EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(DOKTORA TEZİ)

ÇOKLU ETMEN SİSTEM GELİŞTİRİMİNDE YENİDEN YAPILANDIRMA

Ali Murat TİRYAKİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilim Dalı Kodu: 619.01.00

Sunuş Tarihi: 09.03.2009

Tez Danışmanı: Prof. Dr. Oğuz DİKENELLİ

Bornova-İZMİR

Ali Murat TİRYAKİ tarafından DOKTORA tezi olarak sunulan "**ÇOKLU ETMEN SİSTEM GELİŞTİRİMİNDE YENİDEN YAPILANDIRMA**" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve **09/03/2009** tarihinde yapılan tez savunma sınavında aday oy birliği/oy çokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:	**İmza**
Jüri Başkanı: Prof. Dr. Oğuz DİKENELLİ	.....
Raportör Üye: Yrd. Doç. Dr. R. Cenk ERDUR	.....
Üye: Prof. Dr. N. Yasemin TOPALOĞLU	.....
Üye: Yrd. Doç. Dr. Tuğkan TUĞLULAR	.....
Üye: Yrd. Doç. Dr. Adil ALPKOÇAK	.....

TEŞEKKÜR

Doktora eğitimim boyunca bana sadece doktora çalışmamda değil, her konuda destek olan danışmanım Prof. Dr. Oğuz Dikenelli'ye çok teşekkür ederim. Özellikle de her zaman anlayışlı ve iyi niyetli olduğu için...

Tez izleme komitesinde bulunan Yrd. Doç. Dr. Rıza Cenk Erdur ve Yrd. Doç. Dr. Tuğkan Tuğlular'a katkılarından ve değerli yönlendirmelerinden dolayı teşekkür ederim.

Doktora savunması değerlendirme jürisinde bulunan diğer hocalarım; Prof. Dr. N. Yasemin Topaloğlu ve Yrd. Doç. Dr. Adil Alpkoçak'a değerlendirmeleri için teşekkür ederim.

Tez çalışması kapsamında test sürecinin belirlenmesi ve test aracının geliştirimi sırasında birlikte çalıştığım Erdem Eser Ekinci ve Seagent geliştiricileri grubunun diğer üyelerine teşekkür ederim.

Tezin yazımı sırasında yardımcı olan Araş. Gör. Önder Gürcan'a teşekkür ederim.

Eğitimime verdikleri sınırsız destekten dolayı annem Latife Tiryaki ve doktora eğitimim sırasında kaybettiğim babam Hikmet Tiryaki'ye sonsuz teşekkürlerimi sunarım. Onlar olmasaydı bu satırları yazamazdım.

Ve son olarak, doktora eğitimim sırasında kazandığım eşim Simge Pervin Tiryaki'ye bitmek bilmeyen çalışma saatlerine karşı her zaman anlayışlı olduğu için çok teşekkür ederim.

ÖZET

ÇOKLU ETMEN SİSTEM GELİŞTİRİMİNDE YENİDEN YAPILANDIRMA

TİRYAKİ, Ali Murat

DOKTORA Tezi, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Oğuz DİKENELLİ

09/03/2009, 213 sayfa

Çoklu etmen sistemlerinin karmaşık ve değişimlere açık doğası, bu sistemlerin geliştirimi sırasında gereksinimlerin sürekli değişimi ile baş edebilecek evrimsel geliştirim yaklaşımlarını gerekli kılmaktadır. Bu tezde, sistem tasarımındaki değişiklikleri yöneterek evrimsel geliştirimi mümkün hale getiren geleneksel yeniden yapılandırma pratiğinin çoklu etmen sistem - ÇES geliştirimi için uyarlanması amaçlanmıştır. Bu amaç doğrultusunda, yeniden yapılandırma pratiğinin gerektirdiği test altyapısını oluşturan etmen tabanlı test güdümlü geliştirim yaklaşımı, bu yaklaşımdaki test etkinliklerinin yürütülmesini sağlayan hedef yönelimli bir test süreci ve ÇES geliştiriminde kullanılabilecek bir yeniden yapılandırma yaklaşımı tanıtılmaktadır. Geliştirilen yeniden yapılandırma yaklaşımı içerisinde, ÇES geliştirimi sırasında elde edilen yeniden yapılandırılabilir ürünler belirlenmiş, bu ürünlerin oluşturduğu tasarım yapılarında sıklıkla karşılaşılan problemler ve bu problemleri çözen

yeniden yapılandırma desenleri tanımlanmıştır. Tez çalışması sırasında, evrimsel ÇES geliştiriminde kullanılmak üzere önerilen yaklaşımları destekleyen SeaUnit adında bir test aracı ve ReSeagent adında bir yeniden yapılandırma aracı SEAGENT etmen çerçevesi üzerine geliştirilmiştir.

Anahtar Sözcükler: Etmen tabanlı yazılım geliştirme, çevik yazılım geliştirme süreçleri, çoklu etmen sistem geliştirimi, test etme, yeniden yapılandırma.

VIII

ABSTRACT

REFACTORING IN MULTI AGENT SYSTEM DEVELOPMENT

TİRYAKİ, Ali Murat

PhD. in Computer Engineering
Supervisor: Prof. Dr. Oğuz DİKENELLİ
09/03/2009, 213 pages

The complex and open nature of multi agent systems requires evolutionary development approaches that can cope with frequently changing requirements during the development of these systems. This thesis aims to adapt and apply the traditional refactoring practice that makes the evolutionary development possible by managing the changes in design, into the multi agent system - MAS development. For this purpose, a test driven approach called agent oriented test driven development that provides a testing infrastructure for refactoring, a goal oriented testing process that supports to apply testing activities in this test driven approach and a refactoring approach that can be used in MAS development are introduced. In the proposed refactoring approach, the refactorable development artifacts in MAS development are specified, the frequently encountered problems on the structures of these refactorable artifacts and the refactoring techniques that solve these problems have been identified. A testing tool called

SeaUnit and a refactoring tool called ReSeagent that support the proposed test driven development and refactoring approaches, were developed on the SEAGENT agent framework.

Keywords: Agent oriented software engineering, agile software development processes, multi agent system development, testing, refactoring.

İÇİNDEKİLER

XVIII

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

XXIV

KISALTMALAR

Kısaltma	Açılım
AVKD	Anlamsal Veb Kural Dili (*Semantic Web Rule Language - SWRL*)
BMD	Birleştirilmiş Modelleme Dili (*Unified Modeling Language - UML*)
ÇES	Çoklu Etmen Sistemi (*Multi Agent System - MAS*)
EBMD	Etmenler için Birleştirilmiş Modelleme Dili (*Agent UML - AUML*)
EDPÇ	Eşgüdümlü Dağıtık Problem Çözme (*Coordinated Distrubuted Problem Solving - CDPS*)
EYTGG	Etmen Yönelimli Test Güdümlü Geliştirim (*Agent Oriented Test Driven Development - AOTDD*)
EYYG	Etmen Yönelimli Yazılım Geliştirim (*Agent Oriented Software Development - AOSD*)
EYYM	Etmen Yönelimli Yazılım Mühendisliği (*Agent Oriented Software Engineering" - AOSE*)
HYG	Hedef Yönelimli Geliştirim (*Goal Oriented Development - GOD*)
KİN	Kanı-İstek-Niyet (*Belief-Desire-Intetion - BDI*)
MGG	Model Güdümlü Geliştirim (*Model Driven Development - MDD*)
MGM	Model Güdümlü Mühendislik (*Model Driven Engineering - MDE*)

Kısaltma	Açılım
NYG	Nesneye Yönelik Geliştirim (*Object Oriented Development - OOD*)
RBS	Rasyonel Birleştirilmiş Süreç (*Rational Unified Process - RUP*)
RED	Rol Etkileşim Diyagramı (*Role Interaction Diagram - RID*)
SGA	Sıradüzensel Görev Ağı (*Hierarchical Task Network - HTN*)
TAG	Test Altındaki Görev (*Task Under Test - TUT*)
TAH	Test Altındaki Hedef (*Goal Under Test - GUT*)
TGG	Test Güdümlü Geliştirim (*Test Driven Development - TDD*)
TGP	Test Güdümlü Programlama (*Test Driven Programming - TDP*)
UP	Uç Programlama (*Extreme Programming - XP*)
VOD	Veb Ontoloji Dili (*Web Ontology Language - OWL*)
ZFEK	Zeki Fiziksel Etmenler Kuruluşu (*Foundation for Intelligent Physical Agents - FIPA*)

1 GİRİŞ

Etmen yönelimli yazılım mühendisliği - EYYM (*"Agent Oriented Software Engineering" - AOSE*) etmen soyutlaması üzerine kurulan karmaşık ve dağıtık yazılım sistemlerinin geliştirilmesini amaçlayan bir araştırma alanıdır(Wooldridge and Ciancarini, 2001). Bu alan, etmen tabanlı sistemlerin bir mühendislik yaklaşımı olarak endüstriye tanıtılması amacıyla ortaya çıkmıştır. Bu tür sistemlerin geliştirilmesi için, geliştirim süreci boyunca geliştiricilere rehberlik edecek etmen tabanlı yöntemlere gereksinim duyulmaktadır. Bu yüzden, etmen tabanlı yazılım geliştirme yöntemlerinin tanımlanması EYYM alanı içerisindeki önemli konulardan biridir. Son yıllarda etmen tabanlı yöntemlerin tanımlanması için büyük bir çaba harcanmıştır.

Literatürde çoklu etmen sistem - ÇES (*"Multi Agent System" - MAS*) geliştirini için bir çok yöntem tanıtılmıştır. Bu yöntemlerden Gaia (Wooldridge et al., 2000; Zambonelli et al., 2003), SODA (Omicini, 2001), Prometheus (Padgham and Winikoff, 2002), MaSE (DeLoach S. A., 1999) gibi bazıları şelale benzeri süreçler kullanmaktadır. Bunların dışındaki INGENIAS (Gomez-Sanz and Pavon, 2005), Tropos (Bresciani et al., 2004), Passi (Cossentino et al., 2003) gibi bazı yöntemler ise artışlı süreçler takip etmekle birlikte gerçekleştirim aşamasını bu artışlı sürecin dışında tutarak bu aşama için herhangi bir destek sunmamaktadırlar. Şelale benzeri veya artışlı süreç modellerine sahip olan bu yöntemlerin hemen hemen hepsi gereksinimlerin belirlenmesi, analiz, tasarım, gerçekleştirim ve test etme işlemlerini sırası ile uygulayan şelale benzeri bir süreç akışını takip et-

mektedirler.

Çoklu etmen sistemleri gibi karmaşık sistemlerin geliştirilmesi sırasında, geliştirim sürecinin henüz başlangıcında tüm gereksinimlerin belirlenmesi neredeyse imkânsızdır. Bununla birlikte, çoklu etmen sistemleri işlevsel ve işlevsel olmayan gereksinimleri geliştirim sürecinin ilk aşamalarında anlaşılamayacak kadar karmaşık olabilen açık organizasyonlar olarak çalışabilirler. Bu nedenlerden dolayı, çoklu etmen sistemlerinin şelale benzeri süreçler ile geliştirilmesi oldukça zordur.

Gereksinimlerin dinamik olarak değiştiği alanları modellemek için geleneksel yaklaşımlardan farklı bir geliştirim sürecine olan gereksinim geleneksel yazılım mühendisliği alanı içerisinde de fark edilmiştir (Beck and Andres, 2004). Değişikliklerin kaçınılmaz olduğu fikri üzerine kurulan ve süreç boyunca meydana gelen değişiklikleri yönetmeye çalışan çevik süreçler[1] bu problemleri çözebilmek amacıyla tanımlanmıştır. Çevik süreçlerin tümü temel olarak kontrol edilebilir bir yol içerisinde yinelemeli ve artışlı geliştirim uygulamak amacıyla bazı hafif pratikler tanıtmaktadırlar. Yinelemeli ve artışlı geliştirim tarzları ortak kullanılarak evrimsel bir geliştirim süreci oluşturmaktadırlar.

Yazılım geliştirme topluluğu tarafından en iyi pratiklerden biri olarak kabul edilen ve rasyonel birleştirilmiş süreç - RBS (*"Rational Unified Process" - RUP*) (Kruchten, 1998), uç programlama - UP (*"Extreme Programming" - XP*) (Beck and Andres, 2004) gibi tüm güncel yazılım geliştirme yöntemlerine entegre edilmiş olan evrimsel yazılım geliştirme yaklaşımı çoklu etmen sistemler gibi dinamik sistemlerin geliştirilmesi için

[1] http://agilemanifesto.org/ , son erişim: 25-Ağustos-2008

uygundur. Etmen teknolojisinin endüstri tarafından kabul edilebilirliğinin arttırılması için gereksinimlerin karmaşıklığı ve sürekli olarak değişimi ile baş edebilecek, tüm geliştirim adımlarını içine alarak evrimsel geliştirimi destekleyen UP benzeri çevik süreçlere gereksinim duyulmaktadır. Bu gereksinim EYYM topluluğu içerisinde büyük oranda kabul edilmektedir (Cernuzzi et al., 2005; Zambonelli and Omicini, 2004).

Evrimsel geliştirim ancak sistem tasarımının geliştirim süreci boyunca sürekli olarak güçlendirilmesi ile başarıyla uygulanabilir. Bu yüzden, sistem mimarisinin ve buna bağlı olarak sistem tasarımının sürekli olarak evrimleşmesinin kontrol edilmesi evrimsel geliştirimin anahtar konularından biridir. Evrimsel geliştirim tarzının ÇES geliştirimine başarılı bir şekilde aktarılabilmesi için, evrimsel geliştirim süreci boyunca tasarımdaki değişiklikleri kontrol altında tutan bakım stratejilerine gereksinim vardır.

En yaygın kullanılan çevik süreçlerden biri olan uç programlamada - UP (*"Extreme Programming" - XP*) (Beck and Andres, 2004) yazılım mimarisinin evrimleşmesinin kontrolü geliştirim süreci boyunca uygulanan yeniden yapılandırma (Fowler, 1999) pratiği ile sağlanmaktadır. Yeniden yapılandırma yazılım sisteminin dışsal davranışını değiştirmeksizin iç yapısının güçlendirilmesi sürecidir. Yeniden yapılandırma sayesinde, sistem tasarımı sürekli olarak iyileştirilerek sistemin daha sonra oluşabilecek değişikliklere hazırlıklı olması sağlanır.

Yeniden yapılandırma operasyonu sırasında sistem tasarımında yapılan değişiklik çalışan sistemin bozulmasına sebep olabilir. Bu yüzden, yeniden yapılandırma pratiğinin uygulanabilmesi için sistemin güvenilir-

liğinin otomatikleştirilmiş testler ile garanti altına alınması gerekir. UP'nin yeniden yapılandırma pratiği ile birlikte evrimsel geliştirimi mümkün hale getiren diğer bir önemli pratiği test güdümlü geliştirimdir(Link and Frolich, 2003). Test güdümlü geliştirim - TGG sistemin güvenilirliğini garantileyen otomatikleştirilmiş testlerin oluşturulması için bir yol sunar: Yinelemeler sırasında geliştirilen her sınıf için test kodu üretir ve bu testler sayesinde kodun işlevsel doğruluğunu garanti ederek çalışan kodun değiştirilmesi sonucunda oluşabilecek bozukluklara karşı koruyucu bir kalkan oluşturur.

Çoklu etmen sistemleri geleneksel yazılım sistemlerinden farklı soyutlamalar ve teknikler kullanılarak geliştirilir (Zambonelli et al., 2001). Bu yüzden, evrimsel geliştirim imkanı sunan geleneksel yeniden yapılandırma pratiği, bu pratiğin uygulanabilmesini sağlayan TGG pratiği ve bu iki pratiği destekleyen yazılım araçları ÇES geliştirinde doğrudan kullanılamazlar. Bu pratikler ve araçlar ÇES geliştirimi için yeniden tanımlanmalıdır.

EYYM alanında bu güne kadar tanımlanan yöntemler, yeniden yapılandırma ve TGG pratikleri için oldukça önemli olan test seviyesi için çok az sayıda destek sunmaktadırlar (Dam and Winikoff, 2003; Gomez-sanz and Pavon, 2004). ÇES geliştiriminde test etme konusu üzerine odaklanan çalışmaların sayısının azlığı bu sistemlerin geliştirimi sırasında kullanılacak test sürecinin belirginleşmesini önlemektedir. Test etme süreciyle önemli derecede bağımlılıkları olan TGG pratiğinin ÇES geliştirimine aktarılabilmesi için bu sistemlerin geliştiriminde uygulanacak test sürecinin belirginleştirilmesi gerekmektedir.

Bu tezde, geleneksel yeniden yapılandırma pratiğinin ÇES geliştirimine aktarılması için gerekli olan yaklaşımlar ve bu yaklaşımları destekleyen araçlar geliştirilmiştir. Bu yol içerisinde ilk olarak, etmen yönelimli test güdümlü geliştirim - EYTGG adında bir test güdümlü ÇES geliştirim yaklaşımı (Tiryaki et al., 2006; Tiryaki et al., 2007) tanıtılmaktadır. EYTGG uç programlamanın TGG pratiğini temel alarak çoklu etmen sistemlerinin yinelemeli ve artışlı bir yol içerisinde evrimsel geliştirimini sağlar. Önerilen yaklaşım, TGG sırasında her ÇES senaryosu için etmen görevlerini, etkileşimsel ve organizasyonel sorumlulukların yinelemeli ve artışlı bir şekilde tanımlanması üzerine kurulmuştur.

EYTGG yaklaşımı içerisindeki test etkinliklerinin gerçekleştirilebilmesi için "test hedefi" kavramı üzerine kurulan yeni bir hedef yönelimli test etme yaklaşımı (Ekinci et al., 2008) tanıtılmaktadır. Bu yaklaşım, ÇES geliştirimi için tanıtılan diğer tüm test yaklaşımlarından ve araçlardan farklı olarak, bu sistemlerin geliştirimi sırasında test edilebilecek en küçük birimlerin etmenler değil, etmenler tarafından başarılan hedefler olduğunu savunmaktadır. Önerilen hedef yönelimli test etme yaklaşımının gerektirdiği altyapıyı sunan SeaUnit adından bir test çerçevesi SEAGENT (Dikenelli et al., 2005a; Dikenelli et al., 2005b) ÇES çerçevesi üzerine gerçekleştirilmiştir. Bu altyapı, geliştiricilerin hedef geliştirimi sırasında test hedeflerini tanımlamasına izin vererek gerçekleştirim döngüsü sırasında birim ve bütünleştirme testlerinin gerçekleştirimini destekler.

Tezde tanıtılan diğer bir yaklaşım evrimsel ÇES geliştirimini mümkün hale getiren bir yeniden yapılandırma yaklaşımıdır (Tiryaki et al.,

2008). Bu yaklaşım, geleneksel yeniden yapılandırma yaklaşımının yolunu takip eder ve ÇES geliştirimi için bazı yeni yeniden yapılandırma desenleri sunar. Önerilen yaklaşım, geleneksel yeniden yapılandırma pratiğinin etmen yönelimli yazılım geliştirme - EYYG alanına aktarılması için, ÇES geliştirimi sırasında elde edilen yeniden yapılandırılabilir tasarım unsurları üzerinde üç yeniden yapılandırma seviyesini, bu sistemlerin geliştiriminde karşılaşılan "kötü kokular" adındaki bazı ortak problemleri ve bu problemleri çözmek için "yeniden yapılandırma desenleri" adı verilen bakım tekniklerini tanıtmaktadır. Bu yaklaşımda tanıtılan yeniden yapılandırma desenlerinin her biri ÇES geliştirilmesinde karşılaşılan kötü kokuların bir veya birkaçını çözmek üzerine odaklanmıştır.

ÇES geliştiriminde kullanılmak üzere önerilen yeniden yapılandırma yaklaşımını desteklemek amacıyla, SEAGENT çerçevesi üzerine ReSeagent adında bir yeniden yapılandırma aracı geliştirilmiştir. Bu araç içerisinde, yeniden yapılandırma teknikleri sıradüzensel görev ağı - SGA (*"Hierarchical Task Network" - HTN*) (Williamson et al., 1996; Paolucci et al., 1999) planlama paradigması kullanılarak oluşturulmuş etmen planları olarak tanımlanır. Araç, SEAGENT geliştirim ortamındaki plan editör ve rol-hedef editörden aldığı yeniden yapılandırma istekleri karşısında bir yeniden yapılandırma etmeni yaratır. Daha sonra, isteğe uygun yeniden yapılandırma planı bu etmen tarafından ilgili plan veya rol-hedef model üzerinde çalıştırılır.

Bir sonraki bölümde etmen sistemler, bu sistemlerin geliştirilmesi için tanımlanmış yöntemler ve tezde etmen sistemlere aktarılmaya çalışılan çevik yaklaşımlar hakkında temel bilgiler verilmiştir. Üçüncü bölümde tez

çalışması ile ilgili literatürde tanıtılmış diğer çalışmalar incelenmektedir. Bu amaçla, çevik pratiklerin etme sistemlere aktarılması, yeniden yapılandırma pratiğinin farklı yazılım geliştirme alanları içerisine aktarılması ve ÇES geliştiriminde otomatik test etme konuları üzerine daha önceden tanıtılmış çalışmalar incelenmiştir. Dördüncü bölüm, çoklu etmen sistemlerin artışlı ve yinelemeli bir yol içerisinde geliştirilmesini sağlayan etmen yönelimli test güdümlü geliştirim - EYTGG adında bir test güdümlü geliştirim yaklaşımı tanıtmaktadır. Bu bölümde aynı zamanda EYTGG yaklaşımına zemin oluşturan etmen sistem geliştiriminde hedef yönelimli test etme yaklaşımı ve bu test yaklaşımını desteklemek için gerçekleştirilen SeaUnit adında bir test çerçevesi tanıtılmaktadır. Beşinci bölümde evrimsel ÇES geliştirimi sırasında kullanılabilecek bir yeniden yapılandırma yaklaşımı tanımlanmaktadır. Altıncı bölümde, önerilen yeniden yapılandırma yaklaşımını desteklemek üzere SEAGENT çerçevesi üzerine geliştirilen ReSeagent adında bir yeniden yapılandırma aracı gerçekleştirim detaylarıyla tanıtılmaktadır. Yedinci bölümde, tezde önerilen yeniden yapılandırma yaklaşımı ve aracının gerçek bir ÇES uygulamasının evrimsel geliştirimi sırasında nasıl kullanılacağını gösteren bir örnek verilmiştir. Son bölümde ise, tezin bütünü üzerine bir tartışma bulunmaktadır. Bu bölümde ayrıca, tezde odaklanılan konu ile ilgili ileriye dönük araştırma olanaklarına değinilmektedir.

2 ALTYAPI

2.1 Etmen Teknolojisi

Bu bölümde etmen ve çoklu etmen sistem - ÇES kavramları kısaca açıklanmaktadır.

2.1.1 Yazılım Etmenleri

Yazılım sistemlerinin tasarımı ve geliştirilmesi için yeni bir paradigma olarak ortaya çıkan etmen-tabanlı teknoloji son yıllarda yaygın olarak kullanılmaya başlanmıştır. Özellikle internet gibi dağıtık ve açık ortamlarda çalışabilen yazılımların gerçekleştirilebilmesi bu teknolojiyi oldukça çekici kılmaktadır.

Günümüzde pek çok araştırmacı etmenler üzerine çalışıyor olmasına rağmen tam bir etmen tanımı verilmiş değildir. Bilinen en basit tanım Russel ve Norvig tarafından verilmiştir: “Etmen algılayıcılarıyla ortamı algılayan ve etkileyicileriyle ortamı etkileyen bir sistemdir” (Nilsson et al., 1996).

