexible infrastructure, without the discussed prerequisites, has been proposed with this thesis study.

Key words: role based programming, context support, java role model

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Gerçek dünya varlıklarının ve davranışlarının bilgisayar dünyasında modellemesini gerçekleştirebilmek adına sunulan çözüm önerilerinden bir tanesi rol tabanlı programlamadır.

Rol tabanlı programlama fikrinin temeli nesne yönelimli programlamadaki modellemenin kısıtlarını aşma amacıdır. Çünkü nesne yönelimli programlamada (OOP) modellenen varlıkların özellikleri ve yetenekleri sabittir. Yani modelleme gerçekleştirildikten sonra ilgili nesne her zaman aynı özelliklere sahip olarak kalacaktır. Halbuki gerçek dünya varlıkları dinamik olarak farklı durumdalarda farklı rollere bürünebilir, farklı özellikler ve yetenekler kazanabilir.

Örneğin bir kişi, okul bağlamında öğrenci ya da öğretmen rollerine bürünüp bunlara göre farklı davranışlar sergiliyor olmasına rağmen, aynı kişi hastane bağlamına girdiğinde tamamen farklı roller olan hasta ya da doktor rollerine bürünüp apayrı davranışlar sergileyebilir.

2000'lerin başından bu yana, rol tabanlı programlamalar üzerinde çeşitli çalışmalar gerçekleştirilmiştir. Bunlar öncelikle rol tabanlı programlamanın genel konseptlerinin belirlenmesi ile başlamıştır. Bu aşamada Steimann tarafından genel prensipler ortaya atılmıştır. [10] Öte yandan ilerleyen dönemlerde bu prensipleri gerçekleyen Java Rol Paketi [8], Jawiro [1] ve OT/J [11] tarafından gerçeklenmeye başlanmıştır. Aynı zaman dilimi içerisinde standart nesneye dayalı programlama çerçevesi içerisinde de benzer sorunları adreslemek adına state ve strateji gibi tasarım kalıpları [7] ortaya atılmıştır.

1.2 Tezin Amacı

Rol tabanlı programlama için çeşitli gerçekleştirimler yapılmıştır. Bunların hemen hepsi bir OOP dilini temel alarak, ilgili dile rol tabanlı programlama desteği eklemeyi hedeflemiştir. Bu gerçekleştirimlerin büyük kısmı ise ilgili dilin yeteneklerini genişletirken yeni bir dil üretmeye doğru yönlenmiştir. Halihazırda yapılmış bazı rol tabanlı programlama gerçekleştirimleri ise, kullanmış oldukları OOP dili üzerinde herhangi bir değişiklik yapmadan kullanıcıya rol desteğini kısıtlı olarak sunabilmiştir.

Bu tez kapsamında geleneksel OOP yaklaşımını bozmadan kullanıcılara rol desteği sağlayan bir gerçekleştirim olan JAWIRO uygulaması [1] ile java programlama diline yaptığı eklemelerle özelleştirilmiş bir programlama dili sunan Object Teams / Java (OT/J) uygulaması [2] incelenmiş, OT/J uygulaması tarafından sunulan bağlam desteğinin JAWIRO gibi sadece OOP temelli bir uygulama üzerinden sağlanabileceği bir çalışma sunulmuştur.

1.3 Hipotez

Bu tez çalışması kapsamında literatür özetinde bahsedilmiş olan çeşitli gerçekleştirimlerin getirdiği avantajlar ve bunlarla beraber programcıya getirdiği ek gereksinimler ile kısıtlar incelenmiş, bunlara alternatif olarak programcıya daha az ek gereksinim ile daha fazla esneklik sunabilecek, özellikle sadece Java programlama dili üzerine kurulu bir anaçatı sunulmuştur.

Tez kapsamında sunulan JCORE anaçatısı, programcıların standart java uygulamalarını rol tabanlı programlama konseptine uyarlayabilecekleri aktör, rol ve bağlam temelleri üzerine kurulmuş bir java kütüphanesidir.

BÖLÜM 2

ROL, BAĞLAM VE GERÇEKLEŞTİRİMLER

Bugüne kadar gerçekleştirilen çalışmaların büyük çoğunluğu modellemede ve gerçekleştirimlerde rol kavramının kullanımının getirileri üzerine incelemeler gerçekleştirmiştir. Bunlara göre roller nesnelerin zaman içerisinde değişmesi ve gelişmesi için uygun altyapıyı sağlar. Roller, bir nesnenin farklı açılardan farklı arayüzlerini eş zamanlı ya da ayrık olarak gösterebilmesini sağlayabilecek modellemeye imkan tanır. Bu modellemeyi yaparken de içinde bulunulan bağlama göre farklı arayüzler modellenerek ilgili kısımları bir araya toplayıp, ilgisizleri birbirinden ayırma imkanı (Seperation of Concerns, SoC) sunar. [3]

2.1 Dinamik ve Statik Modelleme

Alışılagelmiş OOP yaklaşımlarında sınıflarla nesneler arasında statik yapı kullanılır. Yani nesneler modellenirken çeşitli özellikler ve metotlar ile tanımlanır. Nesneler arasındaki ilişki de yine statik olarak tanımlanır ve kalıtım hiyerarşisi tanımlanır. Ancak programlama esnasında hiyerarşi ve model bir kez tanımlanır ve çalışma zamanında (runtime) bu model değişmeden korunur. [4] Statik tabiri ile kastedilen de budur.

Rol ve bağlam kavramları kullanılarak yapılan dinamik modellemelerde, nesneler farklı bağlamlarda farklı rollere sahip olarak farklı görünümler (view'ler) kazanabilir. [3] Böylelikle runtime'da aynı nesnenin farklı view'ler üzerinden diğer nesnelerle etkileşime girmesi mümkün olur. Bunu sağlayabilmek adına nesnelerin bir runtime hiyerarşisi tanımlanır.

Diğer bir deyişle, dinamik modelleme, runime'da nesnelerin hangi bağlamda hangi hangi view'lere sahip olacağının tanımlandığı runtime hiyerarşisinin oluşturulabilmesi imkanını sağlar.

2.2 Bağlam Kavramı

Her bir bağlam bir ya da birden çok kullanım durumunu (use case) ve bunu tanımlayan kuralları temsil eder. [5] Bunu temsil edebilmek adına, hangi nesnelerin ilgili bağlamda hangi davranış özelliklerine sahip olabileceğini tanımlar. Diğer bir deyişle, runtime'da bir nesne ilgili bağlam içerisinde sahip olabileceği rolleri tanımlar.

2.3 Rol Kavramı

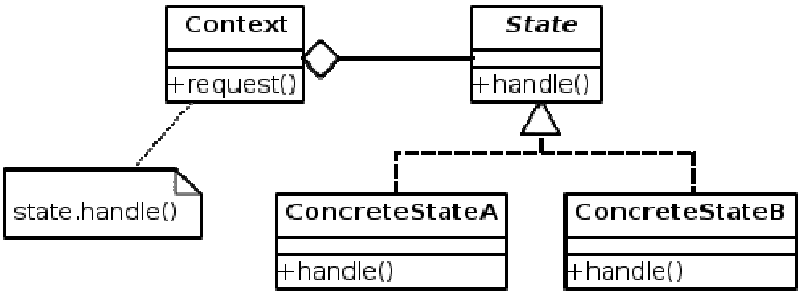
Rol kavramı nesnelerle bağlamların kesişimi olarak tanımlanabilir. [6] Bir bağlama dahil olan bir nesneye, bağlam üzerinde tanımlanan kullanım durumlarına uygun olarak atanan roller ile ilgili nesnenin runtime'da nasıl bir view sunacağı belirlenmiş olur.

2.4 Diğer sunulan çalışmalar

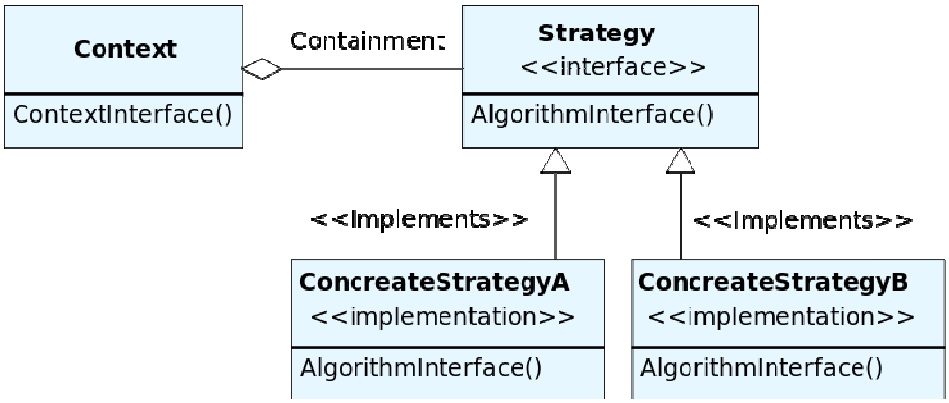
Bu kısımda bugüne kadar nesnelerin dinamik modellenmesi ve rol modelleri üzerine gerçekleştirilen çeşitli yaklaşımlardan bazıları sunulmuştur.

2.4.1 OOP Tasarım Kalıpları

Nesneye yönelimli programlama yapan kişilerin önemli bir kısmı tarafından benimsenen ve kullanımı tavsiye edilen [7] OOP kalıplarından bazıları runtime'daki dinamik yapının modellemesini amaçlamaktadır. Sıklıkla kullanılan tasarım kalıplarından ikisi state (durum) ve strategy (strateji) tasarım kalıplarıdır. Her iki tasarım kalıbı da bir nesnenin içerisinde bulunduğu duruma (state) göre farklı işlemleri gerçekleştirme şansı sunarlar.



Şekil 2. 1 State Tasarım Kalıbı



Şekil 2. 2 Strategy Tasarım Kalıbı

Şekil 2. 1 ve Şekil 2. 2'de gösterildiği üzere her iki tasarım kalıbında da runtime'da farklı özellik gösterimi için standart bir arayüzü (interface) gerçekleyen (implement eden) nesneler kullanılmıştır.

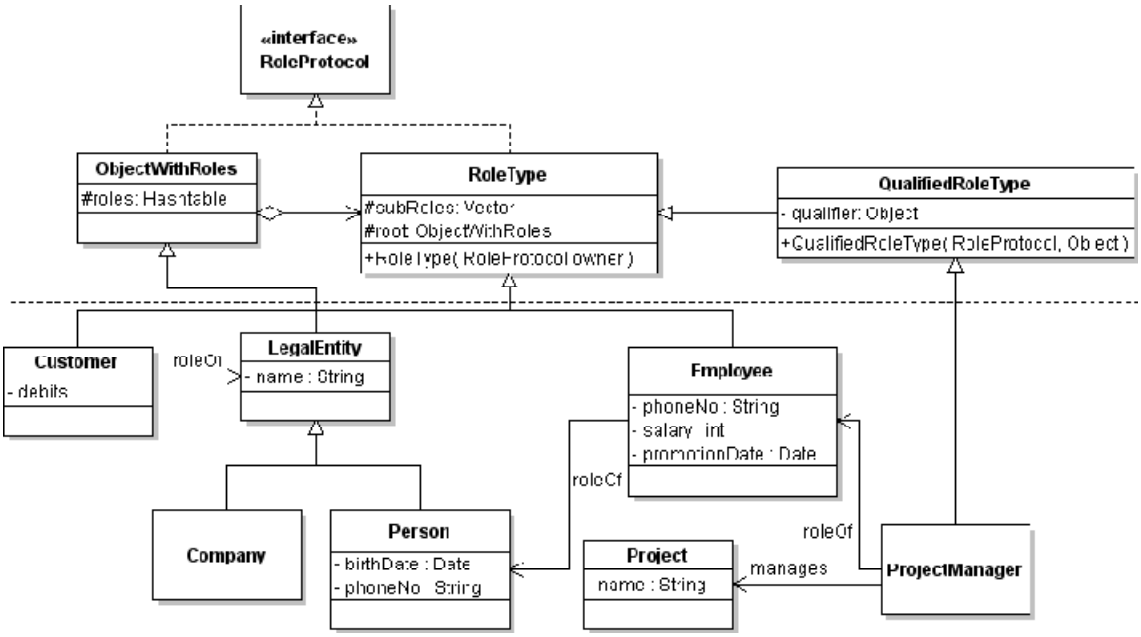
Bu tasarım kalıpları bağlam ve rol kavramlarını kullanmamakla birlikte, runtime'da aynı nesnenin aynı metotlarının farklı çıktılar üretmesini sağlayabilmektedir. Ancak bu durum runtime'da değişebilen yeteneklerin tüm farklı strategy ve state'ler için aynı olması zorunluluğunu getirmektedir.

2.4.2 Java Rol Paketi

Java rol paketi [8], rol tabanlı programlamanın Java programlama dilindeki örneklerinden birisidir. Bu programlama dilinde nesnelere roller eklenebilmekte ve nesnelerin runtime'da sahip olduğu rol bilgilerine erişilerek bu bilgiler kullanılabilmektedir.

Şekil 2. 3'te [1] gösterildiği üzere nesneler çeşitli rollere sahip olabilmekte ve roller arasında da bir kalıtım ilişkisi kurulabilmektedir.