Bir donanım veya yazılım sisteminin etmen olarak tanımlanabilmesi için etmenleri geleneksel donanım ve yazılım sistemlerinden ayıran birincil etmen özelliklerini barındırması gerekmektedir. Bu özellikler özerklik, karşıt eylemlilik, amaç yönelimlilik, sosyal yetenek ve süreklilik özellikleridir (Shoham, 1997). (Erdur, 2001)’de bu birincil özellikler temel alınarak diğer tanımlamaları birleştirmeyi amaçlayan genel bir etmen tanımı verilmiştir;

"Etmen, kendisinden beklenenleri yerine getirmek için belli bir ortamda belli derecede özerklik çerçevesinde çalışan, algılayıcıları ile ortamdaki dinamik değişimleri algılayan ve elde ettiği algılara göre bilgisini, amaçlarını yeniden değerlendiren, amaçları doğrultusunda planlama yaparak bu planlara ilişkin eylemleri yapan, diğer etmenler ile bir etmenler arası iletişim dili aracılığı ile iletişimde bulunma yeteneği olan ve bulunduğu ortamda süreklilik gösteren yazılım veya donanım tabanlı sistemdir" (Erdur, 2001).

Bir etmen her şeyi bilen (*"omniscent"*) ve her şeyi yapabilme gücü olan (*"omnipotent"*) bir sistem olmak zorunda değildir. Etmen değişik durumlarda yerel bilgi tabanını, ortamdan elde ettiği algıları ve diğer etmenlerle olan iletişiminden elde ettiği bilgileri kullanarak amaçları doğrultusunda davranışta bulunmaya çalışmaktadır. Bunun için de yapay zeka alanı içerisinde yer alan planlama ve akıl yürütme gibi süreçleri kullanmaktadır.

Etmenler donanım veya yazılım etmenleri olarak sınıflandırılmaktadır. Bu tezde bahsedilen etmenler yazılım tabanlı oldukları için yazılım etmenleridir. Yazılım etmenleri genel olarak bir ortamda sürekli ve özerk bir biçimde işleyiş gösteren ve çoğunlukla diğer etmenler ile bir bütün olarak çalışan yazılım içerikleridir (Shoham, 1997).

2.1.2 Çoklu Etmen Sistemleri

Tek bir etmenin yalnız başına kendi bilgi ve bireysel yeteneklerini kullanarak çözemediği veya etkin bir biçimde çözemeyeceğini düşündüğü problemleri birbiriyle işbirliği yaparak eşgüdümlü bir biçimde çözmek için bir

araya gelen etmenlerin oluşturduğu ağ çoklu etmen sistem - ÇES (*"multi agent system" - MAS*) olarak adlandırılmaktadır (Durfee and Lesser, 1989; Sycara, 1998).

Farklı hızda yada kalitede olan etmenlerin aynı görevi yerine getirmek için toplanmaları bazı sorunlara neden olabilir. Kendisi için çalışan *("Self–Interested"*) etmenler kavramı ve eşgüdümlü dağıtık problem çözme - EDPÇ (*"Coordinated Distrubuted Problem Solving" - CDPS*), öğrenme (*"learning"*) ve görüşme (*"conversation"*) kavramları çoklu etmen sistemlerindeki bu sorunlara çözüm bulmak amacıyla ortaya çıkmışlardır. ÇES araştırmaları, zeki ve tekil olan etmenlerin eşgüdümlü bir şekilde çalışması ile ilişkilidir. Genel olarak bilgilerinin, yeteneklerinin, amaçlarının ve planlarının eşgüdümlenmesinin sağlanmasını amaçlarlar.

Çoklu etmen sistemlere ilişkin araştırmalar dağıtık yapay zeka başlığı altında gerçekleşmektedir. Geleneksel yapay zeka, bireysel etmen mimarilerini konu edinirken, dağıtık yapay zeka çoklu etmen sistemlerinde etmenlerin bilgi, yetenek ve hedeflerinin eşgüdümlü bir biçime getirilmesi üzerinde durmaktadır.

Çoklu etmen sistemleri diğer dağıtık sistemlerden ayıran özellikler şu şekilde listelenebilir;

- Sistemdeki herhangi bir etmende problemin çözümüne ilişkin bilgilerin tamamı bulunmamaktadır,
- Sistemdeki herhangi bir etmende problemin çözümüne ilişkin yeteneklerin tamamı bulunmamaktadır,
- Sistem kontrolü dağıtıktır,

- Veri merkezi olarak tutulmamaktadır, dağıtıktır,
- İşleyiş eşzamanlı değildir.

Çoklu etmen sistemlerinin oluşturduğu organizasyonlar, aralarında etkileşimi sağlamak için etmenlere bir çerçeve sunmaktadır. Bu organizasyonlar etmenlerin problem çözme kapasiteleri ve etmenler arasındaki bilgi ilişkilerinin dağılımının bir desenidir. Açık ortamlarda, etmenler bir organizasyona dinamik olarak girip çıkabildikleri için sistemde var olan etmenleri önceden belirlemek mümkün değildir (Sycara, 1998). Aşağıda çoklu etmen sistem literatüründe geçen organizasyonlar verilmektedir:

- Sıradüzensel: Karar verme ve kontrol yetkisi bir tek etmende toplanmaktadır. Etkileşim dikey olarak yapılmakta ve en baştakinden alttaki etmenlere doğru inmektedir.
- Uzmanlar Topluluğu: Organizasyonda her etmen özel bir konuda uzmandır. Etmenler önceden belirlenen kurallar çerçevesinde etkileşirler.
- Pazar: Kontrol, görevleri anlaşma ve teklif usulü tamamlayan etmenlere dağıtılmıştır. Etmenler birbirleriyle değer servisleri (*"value services"*) için kullanılan fiyat değişkeni aracılığıyla etkileşirler ve fiyatların karşılıklı ayarlanması aracılığıyla koordine olurlar.
- Bilimsel Topluluk: Bu model çoğulcu toplulukların nasıl işlediğini gösteren bir modeldir. Çözümler yerel olarak oluşturulur, daha sonraki aşamada çözümü test edecek, inceleyecek etmenlerle iletişim kurulur.

Çoklu etmen sistemlerde etmenler arası iletişimin sağlanabilmesi için iki temel unsura gereksinim duyulur; Birincisi ortak bir iletişim dilidir. Diğer unsur ise etmenler arasında gidip gelen mesajların içeriğinin tüm etmenler tarafından anlaşılabilmesi için ortak bir sözlük sağlayan ontolojilerdir.

Ontoloji, bir alandaki kavramlar ve bu kavramlar arasındaki ilişkileri makinelerin okuyabileceği bir terminoloji kullanarak mantıksal olarak tanımlayan yapıdır (Daconta et al., 2003). Bir alanda ortak bir sözlük sağlayıp, terimlerin anlamlarını ve bunlar arasındaki ilişkileri tanımlarlar. Etmenler arası iletişimde kullanılan terminoloji üzerinde ortak bir anlayış sağlanması gerekliliği ontolojilerin kullanımını zorunlu kılar.

2.1.3 Etmen Mimarileri

Etmen mimarileri, etmenlerin içsel yapısı ve işleyişine ilişkin olarak aşağıda listelenen unsurları belirlemektedir (Sycara, 1998).

- Etmenlerin karar verme sürecinin türü ve işleyişi,
- Etmenlerin bulunduğu ortama ilişkin bilgilerin gösterimi,
- Etmenin kullandığı veri yapıları,
- Etmenin kullandığı veri yapıları üzerinde uygulanabilecek işlemler,
- Etmenin veri yapıları arasındaki kontrol akışının belirlenmesi.

Literatürde tanıtılmış dört adet etmen mimarisi modeli bulunmaktadır. Bu mimari modelleri; mantık tabanlı mimariler (*"logic based architectures"*), karşıt eylemli mimariler (*"reactive architectures"*), kanı-istek-niyet - KİN

mimarileri (*"belief-desire-intention- BDI architectures"*) ve katmanlı mimarilerdir (*"layered architectures"*). Bu tez kapsamında KİN mimarisini temel alan çoklu etmen sistemleri için bir geliştirim tarzı ve pratikler önerildiğinde bir sonraki alt-bölümde bu mimarinin genel çerçevesi sunulmuştur.

2.1.3.1 Kanı-İstek-Niyet - KİN Mimarisi

İleri seviye bilgi sistemlerinin yönetim karmaşıklığının ve bakım maliyetinin artmasıyla birlikte son yıllarda özerk işletim yaklaşımlarına ve özerk sistemlere ilgi artmıştır. (Morreale et al., 2006)'da yazarlar özerk bir sistem geliştirme yollarında biri olarak aynı işlevsellik ile ilgili olası davranışların bir uzayının tanımlanmasını destekleyen bir tasarım yaklaşımını tartışmaktadırlar. Bu yaklaşıma göre sistem daha önceden tanımlanmış davranışlar içerisinden en uygununu ortam durumuna göre çalışma zamanında seçebilmelidir. Hedefler sistemin özerk olarak uygun davranışı seçmesi yolunda sistemdeki işlevselliklerin modellenmesi için bir soyutlama olarak kullanılabilirler.

Bu yaklaşım tarzında hedeflerin ve bu hedeflerlere ulaşmak için kullanılacak yeteneklerin net olarak gösterilmesi çeşitli gereksinim analizi ve modelleme teknikleri içerisinde önemli bir rol oynar. Hedef soyutlaması özellikle gereksinimleri tasarım zamanında tam ve kesin olarak bilinmeyen yazılım sistemlerinin geliştirilmesi için, özerk birimler ile ilgili soyutlamalar üreten etmen tabanlı paradigmada kullanıldığında son derece faydalı ve uygun olabilir.

En popüler ve başarılı etmen modellerinden biri felsefeci Michael

Bratman tarafından geliştirilen deneysel akıl yürütme (*"practical reasoning"*) kuramına dayanan kanı-istek-niyet - KİN (Rao and Georgeff, 1995) modelidir. Deneysel akıl yürütme iki süreçten oluşmaktadır. Birinci süreç, hangi hedeflere varılmak istendiğinin belirlenmesidir. İkinci süreç ise belirlenen hedeflere nasıl varılacağının ortaya çıkartılmasıdır. Bu modele göre etmenler hedeflerine ulaşmak için hangi eylemleri çalıştıracağına özerk olarak çalışma zamanında karar verirler.

Literatürdeki KİN modeline dayanan Jack (Norling and Ritter, 2001), Jam (Huber, 1999) gibi etmen platformlarının çoğu istek kavramı yerine hedef kavramını kullanmaktadırlar. Decaf (Graham et al., 2003) ve Jade (Bellifemine et al., 2007) platformlarında olduğu gibi yine bu platformların çoğunda niyet kavramına karşılık ise tanımlanan hedefleri başarmak için geliştirilen planlar geliştirilmektedir.

KİN mimarisine dayanan bir etmen temel olarak kanı tabanı (*"belief base"*), bekleyen hedefler (*"goals"*), plan kütüphanesi ve niyet tabanını (*"intention base"*) içermektedir. Kanı tabanı, bir etmenin dünyaya ilişkin bilgisini kodlamaktadır. Plan kütüphanesi, plan kurallarını içermektedir. Niyet tabanında ise o an bir amacı gerçekleştirmek için etmenin kullandığı planlar bulunmaktadır. Etmen, sistemin bir hedefini gerçekleştirebilmek için plan kütüphanesinden bir plan kuralı seçer ve bu kurala ait plan programını niyet tabanına yerleştirerek bekleyen hedefe ulaşılması işini kendisine sevk eder. Bu programın çalıştırılması sırasında, başarılması gereken alt-hedeflerle (*"sub-goals"*) karşılaşılabilir. Herhangi bir anda programın başarılı olamadığı durum söz konusu olduğunda alternatif bir plan kuralı seçilir ve bu plan kuralına ait plan programı işletilmesi için

niyet tabanına eklenir. Bu süreç, plan başarıyla son bulana kadar devam etmektedir.

2.1.4 Planlama ve Sıradüzensel Görev Ağı - SGA

Son zamanlarda planlama üzerine yapılan araştırmalarda başlıca iki konu üzerine odaklanılmıştır. Bunlardan ilki bilgi toplayan, anlayan ve bilgi üreten eylemlerdir. İkincisi ise koşullu dallanmalar, döngüler gibi gelişmiş kontrol yapıları içeren planlardır. Bu iki konu üzerine yapılan çalışmalar bilgi toplayan eylemlerin planın çalışması sırasında bir çok kez çalışabildiği ve eylemler arasında bilgi ve kontrol akışını sağlayan karmaşık bir ilişki ağına sahip olduklarını göstermektedir. Bu karmaşık yapıların oluşturulabilmesi için farklı planlama paradigmaları tanımlanmıştır.

Bu planlama paradigmaları içerisinde RETSINA (Sycara et al., 2003), SEAGENT (Dikenelli et al., 2005b; Dikenelli et al., 2005a) ve DECAF (Graham et al., 2003) gibi iyi bilinen etmen çerçeveleri tarafından yaygın olarak kullanılanlardan biri 1995 yılında Erol ve ark. tarafından tanıtılan sıradüzensel görev ağı - SGA (*"hierarchical task network" - HTN*) (Erol et al., 1995) planlama paradigmasıdır. SGA planlamasında iki görev tipi bulunmaktadır: içerisinde görevlerden oluşan bir yapı tutan karmaşık görevler (tezde davranış olarak adlandırılmaktadırlar) ve çalışabilir koda sahip ilkel görevler (tezde eylem olarak adlandırılmaktadırlar). Temel fikir, karmaşık görevlerin ilkel görevlere ulaşıncaya kadar daha küçük altgörevlere indirgenmesidir.

Bir karmaşık görevin indirgenmesi planlama sisteminin hemen çalıştırabileceği, daha fazla indirgenmesi mümkün olmayan bir görev (ilkel

görev) bulununcaya kadar devam etmektedir. SGA planlamasının temel amacı, verilen kısmi sıralı görevlerin (plan) gerçekleştirilmesidir. Bu da şu anlama gelmektedir; geleneksel planlayıcılar istenen bir duruma ulaşmaya odaklanmaktayken SGA planlayıcılar görevlerin gerçekleştirilmesine odaklanmaktadırlar.

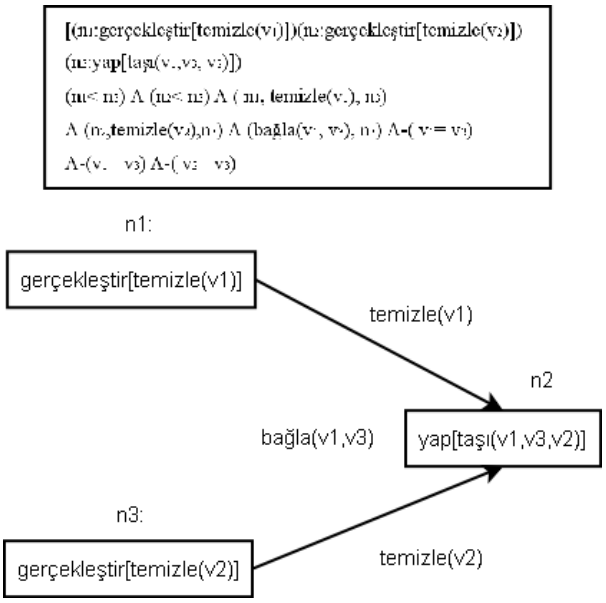
Bir görev ağı işleme konulma zorunluluğu olan görevlerle birlikte onların uygulama sırasını belirleyen kısıtlamaların bütünüdür. Formal olarak bir görev ağı şu sözdizimsel yapıdadır:

$[(n_0 a_0) \ldots\ldots\ldots\ldots\ldots (n_m a_m)\,, \beta]$

Burada a_i bir görevi tanımlamaktadır ve n_i, a_i'nin etiketi olarak a_i'yi ağdaki başka tekrarlanışlarından ayırmaktadır. β ise $(v = v')$ ve $(v = c)$ gibi değişken bağlayıcı kısıtlamalar, $(n < n')$ gibi sıralama kısıtlamaları ve (başlangıç olarak l), $(n, 1)$, $(1, n)$ ve $(n, 1, n')$ gibi durum kısıtlamalarından oluşan mantıksal bir formüldür. Yukarıda v, v' değişkenler, l bir "hazır bilgi" (*"literal"*), c bir sabit, ve n, n' düğüm etiketleri olarak kabul edilmektedirler. Buna göre $(n < n')$ n ile etiketlenen görevin n' ile etiketlenenden önce gelmesi anlamına gelmektedir. $(n, 1)$ ifadesi, n'den sonraki durumda l doğrudur (*"true"*) anlamına gelmektedir. $(1, n)$ ifadesi n'den önceki durumda l doğrudur anlamına gelmektedir. $(n, 1, n')$ ise n'den sonraki ve n''den önce gelen durumlarda l doğrudur anlamına gelmektedir.

Şekil 2.1 görev ağını ve onun grafiksel tanımlamasını göstermektedir.. Bu görev ağında üç tane görev yer almaktadır: $v1$'in temizlenmesi, $v2$'nin temizlenmesi ve $v1$'in $v2$'ye taşınması. Görev ağı ayrıca $v1$'in en son taşınması, $v1$'in taşınmadan önce $v1$ ve $v2$'nin temizlenmesi ve $v1$

taşınmadan önce değişken $v3$'ün $v1$'in yerine bağlanması kısıtlamalarını içermektedir.



Şekil 2.1. Örnek bir görev ağı

Bir görev ağı, sadece tek bir ilkel görev içeren basit bir görev ağı olabilmektedir. Ancak çoğunlukla görev ağları, basit olmayan görevler içeren görev ağları şeklinde bulunmaktadırlar. Basit olmayan görevler doğrudan işlenemeyen görevlerdir, çünkü birden fazla görevin gerçekleştirilmesiyle oluşturulan etkinlikleri temsil etmektedirler. Örneğin İzmir'den Ankara'ya uçak, otobüs ve trenle olmak üzere birkaç farklı yolla seyahat edilebilmektedir. Uçakla Ankara'ya gitmek rezervasyon yapmak, havaalanına gitmek, uçak bileti almak, uçağa binmek görevlerinin gerçek-

leştirilmesini gerektirirken uçmak ancak bilet bulabilme, havaalanında zamanında olabilme, bilet için yeterli paranın bulunması gibi şartların sağlanması ile gerçekleşmektedir.

Basit olmayan görevlerin gerçekleştirilmesi metotlarla temsil edilmektedir. Bir metot (a, d) sözdizimsel formundadır. Burada a basit olmayan görevi, d ise bir görev ağını temsil eder. a'yı başarmak için d'de yer alan tüm görevler hiçbir kısıtlama göz ardı edilmeden gerçekleştirilmelidir.

1) Planlama problemi P'yi gir.
2) Eğer P sadece ilkel görevler içeriyorsa , P'deki çelişkileri çöz ve sonucu döndür. Eğer çelişki çözülemiyorsa hata mesajı döndür.
3) P'de basit olmayan bir t görevini seç.
4) t için bir genişletme seç.
5) t'yi genişlemeyle değiştir.
6) P'deki görevler arasındaki etkileşimi bulmak için kritikleri kullan ve onlarla baş etmek için yollar öner.
7) 6'da önerilen yollardan birini uygula.
8) 2. basamağa git.

Şekil 2.2. SGA algoritması

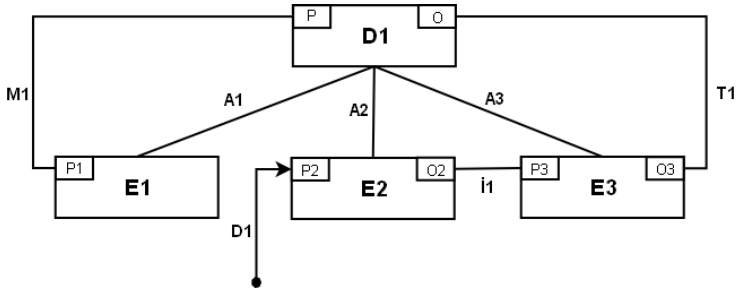
SGA planlama problemi $P = (d, I, D)$ şeklinde bir üçlü olarak gösterilmektedir. Burada d planlamasının yapılacağı görev ağı, I başlangıç durumu ve D'de planlama alanına ait metot ve operatörlerin kümesini temsil etmektedir. SGA planlama için kullanılabilecek algoritma şekil

2.2'te verilmiştir. SGA planlama, ilkel görevler içeren çelişkisiz bir plan bulana kadar görevlerin genişletilmesi ve çelişkilerin yinelemeli olarak çözülmesi ile gerçekleştirilmektedir. Basit olmayan görevlerin her birinin genişletilmesi veya azaltılması (3-5 numaralı basamaklar), basit olmayan bir görevi gerçekleştirebilecek bir metot bulup metot tarafından üretilen görev ağına basit olmayan görevin yerleştirilmesi ile sağlanmaktadır. Beşinci. basamakta üretilen görev ağı görevler arasındaki etkileşimler sonucunda çelişki içerebilmektedir. Böyle etkileşimleri bulma ve çözme işi eleştirilerle yapılmaktadır. Her bir azaltmadan sonra, bir grup kritik kontrol edilir, böylece bu ve diğer azaltmalar arasında etkileşimler tanınır ve çözümlenir. Bu durum altıncı ve yedinci basamaklara yansıtılmıştır.

Williamson ve ark. 1996'da tanıttıkları bir çalışmada (Williamson et al., 1996) mevcut planlama araştırmalarının birbiriyle ilgili iki konuya (birincisi, bilgi-toplama eylemleri; ikincisi ise, döngüler ve koşullu dallanmalar gibi gelişmiş denetim yapılarına sahip planlar) yoğunlaştığına, ancak bu iki araştırma alanının birleştirilmesinin gösterimsel bir probleme yol açtığına dikkat çekmişlerdir. Çalışmada, bilgi üreten eylemler, koşullu dallanmalar, periyodik eylemler ve döngülere sahip sıradüzensel planların gösterimi ve çalıştırılması için bir çerçeve ortaya konmuştur. İndirgenmiş görevi alt-görevleriyle değiştiren diğer bazı SGA planlaması biçimlerinin (Erol et al., 1995) aksine, bu çerçevede görev ağı ağaç benzeri bir yapıda ifade edilmektedir. Ayrıca bu çerçevede *indirgeme* kavramı, yeni alt-görevlerin ihtiyaçlarının ve çıktılarının birbirleriyle ve ebeveyn görevleriyle olan ilişkisini tanımlar. Bu altyapıyı temel alan birçok sistem geliştirilmiştir (Paolucci et al., 1999; Graham et al., 2003).

Bu yaklaşıma göre bir SGA planı görevlerden ve bu görevler arasındaki iki tip bağdan oluşur. İndirgeme bağları karmaşık görevleri daha küçük görevlere indirgemek için kullanılırken, diğer bağ türü olan bilgi akışı bağları görevler arasındaki bilgi geçişlerini göstermektedir.

Bu bölümün devamında Vilyımsın ve ark. tarafından tanıtılan SGA formalizminde kullanılan kavramlar şekil 2.3'de sunulan örnek SGA yapısı kullanılarak açıklanmaktadır.



Şekil 2.3. Örnek SGA yapısı

SGA gösteriminde davranış (karmaşık görev) bir veya birden fazla alt-görevi içeren bir görevdir. Örnek plan yapısında *D1* bir davranıştır. Doğrudan çalıştırılabilir görevler eylemler (ilkel görevler) olarak adlandırılır. Örnek planda *E1, E2* ve *E3* eylemleri bulunmaktır. Bir ihtiyaç (*"provision"*) görevlerin bilgi gereksinimlerini gösterir ve çalışma zamanı değişkeni veya plana etmen tarafından başlangıçta verilen bir parametre olabilir. Örnek planımızda *D1* görevi *P* ihtiyacına, *E1* görevi *P1* ihtiyacına, *E2* görevi *P2* ihtiyacına ve *T3* görevi *P3* ihtiyacına sahiptir. Bununla birlikte, *P2* diğer bir etmenden mesaj yolu ile elde edilebilecek bir bilgiyi

simgeleyen bir dışsal ihtiyaçtır. Sonuç durumu (*"outcome"*), bir görevin ne şekilde bittiğini gösteren bir durumdur ve görevin bu durum karşısında ürettiği sonuçları döndürür. Şekil 2.3'deki örnek planımızda *O, O2* ve *O3* sonuç durumları bulunmaktadır. *A1, A2* ve *A3* bağları *D1* görevinin daha küçük görevlere indirgenmesini temsil eden indirgeme bağlarıdır.

Dört tip bilgi akış bağı bulunmaktadır; ihtiyaç-sonuç durumu bağları, dışsal ihtiyaç bağları, kalıtım bağları ve "ters kalıtım" bağları.

Her bir ihtiyaç-sonuç durumu bağı $\left(G_\alpha, o_{G_\alpha}, G_\beta, p_{G_\beta}\right)$ formunda gösterilmektedir. Bu ifadede G_α ve G_β aynı seviyedeki iki görevdir. Bu bağ gösterimi G_α'nın işletilmesi sonucunda oluşan o_{G_α} sonucunun G_β görevinin p_{G_β} ihtiyacını sağlayacağı anlamına gelmektedir. Örnek planımızda *E2* görevinin *O2* sonuç durumu ile *E3* görevinin *P3* ihtiyacı arasında *İ1* isimli bir ihtiyaç-sonuç durumu bağlantısı bulunmaktadır.

Dışsal ihtiyaç bağları bir göreve plan dışından bilgi geçirilmesini sağlarlar. Etmen sistemlerde dışarıdan beklenen bilgiler diğer etmenlerden mesaj yoluyla elde edilir. Örnek planda *D1, E2* eyleminin sahip olduğu *P2* ihtiyacını sağlamak için kullanılan bir dışsal ihtiyaç bağıdır.

Bir kalıtım bağı (D, p_D, G, p_G) şeklinde ifade edilir. Bu ifadedeki G görevi D davranışının bir alt-görevini temsil etmektedir. p_D D davranışının, p_G ise G görevinin ihtiyaçlarıdır. İfade p_D'yi sağlayan her değerin p_G'ye geçirileceği anlamına gelmektedir. Örnek plan yapısında *D1* davranışının *P* ihtiyacı ile *E1* eyleminin *P1* ihtiyacı arasında *M1* kalıtım bağı bulunmaktadır.

Benzer olarak, "ters kalıtım" bağlantısı (G, o_G, D, o_D) ifadesiyle temsil edilir. Bu ifadede yine G görevi D davranışının bir alt-görevini

temsil etmektedir. o_G G görevinin, O_D ise D davranışının birer sonuç durumudur. Bu ifade o_G sonuç durumu oluşması durumunda bu sonuç durumunun içerdiği değerlerin o_D'ye geçirileceği ve D davranışının o_D ile biteceği anlamına gelmektedir. Şekil 2.3'deki plan yapısında *E3* eyleminin *O3* sonuç durumu ile *D1* davranışının *O* sonuç durumu arasında *T1* "ters kalıtım" bağı bulunmaktadır.

2.2 Etmen Sistem Geliştirim Yöntemleri

ÇES geliştirim yöntemlerinin tanımlanması EYYM alanı içerisinde önemli konulardan biridir. Son yıllarda ÇES geliştirimine rehberlik edecek etmen tabanlı yöntemlerin geliştirilmesi için büyük bir çaba harcanmıştır. Bu çaba sonucunda pek çok etmen yönelimli yöntem tanımlanmıştır. Bu bölümde, ÇES geliştirimi için literatürde tanıtılmış yöntemlerinin en yaygın olarak kullanılanlarından bazıları şelale benzeri yöntemler ve evrimsel yöntemler adında iki ana sınıfa ayrılarak incelenmektedir.

2.2.1 Şelale Benzeri Yöntemler

2.2.1.1 GAIA

Gaia (Wooldridge et al., 2000; Zambonelli et al., 2003), etmen sistem geliştirimini analiz ve tasarım aşamalarında destekleyen ve ETYG alanında yaygın şekilde kullanılan geniş kapsamlı bir yöntemdir. Geleneksel şelale süreç modeli üzerine geliştirilmesi nedeniyle, tüm süreç boyunca katı kurallarla tanımlanmış modeller elde etmeyi amaçlar.

Analiz aşaması üç adımda incelenmektedir. Birinci adımda sis-

temdeki roller belirlenir. Bu amaç için, her biri biçimsel olmayan tanımlamaları içeren anahtar rollerin bulunduğu rol modeli kullanılır. İkinci adımda her rol ile ilişkili bir protokol seti tanımlanır. Protokoller sistem içinde tanımlanmış roller arasındaki etkileşim kalıplarıdır. Bu adımın sonucunda roller arası etkileşimin gösterildiği etkileşim modeli oluşturulur. Üçüncü adımda protokoller kullanılarak rol modeli detaylandırılır. Detaylı rol modelde, sistemdeki her anahtar rol kısıt, yetki ve görevleri cinsinden tanımlanmaktadır.

Gaia'nın tasarım aşaması da üç adımdan oluşmaktadır. Birinci adımda, geliştirilmekte olan sistem için bir etmen modeli oluşturulur. Bu modelde roller etmen tipleri ile eşleştirilerek etmen tipi hiyerarşisi sağlanır. İkinci adımda, rollerin protokol, etkinlik ve güvenlik özellikleri incelenerek servis modeli geliştirilir. Son olarak, etkileşim ve etmen modelleri kullanılarak bilgi modeli oluşturulur. Bu model etmen tipleri arasındaki iletişim bağlantılarını gösterir.

2.2.1.2 Multiagent Systems Engineering (MaSE)

MaSE (DeLoach S. A., 1999) yönteminin hedefi, ÇES geliştiricilerine sistem tanımlamalarından sistemin gerçekleştirimine kadar tüm süreci kapsayacak şekilde yol göstermektir. Süreç, iki ana aşamaya ayrılan yedi adımdan oluşur.

MaSE analiz aşaması, ilk olarak hedeflerin elde edilmesi yönünde analistin hedef ve yapıları belirlemesinden, bunları sıradüzensel sırasına göre tanımlamasına kadar rehberlik eder. İkinci aşamada, daha sonra sıralı diyagramların oluşturulmasında kullanılacak olan kullanım senaryoları

netleştirilir. Analiz aşamasının son adımı rollerin netleştirilmesini içerir. Bu aşamada rol modeli ve koşut zamanlı görev modelleri oluşturulur. Rol modeli sistemdeki görevleri tanımlar. Aynı zamanda hedefleri rollerle paylaştırarak, hangi rolün hangi hedefe karşılık geldiğini, her rolün sorumlu olduğu görevleri ve roller arasındaki iletişim yollarını tespit eder. Daha sonra görevler grafiksel olarak detaylandırılır.

Tasarım aşamasının ilk adımında etmen sınıfları oluşturulur. Bu adımın sonunda tüm ÇES'i tanımlayan etmen sınıf diyagramı ortaya çıkar. Bu diyagram etmenleri ve bu etmenlerin üstlendikleri rolleri gösterir. Etmen sınıfları arasındaki bağlantılar etkileşimi gösterir ve ismi ile etiketlendirilirler. Bu etkileşimlerin detaylandırılması, tasarım aşamasının ikinci adımında gerçekleşir. Üçüncü adımda etmenlerin sahip olduğu mimari ve bu mimarinin bileşenleri tanımlanır. Son olarak sistem tasarımı adımında etmenlerin sistem içindeki konumları netleştirilir.

2.2.1.3 Prometheus

Prometheus (Padgham and Winikoff, 2002) yöntemi KİN platformları için etmenlerin yapılandırılması ve kısıtlı etmen geliştirim deneyimi olan uygulamacılar için detaylı bir yol haritası sağlamak üzerine odaklanmıştır.

Yaratıcılarına göre Prometheus çoklu etmen sistemlerinin tanımlanması, tasarımı ve gerçekleştirimi için detaylı ve eksiksiz bir süreçtir. Yöntem üç aşamadan oluşmaktadır: sistem tanımlama, mimari tasarım ve detaylı tasarım.

Sistem tanımlama aşaması sistem hedeflerinin (bu hedefler bir veya birden fazla işlevsellik ile sonuçlanmalıdır) bir senaryo seti olarak

modellenmesini içerir. Mimari tasarım aşaması etmenlerin, sistem genel bakışının ve etmenler arası protokollerin etmenler için birleştirilmiş modelleme dili - EBMD (*"Agent UML" - AUML*) (Huget, 2004) kullanılarak modellenmesini içerir. Detaylı tasarım aşaması ise her etmenin içsel yapısının yetenekleri doğrultusunda geliştirilmesi üzerine odaklanmıştır. Her yetenek olay, veri ve plan tanımlayıcılarının bir seti ile tanımlanmaktadır. Prometheus aynı zamanda, geliştiricilerin büyük ölçekli sistemler tasarlamasını kolaylaştıran çoklu soyutlama seviyeleri modelleyebilmesi için sıradüzensel bir mekanizma sunar.

Bunlara ek olarak aşağıdaki özelliklere sahiptir;

- Yöntem, Jack geliştirim ortamı - JGO (*"Jack Development Environment"* - JDE) ve Prometheus tasarım aracı ile desteklenmektedir. Bu iki araç yöntemin tasarım aşamasının sonuçlarının gerçekleştirilmesi sırasında kullanılabilmektedir.

- Test etme ve hata ayıklama etkinlikleri için bir kılavuz tanımlanmıştır.

- Gereksinimlerin elenmesi ile ilgili bazı etkinlikler analiz aşamasında kısmen bulunmaktadır (gereksinim analizi analiz şemasının bir bölümü olarak sunulmaktadır). Bu yüzden, Prometheus yönteminin analiz ve tasarımın yanında gereksinim eleme, kodlama ve test etme aktivitelerini de kapsadığı söylenebilir.

Prometheus yöntemi tarafından tanımlanan süreç, gereksinimlerin elenmesinden test etmeye kadar sıralı şekilde yürütülen tamamen doğrusal bir süreçtir. Süreç içerisinde geri bildirimlerin geliştirime katkısı kısıtlıdır. Bu

durum sürecin geliştirim aşamalarının mimari tasarım ve detaylı tasarım modelleri arasındaki yeniden gözden geçirmeler dışına çıkamamasına neden olmaktadır. Bu nedenden dolayı, Prometheus şelale tabanlı yöntemlerin bilinen sınırlamalarına sahiptir.

2.2.1.4 Şelale Süreç Modeli Kullanan Yöntemler için Değerlendirme

Yazılım sistemlerinin geliştirilmesi için disiplinli bir süreç öneren ilk model olması nedeniyle, şelale modelinin yazılım mümendisliği tarihinde önemli bir yeri bulunmaktadır. Bununla birlikte, şelale modeli esnek olmayan, sert bir karakteristiğe sahiptir. Bu model içerisinde geri bildirimlerin maliyeti oldukça yüksektir. Gereksinimleri sürekli değişen yazılım sistemlerinin geliştirim süreci boyunca elde edilen geri bildirimleri değerlendirmeyen, doğrusal bir model ve sıkı kurallar dizisini kullanarak geliştirilmesi maliyetlidir. Bir sistem çok durağan ve belirgin gereksinimlere sahip olmadıkça ve kapalı, durağan bir ortamda çalışmak üzere tasarlanmıyorsa süreçte daha fazla esneklik ve adapte olabilme özelliklerine gereksinim duyulur. ÇES uygulamalarının büyük çoğunluğu çok sayıda etmen tarafından oluşturulmuş ve gereksinimleri sürekli değişen dinamik bir ortam içerisinde çalışan çoklu etmen sistemleri üzerine odaklanır. Bu yüzden, şelale modelinin ÇES geliştirimi için iki temel kısıtlaması vardır. Bu kısıtlamalar; gereksinimlerdeki değişiklikleri önceden tahmin etmeye çalışması ve birçok farklı koşul içerisinde olabilecek bu değişikliklere karşı yazılım evrimini önceden planlamaya çalışmasıdır.

2.2.2 Evrimsel Yöntemler

2.2.2.1 Tropos

Tropos (Bresciani et al., 2004) Kanada ve İtalya'da bulunan çeşitli üniversitelerden araştırmacılar tarafından ortaya atılan bir etmen yönelimli yazılım geliştirim yöntemidir. Tropos'u diğer yöntemlerden ayıran en önemli faktör, alan aktörlerinin ve bu aktörlerin hedeflerinin belirlendiği erken gereksinim analizi aşamasıdır. Tropos'un yazılım geliştirim süreci erken gereksinim analizi, ileri gereksinim analizi, mimari tasarım, detaylı tasarım ve gerçekleştirim olmak üzere beş aşama içerir.

Erken analiz aşamasında, yukarıda söz edildiği gibi aktörler ve bu aktörlerin hedefleri belirlenir. Tropos hedefleri işlevsel ve işlevsel olmayan gereksinimler olmak üzere ikiye ayırır. Bu aşamada iki model kullanır. İlk model, aktörleri ve aralarındaki ilişkileri ortaya koyan, hedeflere ulaşmak üzere aktörler arasındaki ilişkileri tanımlayan aktör diyagramıdır. Diğer model ise, hedeflerin analizini ve bunları gerçeklemekle sorumlu olan aktörlerin tanımlandığı hedef diyagramıdır.

İleri gereksinim aşamasında, önceki aşamada oluşturulan modeller genişletilir. Bu aşamanın önemi sistemin kendi ortamında modellenmesidir. Geliştirilen sistem bir aktör olarak betimlenir ve bu aktörün diğer aktörlerle olan bağımlılığı sistemin ortamına olan bağımlılığını işaret eder. Ayrıca sistemin kendi ortamı içerisinde bulunan aktörlerden beklentisi netleşir. Bu aşamada hedefler alt-hedeflere ayrıştırılır.

Mimari tasarım aşaması üç adıma ayrılır. İlk adımda yeni aktörler tanımlanır ve genişletilmiş aktör diyagramına eklenir. Yeni aktörler mimari

seçime bağlı olarak ortaya çıkmış olabileceği gibi alt-hedefleri desteklemek amacıyla da eklenebilir. İkinci adımda her aktörün yetenekleri ve kapsamı tanımlanırken, üçüncü adım bu yeteneklerin gruplanması ve etmen tiplerinin oluşturulmasını içerir.

Detaylı tasarım aşaması etmen tanımlamalarının detaylı bir biçimde oluşturulmasını gerektirir. Bu aşamada tasarımcı, yeteneklerin, etmen planlarının ve etkileşimlerinin tanımlanabilmesi için üç çeşit diyagrama gereksinim duyar. Tropos yeteneklerin ve etmen planlarının gösterimi için birleştirilmiş modelleme dili - BMD (*"unified modeling language" - UML*) etkinlik diyagramları kullanır. Etmenler arası etkileşimlerin gösterimi için ise EBMD etkileşim diyagramları kullanır.

Tropos, gerçekleştirim seviyesi desteği için bir kanı-istek-niyet platformu olan JACK (Norling and Ritter, 2001) platformunu seçmiştir. JACK etmenler, yetenekler, veritabanı ilişkileri, olaylar ve planları içeren beş ana dil yapısı sunar. Bu aşamada, geliştirici tasarım aşamasındaki her kavramı bu beş yapıdan birisi ile eşleştirmek durumundadır. Tropos, bu kavramların KİN kavramlarına ve KİN kavramlarının JACK yapılarına eşleştirilmesinde yol gösteren birkaç yönteme sahiptir.

2.2.2.2 Passi

Passi (Cossentino et al., 2003) gereksinim analizinden sistem gerçekleştirimi ve yükleme aşamalarına kadar tüm fazları kapsayan bir çoklu etmen geliştirim yöntemidir. Sistem tasarımı artışlı arıtma (*"incremental refining"*) ile geliştirim tekniğinin gereksinimlerini karşılayabilecek beş sıralı aşama ile sağlanmaktadır.

Passi yazılım geliştiriminin şu fazlarını içermektedir; (i) Sistemin gerektirdiği işlevselliklerin bir kullanım durumu tabanlı tanımlama yöntemi ile tanımlanarak sistem gereksinimlerinin üretilmesi ve bu ürünlerin etmen paradigmasına başlangıç olarak ayrıştırılması. (ii) Etmenler arası etkileşimlerin ve bağımlılıkların bir modelinin oluşturulmasını hedefleyen bir analiz fazı (etmen toplumu). Bu aşama, alan ontolojilerinin belirlenmesi ve etmenlerin rolleri arasındaki iletişimlerin tanımlanmasını içermektedir. (iii) Gerekli etmenler, sınıflar ve metotlar doğrultusunda bir çözüm mimarisinin modellenmesini amaçlayan bir tasarım aşaması olan etmen gerçekleştirim aşaması. Bu aşama tüm sistem için bir davranış tanımlamasını ve bir yapısal tanımlamayı içerir. (iv) Kodlama aşaması çözümün kod seviyesinde modellenmesini amaçlar. Bu aşama desenlerin yeniden kullanılması ve otomatik kod üretimi ile desteklenmektedir. (v) Yükleme, sistem parçalarının dağıtımının modellenmesi ile dağıtık bir platform oluşturulmasını amaçlamaktadır. Passi aynı zamanda iki farklı aşamaya bölünmüş bir test etme yaklaşımının tanımlamasının da içerir. Bu aşamalar; her bir etmenin gerçekleştiriminden sonra test edildiği etmen test etme ve ÇES'in yüklemeden sonra test edildiği topluluk test etme aşamalarıdır.

2.2.2.3 Evrimsel Süreç Modelleri Kullanan Yöntemleri için Değerlendirme

Süreç modelleri için esnek yaklaşımlar, diğer adıyla evrimsel yaklaşımların gereksinimi literatürde geniş çapta kabul edilmektedir (Gomez-sanz and Pavon, 2004; Cernuzzi et al., 2005; Zambonelli and Omicini, 2004). Bununla birlikte evrimsel yöntemler kategorisi içerisinde yer alan Passi

(Cossentino et al., 2003), Massive (Lind, 2001) ve Tropos (Bresciani et al., 2004) yöntemleri de şelale modeline benzer bir akış takip etmektediler. Bu yöntemlerin her biri analiz ve tasarım fazlarındaki kararların geliştirilmesi ile geliştiricilere artışlı bir yol sunmaktadırlar. Her ne kadar bu süreçlerin artışlı ve yinelemeli yapısı gereksinimlerin elenmesi, analiz ve tasarım aşamalarında kullanılabilse de gerçekleştirim aşaması bu döngünün dışında tutulmuştur. Bu yöntemler gereksinimlerin durağan olduğu, sürecin ilk aşamalarında tam olarak tanımlanabildiği ve bu gereksinimler için sabit tasarım kararları tanımlanmadan önce artışlı bir sürecin takip edilmesi gereken çoklu etmen sistemlerinin geliştirilmesi için uygundur. Bununla birlikte, çabuk prototipleme ve hızlı dağıtım gerektiren projeler için uygun değildirler.

2.2.3 Etmen Tabanlı Yöntemlerin Evrimsel Geliştirim Açısından Değerlendirilmesi

Var olan etmen sistem geliştirim yöntemlerini karşılaştırma konusunda literatürde (Dam and Winikoff, 2003; Gomez-sanz and Pavon, 2004; Henderson-Sellers, 2005) gibi birçok çalışma bulunmaktadır. Bu çalışmalar yöntemlerin hepsinin özerklik, akıllı davranma ve tepki verme gibi etmen yönelimli kavramlar için akılcı bir destek sunduğunu belirtmektedirler. Yazılım geliştirim yaşam döngüsü bakış açısından bakıldığında, tüm yöntemler gereksinimlerin belirlenmesi, mimarisel tasarım ve detaylı tasarım aşamalarını yerine getirmektedirler. Güncel yöntemlerin bu aşamalarının etmen sistemler için olumlu tarafları vardır; yöntemlerin analiz ve tasarım aşamaları iyi tanımlanmıştır ve faydalı örneklerle desteklen-

Tablo 2.1. ÇES geliştirim yöntemlerinin sunduğu destekler

	Metodolojiler	Gereksinim Eleme	Gereksinim Analizi	Tasarım	Gerçekleştirim	Doğrulama ve Test Etme	Yükleme
Şelale Akışı	Gaia		X	X			
	Roadmap	X (kısmen)	X	X			
	Prometheus	X (kısmen)	X	X	X	X	
	MaSE	X (kısmen)	X	X	X	X (kısmen)	
	AOR		X	X	X		
Evrimsel ve Artırımlı	OPM/MAS	X	X	X			X
	MASSIVE		X	X	X	X	X
	Ingenias		X	X	X		
	Tropos	X	X	X	X		
	Passi ve Agile Passi		X	X	X	X	X

miştir. Bu, geliştiriciye nesne yönelimli düşünmeden etmen yönelimli düşünmeye geçişte yardımcı olur.

Tüm etmen geliştirim yöntemlerinin desteklerine genel olarak bakıldığında Tablo 2.1'deki tablo ortaya çıkmaktadır. Bu tabloda görüldüğü gibi iki sınıftaki yöntemlerin çoğu analiz ve tasarım sonucunda elde edilen modellerin gerçekleştirimin kolaylaştırmak üzere gerçekleştirim destekleri sunmaktadırlar. Fakat, gerçekleştirim aşaması bu yöntemlerin hiçbirisi tarafından tam olarak desteklenmemektedir. Örneğin; Tropos yönteminde tasarım aşaması sırasında geliştirilen kavramların JACK çoklu etmen sistem çerçevesi kavramları ile eşleştirilmesinden kısaca bahsedilmesine rağmen detaylı bir süreç, örnek veya bu konuların bir tartışmasını sunulmamıştır. Bununla birlikte, evrimsel geliştirim süreci kullanan Passi ve Tropos gibi yöntemlerde de bu gerçekleştirim destekleri daha önceki aşamalarda tam olarak belirginleşmiş modeller üzerinedir. Gerçekleştirim aşaması evrimsel geliştirimin dışında tutulmuştur. Diğer taraftan, evrimsel geliştirim için önemli olan test aşamasını üzerine çok fazla odaklanılmamıştır.

Çoklu etmen sistemleri gibi dinamik sistemlerin geliştirmi için artışlı ve yinelemeli geliştirim tarzlarını içeren evrimsel bir geliştirim yaklaşımı son derece uygundur. Fakat, gerçekleştirim aşamasının bu artışlı ve yinelemeli geliştirim yaşam döngüsüne dâhil etmeyen evrimsel süreçler, gereksinimleri dinamik olarak değişen çoklu etmen sistemlerinin geliştirimi için yeterli değildir. Bu yüzden, ÇES geliştirimi için gerçekleştirim aşamasını da evrimsel geliştirim süreci içerisine alan çevik yaklaşımlara gereksinim duyulduğı son derece açıktır.

2.3 Yazılım Geliştiriminde Çevik Pratikler

Gereksinimlerin dinamik olarak değiştiği alanları modellemek için geleneksel yaklaşımlardan farklı bir geliştirim sürecine duyulan gereksinim yazılım mühendisliği konusunda çalışan araştırmacılar tarafından fark edilmiştir. 2001 yılında Beck ve ark. tarafından yazılımların kontrolünün zor olduğunu kabul eden çevik yazılım geliştirim yaklaşımını tanıtan çevik manifesto [1] yayınlanmıştır. Çevik yazılım geliştirim yaklaşımı yazılımların kontrol edilebilir bir yol içerisinde, yinelemeli ve artışlı bir şekilde geliştirilmesini sağlamak için aşağıda sıralanan dört önemli değer tanıtmaktadır:

- Süreçler ve araçlar yerine bireyler ve etkileşimler,
- Kapsamlı dokümantasyon yerine çalışan kod,
- Sözleşme pazarlığı yerine müşteri ile işbirliği,
- Bir plan takip etmek yerine değişikliklerin karşılanabilmesi.