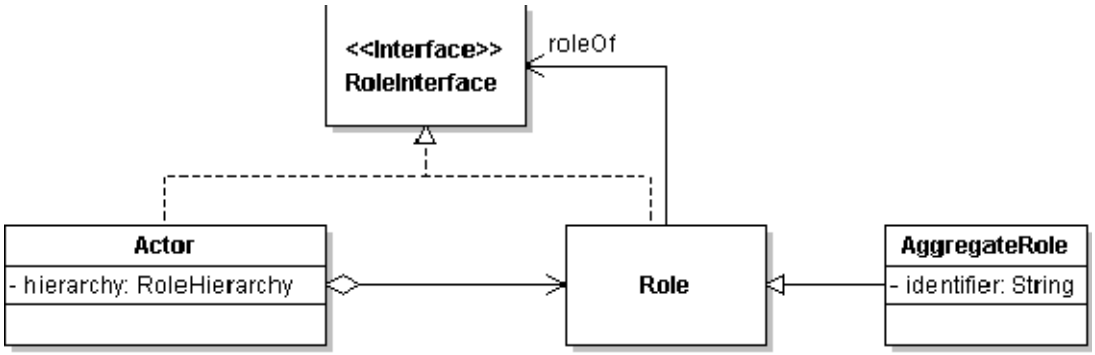
Rol kavramı çeşitli özellikleri ile sunulabilmiş olmasına rağmen, context kavramı bu çözümün içerisinde yer almamaktadır. Ayrıca rollerin özelliklerinin aktiflenebilmesi için kullanıcının ilgili nesnede rol bilgisinin olup olmadığını kontrol ederek talep edilen rol ilgili nesne ile ilişkilendirilmiş ise rol nesnesine yeni bir nesne referansı ile erişimin sağlandığı bir gerçekleştirimdir.



Şekil 2. 3 Java rol paketi ve rol hiyerarşi modeli [8]

2.4.3 JAWIRO (Extending Java With Roles)

2004 yılında Selçuk tarafından sunulan JAWIRO [1], Java rol paketindeki özelliklerin geliştirilmesi ve yeni özelliklerin eklenmesi ile oluşturulmuştur. Şekil 2. 4'te gösterildiği üzere, temel yapı Java Rol Paketi ile aynı olmakla birlikte gerçekleştirimdeki farklılıklar ile ek özellikler sağlanmıştır.



Şekil 2. 4 JAWIRO temel yapısı

Temel olarak Java Rol Paketindeki hiyerarşi korunmakla birlikte aşağıdaki özellikler de eklenmiştir: [1]

- Uygun olmayan rol kazanımları engellenmesi
- Oluşturulan rol hiyerarşisinin kaydedilip tekrar yüklenebilmesi
- Rol kazanımlarının askıya alınabilmesı, gerektiğinde geri aktiflenebilmesi
- Rollerin sahip olduğu rolleri kaybetmeden başka bir aktöre aktarma imkanı
- Nesne seviyesinde çoklu kalıtıma izin verilmesi
- Rol hiyerarşisindeki bir metota ya da değişkene isim ile erişim sağlanması
- Rol hiyerarşisi üzerinde baskın olanların işaretlenmesi ile hangi kuralların hangilerini ezeceği belirlenebilmesi imkanı

JAWIRO yaklaşımı ile sadece Java programlama dili kullanılarak rol modellemeleri yapılabilmektedir. Ancak rol modellemesi yapılmasına rağmen, bağlam kavramı sunulamamaktadır. Dolayısıyla, aktörlerin rolleri sahip olması, rol geçişleri gibi süreçler tamamen kullanıcı tarafından gerçekleştirilen manuel şekilde yönetilmektedir.

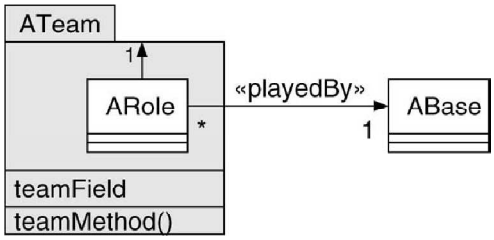
2.4.4 Object Teams / Java (OT/J)

Object Teams/Java (OT/J) programlama dili 2000 yılında Steimann tarafından tanımlanan [10] rol modellerinde yer alması gereken özelliklerin incelendiği kategorileri temel alarak 2000'li yılların başlarında akademik çalışmalar halinde sunulmaya başlanmıştır. [11]

Sunulan programlama dili Java tabanlı br programalama dili olmakla birlikte üzerine yapılan yeni eklentilerle, farklı anahtar kelimeler kullanılarak yeni yetenekler dile eklenmiştir. Dolayısıyla kendine özgü bir senktaks ile geliştirilen programlama dili, yine kendine özgü bir derleyici üzerinden derlenerek çalıştırılabilir bir hale getirilebilmektedir.

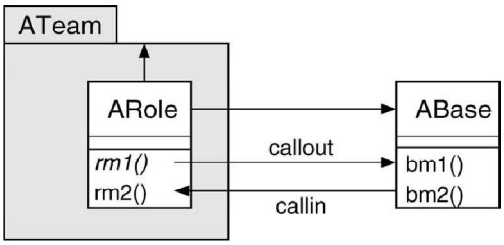
Rollerin ve bağlamların temel alındığı programlama dilinin temelinde "team" kavramı ile bağlamlar tanımlanmış, her bir rol de bağlam içerisine yerleştirilerek bir bağlama dahil olan nesnelerin uygun rolleri kazanmaları sağlanmıştır. Şekil 2. 5'te de gösterildiği

üzere “rol”ler “team”ler içerisinde tanımlanmaktadır. Herhangi başka bir nesnenin bu rolü oynayabilmesi için de ilgili “team”e dahil olması gerekmektedir.



Şekil 2. 5 OT/J üzerinde team ve rol kavramları

Öte yandan OT/J tarafından sunulan en önemli özelliklerden birisi de “callin” ve “callout” method binding’leridir. Şekil 2. 6’da görülebileceği üzere, roller üzerinden bu rolleri oynayan nesnelerin metotlarına, benzer şekilde rolleri oynayan nesneler üzerinden rollerin metotlarına erişim mümkün olmaktadır.



Şekil 2. 6 OT/J ile callin ve callout method binding’leri

Method binding özelliği runtime’da metotların değiştirilebilmesini, bir nesnenin bir bağlama dahil olduktan sonra halihazıda kullanmakta olduğu metotların bile değişebilmesini sağlayabilmektedir.

Bağlam tanımının sayesinde de da hangi metotun hangi metota ne zaman bağlanması (bind olması) gerektiği de bağlama dahil olma ya da bağlamdan çıkma durumlarında, kullanıcılar tarafından tanımlanmış olan kurallar işletilerek otomatik olarak belirlenebilmektedir.

Sunulan programlama dili, Steimann tarafından sunulan 15 adet rol modelleme prensiplerinin 13’ünü tam olarak 1 tanesini de kısmen karşılayabilecek güçlü bir rol modelleme aracı olarak değerlendilirmektedir. [9]

2012 Java One konferansında Herrmann S. Tarafından da belirtildiği üzere; OT/J her ne kadar Java programlama diline JVM üzerinde çalışan diğer programlama dillerinden

daha yakın da olsa, nesneye yönelimli programlamaya yeni boyutlar ve bakış açıları getirebilmek adına OT/J için özelleştirilmiş bir IDE kullanımı zorunlu kılınmıştır. [12]

BÖLÜM 3

BAĞLAM DESTEĞİ İLE JAVA ÜZERİNDEN ROL MODELLEME ÖNERİSİ

Bir önceki bölümde literatürde yer alan rol modelleme yaklaşımlarından önde gelenleri listelenmiştir. Bu bölümde ise tez çalışması kapsamında sunulan öneri (JCORE – Java COntextual Role Enablement) açıklanmıştır. İlk kısımda projenin başarım kriterleri açıklanmıştır. İkinci kısımda ise sunulan çözümün tanımı yapılarak JCORE anaçatısının sunmuş olduğu yetenekler anlatılmıştır. Bir sonraki bölümde ise JCORE mimarisi sınıf yapısı ve kullandığı kütüphaneler detaylı olarak incelenmiştir. Takip eden kısımda ise örneklerle süreç açıklanmıştır. Son kısımda JCORE çözümünün Steimann prensiplerine [10] uyumu incelenmiştir.

3.1 Başarım Kriterleri

Bu projenin motivasyonu halihazırdaki yaklaşımlardan sadece Java tabanlı olanlarında bağlam desteğinin olmaması, bağlam desteği olan yaklaşımların da Java programlama dili dışına çıkmış olmalarıdır.

Bu proje kapsamında sunulan rol modellemelerine yaklaşımında, aşağıdaki özelliklerin yer alması öncelik olarak belirlenmiştir:

- Sunulacak altyapı kullanılarak kod geliştirirken, geliştirilen kodu derlerken ya da çalıştırırken standart Java uygulamalarından farklı bir ortama ihtiyaç duyulmaması
- Bağlam ve rol arasındaki ilişkinin, OT/J geliştiricileri tarafından tanımlandığı şekilde (nesnelerle bağlamların kesişimi şeklinde) sunulması [6]

- Rol geçişlerinin dinamik olarak gerçekleştirilmesi
- Steimann tarafından sunulan prensiplere olabildiğince yüksek uyum gösterilmesi

3.2 Çözümün Tanımı

Sunulan çalışmada rol modellemesi için JCORE anaçatısı oluşturulmuştur. Bu anaçatı aktör, bağlam ve rol kavramları üzerine kurulmuştur. JCORE çalışma zamanında aktörlerin dahil olacakları ve çıkacakları bağlamlara göre rollerin aktörler tarafından oynanabilmesini sağlar.

3.2.1 JCORE Tasarım Kararları

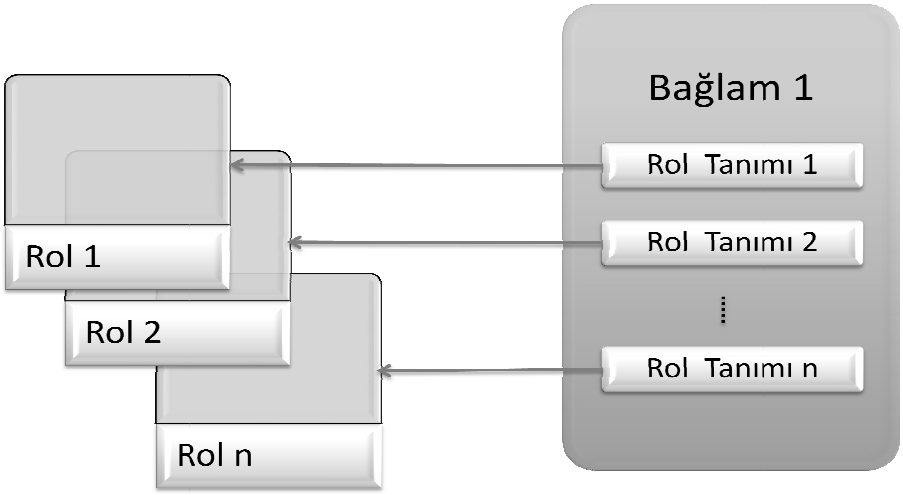
JCORE tasarımı yapılırken varolan çalışmalar incelenerek aşağıdaki temel tasarım kararları alınmıştır:

- Roller ancak aktörler tarafından oynanabilir.
- Her bir rolün aktiflenebilmesi, ilgili aktörün rolün tanımlandığı bağlam içerisine girmesi ile gerçeklenir.
- Rollerin aktiflenmesi programcı tarafından harici olarak gerçekleştirilemez, sadece bağlama dahil olma esnasında anaçatı tarafından otomatik olarak gerçekleştirilebilir.
- Hangi bağlamlarda hangi rollerin yer alacağı, bu bağlama giren hangi aktörlerin nasıl bir role sahip olacakları gibi ayarların programcı tarafından belirlenebileceği bir altyapı sunulur.
- Aktörler bağlama dahil olduklarında hiçbir rol almayabildikleri gibi bir ya da birden çok da rol alabilirler.
- Aktörler eş zamanlı olarak birden çok bağlamda yer alabilirler.
- Aynı rol tanımlamaları farklı bağlamlarda yer alabilir.

3.2.2 Bağlama Dahil Olma ve Bağlamdan Çıkış

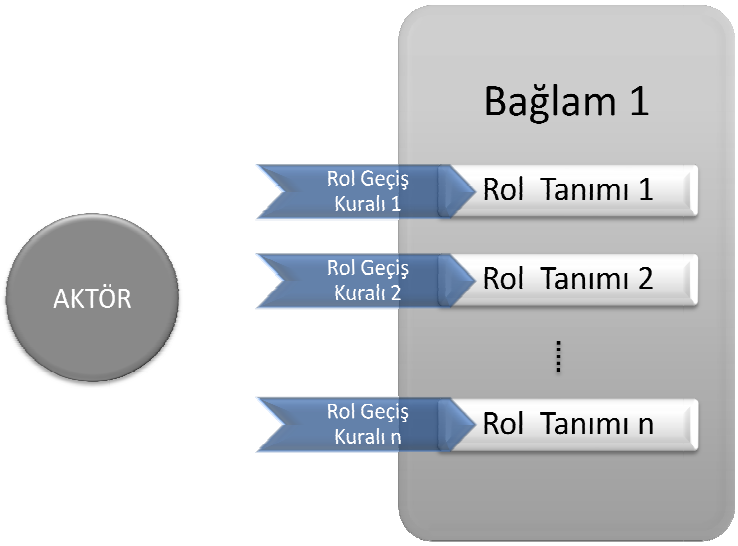
Buna göre JCORE rol sınıflarının ayrıştırıcı özelliği ortak bir interface'i implement ediyor olmalarıdır. Dolayısıyla rol sınıfları bir sınıfın sahip olabildiği tüm yetenekleri içerisinde barındırabilir. Rollerin alanları ve metotları bulunur. Kendi içlerinde kalıtımla türetilebilir.

Bağlam tanımı yapabilmek için ise JCORE içerisinde abstract olarak tanımlanmış olan bir sınıftan türeyen bağlam sınıfları yaratılır. Bağlam sınıflarının da yine kendi alanları ve metotları bulunur.



Şekil 3. 1 JCORE rol-bağlam ilişkisi

Aktörler JCORE bağlamlarına dahil olduklarında, Şekil 3. 2'de gösterildiği üzere, her bir rol tanımı için oluşturulmuş olan rol geçiş kuralları çalıştırılır. Dolayısıyla her bir aktör her bir bağlam içerisinde sadece uygun olan rolü/rolleri edinebilir.

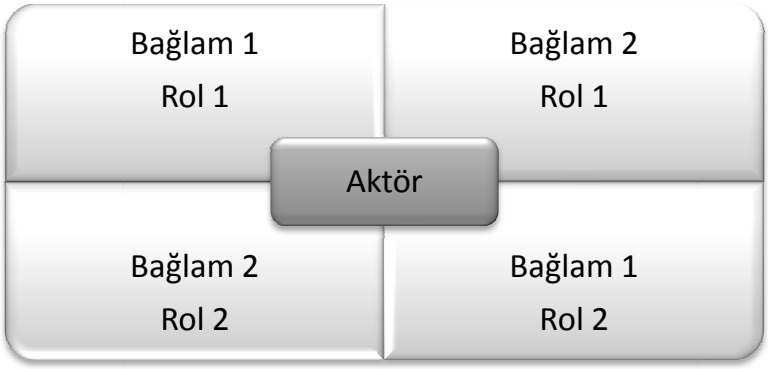


Şekil 3. 2 JCORE'da aktör nesnesinin bağlama dahil olma süreci

Rol geçiş kuralları, programcı tarafından, rollerle birlikte tanımlanan ortak bir interface'i implement eden sınıflardır. Bu sınıfların spesifik bir metoduna yapılabilen bir çağrı ile spesifik aktör'ün ilgili role sahip olup olamayacağı bilgisi edinilir. Şekil 3. 25'te sınıf hiyerarşisi içerisinde rol geçiş kurallarının yeri görülebilir.

Bir aktör eş zamanlı olarak birden çok bağlama dahil olabilir, dahil olduğu tüm bağlamlardan farklı ya da aynı rollerin farklı örneklerini (instance) edinebilir. Dolayısıyla herhangi bir çalışma zamanında bir aktörün birden çok aktif rolü bulunabilir. Şekil 3. 3'de gösterildiği üzere, herhangi bir çalışma anında bir aktörün birden çok aktif rolünden, anlık olarak uygun olan role göre çalışması sağlanabilir.

Aktörler bağlamlardan çıktıklarında ise, ilgili bağlama dahil olma esnasında atanmış olan rollerden de çıkmış olurlar. Bu sayede aktörlerin davranışları bağlam içinde ve dışında farklı olabilmektedir.



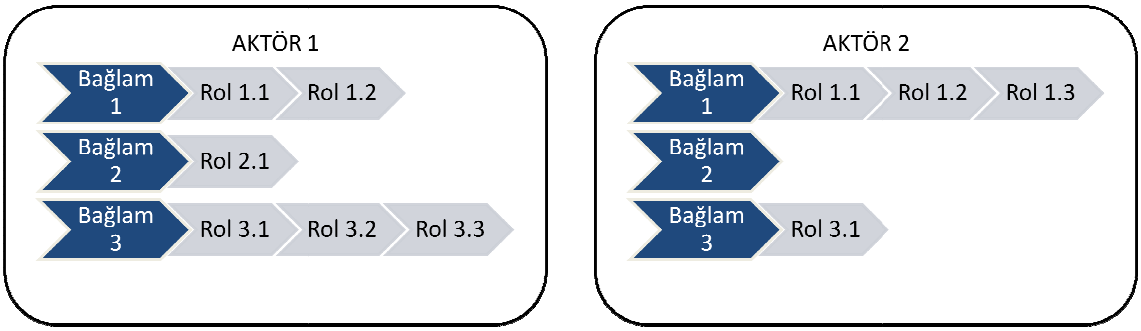
Şekil 3. 3 Runtime'da aktör görünümü

3.2.3 Aktörlerde Role Sahip Olma

Bir aktör herhangi bir anda bir ya da birden çok bağlamda yer alabilir. Yer aldığı her bir bağlamda bir ya da biden çok role sahip olmuş olabilir. Bir aktörün bir role sahip olması ile ilgili aktör üzerinde, rolde ve bağlamda tanımlanmış olan aşağıdaki özellikler aktiflenir:

- Aktör sahip olduğu rollerin tanımlamış olduğu yeni metotları çalıştırabilme özelliğini edinir.
- Aktör üzerinde tanımlı olan metotlardan, kullanıcı tarafından belirletilen metotlar kullanıcının belirttiği şekilde rol ya da bağlam metotları ile değiştirilir, öncesinde ya da sonrasında çalıştırılabilir.

Şekil 3. 4'te gösterildiği üzere aktörler dahil olmuş oldukları bağlamlar ve edindikleri rollerle ilgili kayıtları kendi örnekleri (instance'ları) üzerinde tutarlar. Böylelikle gerekli her bir aktör anlık olarak hangi bağlamlara dahil olduğu bilgisine ve hangi bağlam üzerinden hangi rolü edindiği bilgisine sahiptir. Bir aktörün tir rol örneği ise aynı anda tek bir bağlama dahil edilebilir.



Şekil 3. 4 Aynı bağlamlara dahil olmuş iki farklı aktör üzerinde rol kayıtları

Roller herhangi bir java sınıfından farklı olarak sadece ortak bir interface'i implement ettiği için, rollerin kendi sınıf değişkenleri ve metotları vardır. Benzer şekilde bağlamlar da ortak bir atadan türemiş java sınıfları oldukları için kendi sınıf değişkenleri ve kendi metotları bulunmaktadır. Aktörler belli bir bağlama dahil olduklarında edindikleri rollere göre aktör ya da bağlam metotları ve sınıf değişkenleri ile çeşitli etkileşimlere girebilmektedirler.

3.2.3.1 Aktör Üzerinden Değişken Erişimi

Aktörler dahil oldukları tüm bağlam ve edindikleri tüm rol instance'ları üzerindeki public değişkenlere erişebilmektedirler. Erişimi sağlamak üzere aktör üzerindeki tüm bağlamların ve rollerin isimleri yer almaktadır. Dolayısıyla her bir aktör sınıfı üzerinden dahil olunmuş olan bağlamlar ve roller üzerindeki "public" access modifer'ı ile tanımlanmış olan değişkenlere aşağıdaki örneklerde belirtildiği üzere erişebilir.

```
getContextVariable("contextId", "name");
```

Şekil 3. 5 Aktör üzerinden bağlam değişkenine erişim

```
getRoleVariable("contextId", "roleId", "fieldName");
```

Şekil 3. 6 Aktör üzerinden rol değişkenine erişim

Bağlama dahil olma ve rol edinme işlemleri runtime'da gerçekleştirildiği için her bir aktör halihazırda dahil olduğu bağlamlar ve rol bilgilerini de yine runtime'da edinebilmek üzere aşağıdaki metotları kullanabilmektedir.

```
getActiveContextIdList();
```

Şekil 3. 7 Aktör üzerinden dahil olunmuş olan bağlam listesi alma

```
getActiveRoleList();
```

Şekil 3. 8 Aktör üzerinden dahil olunmuş olan bir bağlamdan edinilen rol listesini alma

3.2.3.2 Rol Üzerinden Değişken Erişimi

Her bir rol instance'ı ancak bir bağlam içerisinde yer alacağından ve sadece bir adet aktör ile eşlenmiş olacağı için, her bir rol üzerinden ilgili bağlamdaki ya da aktördeki "public" değişkenlere erişmek mümkündür. Ancak rollerin tek ortak özelliği ortak bir interface'i implement etmiş olmaları olduğundan ötürü bu işlemi bağlam sınıfları üzerinden gerçekleştirebilirler.

JCORE için yapılmış olan konfigurasyon uygulama açıldığında heap belleğe aktarılır. Runtime'da oluşturulan bağlamlar ve bu bağlamlardaki aktörler ile bu aktörlerin kazanmış oldukları roller de yine heap bellek üzerinde tutulur ve bağlamlar tarafından erişilebilir durumdadır. Dolayısıyla her rol kendi instance'ını vererek ilgili rolü oynayan

aktör nesnesine ve bu nesnenin "public" access modifier'ı ile işaretlenmiş değişkenlerine erişebilir.

```
JcoreContext.getActor(this);
```

Şekil 3. 9 Roller üzerinden aktör erişimi

```
JcoreContext.getActorVariable(this,"fieldName");
```

Şekil 3. 10 Roller üzerinden aktör değişkenlerine erişim

3.2.3.3 Aktör Üzerinden Metot Çağrısı

Aktör nesneleri üzerinden bağlam ve rol metotları çağrıları yapılabilmektedir. 3.2.3.1 bölümünde belirtildiği üzere, aktörler üzerinde dahil olmuş oldukları bağlamlar ve edindikleri rollerler ilgili kayıtların referansları yer almaktadır. Bu sayede aktörler üzerinden içerisinde bulundukları bağlamlardaki ve edinmiş oldukları rollerdeki "public" access modifier'ı ile işaretlenmiş olan metotların çağrıları yapılabilmektedir.

Metot çağrıları Java 1.5 ile sunulan reflection özellikleri ile gerçekleştirilmekte olup, gerçekleştirilen metot çağrıları için uygun parametreler kullanıcı tarafından bir liste halinde sunulmalı, dönülen nesneler ise uygun sınıf tiplerine cast edilmelidir.

Aktör üzerinden bağlam ve rol metot çağrıları aşağıda örneklenmiştir.

```
executeContextMethod("contextId", "methodName", new Object[0]);
```

Şekil 3. 11 Aktör üzerinden bağlam metot çağrımı

```
executeRoleMethod("contextId", "roleId", "methodName", new Object[0]);
```

Şekil 3. 12 Aktör üzerinden rol metot çağrımı

3.2.3.4 Rol Üzerinden Metot Çağrısı

Rol metotları içerisinden aktör metotlarına erişim rol tabanlı programlamadaki en temel özelliklerden birisi olarak sunulmaktadır. Roller interface implement eden sınıflardan oluştuğu için, roller üzerinden metot çağrıları yine değişken erişimlerinde olduğu gibi bağlamlar üzerinden gerçekleştirilebilmektedir. Bölüm 3.2.3.2'de belirtildiği üzere, metot çağrıları gerçekleştirilirken, tüm bağlamlar tarafından

erişilebilen runtime konfigurasyonu kullanılarak metot çağrısı gerçekleştirilebilir. Gerçekleştirilen metot çağrıları yine Java 1.5 ile sunulan reflection yetenekleri kullanılarak gerçekleştirilir. Çağrılan metotlara parametreler liste halinde geçilir, geri dönüş parametresi de uygun bir sınıfa programcı tarafından cast edilerek kullanılır.

```
JcoreContext.executeActorMethod(this, "methodName", new Object[0]);
```

Şekil 3. 13 Rol üzerinden aktör metot çağrısı

```
JcoreContext. executeContextMethod(this, "methodName", new Object[0]);
```

Şekil 3. 14 Rol üzerinden bağlam metot çağrısı

3.2.4 Dinamik Metot Modifikasyonu

JCORE ile sunulan en büyük özelliklerden birisi, aktörler belli rollere büründükten sonra programcı tarafından belirlenmiş metotlarının eskisinden farklı davranış göstermesini sağlayabilmektedir. JCORE altyapısında, aktörler bağlamlara dahil olduklarında eğer belirli kurallara uymuşlarsa çeşitli rollere bürünürler. Bu rollerle ilgili kullanıcı tarafından belirlenen bazı özelliklere göre metotların öncesinde ya da sonrasında çalıştırılması istenilen rol ya da bağlam metotları otomatik olarak çalıştırılır. Ya da aktör metotlarından belirlenmiş olanları rol veya bağlam metotları ile değiştirilir. Bu özellikler XML konfigurasyon dosyası ile uyarlanır.

```
<Jcore><Context class="edu.yildiz.sample.Context1">
        <Role class="edu.yildiz.sample.Role1">
            <BindingRule actor="edu.yildiz.sample.Actor1"
rule="edu.yildiz.sample.BindingRule1">
                <ReplaceRole actorMethod="actorMethod1" roleMethod="roleMethod1"/>
                <ReplaceContext actorMethod="actorMethod2"
contextMethod="contextMethod1"/>
                <BeforeRole actorMethod="actorMethod4" roleMethod="roleMethod2"/>
                <BeforeContext actorMethod="actorMethod3"
contextMethod="contextMethod2"/>
                <AfterRole actorMethod="actorMethod6" roleMethod="roleMethod3"/>
                <AfterContext actorMethod="actorMethod5"
contextMethod="contextMethod3"/>
            </BindingRule>
        </Role>
    </Context></Jcore>
```

Şekil 3. 15 Örnek JCORE XML Konfigurasyonu

JCORE XML konfigurasyonu ile uygulama içerisinde yer alan tüm bağlamlar, bu bağlamların altındaki roller ve ilgili rolleri sahiplenebilecek aktörler, sahiplenme kuralları ile birlikte Şekil 3. 15'te belirtildiği şekilde tanımlanır. Ayrıca her bir aktör üzerinde gerçekleştirilebilecek dinamik metot modifikansyonları XML konfigurasyonu üzerinden belirtilir.

3.2.4.1 Context Seviyesi Özellikler

Programcı, kullanmak istediği bağlamların her birini ayrı bir "Context" seviye XML tag'i ile belirtir. Her bir "Context" seviyesi XML elemanı içerisinde tanımlanmış olan rol listesi yer alır.

"Context" seviyesi XML elemanının "class" özelliği bağlam tanımının gerçekleştirdiği "edu.yildiz.jcore.framework.JcoreContext" sınıfını extend eden sınıfın tam adresini içermelidir.

3.2.4.2 Rol Seviyesi Özellikler

Programcı her bir bağlam altında çeşitli rol tanımlamaları gerçekleştirebilir. "Context" seviyesi XML elemanının alt elemanı olarak yer alan "Role" seviyesi XML elemanları birden çok sayıda olabilir ve her bir rol için ilgili role sahip olabilecek aktörler için kurallar listesi yer alır.