[1] http://agilemanifesto.org/ son erişim: 03-Ağustos-2008

Çevik geliştirim yaklaşımı bu değerler doğrultusunda geleneksel yazılım geliştiriminden farklı olarak etkileşim odaklı, daha esnek bir süreç modeli önermektedir. Yazılım sistemlerinde gereksinimlerin değişiminin kaçınılmaz olduğunu kabul eder. Bu yüzden, her şeyin başlangıçta kapsamlı olarak planlanması yerine geliştirim süreci boyunca etkileşimlere önem vererek daha küçük planlar yapılması gerektiğini savunur. Bu değerlerin üzerine odaklanılabilmesi için bazı temel prensipler sunmaktadır. Bu prensiplerin en önemlileri aşağıda sıralanmıştır:

- Hızlı geri bildirim,
- Basit düşünme,
- Artışlı değişimler,
- Değişikliklerin kabul edilmesi,
- Kaliteli iş.

Çevik yaklaşıma göre yazılım geliştirimi sırasında değişiklikler mutlaka olacaktır. Bu yüzden, bu değişimlerle başa çıkabilecek bir yol takip edilmelidir. Geliştirim sürecinin başlangıcında kapsamlı bir planlama yerine basit düşünerek basit bir tasarım ile yola çıkılır. Hızlı geri bildirimlerle müşteri ve geliştiriciler sürekli etkileşim içerisindedir. Bu şekilde, sürekli olarak projenin gidişatı kontrol altında tutularak değişimler artışlı olarak tasarıma dahil edilir.

Çevik yaklaşımın önerdiği değerleri temel alarak bazı hafif (*"light-weight"*) pratikler ile çevik yaklaşımın prensipleri uygulamak için geliştirilmiş SCRUM (Schwaber and Beedle, 2001), Cyristal (Cockburn, 2004)

ve uç programlama - UP (*"Extreme Programming" - XP*) (Beck and Andres, 2004) gibi birçok çevik yazılım geliştirim süreci tanıtılmıştır. Bu süreçlerin her biri çevik yaklaşımın prensiplerini yazılım geliştirimine uygulamak için bazı hafif pratikler sunmaktadır. Yazılım mühendislerinin projenin daha küçük birimleri üzerinde odaklanmasını sağlayarak riski en aza indirmeyi amaçlarlar. Bu süreçler içerisinde UP en yaygın kullanılmakta olanlardan biridir.

2.3.1 Uç Programlama - UP (*"Extreme Programming" - XP*)

Uç programlama (Beck and Andres, 2004) 1999 yılında Beck tarafından hafif, etkili, düşük riskli, öngörülebilir, bilimsel ve sıkıcı olmayan bir yazılım geliştirme yolu olarak tanıtılmıştır. Diğer süreçlerden aşağıdaki özellikleri ile ayrılır:

- Erken, yoğun ve sürekli geribildirimler,
- Projenin tüm yaşam süreci boyunca elde edilmesi beklenen tüm planları içeren arttırımlı planlama yaklaşımı,
- İş hayatının değişken isteklerine karşı esnek fonksiyon tanımlama planlaması,
- Sistemin evrimini gösteren, hataları erken yakalamaya imkan veren, müşterilerin projenin gelişim aşamasını izleten programcılar tarafından yazılmış testler,
- Sistem yapısıyla sözlü olarak, testle ve kaynak kodla iletişim imkanı,

- Proje sürdükçe süren evrimsel tasarım süreci,
- Programcıların sıkı işbirliğine olan katkısı.

UP, çevik yazılım geliştirme yaklaşımından gelen değerler doğrultusunda bu yaklaşımın temel prensiplerini yerine getirebilmek için on iki tane pratik tanıtmaktadır:

- Planlama oyunu: Bir sonraki sürümün kapsamına ticari öncelikleri göz önüne alarak hızlıca karar verilir.
- Küçük sürümler: Hızlıca üretilebilecek basit sistemler planlanır. Daha sonra yeni sürüm çok kısa bir sürede üretilir.
- Metafor: Tüm sistemin nasıl çalışacağının basit bir tanımı ile geliştirime yol gösterir.
- Basit tasarım: Sistem mümkün olduğunca basit tasarlanmalıdır. Gereksiz karmaşıklık farkedildiğinde hemen ortadan kaldırılmalıdır.
- Test güdümlü geliştirim - TGG: Programcılar sürekli olarak sistem güvenilirliğini garanti altına alan birim testleri yazarlar. Müşteriler de istenilen özelliklerin bittiğini kanıtlayan testleri yazarlar.
- Yeniden yapılandırma: Programcılar tekrarlamaların ortadan kaldırılması, iletişimin güçlenmesi, basitlik veya esneklik eklemek için sistemi dışsal davranışlarını değiştirmeden yeniden yapılandırırlar.
- Eşli programlama: Tüm kod bir makinede çalışan ikili programcı grupları ile yazılır.

- Kolektif sahiplik: Kodun herhangi bir kısmına katkıda bulanma fırsatı bulan herhangi biri bunu herhangi bir zamanda yapabilir.
- Müşteriyle etkileşim: Gerçek bir müşteri takımla beraber olmalı, sorulara cevap verebilmeli ve anlaşmazlıkları çözebilmelidir..
- Sürekli bütünleştirme: Bir gelişimden az bir süre sonra kod bütünleştirilir ve test edilir.
- Haftada 40 saat çalışma süresi: Bir haftada 40 saatin üzerinde çalışmamak bir kuraldır. Bu aşıldığında ikinci haftada verimin düştüğü savunulmaktadır.
- Kodlama standartları: Programcılar belirli kurallara bağlı kalarak kod yazarlar. Bu, tüm grubun kod üzerinde iletişim kurabilmesini sağlar.

Bu pratikler işbirliği içerisinde çalışarak artışlı ve yinelemeli geliştirim tarzlarını içeren evrimsel bir geliştirim sürecini oluştururlar. Evrimsel geliştirim için tüm pratiklerin önemi olmakla birlikte, bu pratiklerden bazıları evrimsel geliştirim için odak noktadadır. Test güdümlü geliştirim ve yeniden yapılandırma evrimsel geliştirim yaklaşımını oluşturan en önemli iki pratiktir. Diğer pratikler bu iki pratiğin çevresine yerleştirilmiştir ve bu iki pratiğin uygulanabilmesini kolaylaştırmayı amaçlarlar. Tezde çoklu etmen sistemlerinin evrimsel geliştirilmesini sağlamak için bu iki pratik üzerinde çalışılmıştır. Bu yüzden, devem eden iki alt-bölümde test güdümlü geliştirim ve yeniden yapılandırma pratikleri detaylı olarak anlatılmaktadır.

2.3.2 Test Güdümlü Geliştirim - TGG

Geleneksel yazılım geliştirme yaklaşımlarında programcıların herhangi bir geribildirim aracı olmaksızın genel bir hedef ile baş başa kalmaları yazılım projelerinde bazı sorunları beraberinde getirmektedir. En sık karşılaşılan sorun yazılım içerisinde gözden kaçmış hatalardır. Yazılımın giderek büyümesi ve karmaşıklaşması sonucunda hataların saptanması güçleşir. Bununla birlikte, değişen gereksinimlerin yazılıma yansıtılması yazılımın yönetilebilirliğini zayıflatmaktadır. Projede görevli personelin değişmesi yeni sorunları da beraberinde getirir. Tüm bu sorunlar projenin zaman hedefinden sapması, sistemin hayata geçirilmesinden sonraki aşamada gereksinimleri karşılayamaması yada proje hayat bulamadan vazgeçilmesi gibi başarısızlıkları beraberinde getirmektedir.

Bu noktada testin önemi tartışılmazdır. Test kodu, bir modülün doğru çalıştığından emin olmak için yazılan programdır (Beck and Andres, 2004). Ancak geleneksel test yaklaşımları ile ilgili aşağıda sıralanan sorunlar göz ardı edilemez:

- Testler yeterince kapsamlı değilse ortaya çıkarılamayan hatalar kullanım aşamasında problem çıkarabilmektedir.
- Geleneksel yöntemlerde test genellikle kodlama tamamen bittikten sonra gerçekleştirilir. Programcının yazdığı kod üzerine odaklanmadığı bu süreçte, testler ile ortaya çıkarılabilen hataları düzeltmek büyük miktarda emek ve zaman gerektirmektedir.
- Testi yazan kişi ile programı yazan kişi birbirinden farklıysa, testi yazan kişi kod üzerindeki tüm detayları algılayamayabilir. Bu da

bazı önemli testlerin göz ardı edilmesine yol açabilmektedir.

- Testler otomatikleşmiş değilse, yeterince sık, düzenli ve tam olarak gereksinim duyulan şekilde çalıştırılamayabilir. Bununla birlikte, eklenen her yeni unsur testler ile eş zamanlı ilerleyemeyeceğinden, değişikler üzerindeki potansiyel hataları yakalayabilecek test koşulları göz ardı edilebilir.

Test güdümlü programlama - TGP (Beck and Andres, 2004; Link and Frolich, 2003) bu problemlere çözüm sağlamaktadır. Test kodunun amacı, çalışan programlar üzerinde yapılan değişikliklerin programın daha önce çalışan kısımlarında herhangi bir hataya sebep olup olmadığını ve istenen çıktıları verip vermediğini kontrol etmektir (Beck and Andres, 2004). Test kodu ile, programda sadece sonuca odaklanılarak yapılan testlerin yanında, her türlü test senaryosu ile daha kapsamlı testlerin yapılması mümkün olmaktadır.

Bir yazılım geliştirim yöntemi olarak test güdümlü programlama - TGP, uç programlamanın en önemli unsurlarından birisidir. Test güdümlü programlamanın altında yatan temel fikir, yetersiz test edilen yazılımın kalitesindeki düşüş ve buna bağlı olarak başarısızlıkla sonuçlanan yazılım projelerinin varlığıdır. Ancak Kent Beck bu problemin sadece yazılım hatalarından kaynaklanmadığını ortaya atmıştır. Problem daha çok yazılım geliştirme etkinliği için esas alınan yöntemde yatmaktadır (Massol and Husted, 2003).

TGP geleneksel yazılım geliştirim yöntemlerine alternatif bir yazılım geliştirme tarzıdır. Temel olarak işlevsel koddan önce ilgili birim için test koşullarının tasarlanması ve gerçekleştirilmesi fikrine dayanır.

Test koşullarının gerçekleştiriminden sonra bu testi başarılı olarak çalıştıracak işlevsel kod gerçekleştirilir. Yazılım geliştirici önce testleri, daha sonra bu testleri çalıştıracak fonksiyonları kodlayarak ilgili program birimine işlevler ekler. Bu sırada oluşturulan kodun tasarımını basit ve anlaşılır tutmak amacıyla, kod üzerinde gerekli biçimde yeniden yapılandırma işlemi gerçekleştirmek ve testleri yeniden çalıştırmak gereklidir.

Farklı programlama paradigmaları için geliştirilmiş JUnit (Massol and Husted, 2003; Link and Frolich, 2003) dbUnit (Haftmann et al., 2007), HttpUnit (Tappenden et al., 2005) gibi bir çok test çerçevesi bulunmaktadır (Hamill, 2004). Genel olarak xUnit (Meszaros, 2007) olarak adlandırılan bu çerçevelerin tümü benzer biçimde çalışır; test metotlarını toplar, otomatik olarak çalıştırır ve sonuçları organize ederler. Test çerçevelerin içerisinde yazılan test kodunun çerçeveye entegre edilmesini sağlayan soyut test sınıfı bulunmaktadır. Bu test sınıfı kalıtım yoluyla genişletilir. Yazılım geliştirici oluşturduğu sınıf içerisine kendi test koşullarını kodlar. Bunu yaparken uç değerler de dahil olmak üzere her durumu göz önüne alması testin eksiksizliği açısından önemlidir.

Test güdümlü yazılım geliştirmede temel prensipler şu şekilde sıralanabilir;

- Testler ilk yazıldığında başarılı sonuç vermezler. Ancak testleri çalıştıracak kod doğru biçimde gerçekleştirildiğinde başarılı sonuç vereceklerdir.

- Kaynak kod mümkün olduğunca basit tutulur. Bu sayede kodun taşınabilir, yeniden kullanılabilir ve kolay değiştirilebilir olması sağlanır.

- Kaynak kodun mümkün olduğunca basit olabilmesi, gereksinim duyulmayan tüm unsurların koddan ayıklanması ile mümkün olabilir. Bu nedenle, gerçekleştirim aşamasından sonra kod yeniden yapılandırılarak gereksiz bölümlerden arındırılmalıdır.

Tamamen farklı bir kod geliştirim tarzı sunması, TGP'nin programcılar tarafından kolay kabul edilememesine neden olmaktadır. Bununla birlikte, ilerleyen süreçte kod yazımı sırasında hataların yakalanması sayesinde ürün tesliminin ardından çok sayıda program hatası dönmemesini sağlaması ve böylece hataları gözden kaçırma riskini ortadan kaldırarak ürünün kalitesini arttırması programcılar tarafından kabul edilmesini sağlamaktadır.

Bazı araştırmalarda test güdümlü yaklaşımın geleneksel yaklaşımlara tercih edilmesi ile yazılım kalitesinde yaklaşık %18 artış görülmüştür. Bunun yanında, yazılım geliştirme süresi %15e kadar artsa da daha sonraki test ve doğrulama sürecinde bu farkın kapandığı gözlenmiştir (George and Williams, 2003).

2.3.3 Yeniden Yapılandırma

Uç programlamanın diğer bir önemli pratiği olan yeniden yapılandırma (*"Refactoring"*) var olan kod ve tasarım dokümanlarının yeniden düzenlenmesi için disiplinli bir tekniktir. Yeniden yapılandırma Fowler tarafından (Fowler, 1999)'de bir yazılım sistemi içerisinde kodun dışsal davranışını değiştirmeksizin içsel yapısını güçlendirmek için kullanılan süreç olarak tanımlanmaktadır. Bu sürecin merkezi küçük davranış dönüşümleridir. Her dönüşüm (yeniden yapılandırma olarak adlandırılır)

küçük çaplıdır. Fakat, bu dönüşümlerin bir serisi anlamlı bir yeniden yapılandırma oluşturur. Yeniden yapılandırmaların küçük olması sistemin bozulma riskini azaltır. Sistem bu küçük yeniden yapılandırmaların her birinin sonunda tam olarak çalışır halde tutularak yeniden yapılandırma sırasında sistemin çalışmasına önemli ölçüde zarar verebilecek değişiklikler kontrol altında tutulur.

(Fowler, 1999)'de nesneye yönelik geliştirim sırasında sık karşılaşılan problemlerin bir kısmı sistem tasarımındaki kötü kokular (*"bad smells"*) adı verilerek detaylı olarak tanımlanmıştır. Yeniden yapılandırma yaklaşımı geliştirilmekte olunan sistemin tasarımda bu kötü kokulardan bir tanesi tespit edildiğinde problemin aynı kitapta tanıtılan yeniden yapılandırma tekniklerinden biri veya bir kaçı kullanılarak çözülmesi fikrine dayanır. Geliştirim sürecinin başında sistem tasarımı düzgün bir şekilde modellense dahi, geliştirim sırasında oluşan program hatalarının (*"bug"*) düzeltilmesi ve sisteme yeni işlevselliklerin eklenmesi ile tasarımda kötü kokular büyük olasılıkla oluşmaktadır. Bu yüzden, bu etkinliklerin sonunda tasarım, uygulanabilecek yeniden yapılandırma fırsatları için gözden geçirilmelidir.

Yeniden yapılandırmanın getirdiği avantajlar şu şekilde sıralanabilir;

- Sistem tasarımının iyileştirilmesi daha sonraki değişimlerin getireceği maliyeti azaltır,

- Kodu ve tasarımı daha kolay anlaşılabilir hale getirerek bakımı kolaylaştırır,

- Daha iyi tasarım yazılımın daha kolay değiştirilebilmesini sağlar, bu yüzden verimliliği arttırtır.

- Test güdümlü geliştirim gibi evrimsel yazılım geliştirme yaklaşımları içerisinde, sistem tasarımını sürekli olarak iyileştirerek evrimsel geliştirimi mümkün kılar.

Çalışan bir sistemin tasarımında üzerinde uygulanan bir yeniden yapılandırma operasyonu sistemin hatasız çalışmasını etkileyebilir. Her ne kadar yeniden yapılandırma sistemin ilgilenilen bölümünün dışsal davranışını değiştirmeden iç yapısında bazı düzenlemeler yapsa da, sistemin geri kalan kısmındaki bazı birimler değiştirilen bu iç yapıya bağımlı olarak gerçekleştirilmiş olabilir. Böyle bir durumda, uygulanan yeniden yapılandırma operasyonu iç yapıya bağımlı olan diğer birimlerin çalışmasını etkileyerek sistem çalışmasını bozabilir. Bu yüzden, yeniden yapılandırma yaklaşımının uygulanabilmesi için sistem güvenilirliğinin daha önceden yazılmış testler ile garanti altına alınmış olması gerekmektedir. Test güdümlü geliştirim çalışan sistemin işlevselliğin garanti altına alan testleri sağlamaktadır.

3 İLGİLİ ÇALIŞMALAR

Bu bölümde, tez konusu ile ilgili literatürde tanıtılmış bazı önemli çalışmalar incelenmektedir. İlk alt-bölümde, yeniden yapılandırma pratiğinin farklı geliştirim tarzları içerisinde kullanımını öneren bazı çalışmalar incelenmektedir. Daha sonraki alt-bölümde, çevik geliştirim yaklaşımının ve bu yaklaşımdan gelen hafif pratiklerin EYYM alanı içerisine aktarılmasıyla ilgili önemli çalışmalar bulunmaktadır. Son alt-bölümde ise, çoklu etmen sistemlerinde yeniden yapılandırma yaklaşımına zemin oluşturan bir test sürecinin tanımlanması çalışmasıyla ilgili olarak, çoklu etmen sistemleri için otomatikleştirilmiş testler elde etmek üzerine odaklanan iki çalışma incelenmiştir.

3.1 Farklı Geliştirim Alanlarında Yeniden Yapılandırma

Yeniden yapılandırma yaklaşımının farklı yazılım geliştirim ortamlarına aktarılması üzerine pek çok çalışma bulunmaktadır. Örneğin (Thompson and Reinke, 2001)'de fonksiyonel programlama sırasında yeniden yapılandırmanın genel çerçevesi çizildikten sonra uygulanabilecek yeniden yapılandırmalar Haskell ve Erlang adındaki fonksiyonel programlama dilleri üzerinde tanıtılmıştır. Bu çalışmanın devamı olarak, (Li and Thompson, 2008)'de bahsedilen iki programlama dili ile fonksiyonel programlama sırasında yeniden yapılandırmayı destekleyen HeRa ve Wrangler isimlerinde yeniden yapılandırma araçları tanıtılmaktadır.

Veri tabanlarının evrimsel geliştirimini sağlamak için yeniden yapı-

landırma pratiğinin veri tabanı geliştirimi alanına aktarılması ile ilgili (Ambler and Sadalage, 2006; Ambler, 2007) gibi çalışmalar bulunmaktadır. (Ambler and Sadalage, 2006)'de ilişkisel veri tabanlarındaki tablo yapıları, veriler, depolanan yordamlar ve tetikleyiciler üzerinde küçük değişikliklerin anlamsallığın değiştirilmeden nasıl yapılabileceği, bu tür sistemlerin geliştiriminde kullanılabilecek bir yeniden yapılandırma yaklaşımı çerçevesinde gösterilmektedir. Bu kitapta, veri tabanı şemalarının kaynak kod ile birlikte adımlar halinde evrimleştirilmesi ile yürütülen artışlı ve çevik veri tabanı geliştirimi için yeniden yapılandırma odak pratik olarak gösterilmektedir. Veri tabanlarının çevik geliştirilmesi için bir yeniden yapılandırma yaklaşımının temel prensipleri verildikten sonra, veri tabanı geliştiriminde kullanılmak üzere tanımlanan yeniden yapılandırma teknikleri beş ana bölüme ayrılmış bir katalog içerisinde tanıtılmaktadır.

Yeniden yapılandırmanın kullanılmakta olduğu diğer bir alan bilgi tabanı geliştirimidir. (Baumeister et al., 2004)'de büyük bilgi tabanı sistemlerinin geliştiriminin çok zorlu ve hataya açık bir süreç olduğu, bu yüzden sistemin yeni gereksinimlerle başa çıkabilmesi için sürekli olarak var olan tasarımının güçlendirilmesi gerekliliğinden bahsedilmektedir. Problemin çözümü yolunda ilk olarak bilgi tabanlarının tasarımını zayıflatan bazı aksaklıklar tanımlandıktan sonra, bu aksaklıkların çözümü için ufak bir yeniden yapılandırma katalogu tanıtılmaktadır. Ayrıca, önerilen yeniden yapılandırma yaklaşımını desteklemek amacıyla bilgi tabanı tasarımı sırasında tanıtılan yeniden yapılandırma tekniklerini otomatik olarak işleten bir çerçeve tanıtılmaktadır.

Ontolojiler üzerinde yeniden yapılandırma yaklaşımlarının tanım-

lanması da son yıllarda oldukça ilgi çeken araştırma konularından biridir. Bu konudaki çalışmalardan biri olan (Baumeister and Seipel, 2006)'de anlamsal veb kural dili - AVKD (*"Semantic Web Rule Language" - SWRL*) ile oluşturulmuş kurallar kullanılarak ontoloji yapılarının yeniden yapılandırılmasını temel alan, kural tabanlı bir yeniden yapılandırma yaklaşımı tanıtmaktadır.

Çoklu etmen sistemlerinin tasarımın sürekli evrimleşmesini gerektiren dinamik ve açık doğası nedeniyle, bakım süreci çok zorlu ve maliyetli bir süreçtir (Dastani and Gomez-sanz, 2005). Bu tür sistemlerin geliştiriminde yeniden yapılandırma gibi çevik yaklaşımdan gelen pratiklerin kullanılmasının faydalı olacağı EYYM alanı içerisinde geniş oranda kabul edilmektedir (Cernuzzi et al., 2005; Zambonelli and Omicini, 2004). Çevik yazılım geliştirim tarzının çoklu etmen sistem geliştirimine entegre edilmesi üzerine bir sonraki alt-bölümde detaylı olarak incelenen (Chella et al., 2004; Clynch and Collier, 2007; Knublauch, 2002; Triebig and Klugl, 2008) gibi önemli çalışmalar bulunmaktadır. Bununla birlikte, bu çalışmaların hiç birinde ÇES geliştirimi sırasında kullanılabilecek kapsamlı bir yeniden yapılandırma yaklaşımı tanıtılmamaktadır.

3.2 Etmen Sistem Geliştiriminde Çevik Yaklaşımların Uygulanması Çalışmaları

Bu bölümde literatüde çevik yaklaşımların ÇES geliştirimine aktarılması konusu üzerine tanıtılmış bazı önemli çalışmalar incelenmektedir.

3.2.1 *AGILE* Yöntemi

Knublauch (Knublauch, 2002; Knublauch and Rose, 2002) en bilinen çevik geliştirim süreçlerinden biri olan uç programlamanın pratiklerini ÇES geliştirimi için kullanmıştır. Çalışmada UP'nin basitlik, iletişim ve geribildirim değerleri temel alınarak ÇES geliştirimi için çevik bir süreç oluşturulması amaçlanmıştır. Bu amaçla, etmen etkileşimlerinin hızlıca modellenebildiği ve değiştirilebildiği çok basit bir ÇES üst modeli ve geliştirim çerçevesi kullanılarak UP pratiklerinin ÇES geliştirimi sırasında nasıl kullanılabileceği üzerinde durulmuştur.