"Role" seviyesi XML elemanının "class" özelliği rol tanımının gerçekleştirdiği "edu.yildiz.jcore.framework.JcoreRole" sınıfını implement eden sınıfın tam adresini içermelidir.

3.2.4.3 BindingRule Seviyesi Özellikler

Her bir rol bağlamlar altında tanımlanır, ancak her bir role sahip olabilecek aktör sınıfı ve bu rolü sahiplenebilme kuralları ayrıca belirlenir. Dolayısıyla aynı role her bir aktör sınıfı farklı metot modifikasyonlarına sahip olabilir. "Role" seviyesi XML elemanının alt elemanı olarak yer alan "BindingRule" seviyesi XML elemanları birden çok sayıda olabilir ve her bir BindingRule için ilgili aktör üzerinde yapılması gereken metot modifikasyonlarının listesi yer alır.

“BindingRule” seviyesi XML elemanının “actor” özelliği bu role sahip olabilecek aktör sınıfının tanımının gerçekleştirdiği “edu.yildiz.jcore.framework.JcoreActor” sınıfını extend eden sınıfın tam adresini içermelidir. Bu sınıfın tanımlanmasının temel sebebi, ilgili aktör metotları üzerinde modifikasyon tanımlarının doğru şekilde yapılabilmesinin sağlanmasıdır.

“BindingRule” seviyesi XML elemanının “rule” özelliği bu ilgili aktörün belirtilen role sahip olabilmesi için çalıştırılması gereken kontrol algoritmasının yer aldığı sınıfı belirtir. Bu özellik içerisinde belirtilen değerin “BindingRule” tanımının yapıldığı “edu.yildiz.jcore.framework. JcoreBindingRule” sınıfını extend eden sınıfın tam adresi olması gerekmektedir.

3.2.4.4 ReplaceRole Seviyesi Özellikler

“BindingRule” altında tanımlanmış bir aktör instance’ı, belirtilen bağlama dahil olup, ilgili role sahip olduğu durumda bazı metotlarının yerine rol sınıfı üzerindeki başka bir metodun çalıştırılması istenilebilir. Bu durumda “ReplaceRole” seviyesi XML elemanı ile bunu belirten konfigurasyon “BindingRule” seviyesi XML elemanı altına bir ya da birden çok sayıda eklenebilir.

Bu elemana ait iki adet özellikten “actorMethod” özelliği, aktör üzerinde değiştirilmesi istenilen metodun ismini, “roleMethod” özelliği rol üzerinde yer alan ve aktör üzerindeki metodun yerine çalıştırılması istenilen metodun ismini belirtmelidir.

Değiştirilmesi istenilen metodun parametreleri ile, yerine çalıştırılacak parametre listesinin ya aynı olması ya da yerine çalıştırılması istenilen metodun parametre almaması gerekmektedir. Öte yandan her iki metodun da aynı geri dönüş parametre tipine sahip olması ya da her iki metodun da geriye herhangi bir parametre döndürmemesi gerekmektedir.Ancak bir aktör sınıfının bir bağlam altında bir metodunda “ReplaceRole” seviyesi bir operasyon için konfigurasyon yapılmış ise, başka bir operasyon konfigurasyonu gerçekleştirilemez. Aktör metodu yerine çalıştırılan metodun atabileceği “Exception”lar “RuntimeException”dan türeyen “JcoreTargetInvocationException” şeklinde atılır.

3.2.4.5 ReplaceContext Seviyesi Özellikler

“BindingRule” altında tanımlanmış bir aktör instance’ı, belirtilen bağlama dahil olup, ilgili role sahip olduğu durumda bazı metotlarının yerine bağlam sınıfı üzerindeki başka bir metodun çalıştırılması istenilebilir. Bu durumda “ReplaceContext” seviyesi XML elemanı ile bunu belirten konfigurasyon “BindingRule” seviyesi XML elemanı altına bir ya da birden çok sayıda eklenebilir.

Bu elemana ait iki adet özellikten “actorMethod” özelliği, aktör üzerinde değiştirilmesi istenilen metodun ismini, “contextMethod” özelliği bağlam üzerinde yer alan ve aktör üzerindeki metodun yerine çalıştırılması istenilen metodun ismini belirtmelidir.

Değiştirilmesi istenilen metodun parametreleri ile, yerine çalıştırılacak parametre listesinin ya aynı olması ya da yerine çalıştırılması istenilen metodun parametre almaması gerekmektedir. Öte yandan her iki metodun da aynı geri dönüş parametre tipine sahip olması ya da her iki metodun da geriye herhangi bir parametre döndürmemesi gerekmektedir. Ancak bir aktör sınıfının bir bağlam altında bir metodunda “ReplaceContext” seviyesi bir operasyon için konfigurasyon yapılmış ise, başka bir operasyon konfigurasyonu gerçekleştirilemez.

Aktör metodu yerine çalıştırılan metodun atabileceği “Exception”lar “RuntimeException”dan türeyen “JcoreTargetInvocationException” şeklinde atılır.

3.2.4.6 BeforeRole Seviyesi Özellikler

“BindingRule” altında tanımlanmış bir aktör instance’ı, belirtilen bağlama dahil olup, ilgili role sahip olduğu durumda bazı metotlarının çalıştırılma öncesinde rol sınıfı üzerindeki başka bir metodun çalıştırılması istenilebilir. Bu durumda “BeforeRole” seviyesi XML elemanı ile bunu belirten konfigurasyon “BindingRule” seviyesi XML elemanı altına bir ya da birden çok sayıda eklenebilir.

Bu elemana ait iki adet özellikten “actorMethod” özelliği, aktör üzerindeki ilgili metodun ismini, “roleMethod” özelliği rol üzerinde yer alan ve aktör üzerindeki metodun öncesinde çalıştırılması istenilen metodun ismini belirtmelidir.

Aktör metodu çalıştırılmadan önce çalıştırılan rol metodunun ya hiç parametre almaması, ya da aktör metodu ile aynı parametre tiplerini aynı sıra ile alıyor olması gerekmektedir. Öte yandan rol metodunun geri dönüş değeri dikkate alınmaz.

Aktör metodu öncesinde çalıştırılan metodun atabileceği "Exception"lar yakalanarak program akışını etkilememek üzere "WARNING" seviyesi bir log ile loglanır ve akışa devam edilir.

3.2.4.7 BeforeContext Seviyesi Özellikler

"BindingRule" altında tanımlanmış bir aktör instance'ı, belirtilen bağlama dahil olup, ilgili role sahip olduğu durumda bazı metotlarının çalıştırılma öncesinde bağlam sınıfı üzerindeki başka bir metodun çalıştırılması istenilebilir. Bu durumda "BeforeContext" seviyesi XML elemanı ile bunu belirten konfigurasyon "BindingRule" seviyesi XML elemanı altına bir ya da birden çok sayıda eklenebilir.

Bu elemana ait iki adet özellikten "actorMethod" özelliği, aktör üzerindeki ilgili metodun ismini, "contextMethod" özelliği bağlam üzerinde yer alan ve aktör üzerindeki metodun öncesinde çalıştırılması istenilen metodun ismini belirtmelidir.

Aktör metodu çalıştırılmadan önce çalıştırılan rol metodunun ya hiç parametre almaması, ya da aktör metodu ile aynı parametre tiplerini aynı sıra ile alıyor olması gerekmektedir. Öte yandan bağlam metodunun geri dönüş değeri dikkate alınmaz.

Aktör metodu öncesinde çalıştırılan metodun atabileceği "Exception"lar yakalanarak program akışını etkilememek üzere "WARNING" seviyesi bir log ile loglanır ve akışa devam edilir.

3.2.4.8 AfterRole Seviyesi Özellikler

"BindingRule" altında tanımlanmış bir aktör instance'ı, belirtilen bağlama dahil olup, ilgili role sahip olduğu durumda bazı metotlarının çalıştırılması sonrasında rol sınıfı üzerindeki başka bir metodun çalıştırılması istenilebilir. Bu durumda "AfterRole" seviyesi XML elemanı ile bunu belirten konfigurasyon "BindingRule" seviyesi XML elemanı altına bir ya da birden çok sayıda eklenebilir.

Bu elemana ait iki adet özellikten “actorMethod” özelliği, aktör üzerindeki ilgili metodun ismini, “roleMethod” özelliği rol üzerinde yer alan ve aktör üzerindeki metodun çalıştırılması sonrasında çalıştırılması istenilen metodun ismini belirtmelidir.

Aktör metodu çalıştırıldıktan sonra çalıştırılan rol metodunun ya hiç parametre almaması, ya da aktör metodu ile aynı parametre tiplerini aynı sıra ile alıyor olması gerekmektedir. Öte yandan rol metodunun geri dönüş değeri dikkate alınmaz.

Aktör metodu sonrasında çalıştırılan metodun atabileceği “Exception”lar yakalanarak program akışını etkilememek üzere “WARNING” seviyesi bir log ile loglanır ve akışa devam edilir.

3.2.4.9 AfterContext Seviyesi Özellikler

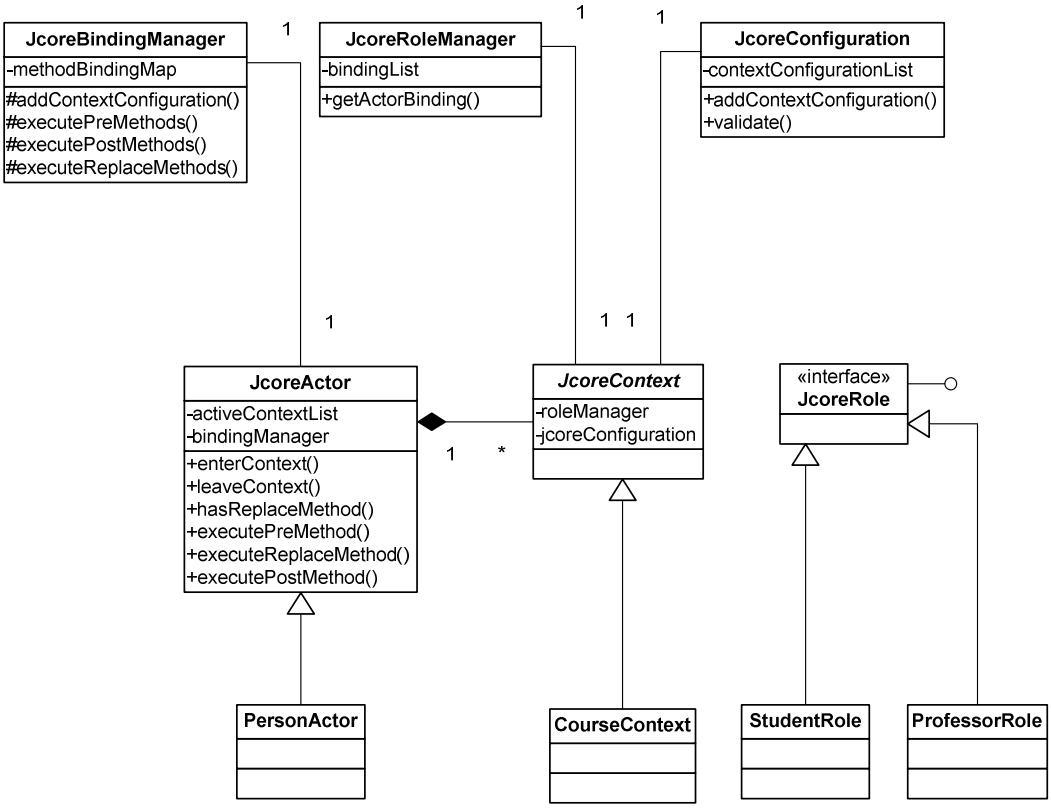
“BindingRule” altında tanımlanmış bir aktör instance’ı, belirtilen bağlama dahil olup, ilgili role sahip olduğu durumda bazı metotlarının çalıştırılması sonrasında bağlam sınıfı üzerindeki başka bir metodun çalıştırılması istenilebilir. Bu durumda “AfterContext” seviyesi XML elemanı ile bunu belirten konfigurasyon “BindingRule” seviyesi XML elemanı altına bir ya da birden çok sayıda eklenebilir.

Bu elemana ait iki adet özellikten “actorMethod” özelliği, aktör üzerindeki ilgili metodun ismini, “contextMethod” özelliği bağlam üzerinde yer alan ve aktör üzerindeki metodun çalıştırılması sonrasında çalıştırılması istenilen metodun ismini belirtmelidir.

Aktör metodu çalıştırıldıktan sonra çalıştırılan rol metodunun ya hiç parametre almaması, ya da aktör metodu ile aynı parametre tiplerini aynı sıra ile alıyor olması gerekmektedir. Öte yandan bağlam metodunun geri dönüş değeri dikkate alınmaz.

Aktör metodu sonrasında çalıştırılan metodun atabileceği “Exception”lar yakalanarak program akışını etkilememek üzere “WARNING” seviyesi bir log ile loglanır ve akışa devam edilir.

3.3 JCORE Mimarisi



Şekil 3. 16 JCORE temel sınıf hiyerarşisi

Şekil 3. 16'da gösterildiği üzere, JCORE mimarisinde temel yapı aktör (JcoreActor), rol (JcoreRole) ve bağlam(JcoreContext) kavramları üzerine kurulmuştur. Bunlardan aktör ve bağlam sınıfları kendi yeteneklerini ortaya çıkarabilecekleri yardımcı sınıflarla (JcoreBindingManager, JcoreRoleManager, JcoreConfiguration) "has-a" ilişkisi ile servis almaktadırlar. Rol sınıfı ise bir interface şeklinde modellenmiş olduğu için yine çeşitli hizmetler bağlam sınıfı üzerinden rol sınıfı kullanıcısı hizmetine sunulmaktadır.