Bu çalışma UP pratiklerinin çoklu etmen sistemlerinin geliştirimi içerisindeki etkinliğini tespit etmesine rağmen, gerçekleştirim sırasında kullanılan etmen geliştirim çerçevesinin ve süreç üst modelinin çok basit olması nedeniyle, UP'nin test güdümlü gerçekleştirim pratiği nesne tabanlı test güdümlü gerçekleştirim gibi görünmektedir.

ÇES senaryoları etmen planları ve bu planlar arasındaki etkileşimler ile gerçekleştirilirler. Bu yüzden, çoklu etmen sistemlerinin test güdümlü gerçekleştirimi etmen planları ve etkileşimlerin test güdümlü gerçekleştirilmesini temel almalıdır. Fakat, Knublaunch'un çalışmasında etmen planlarının test güdümlü bir şekilde nasıl geliştirileceğinden bahsedilmemektedir.

Geliştirim çerçevesinin ve süreç üst modelinin çok basit seçilmesi, aynı zamanda etmenler üzerinde yeniden yapılandırma sürecinin çok basit bir süreç olarak görülmesine neden olmuştur. Bu yüzden, yeniden yapılandırma pratiği etmenler üzerinde uygulanmıştır. Bununla birlikte, gerçekçi bir etmen geliştirim çerçevesi kullanılarak geliştirilen bir etmen,

ÇES içerisinde bir çok rol oynayabilir ve bu roller de bir çok hedefe, sorumluluğa ve yeteneğe sahip olabilirler. Bu yüzden, etmenler test etme ve yeniden yapılandırma için küçük değil, aksine oldukça büyük birimlerdir.

Bu nedenlerden dolayı, gerçekçi bir etmen sistem gerçekleştirim çerçevesi ve üst model kullanılarak uygulandığında bu pratiklerin ölçeklenirliği şüphelidir.

3.2.2 *Agile Passi* Yöntemi

Bu konudaki diğer bir önemli çalışma (Chella et al., 2004; Cossentino and Seidita, 2004; Chella et al., 2006)'de Chella ve ark. tarafından çok bilinen Passi (Cossentino et al., 2003) yöntemini *Agile Passi'* ye dönüştürmek üzere tanıtılmıştır. Bu grup, çoklu etmen sistemlerinin test edilmesi için otomatikleştirilmiş test etme yaklaşımını sunmak üzere bir test çerçevesi geliştirmiştir (Caire et al., 2004). Bu çerçeve etmenlerin içsel davranışlarının test edilebilmesine olanak sağlamasına rağmen, çalışmada çoklu etmen sistemlerinin yinelemeli ve artışlı bir geliştirim tarzı ile geliştirilmesi için bir yaklaşım tanıtılmamıştır. Buna bağlı olarak, tanıtılan test çerçevesi böyle bir yaklaşımı desteklememektedir. *Agile Passi* çalışması çevik ÇES geliştirimini mümkün hale getiren bir yeniden yapılandırma yaklaşımı da sunmamaktadır. Bu grup tarafından geliştirilen test aracı bölüm 3.3.1'de detaylı olarak incelenmiştir.

3.2.3 *SADAAM* Yöntemi

(Clynch and Collier, 2007)'de SADAAM adında bir çevik etmen sistem geliştirim yöntemi tanıtılmaktadır. Çalışmada çevik yaklaşımlardan gelen

test güdümlü geliştirim ve yeniden yapılandırma gibi tekniklerin üzerine kurulmuş genel bir yöntemden bahsedilmektedir. Bu yöntem geliştirimin sonuna kadar yinelemeli bir şekilde devam eden tasarım, test güdümlü gerçekleştirim, dağıtım ve geliştirme adı verilen dört ana aşamadan oluşmaktadır. Fakat, yöntem içerisinde önemli yerleri olan test güdümlü geliştirim ve yeniden yapılandırma pratiklerinin etmen geliştirim için nasıl uygulanacağı konusunda kapsamlı bir tartışma bulunmamaktadır. Bu durum, önerilen yöntemin çevik yaklaşımın önerdiği soyut geliştirim modeline ÇES geliştirimi adına fazla bir şey katmayan, çok yüzeysel bir yöntem olarak kalmasına neden olmaktadır. Yöntemin ÇES geliştiriminde nasıl kullanılacağının belirginleşmesi için, içerisindeki çevik pratiklerin etmen sistemlerde nasıl uygulanacağının ayrıntılı ve kapsamlı olarak incelenmesi gerekmektedir.

3.2.4 Etmen Tabanlı Simülasyon Sistemleri için Yeniden Yapılandırma Yaklaşımı

(Triebig and Klugl, 2008)'de etmen sistemlerin bakımı için yeniden yapılandırma pratiğinin kullanılması önerilmektedir. Çalışma nesneye yönelik yazılım geliştirim için (Fowler, 1999)'de tanıtılan geleneksel yeniden yapılandırma tekniklerinden hangilerinin ve nasıl etmen tabanlı simülasyon sistemlerinde kullanılacağını MASim adı verilen çok etmenli simülasyon sistemleri modelleme paradigması üzerinde tanımlamak üzerine odaklanmıştır. Çalışma sonucunda MASim içerisinde tasarımı güçlendirmek için kullanılabilecek yeniden yapılandırmala tekniklerinde oluşan bir katalog tanıtılmıştır. Ayrıca, tanıtılan bazı yeniden yapılandırma teknikleri gerçek-

leştirilerek SeSam [1] etmen tabanlı simülasyon platformu içerisine entegre edilmiştir.

Bununla birlikte, etmen tabanlı simülasyon sistemlerinin yeniden yapılandırılması çalışmasında ÇES geliştiriminde kullanılabilecek genel bir yeniden yapılandırma yaklaşımı önerilmemektedir. Önerilen yeniden yapılandırma teknikleri sadece belirli tipteki etmen sistemler için uygun olabilir. Ayrıca, etmen sistem geliştiriminde kullanılabilecek yeniden yapılandırma tekniklerinin belirlenebilmesi için öncelikle bu sistemlerin genel karakteristikleri üzerine tanımlanan genel bir yeniden yapılandırma yaklaşımına gereksinim vardır.

3.3 Çoklu Etmen Sistemlerinde Otomatik Test Etme Çalışmaları

Literatürde tanıtılmış etmen yönelimli yazılım geliştirme - EYYG yöntemleri temel olarak çoklu etmen sistemlerinin analizi, tasarımı ve gerçekleştirimi için disiplinli yaklaşımlar önermektedirler. Bununla birlikte, bu yöntemler içerisinde etmen sistemlerin nasıl test edileceği konusundan çok kısıtlı bahsetmektedirler (Cernuzzi et al., 2005).

Bu yöntemlerin sadece birkaçı net bir doğrulama süreci tanımlamaktadır. MaSE (DeLoach S. A., 1999) ve MAS-CommonKADs (Iglesias et al., 1997) yöntemleri etmenler arası iletişimin otomatik doğrulanmasını desteklemek için model kontrolüne (*"model checking"*)[2] dayanan

[1] Sesam: http://www.simsesam.de/ , son erişim 07-Ağustos-2008

[2] Model kontrolü: Verilen bir yapının verilen bir mantıksal formülün bir modeli olup olmadığının kontrol edilmesi sürecidir.

bir doğrulama süreci önermektedir. Desire (Engelfriet et al., 2002) ise matematiksel ispatlara dayanan bir doğrulama süreci önerir. Bu sürecin amacı bir sistemin varsayımların kesin bir seti altında kesin bir özellikler setine bağlı kalmasının ispatlanmasıdır. Sadece PASSI/Agile PASSI (Chella et al., 2006) ve AGILE (Knublauch, 2002) gibi birkaç yinelemeli sürece sahip yöntem, bir test aracının desteği ile artışlı bir test süreci önermektedir.

Literatürde etmen sistem geliştiriminde otomatikleştirilmiş birim testlerin yazılması için geliştirilmiş test araçları tanıtan bir kaç çalışma bulunmaktadır. Bu bölümde bu test araçlarımdan en bilinen iki tanesi incelenmiştir.

3.3.1 *Agile Passi* Test Çerçevesi

Caire ve ark. (Caire et al., 2004)'de gerçekleştirim aşamasında bir test aracı desteğinin ÇES geliştirimi için sarf edilen zaman ve maliyeti düşürebileceğini ve kaliteyi arttıracağını fark etmişlerdir. Bu çalışmada geniş kapsamlı bir test çerçevesi sunmak yerine, basit bir test çerçevesini temel alan yeni bir test yaklaşımı sunmak amaçlanmıştır. Test çerçevesi test suitlerinin artışlı bir yol içerisinde oluşturulmasına izin vermektedir. Bu test çerçevesi geliştiricilerin testleri tekbiçimli ve otomatik bir yol içerisinde oluşturmasını ve çalıştırmasını desteklemektedir. Tekbiçimlilik, tüm testlerin aynı çalışma desenini takip etmesinden ve benzer bir format içerisinde sonuç üretmesinden gelmektedir. Otomatik olması ise, tüm testlerin ne çalışma sırasında ne de testlerin geçip geçmediğinin denetlenmesi sırasında kullanıcı aracılığına gereksinim duymamasından gelmektedir.

Otomatikleşmiş test etme yeni bir yaklaşım değildir. Otomatikleştirilmiş testlerin geliştirimini destekleyen bir çerçeveye gereksinim nesneye yönelik geliştirim topluluğu tarafından daha önceden fark edilmiştir. Bu durum JUnit (Massol and Husted, 2003) gibi bir çok XUnit test çerçevelerinin geliştirilmesini tetiklemiştir. Bu çalışmada tanıtılan test çerçevesi otomatik test etme yaklaşımını ÇES geliştirimine taşımayı amaçlamıştır. Test çerçevesi çok bilinen JADE (Bellifemine et al., 2007) etmen geliştirim çerçevesi üzerine kurulmuştur ve JADE tabanlı çoklu etmen sistemleri için testler oluşturulmasını destekler.

Güncel ÇES geliştirim yöntemlerinde genellikle analiz ve tasarım aşamaları ele alınmış, test aşaması çok fazla irdelenmemiştir. Bu yöntemlerden birisi olan Passi (Cossentino et al., 2003) yönteminde test etkinlikleri iki farklı adımda ilerletilmektedir. İlki, her etmenin davranışının o etmenin sorumluluğu altındaki sistem gereksinimlerine göre doğrulanması adımıdır. İkincisi olan topluluk testi sırasında ise farklı etmenler tarafından üretilen sonuçların tümü onaylanır ve birden fazla etmenin başarılı bütünleştirilmesi doğrulanır. Çalışma genel olarak etmen testi adımı üzerine odaklanmış ve çoklu etmen sistemlerdeki etmenlerin test edilmesini sağlayan bir yaklaşım ve bu yaklaşımı destekleyen bir test aracı tanıtılmıştır.

Etmen test etme aşaması aslında iki farklı etkinlikten oluşmaktadır. İlk olarak, etmen davranışı siyah kutu olarak ele alınmaktadır. Bir etmen genellikle iletişim protokollerini kullanarak sistemin geri kalan kısmıyla etkileşir. Bu gerçetekn yola çıkılarak, etmen davranışının test edilmesi bir etkileşimin başlatılması ve bu etkileşimin sonuçlarının değer-

lendirilmesiyle gerçekleştirilmektedir. Test edilmek istenen etmen etkileşim içerisinde başka bir tetikleyici etmen tarafından uyarılmalıdır. Bu tetikleyici etmenin oluşturulması işlemi etmenin bir davranış desenleri deposu (örneğin standart etkileşim protokollerinin gerçekleştirimi ile ilgili davranışlar seti) ile oluşturulmasına izin veren etmen üretme aracı (*"Agent Factory Tool"*) ile gerçekleştirilmektedir.

Etmenin davranışları, iletişim eylemi sonuçlarının (bir mesaj seti) tasarım modelleri göz önüne alınarak elde edilen beklenen sonuçlar ile karşılaştırılması sayesinde test edilir. Tasarım modelleri etmenler dış ortam ile etkileşime girdiğinde (örneğin algılayıcılar kullanarak) yetersiz kalabileceğinden, test etmeni dışsal bir aktör ile etkileşime girdiğinde bu tür bir doğrulama faydalı olabilir. Böyle bir durumda, tasarımın diğer kısımlarımdan başka bilgiler çıkartılmalıdır.

Etmen testinin ikinci seviyesi etmenlerin dahili davranışlarının test edilmesini içerir ve etmen yapısının beyaz kutu testi olarak değerlendirilebilir. Bu seviyede her bir davranış bir siyah kutu olarak ele alınır. Bu yüzden, sonuç bir alt-sistemin (etmenin) siyah kutu testinin bir tipidir. Bu seviyede beklenen sonuçların belirlenebilmesi için etmen davranışlarının her birini, akışlarını ve bağımlılıklarını gösteren daha detaylı tasarım modelleri kullanılmaktadır.

Çalışmada, önerilen test yaklaşımını destekleyen bir test aracı tanıtılmaktadır. Test aracı JADE etmen çerçevesi üzerine kurulmuştur ve bu yüzden JADE tabanlı çoklu etmen sistemleri için testler oluşturulmasına olanak sağlamaktadır. Araç, tek bir etmen bağlamında testlerin geliştirimi, çalıştırılması ve sonuçların gösterilmesini desteklemektedir.

Test çerçevesi iki seviyeli bir modeli temel almaktadır. İlk seviyede, etmen bir atomik birim olarak tanımlanmaktadır. Tek bir etmen tarafından yürütülen etkinliklerin doğruluğunun test edilmesi için çok sayıda farklı durum test edilir. Bu durum, belirli etmen görevlerinin tanımlandığı ikinci seviye için bir yol oluşturur. Verilen bir etmenin tüm yetenekleri ile ilgili olan testlerden "etmen testleri", belirli görevlerle ilgili testlerden ise "görev testleri" olarak bahsedilmektedir.

Sunulan test yaklaşımının ve test aracının etmen sistem geliştirimindeki birim test etme aşamasını iki seviyeye bölerek bu seviyelerden birinde etmeni, diğerinde ise etmen davranışlarını test ettiği savunulmaktadır. Fakat, çoklu etmen sistemlerindeki etmenler geliştirilen test aracının anlatımında daha fazla üzerinde durulan ilk seviyedeki gibi karşıdan bir mesaj göndererek buna karşı gönderdiği mesajlara göre test edilemeyecek kadar karmaşık yazılımlardır. İkinci seviye bunu kabul etmekte ve daha küçük birimler olan etmen davranışları üzerine odaklanmakla birlikte, bu bölümde de etmen davranışları siyah kutu olarak görülmüş ve test edilmiştir. Bu yüzden, bu seviye de ilk seviyede toplu olarak tek bir siyah kutu kabul edilen etmen davranışlarının her birinin teker teker kapalı kutu olarak test edilmesinden ileri gidememiştir.

Çalışmada sunulan yaklaşım ÇES geliştirimindeki potansiyel test seviyeleri ile ilgili olarak kayda değer fikirler sunmasına rağmen Passi test yaklaşımı yetersiz dokümantasyona sahiptir ve birim test durumlarının alt seviye tasarımında geliştiricilere yardım edecek herhangi bir teknik önermemektedir.

3.3.2 Çoklu Etmen Sistemlerde Sahte Etmen Kullanarak Birim Test Etme

Etmen sistemlerde test üzerine diğer bir çalışma olan (Coelho et al., 2006)'de Coelho ve ark. tarafından ÇES geliştirimi için yeni bir test yaklaşımı sunulmuştur. Yaklaşımın ana amacı, her etmenin diğerlerinden bağımsız olarak test edilmesi için ÇES geliştiricilerine yardım etmektir. Yaklaşım, etmen birim test durumlarının tasarlanması ve gerçekleştirilmesine kılavuzluk eden sahte etmenlerin kullanılması fikrini temel almaktadır. Etmenlerin test etme işlemi sırasındaki eş zamanlı olmayan çalışmasını gözetlemek ve kontrol etmek amacıyla ilgi (*"aspect"*) yönelimli bir altyapı kullanılmıştır. Çalışmada aynı zamanda, önerilen yaklaşımın JADE platformu üzerine kurulmuş bir gerçekleştirimi de sunulmuştur. Gerçekleştirimde JADE birim testlerinin çalışmasını sağlamak için JUnit (Link and Frolich, 2003) test çerçevesi genişletilmiştir.

Çalışmada geleneksel yazılım test etme seviyelerinden (birim, bütünleştirme, sistem ve kabul test seviyeleri) bahsedilmiş ve bu seviyelerin bir ÇES test etme sürecinin tanımlanması için kullanılabileceği vurgulansa da, yaklaşımda ve JADE için geliştirilen test aracında sadece ÇES geliştirimdeki birim test etme seviyesi üzerine odaklanılmıştır.

Önerilen yaklaşım çoklu etmen sistemlerinde en küçük test edilebilir birimlerin etmenler olduğunu savunmaktadır. Temel fikir, her etmenin normal ve anormal durumlar altında diğer etmelerden bağımsız olmasını sağlayan izole bir ortam içerisinde sorumluluklarını gerçekleştirip gerçekleştiremediğinin test edilmesidir.

Çalışmada etmenlerin son derece karmaşık iç yapıya sahip olduk-

ları ve bu iç yapı içerisindeki elemanların hatasız çalışmasını doğrulamak için birim test etme tekniklerine gereksinim duyulduğu söylenmekledir. Bununla birlikte, etmenin çoklu etmen sistemlerinin modülerliği içerisindeki içsel olarak tutarlı ve sistemin geri kalanı ile en az bağımlılığa sahip birim olduğundan dolayı bir bütün olarak test edilmesi gerektiği savunulmaktadır.

Çoklu etmen sistemleri içerisinde sürekli olarak birbirleri ile iletişim kuran etmenlerin nasıl birbirinden bağımsız olarak test edilebileceği üzerine odaklanılmıştır. Bu doğrultuda, birbirine bağımlı birimlerin her birinin izole bir ortamda test edilmesi için sahte birimlerin oluşturulması fikri ÇES test etme yaklaşımına aktarılmış ve "sahte etmen" adında bir kavram tanımlanmıştır. Bir sahte etmen sadece test altındaki etmen ile iletişim kuran normal bir etmendir. Sahte etmenin planı test altındaki etmene gönderilmesi gereken ve bu etmenden beklenen mesajları tanımladığından dolayı bir test komut dosyasına denktir. Bir etmenin sahte etmen kullanılarak izolasyon içerisinde test edilmesi programcının etmenin işbirlikçileri ile olan etkileşimlerini göz önüne almasını sağlar.

3.3.3 Tropos Hedef Yönelimli Test Etme Yaklaşımı

Nguyen ve ark. (Nguyen et al., 2007)'de çoklu etmen sistemlerinin geliştiriminde kullanılabilecek farklı test seviyeleri ve bu seviyelerin içerdiği test etkinliklerini Tropos (Bresciani et al., 2004) yöntemi üzerinde tanımlamışlardır. Bu test yöntemi içerisinde birim test etme, bütünleştirme test etme ve sistem test etme olmak üzere üç seviye bulunmaktadır. Birim test etme seviyesi etmenlerin test edilmesini amaçlar. Tropos yöntemi

geliştirim sırasında etmenlerin hedeflerini tanımladığından, etmenlerin test edilmesi için bu hedeflerin gereksinimleri kullanılır. Farklı etmenlere paylaştırılmış, birbirine bağımlı hedeflein işbirliği içerisinde çalışmasını doğrulamayı amaçlayan bütünleştirme testleri, Tropos hedef modeli içerisindeki hedef bağımlılıkları kullanılarak oluşturulur.

Tanıtılan yaklaşım hedef kavramı üzerine kurulmuş olmasına rağmen, test edilebilir en küçük birim olarak etmenin tamamını test etmeyi amaçlamaktadır. Bir etmenin test edilmesi için bir test etmeni yardımıyla gerekli test mesajı gönderilmektedir. Ancak etmen karmaşık bir altyapıya sahiptir. Bu yüzden etmenlerin özerkliğinin ve her hedef için çalıştırdığı içsel davranışların etmenin dışarısından test edilmesi her zaman mümkün değildir.

4 ETMEN YÖNELİMLİ TEST GÜDÜMLÜ GELİŞTİRİM

Bu bölümde, çoklu etmen sistemlerin artışlı ve yinelemeli bir yol içerisinde geliştirimine olanak sağlayarak tezin ana hedefi olan çoklu etmen sistemleri için yeniden yapılandırma yaklaşımına zemin oluşturan etmen yönelimli test güdümlü geliştirim - EYTGG (Tiryaki et al., 2006) adında bir yaklaşım tanıtılmaktadır. EYTGG yaklaşımı geliştirim boyunca sistemin güvenilirliğini garanti altına alan testlerin elde edilmesini sağlayarak yeniden yapılandırma pratiğinin ÇES geliştiriminde uygulanabilmesini mümkün hale getirir.

EYTGG yaklaşımı uç programlamanın (Beck and Andres, 2004) test güdümlü geliştirim - TGG pratiğini temel alarak, gerçekleştirim sırasında her ÇES senaryosu için etmen görevlerini, etkileşimleri ve organizasyonel sorumlulukları evrimsel bir yol içerisinde tanımlar. Geleneksel TGG gibi EYTGG de tüm şelale benzeri ve evrimsel yöntemlere entegre edilebilen genel bir yaklaşımdır. Herhangi bir etmen sistem geliştirim yöntemi kendi analiz, tasarım ve gerçekleştirim aşamalarının uygulanması sırasında EYTGG yaklaşımın kullanarak artışlı ve yinelemeli geliştirim tarzlarını destekleyebilir.

Önerilen EYTGG yaşam döngüsünün en kritik adımlarından biri olan test adımını gerçekleştirebilmek için, hedef kavramı üzerine kurulan bir hedef yönelimli test etme yaklaşımı (Ekinci et al., 2008) tanımlanmıştır. Bu test yaklaşımı en önemlilerinden üçü Bölüm 3.3'de detaylı olarak incelenen diğer test etme yaklaşımlarından ve test araçlarından farklı olarak

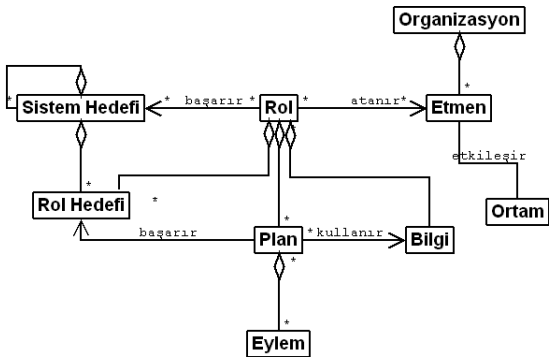
etmenler yerine etmen hedeflerinin çoklu etmen sistemler içerisindeki test edilebilir en küçük birimler olduğunu ileri sürmektedir. Bununla birlikte, hedef yönelimli test etme yaklaşımını destekleyen SeaUnit adından bir test çerçevesi SEAGENT (Dikenelli et al., 2005a; Dikenelli et al., 2005b) ÇES geliştirim çerçevesi üzerine geliştirilmiştir. SeaUnit test çerçevesi, geliştiricilerin ilgilenilen senaryonun işlevselliğini doğrulamak üzere etmen davranışları ve etmenler arası etkileşimler için otomatik testler yazabilmesine olanak sağlar.

4.1 EYTGG Geliştirim Döngüsü

Uç programlama içerisinde test güdümlü geliştirim, ancak geliştirici metafor (*"metaphor"*) olarak adlandırılan basitleştirilmiş genel sistem mimarisi hakkında bilgiye sahipse başarıyla uygulanabilir (Beck and Andres, 2004). Sistem metaforu herkesin (müşteriler, programcılar ve yöneticiler) sistemin nasıl çalışacağını hakkında söyleyebileceklerinden elde elden bir öyküdür. Uç programlamadaki metafor, yazılım araştırmacılarının kullandığı "yazılım mimarisi" kavramının yerini alır. Metaforun bilinmesi, geliştiricilerin test güdümlü geliştirim sırasında mimarisel birimleri ve bu birimlerin işlevsel birimlere bağımlılıklarını tanımlamasına olanak sağlar.

Hedef yönelimli ÇES geliştirimindeki ürünlerinin tanımlanabilmesi için, literatürde önerilmiş olan hedef yönelimli ÇES geliştirim yöntemlerinin tümü için ortak soyutlamaları içeren genel bir metafor (üst model) tanımlanmalıdır. Bu yüzden tezde, evrimsel ÇES geliştirimi sürecinin uygulanması yolundaki hedef ÇES modeli için temel bir üst model tanımlanmıştır. Şekil 4.1'de görülmekte olan bu üst model Gaia,

Tropos ve Passi gibi çok bilinen bazı hedef yönelimli yöntemler tarafından kullanılan üst modellerin bir sentezidir ve sadece bu yöntemlerdeki ortak soyutlamaları (farklı yöntemlerde farklı isimlerle anılabilirler) içermektedir. Geliştirim sırasında her ÇES senaryosu için bu üst modelin bir örneği yaratılır ve önerilen çevik geliştirim döngüsü bu üst model örneği üzerinde uygulanır.

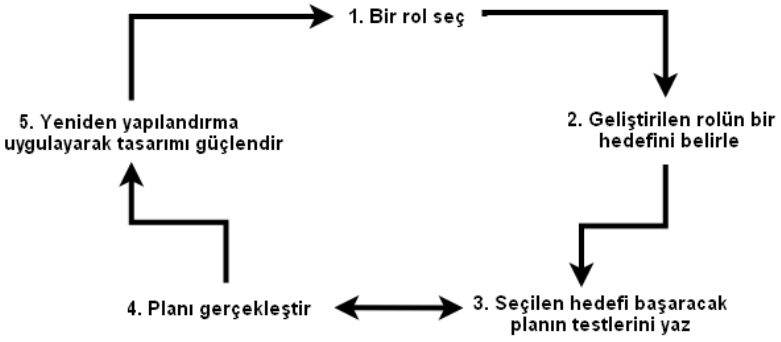


Şekil 4.1. ÇES Metaforu

Her çoklu etmen sisteminin temel gereksinimlerinden gelen genel hedefleri vardır. Bu genel hedefler sistem hedefleri olarak adlandırılmaktadır. Roller işbirliği içerisinde çalışarak sistem hedeflerini başarmaya çalışırlar. Rol sistem içerisinde aynı özelliklere sahip etmenlerin bir grubunu tanımlar. Her rolün diğer rollerle etkileşerek başarımında görev aldığı sistem hedeflerinden gelen bazı sorumlulukları, yetenekleri ve yetkileri vardır. Rol hedefleri bir durum karşısında etmen tarafından başarılması gereken atomik hedeflerdir ve sistem hedeflerinin ayrıştırılması ile

elde edilirler. Bir rolün sahip olduğu rol hedefleri o rolün sorumlulukları olarak düşünülebilir. Rol kendisine atanan rol hedeflerini başarmak için planlarını kullanır. Planlar rolün yetenekleridir. Bir rol hedefi her seferinde tek bir plan ile başarılır fakat etmenler özerk doğaları gereği bir rol hedefini başarmak için farklı zamanlarda farklı planlar kullanılabilir. Bir plan çalışabilir kod içeren eylemlerden oluşan bir yapıdır. Roller çalıştırılmak üzere etmenlere atanırlar. Sistemin çalışması sırasında sistemde oluşturulmuş olan etmenler bütünü ÇES organizasyonunu oluşturur. Etmenler dış ortamla ve diğer etmelerle etkileşim kurarak rol hedeflerini başarmaya çalışırlar.

Şimdi, tanımlanan kavramlar ve üst modeli temel alarak EYTGG sürecini tanımlayabiliriz. Şekil 4.2 EYTGG döngüsünün görsel tanımlamasını göstermektedir. Aşağıda her adım detaylı olarak anlatılmaktadır.



Şekil 4.2. EYTGG yaklaşımının artışlı döngüsü

İlk adımda, bir rol seçilir. Senaryo gerçekleştiriminin başlangıcında geliştirici EYTGG döngüsünü başlatmak üzere senaryoyu başlatacak rolü

belirler.

İkinci adımda, geliştirici ilgilenilen senaryonun içersinde seçilen rolün sorumluluğunu karşılamak üzere bir hedef belirler. Bu aşamada geliştirici ilgilenilen hedefi yerine getirecek başlangıç plan yapısını basit olarak çizebilir ve/veya üzerinde düşünebilir.

Üçüncü adımda, geliştirici geliştirilmekte olan hedefin geçerliliğini sınayacak testleri yazar. Özerk etmenler için otomatik testlerin yazılması, nesneler için test yazılmasına göre farklı bir yaklaşım gerektirir. Bu sistemlerin geliştirim sırasında etmen, rol, hedef gibi farklı kavramlar kullanılmaktadır. Çoklu etmen sistemleri için test seviyelerini belirleyen ve bu sistemlerin test güdümlü geliştirimi için zemin hazırlayan hedef yönelimli bir test etme yaklaşımı ve bu yaklaşımı desteklemek için geliştirilen bir test aracı bölüm 4.2'de tanıtılmaktadır.

Dördüncü adımda, geliştirici ikinci adımda tanımlanan rol hedefinin ve üçüncü adımda bu hedefin doğruluğunu garantilemek için yazılan testleri temel alarak ilgilenilen hedefi başaracak etmen planını gerçekleştirir. Önerilen hedef yönelimli test etme yaklaşımında, testlerin gerçekleştirimde kullanılan programlama dili ile aynı dil kullanılarak tanımlanması desteklendiğinden, geliştirici bir önceki adımda oluşturduğu test planlarındaki kodun yeniden kullanılabilmesi sayesinde yazılan testleri çalıştıracak planı kolaylıkla gerçekleştirir.

Etmen görevleri tüm testlerin tanımlanması ve bütün yapının tek bir adım içerisinde kurulması için çok karmaşık olabilir. Plan yapılarının gerçekleştirimi sırasında geliştiriciler yapıyı yanlış veya eksik kılacak ek gereksinimler tanımlayabilir. Bu noktada, bir önceki adıma dönerek gerekli

testler yazılır ve daha sonra plan yapısı gerçekleştirimine devam edilir. Bu yüzden üçüncü ve dördüncü adımlar arasında dahili bir döngü vardır.

Son adımda, geliştirici başlangıçtaki tasarım kararlarını yeniden yapılandırabilir. Örneğin, önceki plan yapısı görev içerisinde yeniden kullanılabilir görevler tanımlanarak daha iyi bir yapıya dönüştürülebilir ve/veya sistemin modülerliğini arttırmak için yeni etmen(ler) veya organizasyonel etkileşimler tanımlanabilir. Bu adım bir sonraki ana bölümde ayrıntılı olarak tanıtılmaktadır.

Döngünün sonunda, ilk adımda seçilen rolün geliştirilmekte olan hedefini başaracak plan yapısı işleveselliği testler ile garanti altına alınmış bir şekilde gerçekleştirilmiş olur. Daha sonra geliştirim döngüsü geliştirilen bu hedefin etkileşimde olduğu hedefler öncelikli olacak şekilde sistemdeki rollerin tüm hedefleri için tekrar eder. Sonuç olarak, senaryo EYTGG döngüsünün takip edilmesi ile yinelemeli ve artışlı bir yol içerisinde gerçekleştirilir.

4.2 Etmen Sistemlerde Hedef Yönelimli Test Etme için Yeni Bir Yaklaşım

ÇES geliştiriminde kullanılmak üzere bir test süreci tanımlamak için, geleneksel yazılım test tipleri (birim, bütünleştirme, sistem ve kabul testleri) ÇES geliştirim etkinlikleri ile eşleştirilmelidir. Böyle bir eşleme bizi en temel sorunun cevabını aramaya iter; ÇES geliştirimi için birim nedir?

ÇES geliştiriminde kullanılmak üzere daha önceden tanıtılan (Caire et al., 2004; Nguyen et al., 2007) test süreçleri ve (Coelho et al.,

2006; Caire et al., 2004) test araçlarının tümü ÇES geliştiriminde en küçük birimin etmen olduğunu savunmuşlar ve bir etmenin işlevsel doğruluğunu denetlemek için olay tabanlı araç altyapıları kullanmışlardır. Fakat, etmenin kendisi bir birim olarak değerlendirilmek için çok büyük ve karmaşık bir sistemdir. Bir etmen birçok hedefi içerisinde bulunduran bir çok rolü oynayabilir. Bu hedefler de etmenin iletişim altyapısı, bilgi işleme ve planlama gibi görevleri içeren karmaşık mimarisi sayesinde başarılırlar. Bu yüzden, böyle geniş ölçekli bir sistemde hataların ve başarısızlıkların sadece sistem olaylarının dikkate alınarak yakalanması zordur (Hanks et al., 1993). Test etme ve hatanın yerini belirleme süreçlerini kolaylaştıracak daha küçük bir birime gereksinim vardır.

Bu bölümde, rol hedeflerini test edilebilecek en küçük birimler olarak kabul eden yeni bir test etme süreci tanıtılmaktadır. Etmen yönelimli yazılım geliştirim yöntemleri genellikle analiz aşamasında sistem seviyesindeki hedefleri belirler, daha sonra tasarım aşamasında bu hedefler parçalanır ve ÇES mimarisine atanır. Etmen içsel mimarisine bağlı olarak (KİN gibi) bu hedefleri işletir ve işbirliği ile sistem hedefleri başarılır. Bu durumda, etmen seviyesi hedefler tüm etmen yöntemleri için en küçük geliştirim ürünleridir.

Rol hedeflerini en küçük birimler olarak tanımladıktan sonra geliştiricinin bu hedefleri nasıl test edeceğini tanımlamalıyız. Bu tez kapsamında, birim testlerin gerçekleştirilmesi için “test hedefi” olarak adlandırılan yeni bir kavram tanıtılmaktadır. Test hedefleri rol hedeflerinin test edilmesi için yazılan açık tanımlamalardır. Bu fikir uç programlamanın test güdümlü geliştirim pratiğinden esinlenilerek oluşturulmuştur.

Uç programlamada olduğu gibi, test hedefleri etmen çerçevesi tarafından desteklenen ve uygulama hedeflerinin gerçekleştirilmesinde kullanılan etmen programlama dili kullanılarak gerçekleştirilir ve rol hedefleri olarak tutulurlar. Bu yüzden, testlerin hedefler olarak kodlanması test kodunun kaynak koda kolayca dönüşümünü sağlar (van Deursen et al., 2001).

Test hedefleri etmenlerin bilgi tabanı gibi içsel modüllerini doğrulamak için etmen programlama dilini kullandığından, olay tabanlı test etmeye göre daha kapsamlı test etme olanakları sağlar. Bununla birlikte, etmenler arası işbirliği gereksinimleri, sahte etmen kavramını gereksiz hale getiren test hedefleri içerisinde test edilebilir. Gerçek etmen gerçekleştirildikten sonra test hedefi yeni durumu gösterecek şekilde yeniden yapılandırılır.

Tanıtılan "test hedefi" kavramı ve test süreci temel alınarak bir test altyapısı SEAGENT (Dikenelli et al., 2005a; Dikenelli et al., 2005b) çerçevesi üzerine gerçekleştirilmiştir. Bu test altyapısı, hedef gerçekleştirimi sırasında test hedeflerinin tanımlamasına olanak sağlayarak, gerçekleştirim döngüsü sırasında birim ve bütünleştirme testlerini destekler.

4.2.1 Hedef Yönelimli Test Yaklaşımı İçerisindeki Test Seviyeleri

Etmen sistemlerinin otomatik test edilmesi yaklaşımı geliştirim yöntemine ve kullanılan etmen teknolojisine bağımlıdır. Bu yüzden, bu sistemlerin geliştiriminde kullanılacak bir test süreci tanımlamak için etmen teknolojisinde otomatik test edilmesi gereken, test edilebilir ürünlerin belirlenmesi gerekir.

Tezde önerilen test sürecinde, test edilmesi gereken temel ürünler sistem hedefleri ve rol hedefleridir. Etmenler ilgili hedefleri başarmak için belirli rolleri oynarlar. Bununla birlikte, etmenler sistem seviyesi hedefleri başarmak için diğer rollerdeki etmenler ile işbirliği içerisinde çalışmalıdırlar. Geliştiriciler, ilk olarak her rol hedefinin doğru gerçekleştirildiğini, daha sonra da etmenlerin işbirliğini test ederek sistem seviyesi hedefleri doğrulamalıdır. Tanımladığımız test sürecinde etmen test etme süreci için en küçük birim olarak rol hedeflerini ele alınmaktadır. Bu test süreci içerisindeki test tipleri aşağıdaki gibi tanımlanmıştır:

- **Rol hedefi (birim) test**i; rol hedefleri en küçük test edilebilir birimlerdir ve rol hedeflerinin test edilmesi, bir hedefin tek bir etmen bağlamında düzgün işletimini doğrular. Etmen içeriği etmen bilgi tabanını, etmenin doğrudan iletişim kurduğu dış ortamı içerebilir. Rol hedefleri bölüm 4.4'de sunulan test ortamının desteği ile diğer rol hedefleriyle işbirliklerinden soyutlanmış olarak test edilmelidir.

- **Bütünleştirme testi;** rol hedeflerinin işbirliklerine odaklanarak bir sistem hedefinin işlevselliğini doğrular. Etmenlerin hedef işletimleri ve işbirlikleri sayesinde etmenler tarafından başarılan sistem seviyesi gereksinimleri test edilmelidir.

- **Sistem testi;** sistem testleri performans, güvenilirlik, güvenlik gibi işlevsel olmayan gereksinimleri içeren gerçek çalışma ortamında tüm sistemin çalışmasını doğrulamayı amaçlar. Sistem testlerini gerçekleştirmek için, kritik işlevsel olmayan gereksinimleri gerçekleştiren sistem hedeflerinin bir seti seçilmelidir.

- **Kabul testi;** kabul testleri sistem kullanıcıları için kritik olan sistem işlevselliklerini doğrulamak içindir. Bu yüzden, kabul testleri sistem kullanıcıları tarafından seçilmiş bir sistem seviyesi hedef seti için uygulanır.

Test ürünlerini ve bu ürünleri temel alan test tiplerini tanımladıktan sonra bu ürünlerin ne zaman ve nasıl test edileceği tanımlanmalıdır. V-model (Muller-Ettrich, 1997) test etkinliklerini geliştirim etkinlikleri ile eşleştiren en bilinen modellerden biridir. Bu modele göre, her geliştirim adımı ilgili test tipi ile test edilir. Genel bir hedef yönelimli test süreci doğal olarak kabul testlerini analiz aşamasından sonra, sistem hedefleri belirlendiğinde çalıştırmalıdır. Bu aşamada, seçilen sistem hedefleri için test durumları tanımlanır ve raporlanır. Sistem testleri tanımlı bir etmen mimarisini gerektirir. Bu yüzden, sistem test etkinlikleri sistem hedeflerini başaracak etmen mimarisinin tanımlanmış olduğu tasarım aşamasından sonra gerçekleştirilir.

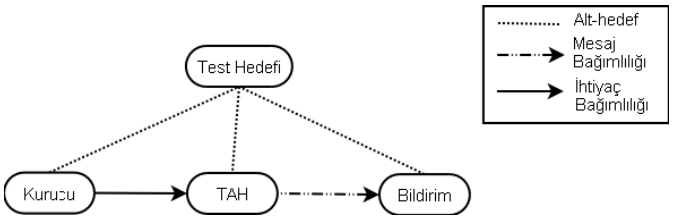
Yaklaşımımızın diğer test yaklaşımlarından temel farklarında biri, birim testlerinin ve bütünleştirme testlerinin gerçekleştirim sürecinde tanımlanmasıdır. Rol hedefleri ve sistem hedefleri test hedefi kavramı kullanılarak gerçekleştirilirler. Test hedeflerinin nasıl tanımlandığı ve gerçekleştirildiği bir sonraki alt-bölümde detaylı olarak anlatılmaktadır.

4.2.2 ÇES Geliştiriminde Hedef Yönelimli Test Etme

Hedef yönelimli ÇES geliştirim yöntemleri ve hedef tabanlı etmen mimarileri ÇES uygulamalarının hedef kavramı üzerine geliştirilmesini amaçlamaktadırlar (Stollberg and Rhomberg, 2006). Bu yöntemlerde ilk olarak,

uygulamanın gereksinimleri sistem seviyesi hedeflere aktarılır, daha sonra sistem seviyesi hedefler, işbirliği içerisinde çalışarak sistem seviyesi hedeflerin başarımını sağlayan etmen seviyesi hedeflere ayrıştırılır. Bu yüzden, bir ÇES uygulamasının test edilmesi süreci, tanımlanan bu hedeflerin doğruluğunu garanti altına almak için sistem ve etmen seviyesi hedefleri doğrulamalıdır.

Önerilen yeni hedef yönelimli test yaklaşımı geliştirim sırasında tanımlanan hedeflerin testlerinin yine hedefler olarak gerçekleştirilmesi fikrine dayanır. Test etme için kullanılan hedeflerin kaynak hedeflerden farkını vurgulanmak amacıyla bu hedeflere "test hedefi" adı verilmiştir. Şekil 4.3'de görüldüğü gibi bir test hedefi kavramsal olarak kurucu (*"setup"*) hedef, test altındaki hedef - TAH ve bildirim (*"assertion"*) hedefi olmak üzere üç alt-hedefe bölünür. Test hedefi tetiklendiğinde, başlangıç olarak kurucu hedef çalıştırılır ve test altındaki hedefin çalışmasını sağlayacak girdileri ve ön koşulları hazırlar. Tüm koşullar hazır olduğunda TAH çalıştırılır ve son olarak bildirim hedefi işletilir. Bildirim hedefleri bilgi tabanı farklılıkları gibi etmen altyapısının içsel detaylarını kontrol eden karmaşık gerçekleştirimler olabilir. Denetimlerin karmaşıklığına göre, bildirim hedefleri daha küçük alt-bildirim hedeflerine bölünebilir ve bazı kaynak hedefler veya diğer test hedefleri bildirim hedefleri içerisinde yeniden kullanılabilir.



Şekil 4.3. Bir hedefin birim testi

Bir test hedefinin hangi test seviyesinde olduğu kapsadığı TAH'a bağlıdır. TAH bir rol hedefi ise birim seviyesi test uygulanmaktadır. Diğer taraftan, TAH rol hedeflerinden oluşan bir sistem hedefi ise yapılan test bütünleştirme testi olarak kabul edilir. Test hedefi kavramı birim ve bütünleştirme testlerinin aynı yaklaşım (bir hedefin başarımını test etmek için bir test hedefinin gerçekleştirilmesi) kullanılarak yapılabilmesini mümkün hale getirmiştir.

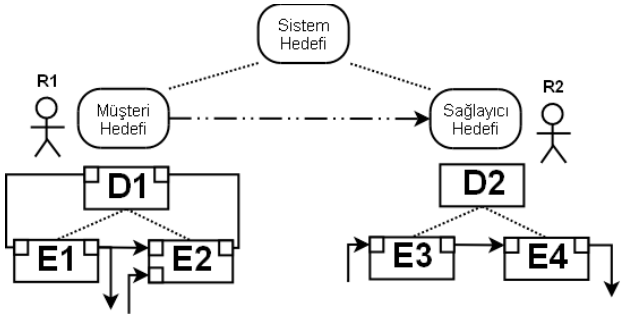
Diğer bir taraftan, sistem seviyesi test etme geçerlilik, stres ve güvenlik gibi bir çok test tipini içerebilen daha kapsamlı bir test aşamasıdır. Bu test tiplerinden geçerlilik testi dışındakilerin test hedefi kavramı kullanılarak test edilmesi mümkün değildir. Bununla birlikte, hangi sistem ve kabul testi tekniklerinin test hedefi yaklaşımı ile uygulanabileceği bu tezin konusu içerisinde değildir.

Bir sonraki alt-bölümde, SGA planlama paradigmasını kullanan bir etmen mimarisi içerisinde birim ve bütünleştirme testleri için test hedeflerinin nasıl gerçekleştirileceği tanımlanmaktadır. Önerilen test hedefi yaklaşımı SGA paradigması kullanılarak örneklenmesine rağmen, bu fikir başka planlama paradigmaları kullanan hedef yönelimli etmen mimarileri için de uygulanabilir.

4.3 Hedef Yönelimli Test Yaklaşımının SGA Tabanlı Bir Etmen Mimarisi Üzerinde Uygulanması

SGA, Retsina (Sycara et al., 2003), Decaf (Graham et al., 2003) ve SEAGENT (Dikenelli et al., 2005a) gibi etmen çerçeveleri tarafından yaygın olarak kullanılan bir planlama paradigmasıdır. Bu çerçevelerde etmen planları tasarım aşamasında SGA planları olarak modellenir ve çalışma zamanında bir SGA planlayıcısı tarafından çalıştırılır. Tez çalışması sırasında, önerilen hedef yönelimli test etme yaklaşımını güçlendirmek amacıyla SEAGENT çerçevesi için hedef yönelimli geliştirime uygun bir SGA planlayıcısı (Ekinci et al., 2007) geliştirilmiştir.

SGA paradigmasını kullanan etmenler içerisinde, rol hedefleri SGA planları tarafından gerçekleştirilir. Önerilen test yaklaşımında bir SGA planının test edilmesi için kurucu ve bildirim hedefleri ayrı SGA planları olarak gerçekleştirilir ve bu planlar çalışma zamanında test hedefini başarmak için ilişkilendirilir.



Şekil 4.4. Örnek bir SGA yapısı

Test yaklaşımının uygulanmasını anlatabilmek için Şekil 4.4'de

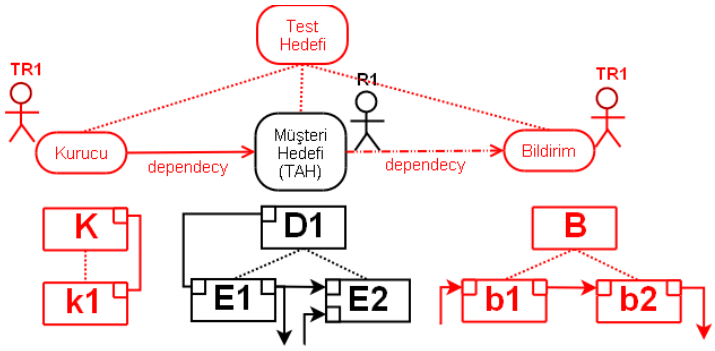
görülen basit bir örnek kullanılmıştır. Bu örnekte bir sistem hedefi iki rol hedefinin işbirliği ile başarılmaktadır. SGA planı olarak gerçekleştirilmiş olan bu rol hedeflerinden her biri farklı rollere sahip bir etmen tarafından işletilmektedir. Şekildeki SGA yapılarında *D* harfi bir davranışı, *E* harfi ise bir eylemi göstermektedir. Şekilde görüldüğü gibi *Müşteri Hedefi R1* rolüne ait *D1* davranışı tarafından gerçekleştirilmektedir. *D1* başlatıldığında bu davranışın ilk eylemi *E1, E3* eyleminin ihtiyacını içeren bir mesaj göndererek *Sağlayıcı Hedefi'*ni tetikler. *E4* eyleminin çalışması ile *D2* davranışı biter ve *E4* eylemi *E2* eylemine bir sonuç mesajı gönderir. *E2* bu mesajı aldığında çalışır ve *Müşteri Hedefi* sonlanır. Bu örnek devam eden alt-bölümlerde hedef yönelimli test yaklaşımımızın SGA gerçekleştirimi üzerinden nasıl uygulandığını tanımlamak için kullanılacaktır.