JCORE anaçatısı kullanımı esnasında aşağıdaki 4 temel süreç işletilir:

- XML konfigurasyonunu yükle, doğrula
- Her bir aktör metodu için yapılacak güncellemeleri belirle
- Aktör sınıflarını güncelle ve JVM'e yükle
- Runtime'da uygun yönlendirmeleri gerçekleştirerek rol ve bağlam özelliklerini aktör üzerine yansıt

Bu işlemleri gerçekleştirebilmek üzere kullanıcı tarafından, Şekil 3. 17'de gösterildiği şekilde, uygulamanın başında JCORE ortamı kurulur. Böylelikle yukarıda belirtilen ilk 3 adım anaçatı tarafından gerçekleştirilmiş olur.

```
new JcoreEnvironmentSetup("Jcore.xml");
```

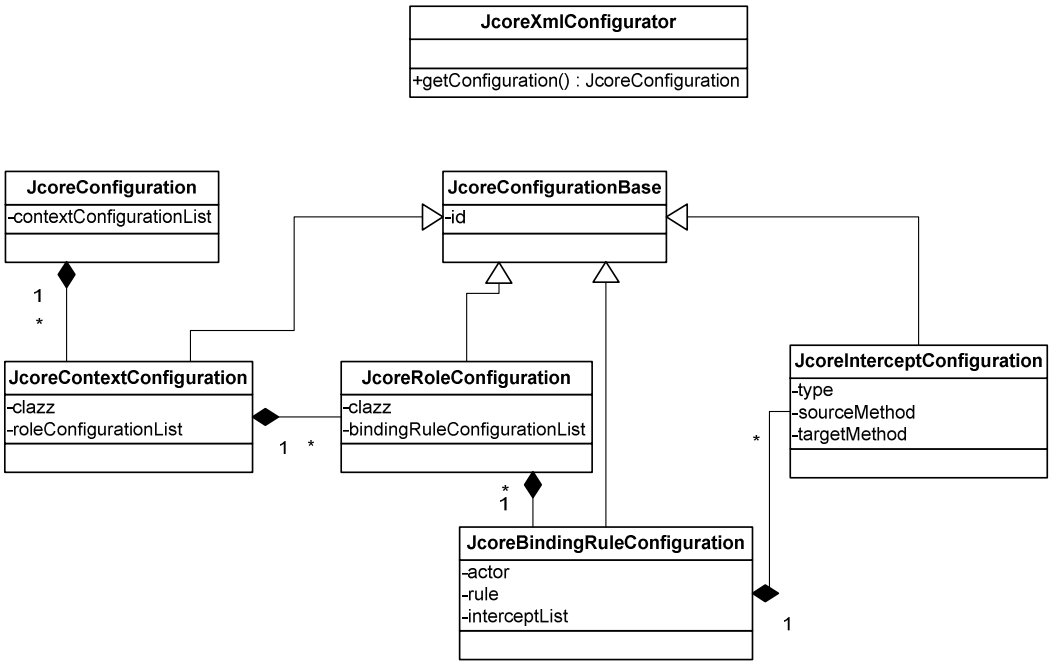
Şekil 3. 17 JCORE ortamının kurulumu

Ortam kurulduktan ve program çalışmaya başlatıldıktan sonra aktörlerin farklı bağlamlara dahil edilmeleri ve/veya çıkmaları durumunda, JCORE anaçatısı tarafından yukarıdaki listede son maddede belirtilen operasyon gerçekleştirilerek bağlam girişlerinde rol atamaları ve aktörler üzerinde dinamik metot modifikasyonları gerçekleştirilir.

3.3.1 XML Konfigurasyon Yükleme ve Doğrulama

XML konfigurasyonun gerçekleştirildiği dosya adresi belirtilerek JCORE ortam kurulumu başlatılır. Başlatılan ortam kurulumunda öncelikle JcoreXmlConfigurator sınıfı classpath içerisinde verilen dosyaya erişmeye çalışır.

Bulunan dosya DOM parser ile parse edilerek JCORE konfigurasyonu XML yapısı korunarak nesnelere aktarılır. Bu adımdaki amaç, programcının XML konfigurasyonundaki modeli olduğu gibi nesnelere aktarmak ve bu nesneler üzerinden doğrulama işlemlerini gerçekleştirebilmektir. Aktarılan nesne hiyerarşisi Şekil 3. 18'de gösterilmiştir.



Şekil 3. 18 JCORE XML Konfigurasyon Sınıf Hiyerarşisi

Buna göre kulanıcı tarafından oluşturulmuş olan JCORE konfigurasyonu JcoreConfiguration sınıfından bir nesne içerisine yerleştirilir. Ancak bu parse ve yerleştirim işlemi esnasında çeşitli doğrulama işlemleri gerçekleştirilir. Gerçekleştirilen doğrulama işlemlerinde aşağıdaki soruların tamamına "EVET" cevabı dönüldüğü durumda doğrulama başarılı sayılmaktadır:

- Verilen bağlam sınıfları JcoreContext sınıfından mı türetilmiştir?
- Verilen rol sınıfları JcoreRole sınıfından mı türetilmiştir?
- Verilen aktör sınıfları JcoreActor sınıfından mı türetilmiştir?
- Verilen "BindingRule" sınıfları JcoreBindingRule sınıfından mı türetilmiştir?
- Verilen "BindingRule" sınıflarının "default constuructor"u var mıdır?
- Dinamik metot modifikasyonu konfigurasyonlarında belirtilen metotlar belirtilen sınıflar içerisinde yer almakta mıdır?

Aktör metotlarında daha sonraki modifikasyon işlemlerinin gerçekleştirilebilmesi için bu doğrulama işlemlerini gerçekleştirirken, ilgili sınıfların JVM'e yüklenmemesi gerekmektedir. Dolayısıyla doğrulama işlemlerini gerçekleştirirken Javassist [13] kütüphanesi kullanılarak ilgili class dosyaları JVM'e yüklenmeden incelenmektedir.

Doğrulama işlemi gerçekleştirildikten sonra oluşturulmuş olan JcoreConfiguration nesnesi, tüm bağlam sınıfları tarafından erişilebilir hale getirilir.

3.3.2 Aktör Metot Değişikliklerinin Belirlenmesi

JCORE sisteminin temel yapısı, aktör metotları üzerinde yapılabilecek değişikliklere göre aynı aktör nesnesinin farklı bağlamlarda farklı şekilde çalışmasının sağlanması üzerine kuruludur. Bu sebeple kullanıcı tarafından yapılmış olan dinamik metot değişiklik konfigurasyonları bir önceki bölümde belirlenen sınıf hiyerarşisi içerisinden işlenerek uygun formata dönüştürülerek bir sonraki adıma hazır hale getirilir.

Bu aşamada hangi aktör sınıflarının hangi metotlarında değişiklik yapılacağı belirlenir. Tüm metotlarda aynı tipte değişiklik gerçekleştirileceği için, kullanıcı tarafından konfigure edilmiş olan dinamik metot değişiklik tipinden bağımsız olarak sadece davranışı değiştirilmek istenen metotların listesi oluşturulur.

3.3.3 Aktör Sınıflarının Güncellenmesi ve JVM'e yüklenmesi

Aktör sınıflarına ait bazı metotlarının davranışlarının dinamik olarak değiştirilebilmesi için ilgili aktör sınıflarına ait byte kodlar güncellenerek JVM'e yüklenmesi gerçekleştirilmektedir.

Sunulması planlanan özellikler bir Aspect Oriented Programming (AOP) tarafından karşılanabilecek özellikler gibi görünse de AOP kullanmanın 2 farklı dezavantajından ötürü tez çalışması kapsamında byte kodların değiştirilerek JVM'e yüklenmesine karar verilmiştir:

- AOP konusundaki en büyük yetenekleri sunan AspectJ gibi AOP uygulamaları Java dışında farklı bir dil ile aspect'lerin programlanmasını gerektirmekte ve farklı bir compiler üzerinden derlemeyi –her ne kadar bu işlemi otomatize hale getirecek farklı çözüm yöntemleri sunulmuş olsa da- koşul olarak kılmaktadır. [14]
- Hem AspectJ hem de alternatif AOP çözümlerinden önde gelenlerinden Spring-AOP üzerinden gerçekleştirilebilecek olan AOP çözümlerinde, AOP uygulanacak olan sınıfların çalışma zamanı öncesinde belirlenmiş ve kodlamanın buna göre

yapılmış olması beklenmektedir, ki bu durumda programcıların kullanacakları tüm aktör sınıfları için uygun aspect konfigurasyonu da yapmaları gerekecektir.

Öte yandan AOP işlemi sonucu yapılan işlem yine AOP altyapısı tarafından class dosyalarının güncellenerek JVM'e yüklenmesi olduğu için, byte kod güncellemesinin, amaca spesifik olarak JCORE içerisinde gerçekleştirilmesinin daha uygun olduğu kararına varılmıştır.

Byte kod güncellemesi yapabilmek adına JVM atyapısı ve classloader'ların çalışma presipleri incelenerek proje kapsamında sektörde geniş kullanım alanı bulan Sun Microsystems tarafından üretilmiş olan HotSpot JVM'i üzerinde geliştirme ve testler gerçekleştirilmiştir.

Derlenmiş Java kodları (class dosyaları) JVM tarafından ilk kullanım esnasında yüklenmektedir. Bu sayede bir projeye eklenmiş olan tüm kütüphane sınıflarının belleğe yüklenmesi yerine sadece kullanılan sınıflar belleğe yüklenmektedir. [14]

Byte kod güncelleme işlemi için Javassist altyapısı kullanılmaktadır. [13] Javassist JVM'e yüklenmemiş olan class dosyalarının JVM'e yüklenmeden önce güncellenmesini sağlamakla birlikte, JVM'e yüklenmiş olan dosyaların güncellenebilmesi için de HotSwapper altyapısını sunmaktadır. [15] Ancak proje kapsamında sadece JVM'e yüklenmemiş class dosyalarının güncellenmesi gerçekleştirilmektedir. Diğer bir deyişle, JCORE altyapısının beklendiği şekilde çalışabilmesi için, program çalıştırılırken ilk gerçekleştirilmesi gereken işlem JCORE altyapısının kurulması olmalıdır. Böylelikle diğer sınıflar JVM'e yüklenmeden class dosyalarını güncelleme ve JVM'e yükleme imkanı tanınmış olur.

Aktör sınıflarında dinamik metot modifikasyonu için, tüm ilgili metotlar aşağıdaki şekilde güncellenir:

- Varolan metodun ismini değiştir
- Orjinal metot imzasına sahip yeni bir metot oluştur
    - Oluşturulan metot öncelikle "BeforeRole" ya da "BeforeContext" ile konfigure edilmiş metotların çağrıları gerçekleştirilir

        - Bu işlem için JcoreActor sınıfında yer alan “executePreMethod” metodu çağrılır

    - Sonrasında “ReplaceRole” ya da “Replacecontext” ile konfigure edilmiş bir metot var ise ilgili metot çalıştırılır, yok ise ismi değiştirilen orjinal metoda çağrı yapılır

        - Bu işlem için JcoreActor sınıfında yer alan “hasReplaceMethod” ile -gerekli ise- “executeReplaceMethod” metotları çağrılır

    - Son olarak da “AfterRole” ya da “AfterContext” ile konfigure edilmiş metotlar var ise, ilgili metotlar çağrılır

        - Bu işlem için JcoreActor sınıfında yer alan “executePostMethod” metodu çağrılır

Yukarıda tanımlanmış olan işlemleri örnekler üzerinden detaylandırılmıştır. Metotlar geri dönüş parametresi olan ve olmayan metotlar olarak ikiye ayrılarak incelenmiştir. Javassist altyapısının sunduğu özellikler kullanılarak iki farklı metot türü için iki farklı metot gövdesi oluşturulmuştur.

```
public void enterRoom() {
    System.out.println("Entering room...");
}
```

Şekil 3. 19 Geri dönüş parametresi olmayan örnek metot

Şekil 3. 19’da geri dönüş parametresi olmayan bir metot görülmektedir. Bu metodun çalışma zamanında farklı davranışlar sergileyebilmesi için ilgili sınıfa ait derlenmiş kod dosyası değiştirilerek Şekil 3. 20’de gösterildiği hale dönüştürülmüştür. Varolan metot, ismi değiştirilerek (sonuna “_jcore” eklenerek) yeni bir metot olarak kaydedilmiştir. Orjinal metodun ise içeriği Javassist kütüphanesinin özellikleri kullanılarak yeniden yazılmıştır.

Yazılan kod içerisinde “`$args`” anahtar kelimesi ilgili metodun almış olduğu tüm parametreleri “Object” tipinden bir Array şeklinde döndürmektedir. “`$$`” anahtar kelimesi ise javanın sunmuş olduğu “Varargs” özelliğinin Javasisst versiyonu olarak

sunulduğundan birbirinden ayrılmış, liste şeklinde olmayan, bir parametre listesi olan metotların çağrısında bu anahtar kelime kullanılabilmektedir.

Öte yandan, yeni metot içerisinde yer alan "executePreMethod", "hasReplaceMethod", "executeReplaceMethod" ve "executePostMethod" gibi metotlar, JcoreActor sınıfında yer alan metotlar olduğundan, bu sınıftan kalıtımla türetilmiş herhangi bir aktör metodu içerisinde kullanılabilmektedir.

```
public void enterRoom_jcore() {
    System.out.println("Entering room...");
}
public void enterRoom() {
    executePreMethod("enterRoom", $args);
    if(hasReplaceMethod("enterRoom")) {
        executeReplaceMethod("enterRoom", $args);
    } else {
        enterRoom_jcore($$);
    }
    executePostMethod("enterRoom", $args);
}
```

Şekil 3. 20 Geri dönüş parametresi olmayan metodun JCORE ile güncellenmiş hali

Geri dönüş parametresi tanımlamış olan metotlar için durum biraz daha farklıdır. Java programlama dilinde geri döndürülen değişkenin metot imzasında belirtilen sınıf tipinden olması zorunluluğu bulunduğundan, Object tipinden bir nesne geri döndürülememektedir. Ancak "AfterRole" ve "AfterContext" tanımları ile, metodun çalıştırılmasından sonra işletilmesi gereken farklı metotlar olabilecektir. Dolayısıyla geri döndürülecek değerin bir değişkende tutulması gerekmektedir.

Şekil 3. 21'de geri dönüş değeri olan bir metot örneği görülmektedir. Bu metot, geri dönüş tipi olarak "String" sınıfını belirtmiş olup, sınıf seviyesinde tanımlanmış olan "String" tipinde bir nesneyi geri döndürmektedir.

```
private String name;
...
public String identify() {
    return name;
}
```

Şekil 3. 21 Geri dönüş parametresi olan örnek metot

Bu metodun body kısmı değiştirilirken, Şekil 3. 22'te gösterildiği üzere, geri döndürülecek olan parametre Object tipinden bir değişken içerisinde tutulmaktadır. İlgili değişkenin içeriği metot içerisinde belirlenip, geri döndürülürken uygun veri tipine cast edilmektedir. Geri dönüş veri tipi Javassist kütüphanesi tarafından "`$r`" anahatar kelimesi ile sağlanmaktadır.

```
public String identify_jcore() {
    return name;
}
public String identify() {
   Object response=null;
   executePreMethod("identify", $args);
   if(hasReplaceMethod("identify")) {
       response=executeReplaceMethod("identify", $args);
   } else {
       response=identify_jcore($$);
   }
   executePostMethod("identify", $args);
   return ($r)response;
}
```

Şekil 3. 22 Geri dönüş parametresi olan metodun JCORE ile güncellenmiş hali

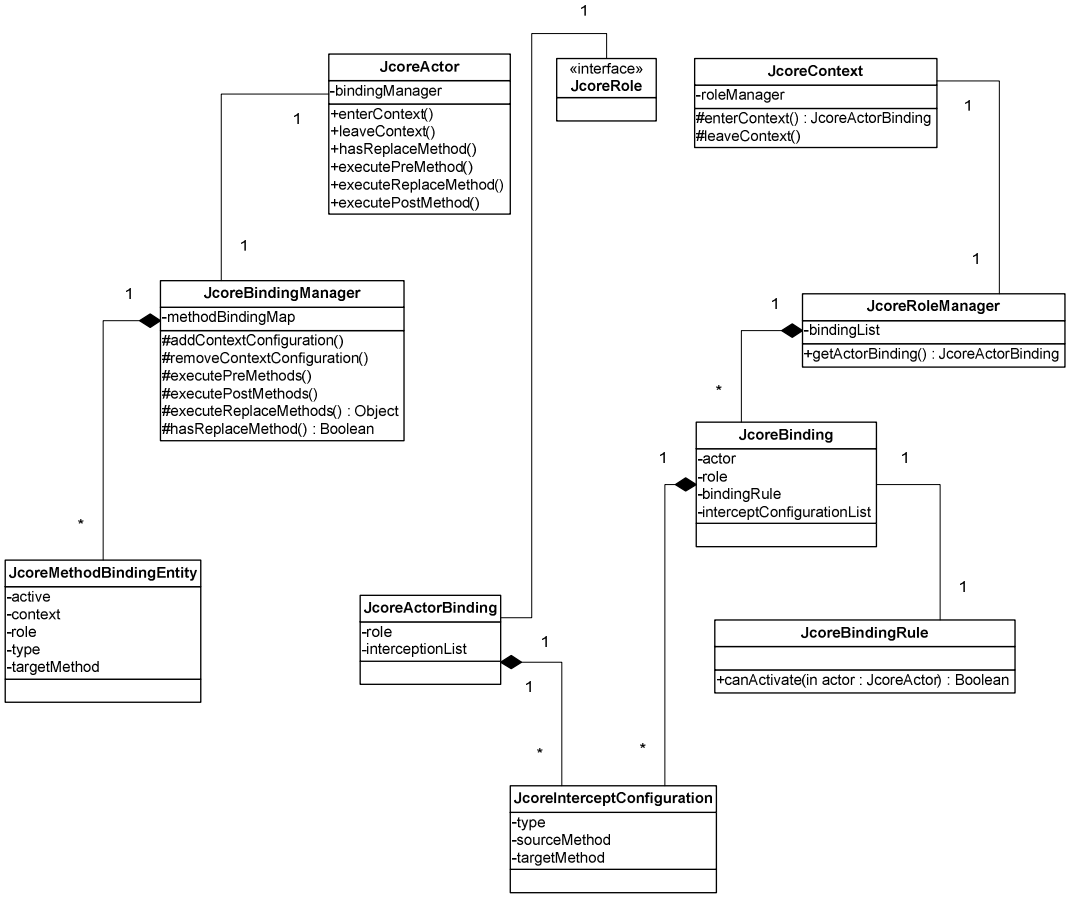
Aktör sınıflarında ilgili metotlar güncellendikten sonra Javassist kütüphanesi tarafından ilgili anahtar kelimeler yorumlanarak ve kod tekrar derlenerek güncellenmiş byte kodlar classloader'lar ile belleğe yüklenir.

Javassist kütüphanesi, yeni oluşturulmuş olan byte kodlarının class dosyalarına yazılmasını da desteklemiş olsa da tez kapsamında, uygulamanın her çalışmasında Javassist üzerinden yeniden derleme yapılması ve sonunda güncel byte kodların yüklenmesini ve çalıştırılmasını ancak class dosyalarının dosya sistemine yazılmaması yöntemi tercih edilmiştir.

3.3.4 Runtime Aktör Metot Yönlendirmeleri

JCORE sistemi üzerinde XML dosyaları ve uygun sınıf tanımlamaları üzerinden gerçekleştirilen konfigurasyonlar statik konfigurasyonlar olup, runtime'daki durumu yansıtmazlar. Sadece aktör-rol-bağlam ilişkilerinin tanımı gerçekleştirilmiş olur. Runtime'da hangi aktörün aktif olarak hangi bağlamda yer aldığı, dolayısıyla hangi

metotların ne gibi bir davranış sergilemesi gerektiği bilgileri runtime konfigurasyonunun yönetildiği mekanizmalar üzerinden gerçekleştirilir. Bu mekanizmalar, JCORE içerisinde bağlam ve aktör sınıfları içerisinde yer alır.



Şekil 3.23 JCORE runtime aktör metot yönlendirmeleri için kullanılan sınıf diyagramı

Şekil 3.23'te belirtildiği üzere runtime metot yönlendirmeleri aktör ve bağlam sınıfları içerisindeki "Manager" sınıfları gerçekleştirmektedir.

3.3.4.1 Bağlam Üzerindeki Runtime Konfigurasyonu

JcoreRoleManager sınıfından oluşturulan "roleManager" değişkeni her bir bağlam içerisinde farklı bir instance şeklinde oluşturulur. Bu nesnenin görevi bağlama giriş yapan her bir nesne için hangi rolleri alması gerektiği bilgisinin sunmasıdır.

Her bir bağlam nesnesi oluşturulurken, ilgili bağlam ile ilgili yapılmış olan konfigurasyon bilgisi "roleManager"a aktarılır. Bu sayede "roleManager" verilen bir aktör instance'ı için uygun bir rol tanımı mevcut ve verilen aktör instance'ı tanımlanmış olan rolü

edinmeye yeterli ise (rol atama kurallarına başarılı sonuç döndürür ise), ilgili rolün atanmasını sağlayabilmektedir.

Rol ataması işlemi iki farklı şekilde yapılabilmektedir:

- Bir bağlama dahil olunduğunda, tanımlanmış olan rol sınıfından yeni bir instance oluşturulur ve aktör sınıfına döndürülür. Yeni instance oluşturulabilmesi için rol sınıfının default constructor metodunun bulunması gerekmektedir. Aksi durumda role instance'ı oluşturulamaz ve RoleInstantiationException fırlatılır.
- Bir bağlama dahil olunurken rol instance'ı halihazırda oluşturulmuş ise ve aktör ilgili rolü almaya yeterli yeteneğe sahip ise, oluşturulmuş olan rol instance'ı aktöre aktarılarak, verilen rol instance'ının kullanılması sağlanabilir. Ancak verilen rol instance'ının kurallarda tanımlanan rol sınıfından oluşturulmuş olması ve halihazırda başka bir aktör tarafından oynanmıyor olması gerekmektedir.

Bunların dışında "roleManager" sınıfı, ilgili bağlama giren ve çıkan aktörlerin bilgisini işleyerek, tüm bağlam sınıfları tarafından erişilebilir rol kayıtlarını güncellemektedir. Böylelikle rol sınıfları bağlamlar üzerinden bind edildikleri aktör bilgilerine erişebilmektedirler.

3.3.4.2 Aktör Üzerindeki Runtime Konfigurasyonu

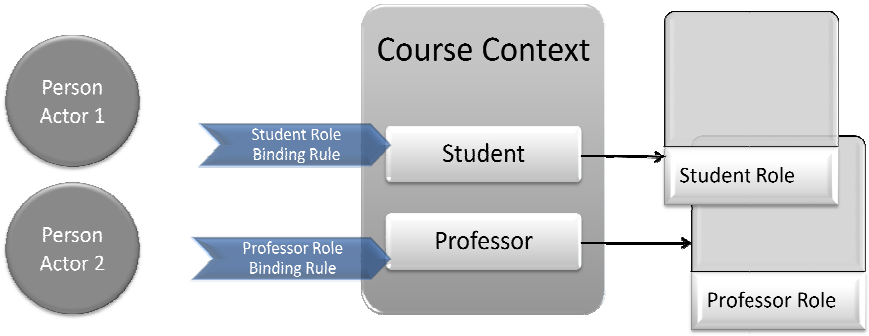
JcoreBindingManager sınıfından oluşturulan "bindingManager" değişkeni her bir aktör içerisinde farklı bir instance şeklinde oluşturulur. Bu nesnenin iki temel görevi bulunmaktadır:

- Bağlam giriş çıkışlarında ilgili aktörün aldığı ya da bıraktığı rol kayıtlarını tutmak:
    - Böylelikle herhangi bir anda aktörün oynamakta olduğu roller, bu rollerin gerektirmiş olduğu değişiklikler ve içinde bulunduğu bağlamlar "bindingManager" tarafından bilinmektedir.
- Aktör metotlarının dinamik olarak farklı davranış sergileyebilmesini sağlamak:

- Bölüm 3.3.3'te açıklandığı üzere, değişiklik gerektiren metotların içeriği değiştirilerek aktör sınıfı içerisindeki "hasReplaceMethod", "executePreMethod", "executeReplaceMethod" ve "executePostMethod" metotları çağrılarak uygun method çağrılarının yapılması sağlanmaktadır.
- Bu aşamada, anlık olarak sahip olunan roller ve bu rollerin gerektirdiği method değişiklikleri "bindingManager" tarafından bilindiği için, hangi method öncesinde/sonrasında/yerine hangi rol metodu ya da bağlam metodu çalıştırılması gerektiği, ilgili bağlam ya da rol nesnesine referanslar gibi bilgiler kullanılarak, uygun metotların çalıştırılması sağlanmaktadır.

3.4 JCORE Kullanım Örneği

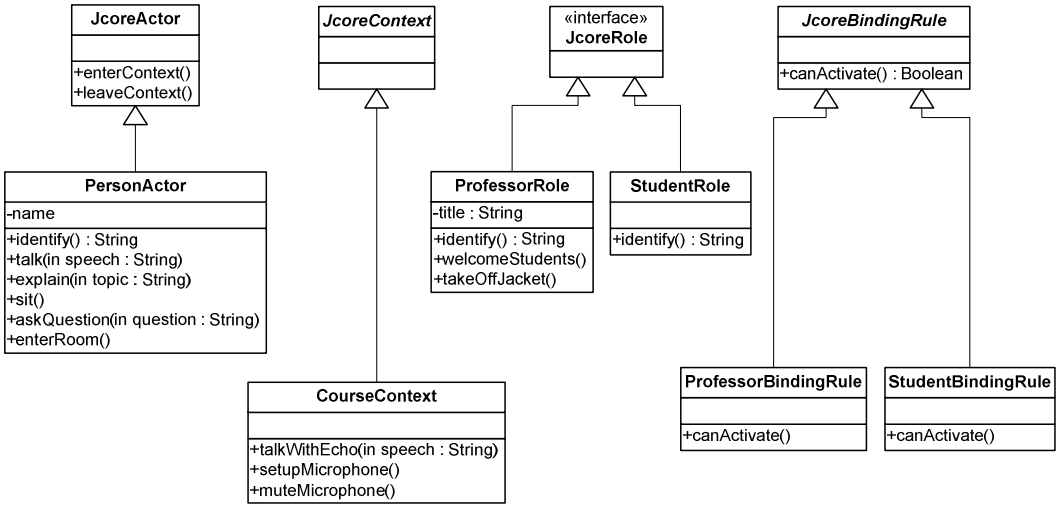
Bu kısımda JCORE altyapısı kullanılarak rol modellemesi gerçekleştirilen bir örnek incelenmiştir.



Şekil 3. 24 JCORE üzerinde bağlam ve aktör ve rol örneği

Şekil 3. 24'te görüldüğü üzere, bir ders bağlamını modellemek üzere "CourseContext" nesnesi kullanılmıştır. Ders bağlamında alınabilecek 2 farklı rolden, öğrenci ve öğretmen rolleri için sırasıyla "Student" ve "Professor" rolleri tanımlanmıştır. Her bir rol tanımı için uygun birer rol kazanım kuralı tanımlanmış ve 2 farklı aktör nesnesi tanımlanmıştır.

Şekil 3. 25'te yer alan sınıf diyagramında JCORE paketindeki JcoreActor, JcoreContext, ve JcoreBindingRule sınıflarını extend eden, JcoreRole sınıfını implement eden sınıflar gösterilmiştir.