4.3.1 Birim Test Etme

Daha önceki bölümlerde bahsedildiği gibi hedef yönelimli test etme yaklaşımı rol hedeflerinin bir ÇES uygulamasının en küçük test edilebilir birimleri olduğu ve birim testler ile doğrulanması gerektiği fikri üzerine kuruludur. SGA paradigmasına göre, plan yapıları içerisinde en üst seviyedeki davranışlar rol hedeflerini yerine getirirler. Bu yüzden, test altındaki hedefi başarmak üzere geliştirilen plan içerisinde en üst seviyedeki davranış test hedefinin gereksinimlerini belirler. Kurucu hedef TAH'in ön koşullarını hazırlamalıdır. Benzer olarak, bildirim hedefleri en üst seviyedeki davranışın gereksinimlerini temel alarak TAH'in son koşullarını doğrulamalıdır.

Şekil 4.4'da tanımlanan örnek içerisindeki *R1* rolüne ait *Müşteri*

Hedefi'nin önerilen yaklaşım ile nasıl test edilebileceği Şekil 4.5'de gösterilmektedir. Şekilde görüldüğü gibi test hedefinin her alt-hedefi (kurucu, TAH ve bildirim hedefleri) ayrı SGA planlarına sahiptir ve bu alt-hedefler birbirleri arasında bağımlılık bağlarına sahiptirler. Test hedefinin çalıştırılması sırasında bu hedefler tanımlanan bağımlılıklara göre çalışma zamanında oluşturulan bağlar kullanılarak çalıştırılırlar.



Şekil 4.5. Birim test etme

Kurucu, TAH ve bildirim hedefleri arasındaki hedef bağımlılıkları aynı zamanda bu hedefleri gerçekleştiren SGA planları arasında da bazı bağımlılıkları gerektirir. Örneğin, *K* kurucu planı TAH'in *D1* davranışının ihtiyaçlarını hazırlar. Kurucu hedefin çalıştırılmasından sonra hedef eşleme motoru (Ekinci et al., 2007) kurucu ve TAH hedefleri arasındaki bağımlılığı tanımlar ve *D1* davranışının ihtiyaçlarını *K* planının çıktıları ile sağlayarak *Müşteri Hedefi'*nin planını tetikler. Böyle bir mekanizma bir hedefin bağımlılıklarını takip ederek alt-hedefleri çalıştırabilen bir hedef eşleme motoru ve ilgili hedeflerin planlarını çalıştıran bir planlama motoru gerektirmektedir.

Benzer olarak, *Müşteri Hedefi* ve *Bildirim Hedefi* aralarında farklı

tipte bir bağımlılığa (şekilde noktalı çizgilerle gösterilen mesaj bağımlılığı) sahiptirler. Bu bağımlılık *Müşteri Hedefi*'nin mesajının (*E1* eyleminden gönderilen mesaj) *Bildirim Hedefi*'ne (*B* davranışının *b1* eylemi) bağlanması gerektiğini göstermektedir. *E1* eylemi bir mesaj gönderdiğinde bu mesaj aynı etmen tarafından alınır ve etmenin hedef eşleme motoru bu mesajı değerlendirebilecek hedef olarak *Bildirim Hedefi'*ni belirler ve bu hedefin planını tetikler. Daha sonra *Bildirim Hedefi*'ni gerçekleştiren plan içerisindeki *b1* eylemi *E1* eyleminin işlevselliğini denetler.

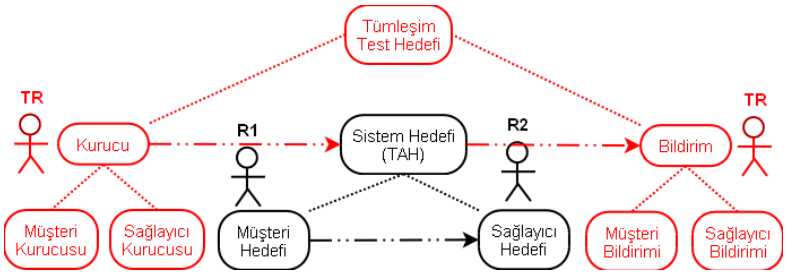
Diğer bir taraftan, TAH işbirliği yaptığı diğer hedeflerden (bu hedefler aynı etmende veya farklı etmenlerde bulunabilir) karşılanacak bazı dışsal ihtiyaçların da sağlanmasını gerektirebilir. Böyle bir durumda, TAH'ın doğrulanması için bu ihtiyaçlar çalışma zamanında sağlanmalıdır. Örneğimizde bu durum *b2* ve *E2* eylemleri arasındaki mesaj bağımlılığı olarak gösterilmektedir. Bu mekanizma TAH'ın gereksinim duyduğu bilgilerin bir bildirim hedefinin görevleri ile toplanabilmesine izin vererek diğer otomatik test araçlarının çekirdek kavramı olan sahte etmen kavramını gereksiz hale getirmektedir. Sahte etmen kullanılan yaklaşımlar etmenler arasında mesaj bağımlılığı gerektirirler. Bu yüzden, hiçbir dışsal mesaj göndermeyen ve almayan bir rol hedefini sahte etmenler ile doğrulanabilmesi mümkün değildir.

4.3.2 Bütünleştirme Test Etme

Önerdiğimiz hedef yönelimli test yaklaşımında bütünleştirme seviyesi testler sistem hedeflerinin doğrulanması ile uygulanmaktadır. Sistem hedefleri farklı rollerdeki rol hedeflerinin işbirliğini içerdiklerinden, sis-

tem hedefini doğrulamak için bu işbirliğinin sonuçları doğrulanmalıdır. Bir bütünleştirme test hedefi birim test hedeflerine benzer olarak kurucu, sistem hedefi (TAH olarak) ve bildirim hedeflerini içerir. TAH'ın alt-hedeflerinin farklı etmenlerde bulunabilmelerine rağmen, bütünleştirme testinde amaç kullanılan etmen çerçevesinin mesajlaşma altyapısını test etmek değildir. Bu yüzden işbirliğinin işlevsel doğruluğunu garanti altına almak için bütünleştirme test hedefi tek bir etmen içerisinde çalıştırılır.

Bütünleştirme testi ile birim testi arasındaki en önemli fark şudur; bütünleştirme testinin kurucu ve bildirim hedefleri alt-hedefler içerir. Bir bütünleştirme testi hedefinin kurucu hedefi, TAH'ın her alt-hedefinin ön koşullarını hazırlamalıdır. Benzer olarak, bildirim hedefinin alt-hedefleri de TAH'ın alt-hedeflerinin işlevselliğinin ve işbirliğinin doğruluğunu kontrol etmelidir.



Şekil 4.6. Bütünleştirme Test Etme

Şekil 4.6 örneğimizdeki ana sistem hedefini doğrulayan bir bütünleştirme test hedefi yapısını göstermektedir. Sistem hedefi *Müşteri Hedefi* ve *Sağlayıcı Hedefi* adında iki alt-hedef içerdiğinden *Kurucu Hedef* TAH'ın her alt-hedefinin ön koşullarını sağlayabilmek için iki bağlantılı alt-hedefe (*Müşteri ve Sağlayıcı Kurucu Hedefleri*) gereksinim duyar.

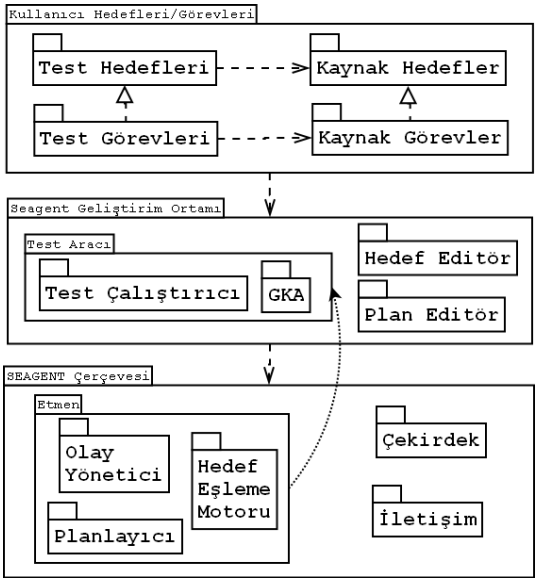
Bir bütünleştirme testi hedefinin çalıştırılması birim testlerdeki gibi bir hedef eşleme motoru ile yönetilir. Bütünleştirme test hedeflerinin çalıştırılması sırasında eşleme motoru iki bağımsız alt-hedef içeren kurucu hedefi çözümler ve bu alt-hedeflerin planlarını çalıştırır. Alt-hedeflerin planlarının tamamlanmasından sonra eşleme motoru *Kurucu Hedef* ve TAH arasındaki bağımlılıkları takip ederek tüm ihtiyaçları sağlanmış olan *Müşteri Hedefi*'ni başlatır. Bütünleştirme test hedefinin çalışması sırasında TAH'ın alt-hedefleri ve *Bildirim Hedefi'*nin alt-hedefleri arasındaki bağımlılıklar eşleme motoru tarafından yönetilir ve ilgili planlar bağımlılıkların gerektirdiği sıra içerisinde çalıştırılır.

4.4 SeaUnit Test Aracı

SeaUnit test aracı, tezde önerilen hedef yönelimli test yaklaşımındaki test hedeflerinin oluşturulması ve çalıştırılmasında geliştiricilere yardımcı olmak üzere geliştirilmiştir. Aracın mimarisi SEAGENT etmen geliştirim çerçevesi üzerine oturtulmuştur. Araç SEAGENT çerçevesinin sunduğu altyapıyı kullanarak test hedeflerini doğrudan çalıştırdığından, çalışma mekanizması oldukça basittir. SEAGENT çerçevesi SeaUnit mimarisinin gerçekleştirimi için kritik olan iki birime sahiptir. Bu birimler; planlama motoru ve hedef eşleme (*"matching"*) motorudur. Planlama motoru SGA paradigmasını temel alır ve rol hedeflerinin başarılması için SGA planlarını çalıştırır. Planlama motorunun mimarisi ile ilgili detaylar (Ekinci et al., 2007)'da bulunmaktadır. Eşleme motorunun temel sorumluluğu gelen istekleri yerine getirecek hedeflerin seçilmesi ve çalışmasının başlatılmasıdır. SEAGENT eşleme motoru, planlama motoru ile birlikte çalışarak

bir rol hedefinin alt-hedefleri arasındaki bağımlılıkları yönetebilir.

SeaUnit *Eclipse* geliştirim ortamı üzerine bir eklenti olarak gerçekleştirilmiş ve yine bir *Eclipse* eklentisi olan SEAGENT geliştirim ortamı içerisine dahil edilmiştir. Test aracının mimarisi ve SEAGENT çerçevesinin ilgili modülleri ile bağımlılıkları Şekil 4.7'de gösterilmektedir.



Şekil 4.7. SeaUnit test aracının mimarisi

Şekilde görüldüğü gibi, test aracının tüm gereksinimleri iki altpaket tarafından yerine getirilmektedir; *Test Çalıştırıcı* ve *Grafiksel Kullanıcı Arayüzü - GKA*. Bir test hedefinin çalıştırılması için, *Test Çalıştırıcı* SEAGENT çerçevesi içerisinde bir etmen örneği yaratır. Bu etmen, test hedefinin çalıştırılması sorumluluğunu üstlenir. Daha sonra, *Test Çalıştırıcı,* test hedefinin başarımı için bir amaç (*"objective"*) göndererek etmeni tetikler.

Test hedefinin çalışması sırasında, *GKA* etmen tarafından gönderilen olayları dinler. SEAGENT etmenindeki *Olay Yönetici* modülü tarafından oluşturulan bu olaylar, *GKA* tarafından toplanır. *Olay Yönetici* tarafından gönderilen sadece iki tip olay test hedefinin çalışmasının izlenmesi için kullanılır. Bu olaylar; *Görev Bitti* ve *Hatalı Görev*'dir. İlk değerlendirilen olay tipi, bir davranış veya eylem başarılı bir şekilde çalıştırıldığında gönderilen *Görev Bitti* olayıdır. Etmen tarafından gönderilen diğer olay olan *Hatalı Görev*, bir görevin çalışması sırasında hata oluştuğunu bildirmek için kullanılır. *Görev Bitti* ve *Hatalı Görev* olayları çalıştırılmakta olan hedefin başarılıp başarılmadığını belirtmektedirler. *GKA* olayları yakalar ve geliştiriciye gösterilmek üzere hata tiplerine dönüştürür. Bir görevin çalışması sırasında oluşan istisnalar hata veya başarısızlık olarak işaretlenebilir. Eğer atılan istisna bildirim (*"assertion"*) hatasının bir örneği ise başarısızlık olarak değerlendirilir.

Diğer taraftan, diğer tüm istisna türleri hata olarak değerlendirilir. Örneğin, bir eylem tarafından atılan *NullPointerException* hatası etmenin dinleyicilerine bir *Hatalı Görev* olayı gönderilmesine neden olur. *GKA* olayı yakalar ve hata olarak değerlendirir, Olay kaynağı bildirim hatası ise *GKA* tarafından bir başarısızlık olarak değerlendirilir ve hedef durumu başarısız olarak raporlanır.

Test aracının sadece test hedeflerinin durumlarını raporlamak için *Olay Yöneticisi*'ni dinlemesi sayesinde, planlama motorunun olaylarının doğrudan kullanılmaması sağlanmıştır. Bu tasarım, test aracını planlayıcının içsel çalışmasından bağımsız hale getirmiştir.

5 ÇOKLU ETMEN SİSTEMLERİNDE YENİDEN YAPILANDIRMA

Çoklu etmen sistemlerinin EYTGG gibi evrimsel geliştirim süreçleri kullanılarak başarılı bir şekilde geliştirimi, ancak geliştirilen ÇES'in tasarımının geliştirim süreci boyunca sürekli olarak iyileştirilmesiyle mümkün olabilir. EYTGG yaklaşımının önerdiği yinelemeli geliştirim döngüsünde sistem tasarımının güçlendirilmesi, döngünün son adımı olan yeniden yapılandırma adımı içerisinde gerçekleştirilmektedir.

Bu bölümde, evrimsel ÇES geliştirimi sırasında kullanılmak üzere (Tiryaki et al., 2008)'de önerilen bir yeniden yapılandırma yaklaşımı tanıtılmaktadır. Bu yaklaşım, geliştirim döngüleri arasında sistem tasarımında oluşan değişiklikleri yöneterek çoklu etmen sistemlerinin evrimsel bir süreç içerisinde geliştirimini mümkün kılar.

Çoklu etmen sistemleri için önerilen yeniden yapılandırma yaklaşımı, geleneksel yeniden yapılandırma (Fowler, 1999) yaklaşımını takip ederek ÇES geliştirimi için bazı yeniden yapılandırma desenleri sunar. Yaklaşım, geleneksel yeniden yapılandırma pratiğinin EYYM alanına aktarılabilmesi için, ÇES geliştirimi sırasında elde edilen yeniden yapılandırılabilir ürünler üzerinde üç yeniden yapılandırma seviyesi tanıtmaktadır. Bununla birlikte, bu sistemlerin geliştirilmesi sırasında karşılaşılan ortak problemler kötü kokular adıyla, bu problemlerle başa çıkmak üzere tanımlanan bakım teknikleri de yeniden yapılandırma desenleri adıyla tanıtılmaktadır. Tanıtılan yeniden yapılandırma desenlerinin her biri ÇES geliştirimi sırasında tasarımda karşılaşılan bir veya birden fazla kötü koku-

nun ortadan kaldırılması üzerine odaklanmıştır.

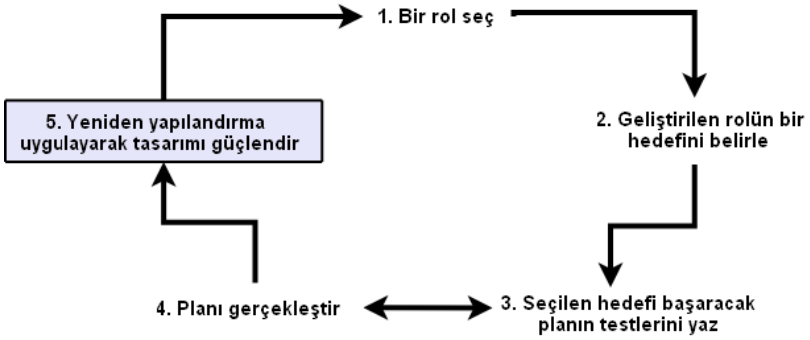
Bir sonraki alt-bölümde, yeniden yapılandırmanın EYTGG yaklaşımının önerdiği evrimsel geliştirim tarzı içerisindeki yeri ve önemi tartışılmaktadır. Daha sonraki alt-bölümde ise, ÇES geliştirimi için yeniden yapılandırma seviyeleri, bu sistemlerin tasarımlarında oluşabilecek kötü kokular ve yeniden yapılandırma desenleri gibi ÇES geliştiriminde yeniden yapılandırma yaklaşımının temel prensipleri tanıtılmaktadır.

5.1 Yeniden Yapılandırmanın Evrimsel ÇES Geliştirimi İçerisindeki Yeri

Bölüm 4'de evrimsel ÇES geliştirimini destekleyen etmen yönelimli test güdümlü geliştirim - EYTGG adındaki bir çevik geliştirim yaklaşımı tanıtılmıştır. EYTGG yaklaşımının yinelemeli ve artışlı geliştirim döngüsü Şekil 5.1'de gösterilmektedir. EYTGG, uç programlamanın test güdümlü geliştirim ve yeniden yapılandırma pratikleri üzerine kurulmuştur. Test güdümlü geliştirim tarzı ve bu geliştirim tarzını destekleyen SeUnit adında bir test aracı daha önceki bölümlerde tanıtılmıştır. Bu bölüm ise, EYTGG yaşam döngüsü içerisindeki "yeniden yapılandırma uygulayarak tasarımın güçlendirilmesi" adımı üzerine odaklanmıştır ve yeniden yapılandırmanın evrimsel ÇES geliştirimi içerisindeki yerini tanımlamayı amaçlamaktadır.

Yeniden yapılandırma adımına kadar, geliştirim döngüsünün üçüncü adımında geliştirilmek üzere seçilen hedefin doğruluğunu kanıtlayan testler yazılır, dördüncü adımda ise bu testleri geçecek ve eldeki

hedefi başarabilecek plan gerçekleştirilir. Bununla birlikte, geliştiriciler kod gerçekleştirimi sırasında sadece eldeki planın işlevselliği üzerine odaklandıklarından, geliştirilmekte olan sistemin tasarımı plan gerçekleştirimi sırasında kötüye gidebilir. Yeniden yapılandırma adımı uygulanmazsa, bu kötüye gidiş birkaç döngü sonra ÇES tasarımının yeni gereksinimleri karşılayamaz hale gelmesine neden olur.



Şekil 5.1. EYTGG döngüsü içerisinde yeniden yapılandırma

Geliştiriciler yeniden kullanılabilir plan parçaları ve geliştirilen ürünler üzerinde esnek tasarım gibi iyi tasarım fırsatlarını değerlendirerek tasarımın kötüye gidişini önlemelidirler. Yeniden yapılandırma adındaki bu süreç EYTGG döngüsünün beşinci adımında uygulanmaktadır.

Yeniden yapılandırma adımında geliştiriciler daha önceden tanımlanmış yeniden yapılandırma desenlerini kullanarak başlangıç tasarım kararlarını güçlendirebilirler. Örneğin, bir plan yapısı plan içerisindeki yeniden yapılandırılabilir görev(ler)in tanımlanması ile daha iyi bir yapıya

dönüştürülebilir veya sistemin modülerliğini ve sağlamlığını arttırmak için yeni roller veya roller arası etkileşimler tanımlanabilir.

Bazı yeniden yapılandırma operasyonları daha önceden bu operasyonun değiştirdiği içsel yapı ile bağımlı olarak yazılmış bazı testlerin de yeniden yapılandırılmasını gerektirebilir. Böyle bir durumda, geliştiriciler aynı roldeki aynı hedef veya yeniden yapılandırma operasyonundan etkilenen başka bir roldeki başka bir hedef için yeni bir döngü başlatmalı ve bu testleri döngünün üçüncü adımında yeniden düzenlemelidirler. Daha önceki döngülerde yazılmış olan plan kodu yeniden düzenlenen bu testleri geçemez ise geliştiriciler aynı zamanda dördüncü adımda bu testleri geçecek plan kodunu da gerçekleştirmelidirler. Bununla birlikte, kodda yapılan bu değişiklikler sistem tasarımını bozabileceğinden, sistem tekrar yeniden yapılandırmaya gereksinim duyabilir. Bu yüzden, bir yeniden yapılandırma operasyonu bir çok geliştirim döngüsünü tetikleyebilir.

5.2 Çoklu Etmen Sistemlerinde Yeniden Yapılandırma Yaklaşımının Temel Prensipleri

Yeniden yapılandırma pratiğinin etmen yönelimli yazılım mühendisliği alanına taşınabilmesi için cevaplanması gereken üç temel soru vardır:

- ÇES geliştirimi sırasında elde edilen geliştirim ürünlerinden hangileri yeniden yapılandırılabilir?
- ÇES geliştirimi sırasında yeniden yapılandırmaya nasıl ve ne zaman karar verilebilir?

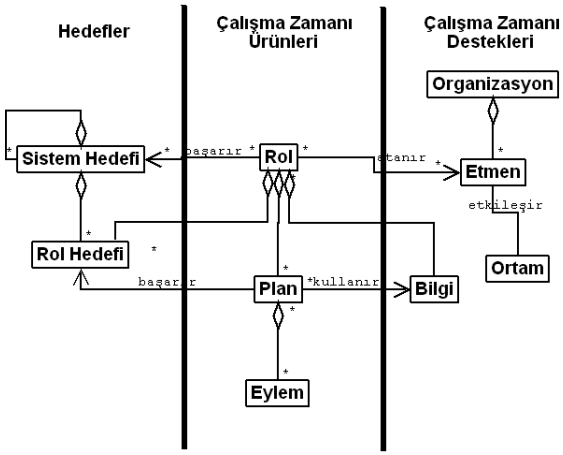
- ÇES geliştiriminde uygulanabilecek yeniden yapılandırma teknikleri nelerdir?

Bundan sonraki alt-bölümlerde bu soruların cevapları detaylı olarak tartışılmaktadır.

5.2.1 ÇES Geliştiriminde Yeniden Yapılandırma Seviyeleri

ÇES geliştirimi sırasında üzerinde yeniden yapılandırma uygulanabilecek geliştirim ürünlerinin belirlenebilmesi amacıyla, Şekil 4.1'de sunulan genel ÇES üst modeli içerisindeki kavramlar Şekil 5.2'de görüldüğü gibi hedefler, çalışma zamanı ürünleri ve çalışma zamanı destekleri adındaki üç dikey katmana yerleştirilmiştir. Sistem hedefi ve rol hedefi kavramları hedef katmanı içerisinde bulunmaktadır. Çalışma zamanı ürünleri katmanındaki elemanlar sistem ve rol hedeflerinin başarılması için geliştirilir. Çalışma zamanı destekleri katmanındaki elemanlar ise tanımlanan çalışma zamanı ürünlerinin çalışmasına yardımcı olurlar.

Hedef yönelimli yazılım geliştirme yöntemleri kullanıcı gereksinimlerinden gelen hedeflerin tanımlanması, analiz aşamasında bu yüksek seviyeli hedeflerin daha küçük hedeflere ayrıştırılması ve tasarım aşamasında bu hedeflerin başarımı için birlikte çalışan plan ve rol gibi çalışma zamanı ürünlerinin tanımlanması etkinlikleri üzerine kurulmuşlardır. Hedeflerin başarılması için, tanımlanan çalışma zamanı ürünleri çalışma zamanı destek birimleri kullanılarak çalıştırılırlar.