Şekil 3. 25 Örnek rol model sistemi sınıf diyagramı

Bu senaryoya göre PersonActor nesnesi kendi başına çeşitli özellikleri ve davranışları olan bir kişiyi modellemektedir. CourseContext nesnesi ise bir sınıf bağlamını modellemektedir. Göz önüne alınan senaryoda sınıf bağlamında mikrofonlar yer almakta ve mikrofonla konuşma esnasında ekolu bir ses ile konuşulmaktadır. ProfessorRole nesnesi ise bir öğretmenin ünvan bilgisini içermektedir. Bir öğretmenin kendisini tanımlarken bu ünvan bilgisini de kullanacağı göz önüne alınmıştır. Ayrıca öğretmenin öğrencileri selamlama ya da ceketini çıkarma gibi metotları da bulunmaktadır. Öğrenci modelinin yansıtıldığı StudentRole nesnesi ise isim bilgisini göz önüne almadan, kendisini bir öğrenci olarak tanımlamaktadır. Bir aktörün ProfessorRole'e sahip olabileceği kararının verildiği ProfessorBindingRule nesnesi ise ilgili aktörün PersonActor tipinden olması ve isminin içerisinde spesifik kelimelerin yer alması zorunluluğunu barındırmaktadır. Benzer şekilde bir aktörün StudentRole'e sahip olabileceğinin kararının verildiği StudentBindingRule nesnesi de ilgili aktörün PersonActor tipinden olması ve isminin içerisinde farklı spesifik kelimelerin yer alması zorunluluğunu barındırmaktadır.

Kendi başlarına bağımsız olan bu nesneler JCORE altyapısında XML konfigurasyonları ile bir araya getirilerek farklı özellikler gösteren nesneler oluşturulabilmektedir. CourseContext'e dahil olan bir aktörün bazı davranışlarını edinmiş olduğu rol metotları ile, bazı davranışlarını ise girmiş olduğu bağlamın barındırdığı rol metotları ile

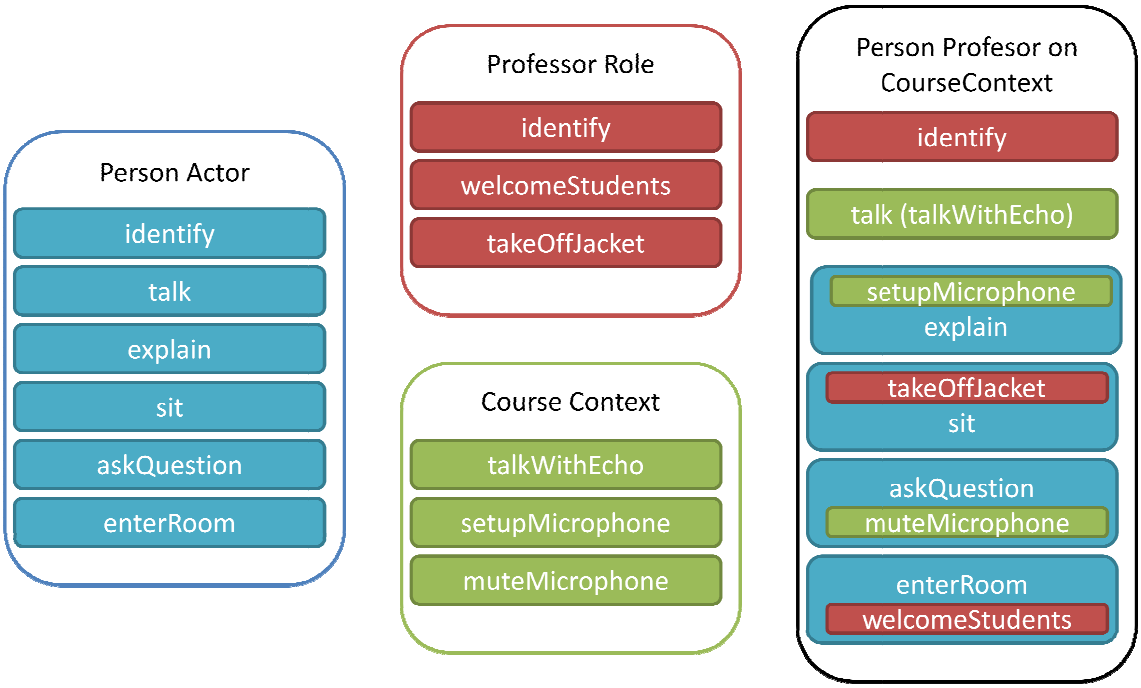
runtime'da güncelleyebilmesi özelliği sağlanmaktadır. Bunun için oluşturulmuş olan JCORE XML konfigurasyonu Şekil 3. 26'da gösterilmiştir.

```
<Jcore>
    <Context id="courseContext" class="edu.yildiz.jcore.impl.CourseContext">
        <Role id="professor" class="edu.yildiz.jcore.impl.ProfessorRole">
            <BindingRule actor="edu.yildiz.jcore.impl.PersonActor"
rule="edu.yildiz.jcore.impl.ProfessorBindingRule">
                <ReplaceRole actorMethod="identify" roleMethod="identify"/>
                <ReplaceContext actorMethod="talk" contextMethod="talkWithEcho"/>
                <BeforeContext actorMethod="explain" contextMethod="setupMicrophone"/>
                <BeforeRole actorMethod="sit" roleMethod="takeOffJacket"/>
                <AfterContext actorMethod="askQuestion" contextMethod="muteMicrophone"/>
                <AfterRole actorMethod="enterRoom" roleMethod="welcomeStudents"/>
            </BindingRule>
        </Role>
        <Role id="student" class="edu.yildiz.jcore.impl.StudentRole">
            <BindingRule actor="edu.yildiz.jcore.impl.PersonActor"
rule="edu.yildiz.jcore.impl.StudentBindingRule">
                <ReplaceRole actorMethod="identify" roleMethod="identify"/>
                <BeforeContext actorMethod="talk" contextMethod="setupMicrophone"/>
                <AfterContext actorMethod="talk" contextMethod="muteMicrophone"/>
            </BindingRule>
        </Role>
    </Context>
</Jcore>
```

Şekil 3. 26 Örnek rol model sistemi JCORE XML konfigurasyonu

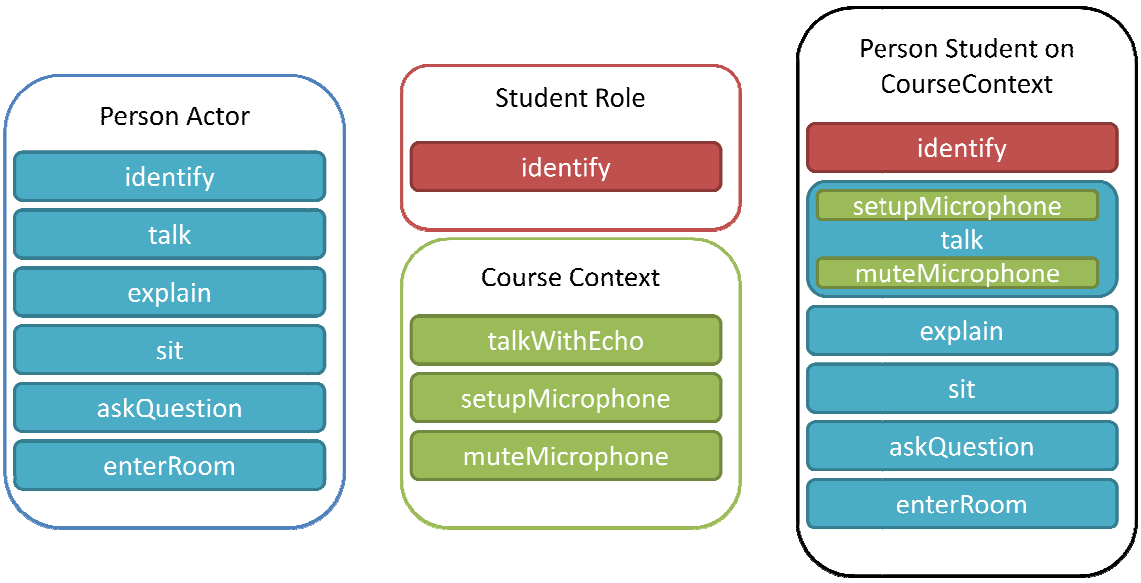
PersonActor tipinden bir aktör CourseContext bağlamı içerisine girdiğinde 2 farklı role bürünme ihtimali olmaktadır. Bunlardan birincisi "professor" rolü, ProfessorBindingRule ile belirlenmekteyken, diğeri "student"rolü StudentBindingRule ile belirlenmektedir. Her iki rol için de farklı dinamik method modifikasyonları tanımlanmıştır.

Şekil 3. 27'de "professor" id'li role sahip olan bir aktörün dinamik method modifikasyonu ile oluşturulan yeni yapısı gösterilmiştir. Bu gösterim içerisinde dinamik method modifikasyonu gerçekleştirilmeden önceki aktör metotları, tanımlanan bağlamın sahip olduğu metotlar ve "professor" id'si ile tanımlanmış olan rolün sunmuş olduğu metotlar gösterilmiştir.



Şekil 3. 27 “professor” rolü için runtime method modifikasyonları ile oluşturulan yapı

Şekil 3. 28’de “student” id’li role sahip olan bir aktörün dinamik method modifikasyonu ile oluşturulan yeni yapısı gösterilmiştir. Bu gösterim içerisinde dinamik method modifikasyonu gerçekleştirilmeden önceki aktör metotları, tanımlanan bağlamın sahip olduğu metotlar ve “student” id’si ile tanımlanmış olan rolün sunmuş olduğu metotlar gösterilmiştir.



Şekil 3. 28 "student" rolü için runtime method modifikasyonları ile oluşturulan yapı

JCORE sisteminin çalışmasını görebilmek üzere Şekil 3. 29'daki gibi bir Demo sınıfı oluşturulmuştur.

```
public class Demo {

    public static void main(String[] args) throws InvalidJcoreConfigurationException,
ConfigurationNotFoundException {
        new JcoreEnvironmentSetup("Jcore.xml");
        PersonActor person1 = new PersonActor("Yunus Emre Selcuk");
        PersonActor person2 = new PersonActor("Mehmet Pekmezci");
        System.out.println("RUNNING FOR PERSON 1:");
        executeMethods(person1);
        System.out.println("RUNNING FOR PERSON 2:");
        executeMethods(person2);
        System.out.println();
        System.out.println("ENTERING CONTEXT");
        System.out.println();
        CourseContext courseContext = new CourseContext();
        person1.enterContext(courseContext);
        person2.enterContext(courseContext);
        System.out.println("RUNNING FOR PERSON 1:");
        executeMethods(person1);
        System.out.println("RUNNING FOR PERSON 2:");
        executeMethods(person2);
        System.out.println();
        System.out.println("LEAVING CONTEXT");
        System.out.println();
        person1.leaveContext(courseContext);
        person2.leaveContext(courseContext);
        System.out.println("RUNNING FOR PERSON 1:");
        executeMethods(person1);
        System.out.println("RUNNING FOR PERSON 2:");
        executeMethods(person2);
    }

    private static void executeMethods(PersonActor person) {
        System.out.println("IDENTIFICATION => " + person.identify()) ;
        person.enterRoom();
        person.sit();
        person.talk("SAMPLE SPEECH");
        person.askQuestion("SAMPLE QUESTION");
        person.explain("SAMPLE TOPIC");
    }
}
```

Şekil 3. 29 JCORE çalışmasını örnekleyen Demo sınıfı

Demo sınıfının içerisindeki main method'da 2 adet aktör nesnesi oluşturulmuştur. Oluşturulan aktör nesnelerinin isimleri "Mehmet Pekmezci" ve Yunus Emre Selcuk" olarak verilmiştir. StudentBindingRule kuralı, isminde "Mehmet" geçen aktörlerin "student" rolünü alabileceğini belirtmiş olduğundan ve ProfessorBindingRule kuralı isminde "Yunus" geçen aktörlerin "professor" rolünü alabileceğini belirtmiş olduğundan, aynı sınıf bağlamına dahil olan bu iki aktörün 2 farklı rol alarak farklı davranışlar sergilemesi beklenmiştir.

Aktörlerin bağlam içinde ve dışında göstermiş oldukları davranışları gösterebilmek üzere nesneler bağlama girmeden önce, bağlama girdikten sonra ve bağlamdan çıktıktan sonra aynı metotlar çalıştırılmıştır. Örnek sınıfların tamamı gerçekleştirdiği işlemi konsola yazmakta olduğundan çıktıların incelenmesi metotların ne şekilde davrandığını açıklamış olacaktır. Bu sebeple Şekil 3. 30'da ilgili sınıfın çalıştırılması ile oluşturulan çıktı gösterilmiştir.

```
RUNNING FOR PERSON 1:
IDENTIFICATION => Yunus Emre Selcuk
Entering room...
Sitting...
talking silently : SAMPLE SPEECH
Asking question : SAMPLE QUESTION
this topic is being explained : SAMPLE TOPIC
RUNNING FOR PERSON 2:
IDENTIFICATION => Mehmet Pekmezci
Entering room...
Sitting...
talking silently : SAMPLE SPEECH
Asking question : SAMPLE QUESTION
this topic is being explained : SAMPLE TOPIC

ENTERING CONTEXT

RUNNING FOR PERSON 1:
IDENTIFICATION => Mr. Professor
Entering room...
Welcome to the class!
Taking the jacket off...
Sitting...
==> talking with echo : SAMPLE SPEECH
```

Şekil 3. 30 Demo sınıfının çalıştırılması ile üretilen çıktılar

```
Asking question : SAMPLE QUESTION
==> Muting the microphone...
==> Setting up the microphone...
this topic is being explained : SAMPLE TOPIC
RUNNING FOR PERSON 2:
IDENTIFICATION => Student
Entering room...
Sitting...
==> Setting up the microphone...
talking silently : SAMPLE SPEECH
==> Muting the microphone...
Asking question : SAMPLE QUESTION
this topic is being explained : SAMPLE TOPIC

LEAVING CONTEXT

RUNNING FOR PERSON 1:
IDENTIFICATION => Yunus Emre Selcuk
Entering room...
Sitting...
talking silently : SAMPLE SPEECH
Asking question : SAMPLE QUESTION
this topic is being explained : SAMPLE TOPIC
RUNNING FOR PERSON 2:
IDENTIFICATION => Mehmet Pekmezci
Entering room...
Sitting...
talking silently : SAMPLE SPEECH
Asking question : SAMPLE QUESTION
this topic is being explained : SAMPLE TOPIC
```

Şekil 3. 31 Demo sınıfının çalıştırılması ile üretilen çıktılar (devam)

Çıktılardan da görülebildiği üzere aynı sınıftan yaratılmış 2 nesne bağlam dışında orjinal method özellikleri le davranış sergilerken, sınıf bağlamına dahil olduklarında farklı roller edinerek farklı method davranışları sergilemektedirler. Bu method davranışlarının bir kısmı roller üzerinden bir kısmı ise bağlamlar üzerinden sağlanmaktadır. Öte yandan çıktılar içerisinde bağlama giriş yapmadan önce ve bağlamdan çıkış yaptıktan sonra her iki aktör için de oluşan çıktıların aynı olmasından yola çıkarak, aynı aktör sınıflarının bağlamlardan çıktıktan sonra eski özelliklerini koruyabildiği görülebilir.

3.5 Steimann Prensipleri ve JCORE

Bu bölümde Steimann tarafından ortaya atılan rol modellerinin taşıması gereken özelliklerin JCORE üzerinden ne ölçüde karşılandığı incelenmiştir: [10]

1. *Roller kendi özellikleri ve davranışları ile varolurlar*
    - Bir rol nesnesi, JcoreRole interface'ini implement etmek dışında standart bir java sınıfı olduğu için kendi sınıf değişkenleri ve metotları yer almaktadır.
2. *Roller ilişkilere bağlıdırlar*
    - Her bir rol kavramı, bağlamlar içerisinde anlam kazanmaktadır. Rol nesneleri oluşturulurken her ne kadar bu ilişki kurulmamış olsa da JCORE XML konfigurasyonu gerçekleştirildiğinde roller, bağlamlar ve aktörler arasındaki ilişki kurulmuş olur.
3. *Bir nesne eş zamanlı olarak farklı rolleri oynayabilir*
    - JCORE anaçatısı ile, bir aktör bir bağlamdan bir ya da birden çok rol edinebilir. Öte yandan bir aktör eş zamanlı olarak birden çok bağlam içerisinde yer alabilir.
4. *Bir nesne aynı rolü eş zamanlı olarak birden çok kez oynabillir*
    - JCORE altyapısı aynı rol sınıfının farklı bağlamlar altında tanımlanmasına izin vermektedir. Bu sayede aynı rol tanımının yapıldığı birden çok bağlama eş zamanlı olarak giriş yapmış olan aktör eğer rol edinme kurallarını sağlıyor ise aynı rolü birden çok kez oynama şansına sahip olacaktır.
    - Rol tanımı ile rolün aktör üzerinde yarattığı dinamik method modifikasyonları birbirinden ayrılmış olduğu için, farklı bağlamlarda aynı rolü oynayıp farklı method modifikasyonları oluşturulabilir. Böylece bir aktör aynı rolü farklı bağlamlar altında eş zamanlı olarak oynarken, aynı rolün farklı method modifikasyonları olabilecektir.

5. *Nesneler rolleri dinamik olarak edinebilir ve bırakabilirler*
    - JCORE altyapısında aktörler bağlamlara giriş yaptıklarında dinamik olarak rolleri edinip, bağlamlardan çıktıklarında yine dinamik olarak bu rolleri bırakabilmektedirler.
6. *Rollerin oynanma ve bırakılma sırası için kısıtlar tanımlanabilir*
    - JCORE rollerin oynanabilmesi için runtime'da aktörün sahip olduğu özelliklere göre kısıt koyma imkanı sağlamaktadır. Bu sayede aktörler bir bağlama dahil olduklarında rol ataması yaparken çalıştırılan kurallar arasında aktörün belirli bir rolü oynuyor olması kuralı eklenerek bu yetenek sağlanabilir.
    - Ancak JCORE üzerinde rollerin bırakılması için kısıt tanımlama imkanı sunulmamaktadır.
7. *Aralarında ilişki bulunmayan nesne türleri aynı rolü oynayabilir*
    - Aynı ya da farklı bağlamlar altında aynı rol sınıfı tamamen farklı aktör sınıfları tarafından oynanabilir.
    - JCORE'un sunmuş olduğu tek ilişki kısıtı aktörlerin JcoreActor nesnesinden türemiş olmalarıdır.
8. *Roller rolleri oynayabilir*
    - JCORE anaçatısında, rolleri oynama işlemi aktörlerin herhangi bir bağlama girmesi ile sağlanır. Yani rolü oynacak olan nesne her zaman aktör olmalıdır.
    - Steimann'ın bu gereksinim için yaptığı açıklama aşağıdaki gibidir:
        - "Person" tipinden bir çalışanın sahip olduğu bir rol olan "Employee" rolünün "ProjectLeader" gibi farklı bir role bürünebilmesi yeteneği. [10]
    - JCORE altyapısında bu yetenek direkt olarak sağlanmamakla birlikte, ancak "ProjectLeader" gibi bir rolün farklı bir bağlamda tanımlanması ile

sadece “Employee” rolüne sahip olan bir aktörün “ProjectLeader” haline gelebilmesi mümkün olmaktadır.

9. *Roller bir nesneden diğerine transfer edilebilir*
    - Halihazırda JCORE altyapısı bu özelliği desteklememektedir.

10. *Nesnenin durumu (state’i) rol-spesifik olabilir*
    - Steimann tarafından bu gereksinimin açıklaması aynı rolün birden fazla kez oynanabildiği durumda her bir rolün farklı instance’lar şeklinde yer alması şeklinde açıklanmaktadır.
    - JCORE altyapısında her bir rol ayrı birer instance olduğu için, aynı rol aynı aktör tarafından her oynandığında farklı bir instance’ı olacaktır.

11. *Nesnenin davranışları rol-spesifik olabilir*
    - Aktör tarafından oynana her bir rol farklı bir instance şeklinde tutulduğu ve bu roller üzerinde durum bilgileri saklanabildiği için, aktörlerin dinamik olarak güncellenen metotları her bir rolün içinde bulunduğu duruma göre belirlenecektir.

12. *Rollerin erişim kısıtları olur*
    - Steimann bu gereksinimi roller kendilerini oynayan nesnelerin özelliklerine erişemezler şeklinde açıklamıştır.
    - JCORE altyapısında da “public” access modifier’ı ile işaretlenmemiş olan aktör sınıfı değişkenleri ve metotları rol tarafından erişilemez durumdadır.

13. *Farklı roller aynı yapıyı ve özellikleri paylaşabilirler*
    - Steimann’ın bu gereksinim için yapmış olduğu açıklamada 2 temel ihtiyaçtan bahsedilmektedir:
        - Roller birbirinden kalıtım ile türetilebilir olmalıdır
        - Roller kendilerini oynayan nesnelere delegasyon yapabilmelidir

- JCORE altyapısı içerisinde bu gereksinimler aşağıdaki şekillerde karşılanmaktadır:
  - Rol sınıfları kullanıcının istediği gibi hiyerarşi kurabileceği, birbirinden türetebileceği standart java sınıflarıdır.
  - Rol içerisinden kendisini oynayan aktör sınıfının metotları çağrılabilmektedir.

14. Nesneler ve rolleri aynı kimliği paylaşırlar

- JCORE anaçatısında roller tamemen bağımsız nesneler olduğu için içerilerinde kimlik bilgisi barındıramazlar. Ancak OT/J tarafından bu özelliği sağlamak için belirtilen özellik [9] JCORE içerisinde yer almaktadır. Aktör sınıfları anlık olarak oynamakta olduğu rollerin listesine sahip olduğu için bu bir aktör ile rol nesnesinin aynı kimliklere sahip olup olmadığının anlaşılması sağlanmaktadır. Bunun için aktör sınıflarının "jcoreEquals" metodu kullanılabilir.

15. Nesneler ve rollerinin farklı kimlikleri bulunur

- JCORE anaçatısında roller ve aktörler tamamen farklı nesneler olduğu için farklı kimliklere sahiptirler. Bunun için "==" operatörü ya da java.lang.Object tarafından sağlanan "equals" metodu kullanılabilir.

JCORE rol modeli Steimann tarafından belirlenmiş olan 15 kriterden 12'sini tam olarak sağlamakta, birini (6) kısmen sağlamakta, ikisini (8,9) sağlayamamaktadır.

BÖLÜM 4

SONUÇ VE ÖNERİLER

JCORE çözümü, Java One 2012 konferansından kendisine yer bulabilmiş [12] güncel bir konuda, rol modellemeleri üzerine bir çözüm sunmaktadır. Sunulan çözüm tamamen java programlama dili içerisinde OOP temelli bir geliştirme yönteminin çalışma zamanında rol tabanlı bir yapıya dönüşmesini sağlar.

Çözümün temel yapı taşları bir XML konfigurasyon dosyası ile çalışma zamanında sınıfların güncellenmesini sağlayan modüldür. XML konfigurasyon dosyası rol ve bağlam kavramlarının tanımlanması ve kurallarının belirlenmesi için programcı tarafından kullanılmaktayken, uygulama çalışmaya başladığı anda sınıfların JVM'e yüklenmeden önce güncellenmesini sağlayan modül programcıya tamemen görünmez bir katmak olarak yer almaktadır.

Kurulan altyapı sayesinde programcı standart geliştirilebilen bir java uygulamasındaki sınıflarının bazılarını belirli sınıflardan türeterek ve ilgili konfigurasyonu XML üzerine ekleyerek gerçekleştirebilmektedir.

Bugüne kadar çeşitli rol modelleme yaklaşımları sunulmuş olmasına rağmen JCORE bu sunulmuş olan çalışmalardan farklı olarak aşağıdaki özellikleri bir arada sunabilmektedir:

- Sadece Java programlama dili ve XML konfigurasyon dosyaları ile çalışması
- Bağlam kavramını sunması

- Kolay yönetilebilir ve dinamik bir yapı sunması

JCORE, birçok özelliği geliştirilerek programcılar için daha kullanışlı bir modelleme aracı olarak sunulabilir. Örnek geliştirme alanları olarak aşağıdaki alanlar görülebilir:

- Multithread çalışma desteği
- Kontrollerin arttırılarak, programcının hata yapma ihtimalinin düşürülmesi
- Dinamik method modifikasyonuna kalıtım desteğinin verilmesi (özellikle Overload ve Override edilen metotlar için)
- Nesne olmayan, “Native” değişken tipleri için destek eklenmesi

KAYNAKLAR

[1] Selçuk, Y E. (2006) Extending Object Oriented Programming with Role Support. PhD Thesis, Istanbul Technical University, Ins. Science and Technology, Istanbul, Turkey.

[2] Herrmann, S. (2007) A Precise Model for Contextual Roles: The Programming Language ObjectTeams/Java. Applied Ontology; 2:181–207.

[3] Graversen, K B.ve Østerbye, K. (2003) Implementation of a Role Language for Object-Specific Dynamic Separation of Concerns

[4] Wikipedia, Data, context and interaction, http://en.wikipedia.org/wiki/Data,_context_and_interaction, 18 Ağustos 2012

[5] Wong R K., Chau L H., Lochovsky F H. (1997) A Data Model and Semantics of Objects with Dynamic Roles.

[6] Herrmann, S. (2005) Programming with Roles in ObjectTeams/Java.

[7] Gamma E, Helm R, Johnson R, Vlissides J (1994) Design Patterns: Elements of Reusable Object-Oriented Software.

[8] Schrefl M., Thalhammer T. (2004) Using roles in Java. Software - Practice and Experience; 34:449–464.

[9] Herrmann, S. (2007) A precise model for contextual roles: The programming language ObjectTeams/Java.

[10] Steimann, F. (2000) On the representation of roles in object-oriented and conceptual modelling

[11] Object Teams, Publications, http://www.objectteams.org/publications/, 18 Ağustos 2012

[12] Redefining Modularity and Reuse in Variants—All with Object Teams, (2012) https://oracleus.activeevents.com/connect/sessionDetail.ww?SESSION_ID=6780, 18 Ağustos 2012

[13] Javassist, http://www.javassist.org/, 18 Ağustos 2012

[14] Wikipedia, AspectJ, http://en.wikipedia.org/wiki/Aspectj, 18 Ağustos 2012

[14] Wikipedia, Hotspot, http://en.wikipedia.org/wiki/HotSpot, 18 Ağustos 2012

[15] Javassist, Online API Manual, Class HotSwapper http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/html/index.html, 18 Ağustos 2012

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mehmet PEKMEZCİ

Doğum Tarihi ve Yeri : 04.06.1984 / İZMİR

Yabancı Dili : İngilizce

E-posta : mehmet.pekmezci@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Mühendisliği	İstanbul Teknik Üniversitesi	2007
Lisans	Konrol Mühendisliği	İstanbul Teknik Üniversitesi	2008
Lise	Fen Lisesi	Manisa Turgutlu Halil Kale Fen Lisasi	2002

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2011-	Alcatel-Lucent	Çözüm Mimarı
2008-2011	Alcatel-Lucent	Mühendis
2007-2008	Valensas Teknoloji Hizmetleri	Mühendis

ÖDÜLLERİ

1.	2007 EMO Proje Yarışması Birinciliği