Şekil 5.2. Katmanlara ayrılmış ÇES üst modeli

Test açısından bakıldığında, hedeflerin başarımının doğrulanması için bu hedefleri başarmak üzere geliştirilmiş olan çalışma zamanı ürünlerini test edilmelidir. Örneğin, her rol hedefinin başarılması için bir veya birden fazla plan geliştirilir. Bu durumda, bir rol hedefinin başarılıp başarılmadığını anlayabilmek amacıyla bu hedefin başarımı için geliştirilmiş olan plan veya planların çalışması test edilmelidir. Bu yüzden, çalışma zamanı ürünlerinin çalışmasını kontrol ederek hedeflerin başarımının doğrulayacak testlerin yazımı için bir test altyapısına gereksinim duyulur. Bu altyapıyı sağlamak amacıyla önerilen ÇES üst modeli içerisindeki çalışma zamanı ürünleri için otomatikleştirilmiş testlerin yazılmasını sağlayan SeaUnit test aracı bölüm 4'de tanıtılmıştır. Bu test altyapısı yeniden yapılandırma yaklaşımımız için de bir zemin oluşturmaktadır.

Yeniden yapılandırma operasyonu yeniden yapılandırılan birimin dışsal davranışını değiştirmemesine rağmen, bu birimin içsel yapısına

bağımlı olan diğer tasarım birimleri bu yeniden yapılandırma operasyonundan etkilenebilirler. Böylece, yeniden yapılandırma hatasız çalışan kodu etkileyerek sistemi bozabilir. Yeniden yapılandırma pratiğinin uygulanabilmesi için sistem güvenilirliğinin daha önceden tanımlanmış otomatik testler ile garanti altına alınması gerekir. Bu yüzden, yeniden yapılandırma çalışma zamanı ürünleri katmanındaki somut ve test edilebilir ürünler üzerinde uygulanabilir. Çalışma zamanı ürünleri üzerinde uygulanan yeniden yapılandırmalar bu ürünlerin dışsal davranışını değiştirmeyeceğinden, bu ürünler tarafından gerçekleştirilen sistem ve rol hedeflerinin başarımını etkilememelidir.

Tez kapsamında Şekil 5.2'de sunulan üst model temel alınarak ÇES geliştirimi için test edilebilir çalışma zamanı ürünleri üzerinde odaklanan üç yeniden yapılandırma seviyesi tanımlanmıştır. Bu bölümün devamında bu yeniden yapılandırma seviyeleri detaylı olarak anlatılmaktadır.

5.2.1.1 Rol Seviyesi

Roller birlikte çalışarak sistem hedeflerini başaran mimarisel elemanlardır. Her rol, sistem hedeflerinden gelen bazı sorumluluklara (rol hedefleri), yeteneklere (planlar) ve kurallara sahiptir. Yükleme (*"deployment"*) aşamasında her rol çalıştırılmak üzere etmen veya etmenlere atanır. Bu yüzden, bir etmenin tüm özellikleri kendisine atanan rollerden gelir. Etmen geleneksel nesne yönelimli geliştirimdeki nesne gibi sadece geliştirilen rollerin çalıştırılması için kullanılan bir taşıyıcıdır.

Rollerin sorumlulukları ve yetenekleri dinamik ve açık sistemler olan çoklu etmen sistemlerinde sürekli olarak değişebilir. Bu yüz-

den, çoklu etmen sistemleri içerisindeki yeniden yapılandırmalar için en önemli elemanlardan biridir. Bu seviyedeki *Sorumluluk Taşı* gibi yeniden yapılandırma teknikleri geliştirilen ÇES'in rol yapısının güçlendirilmesini sağlar. Farklı roller, geliştirilmekte olan sistemin ortak bir sistem hedefini başarmak üzere birlikte çalışabilirler. Roller üzerinde uygulanan yeniden yapılandırma teknikleri rollerin dışsal davranışlarını değiştirmediğinden, sistem hedeflerinin başarımı yeniden yapılandırma operasyonlarından etkilenmez.

5.2.1.2 Plan Seviyesi

Etmenler hedeflerini planların çalışması yoluyla başarırlar. Planlar geleneksel NYG'deki sınıflar gibi ÇES geliştirimi için en küçük test edilebilir ve yeniden yapılandırılabilir birimlerdir. Plan testi, bir hedefin çalışmasını tek bir etmen bağlamında doğrular. Etmen içeriği, etmen bilgi tabanını ve etmenin doğrudan etkileştiği dış ortamı da içerebilir. Bu yüzden, etmen planları üzerinde uygulanan yeniden yapılandırmalar etmen bilgi tabanını ve/veya yeniden yapılandırılan plan veya planlarla ilgili dış ortamı etkileyebilir.

Bir planın bir alt-bölümü farklı bir hedefi başarmak için başka bir plan içerisinde kullanılabilir. Bu durumda, planlar farklı plan yapıları içerisinde aynı görev grubunun tekrarlanmasını azaltacak, yeniden kullanılabilir bir yaklaşım ile yapılandırılmalıdırlar. Bu seviyedeki yeniden yapılandırma teknikleri planların iç yapılarının güçlendirilmesini amaçlamaktadırlar. Yeniden yapılandırma operasyonu sonucunda planların dışsal davranışları değişmediğinden, bu planların başardığı rol hedefi için

yazılmış olan testler yeni yapının doğruluğunun kontrol edilmesi için kullanılabilirler.

Etmen planları SGA gibi bir planlama paradigması kullanılarak geliştirilirler. Bu seviyedeki yeniden yapılandırma teknikleri kullanılan planlama paradigmasına bağımlıdır. Bununla birlikte, bu seviyedeki yeniden yapılandırma tekniklerinin çoğu diğer benzer planlama paradigmaları için genelleştirilebilir.

5.2.1.3 Eylem Seviyesi

Planlar eylem adı verilen çalıştırılabilir görevlerin bir planlama paradigması kullanılarak sıralandırılmalarıyla oluşturulurlar. Eylemleri geleneksel NYG'deki metotlar olarak düşünebiliriz.

ÇES geliştirimi içerisinde en küçük test edilebilir birim plan olmasına rağmen, bazen eylemler NYG'deki metotlar gibi test edilebilir en küçük birimler olarak değerlendirilebilirler. Bir eylem farklı plan yapıları içerisinde kullanılabilir. Bunun yanı sıra, geliştirilmekte olan sistemde "DF'ye bir kitapçı etmeni kaydetme", "DF'ye bir kullanıcı etmeni kaydetme" ve "DF'ye bir pazarlıkçı etmen kaydetme" eylemleri gibi benzer çalıştırılabilir koda ve/veya yapıya sahip bir çok eylem bulunabilir. Böyle durumlarda, geliştiriciler eylem kodlarının ve yapılarının tekrarlanmasını önlemek için yeniden kullanılabilir eylemler yazmalıdırlar. Bu amaçla, tekrar eden eylem kodlarını önlemek için eylem seviyesi yeniden yapılandırma teknikleri kullanılabilir. Örneğin, yukarıda bahsedilen durumdaki "tekrar eden eylem kodu ve yapısı" problemi tekrar eden kodun ve yapının "bir etmenin DF'ye kaydedilmesi" adındaki soyut bir eyleme taşın-

ması ve bu soyut eylemin diğer eylemler tarafından genişletilmesi yoluyla çözülebilir.

5.2.2 Etmen Sistemlerde Kötü Kokular

Bazı ortak tasarım problemleri sistem geliştirimi sırasında geliştiriciler tarafından sıklıkla karşılaşılır. Fowler (Fowler, 1999)'de bu ortak problemleri tasarımda kötü kokular olarak adlandırmış ve yazılımların tasarımındaki bu kötü kokuların üstesinden gelmek için yeniden yapılandırma teknikleri tanıtmıştır. Tanımlanan her yeniden yapılandırma deseni bir veya birden fazla kötü kokunun ortadan kaldırılmasını amaçlamaktadır. Benzer olarak, ÇES geliştirimi sırasında yeniden yapılandırma sürecinin ne zaman başlayacağına karar verebilmek için ÇES geliştirimindeki kötü kokular tanımlanmalı ve daha sonra bu kötü kokuların üstesinde gelecek ilgili yeniden yapılandırma desenleri ortaya konmalıdır.

SEAGENT çerçevesi kullanılarak farklı ÇES uygulamalarının geliştirimi sırasında elde ettiğimiz deneyimler sonucunda, bir önceki bölümde bahsedilen ÇES geliştirimindeki yeniden yapılandırma seviyelerinin her biri için aşağıda sıralanan kötü kokular tanımlanmıştır:

Rol seviyesi kötü kokular: *Yanlış Sorumluluk, Aşırı Yüklenmiş Rol.*

Plan seviyesi kötü kokular: *Büyük Davranış, Plan İçerisinde Çalışma Kararı, İş Yavaşlatan Görev Zinciri, Tekrarlanan Davranış Yapısı, Anlamsız Davranış.*

Eylem seviyesi kötü kokular: *Eylem Kodu ve/veya Parametresi Tekrarı, Büyük Eylem, Eylem İçerisinde Akış Kararı.*

Bu kötü kokuların ayrıntılı tanımlamaları Ek A'de detaylı olarak verilmiştir.

Evrimsel bir geliştirim sürecinin her geliştirim döngünündeki gerçekleştirim aşaması sırasında, geliştiriciler sadece daha önceden tanımlanmış olan testleri geçecek kodun yazımı üzerine odaklanırlar. Bu yüzden, gerçekleştirim sırasında sistem tasarımında bazı kötü kokuların oluşabilmesi kaçınılmazdır. Bu kötü kokular döngünün gerçekleştirimden sonraki adımında yakalanabilir. Önerdiğimiz EYTGG yaklaşımı içerisinde, kötü kokular geliştiriciler tarafından geliştirim döngüsünün "yeniden yapılandırma uygulayarak tasarımı güçlendir" adındaki son adımında yakalanmaktadır.

5.2.3 ÇES Yeniden Yapılandırma Desenleri

Bu bölümde, ÇES geliştirimindeki çalışma zamanı ürünleri üzerinde uygulanarak bir önceki bölümde tanıtılan kötü kokuların üstesinden gelecek yeniden yapılandırma desenlerini tanımlanmaktadır. ÇES geliştirimi sırasında kullanılabilecek yeniden yapılandırma kalıpları, Fowler tarafından (Fowler, 1999)'de kullanılan tanımlama standardı kullanılarak tanımlanmıştır. Bu tanımlama standardı beş bölüm içermektedir; isim, özet, motivasyon, mekanikler ve örnek. Farklı ÇES uygulamalarının geliştirimi sırasında elde ettiğimiz deneyimlerimiz sonucunda tanımladığımız yeniden yapılandırma teknikleri tetikleyici kötü kokularıyla birlikte tablo 5.1'de sunulmuştur.

Tablo 5.1. ÇES geliştirimi için tanımlanan yeniden yapılandırmalar

Seviye	Yeniden Yapılandırma	Kötü Koku
Rol Seviyesi	Sorumluluk Taşı	Aşırı Yüklenmiş Rol Yanlış Sorumluluk
	Rol Böl	Aşırı Yüklenmiş Rol
Plan Seviyesi	Plan Çıkarımı	Plan İçerisinde Çalışma Kararı
	Davranış Çıkarımı	Büyük Davranış Tekrarlanan Davranış Yapısı
	Davranıştan Plana	Plan İçerisinde Çalışma Kararı
	Davranış İsmini Değiştir	Anlamsız Davranış
	Görevleri Bir Görev İle Yer Değiştir	İş Yavaşlatan Görev Zinciri
	Davranıştan Kurtar	Anlamsız Davranış
	Süper-Davranış Çıkarımı	Tekrarlanan Davranış Yapısı
Eylem Seviyesi	Eylem Böl	Büyük Eylem Eylem İçerisinde Akış Kararı
	Süper-Eylem Çıkarımı	Eylem Kodu ve Parametresi Tekrarı

ÇES geliştirimi için önerilen yeniden yapılandırma yaklaşımı içerisinde tanımlanan tüm yeniden yapılandırma desenlerinin Fowler'ın yeniden yapılandırma desenleri tanımlama standardı kullanılarak oluşturulmuş detaylı tanımlamaları daha önce tanıtılan üç yeniden yapılandırma seviyesine ayrılmış olarak ek-1, ek-2 ve ek-3'de sunulmaktadır.

Bununla birlikte, yeniden yapılandırma yaklaşımımız hakkında fikir vermesi amacıyla, her yeniden yapılandırma seviyesinden bir yeniden yapılandırma deseninin isim ve özetini içeren kısa tanımlaması aşağıda listelenmiştir;

Sorumluluk Taşı (rol seviyesi): Farklı bir rol tarafından başarılması gereken bir rol hedefiniz olduğu durumlarda kullanılır. Böyle bir durumda, hedefi eşleştirildiği planlar ve bu planlar tarafından kullanılan bilgi

tabanları ile birlikte başka bir role taşı

Davranışları Birleştir (plan seviyesi): Ortak bir hedef için çalışan bazı anlamsız davranışlar olduğu durumlarda kullanılır. Ortak hedefi göz önüne alarak isimlendirilen yeni bir davranış oluştur ve orijinal davranışlardaki görevleri yeni davranış yapısına taşı.

Süper-Eylem Çıkarımı (eylem seviyesi): Aynı çalıştırılabilir kod parçasının ve\veya parametrelerin bir çok eylem içerisinde tekrarlandığı durumlarda kullanılır. Böyle bir durumda tekrarlanan kodu ve\veya parametreleri tutan soyut bir eylem oluştur ve orijinal eylemlerden bu soyut eylemi genişlet.

Bazı üst seviye yeniden yapılandırma desenleri mekanikleri içerisinde daha alt veya aynı yeniden yapılandırma seviyesinde bulunan diğer yeniden yapılandırma desenlerinden bazılarını içerebilir. Örneğin; rol seviyesindeki *Rol Böl* yeniden yapılandırma deseni, mekanikleri yine rol seviyesindeki *sorumluluk taşı* isimli diğer bir yeniden yapılandırmayı içeren üst seviye bir yeniden yapılandırma desenidir.

6 SEAGENT ÇERÇEVESİ İÇİN BİR YENİDEN YAPILANDIRMA ARACI: RESEAGENT

ÇES geliştirimi sırasında yeniden yapılandırma pratiğinin kullanımını kolaylaştırmak için SEAGENT çerçevesi üzerine bir yeniden yapılandırma aracı gerçekleştirilmiştir. ReSeagent adındaki yeniden yapılandırma aracı, Eclipse[1] ortamındaki yeniden yapılandırma desteğine benzer şekilde, SEAGENT geliştirim ortamı içerisinde bulunan SEAGENT plan editörü ve SEAGENT rol-hedef editörü üzerine geliştirilmiştir. Araç bu editörlerde oluşturulan modellerin yeniden yapılandırılmasını desteklemektedir.

Bir önceki bölümde ÇES geliştiriminde kullanılmak üzere önerilen yeniden yapılandırma yaklaşımı genel bir yaklaşımdır. Bununla birlikte, yaklaşım içerisinde tanıtılan plan seviyesi yeniden yapılandırmalar plan geliştirim için kullanılan planlama paradigmasına, rol seviyesi yeniden yapılandırmalar ise sistemdeki hedeflerin modellenmesi için kullanılan rol-hedef üst modeline bağımlıdır. Çoklu etmen geliştirim çerçeveleri ve yöntemleri farklı planlama paradigmaları ve farklı rol-hedef üst modelleri kullanabilmektedir. Genel bir yaklaşım olarak tanıtılan yeniden yapılandırma yaklaşımını destekleyecek bir yeniden yapılandırma aracının geliştirimi için, ilk olarak hedeflenen plan ve rol-hedef üst modelleri belirlenmelidir.

Biz bu noktada, daha önceden ÇES geliştirim deneyimimiz olan, plan ve rol-hedef üst modelleri net olarak belli olan, plan ve rol-hedef

[1] http://www.eclipse.org/ , son erişim: 10-Ağustos-2008

modellerinin geliştirim süreçlerini birer araç ile destekleyen SEAGENT çerçevesi üzerine bir yeniden yapılandırma aracı geliştirmeye karar verdik. Bununla birlikte, geliştirilen yeniden yapılandırma aracı farklı plan ve rol-hedef üst modelleri kullanan diğer etmen geliştirim çerçeveleri ve yöntemleri için de kullanılabilir. Böyle bir durumda, kullanılan plan ve/veya rol-hedef üst modeli için yeniden yapılandırma planlarının oluşturulması gerekir.

ReSeagent yeniden yapılandırma aracı SEAGENT geliştirim ortamından aldığı yeniden yapılandırma istekleri karşısında daha önceden tanımlanmış yeniden yapılandırma tekniklerini ilgili model üzerinde uygumak üzere tasarlanmıştır. Araç içerisinde, yeniden yapılandırma teknikleri SGA planları olarak tanımlanmaktadır. Yeniden yapılandırm tekniklerinin SGA planları şeklinde tanımlanması devam eden alt-bölümlerde ayrıntılı bir şekilde tartışılmaktadır.

Desteklenen her yeniden yapılandırma için en az bir yeniden yapılandırma planı bulunmaktadır. Bir yeniden yapılandırma için birden fazla plan bulunduğu durumda, yeniden yapılandırma isteğinin yapıldığı andaki duruma göre (istek ile gönderilen girdiler gibi kriterler göz önüne alınarak) bu planlardan en uygunu yeniden yapılandırma operasyonu için seçilir.

6.1 Model Dönüşümü ve Yeniden Yapılandırma

Model güdümlü mühendislik - MGM (*"Model Driven Engineering" - MDE*) yazılım mühendisliği alanı içerisinde, modellerin birinci sınıf varlıklar olduğunu kabul eden ve yazılımların modeller arasında çeşitli model dönüşümleri uygulanarak geliştirilmesi amaçlayan bir disiplindir

(Mens and Gorp, 2006). Model dönüşümü konusu, bu disiplin tarafından önerilen model güdümlü geliştirim tarzının uygulanması için en önemli konulardan biridir (Sendall and Kozaczynski, 2003). Bu güne kadar, farklı tipte modeller üzerine kurulan birçok model dönüşüm yaklaşımı tanıtılmıştır. Bu yaklaşımlar farklı kriterler göz önüne alınarak çeşitli şekillerde sınıflandırılmıştır.

(Mens and Gorp, 2006)'de model dönüşümü için tanımlanan yaklaşımların ve araçların ortak özellikleri temel alınarak gruplandırılmasına izin veren bir taksonomi (sınıflandırma yöntemi) önerilmektedir. Bu amaç doğrultusunda, model dönüşüm yaklaşımları, dilleri ve araçlarının değerlendirilebilmesi için gerekli ölçütler ortaya konmaktadır. Her ölçüt, bu ölçütü sağlayan model dönüşüm yaklaşımlarının birlikte gruplandırılması için kullanılmıştır. Bu çalışmada model dönüşümü ile ilgili verilen tanımlamalar aşağıda sunulmuştur:

Model: Sistem olarak adlandırılan dünyanın bir bölümünün basitleştirilmiş bir gösterimi veya soyut bir tanımıdır (Bézivin and Gerbé, 2001; Seidewitz, 2003). Mühendislik bağlamında, bir model sistem hedef-lerine ulaşmak için gerekli olan uygun eylemlere karar verilmesine yardımcı oluyorsa faydalı olur.

Model dönüşümü: Bir dönüşüm tanımlaması kullanılarak bir kaynak modelden bir hedef modelin otomatik üretilmesidir (Kleppe et al., 2003). Her model dönüşümünde kaynak ve hedef model birbirine denktir.

Dönüşüm tanımlaması: Kaynak modelin nasıl hedef modele

dönüştürülebileceğini birlikte tanımlayan dönüşüm kurallarının bir setidir.

Dönüşüm kuralı: Kaynak model içerisindeki bir veya birden fazla yapının hedef model içerisindeki bir veya daha fazla yapıya nasıl dönüştürülebileceğinin bir tanımlamasıdır.

Model güdümlü geliştirim - MGG (*"Model Driven Development" - MDD*) açısından bakıldığında, yeniden yapılandırma başlangıç model ile güçlendirilmiş model arasındaki bir model dönüşümüdür. Yeniden yapılandırmalar içerisinde tanımlanan mekanikler ise, birlikte kullanılarak model dönüşümünü oluşturan dönüşüm kurallarıdır. Aşağıda, yeniden yapılandırmanın model dönüşümü alanı içerisindeki yeri (Mens and Gorp, 2006)'de model dönüşüm yaklaşımlarının sınıflandırılması için tanımlanan ölçütler kullanılarak tartışılmaktadır.

Kaynak ve hedef model sayısı: Yukarıda model dönüiümü için verilen tanım bazı yaklaşımlarda kaynak ve hedef modelin birden fazla olabileceği şeklinde genellenmiştir. Bu yüzden, model dönüşüm yaklaşımları kaynak ve hedef model sayılarına göre gruplandırılabilmektedir.

Yeniden yapılandırma bir modelin güçlendirilerek daha sağlam bir model elde edilmesi fikrine dayanır. Bu durumda "kaynak ve hedef model sayısı" ölçütünü göz önüne aldığımızda yeniden yapılandırma tek bir kaynak modelden yine tek bir hedef model elde etmeyi amaçlayan bire-bir bir model dönüşüm tipidir.

Teknik alan: Bir teknik alan belirli bir teknoloji için ilişkilendirilmiş kavramlar, araçlar, mekanizmalar, teknikler, diller ve formalizmleri içeren bir model yönetim çerçevesidir. Bir teknik alan, kullanılan

üst üst-model ile belirlenir. Örneğin, yaygın olarak kullanılan XML teknik alanı için üst üst-model *XML Schema* modelidir.

Bir model dönüşümünde kaynak model ve hedef model farklı teknik alanlara sahip olabilir. Böyle bir durumda, teknik alanlar arasında köprü oluşturacak dönüşümleri tanımlayacak araçlara ve tekniklere gereksinim duyulur.

Yeniden yapılandırma birçok geliştirim alanı içerisinde farklı üst üst-modeller için kullanılmaktadır. Bu yüzden, yeniden yapılandırmaların teknik alanı modellerin oluşturulmasında kullanılan üst üst-modele göre farklılık gösterir. Bununla birlikte, SEAGENT çerçevesi içerisinde plan ve rol-hedef modeller veb ontoloji dili - VOL (*"Web Ontology Language" - OWL*)[2] üst üst-modeli kullanılarak oluşturulur ve bu üst üst-modeli temel alarak oluşturulan VOL ontolojilerinde depolanırlar. Bu durumda, SEAGENT modelleri üzerindeki model dönüşümlerinin teknik alanı VOL üst üst-modeli tarafından belirlenir.

İçten veya dıştan kaynaklanan olması: Modelleme dilinin sözdizimi ve anlamsallığı bir üst-model tarafından ifade edilir. Model dönüşümü içerisindeki kaynak ve hedef modellerin oluşturulduğu diller temel alınarak dönüşümler içten ve dıştan kaynaklanan dönüşümler olmak üzere ayrılabilir. İçten kaynaklanan dönüşümler kaynak ve hedef modellerinin aynı dil ile ifade edildği dönüşümlerdir. Dıştan kaynaklanan dönüşümler ise farklı diller ile ifade edilen modeller arasındaki dönüşümlerdir.

Yeniden yapılandırma bir dil ile ifade edilen modelin güçlendiri-

[2]http://www.w3.org/TR/owl-features/ , son erişim: 12-Ağustos-2008

lerek aynı dil ile tanımlanan yeni bir modele dönüştürülmesidir. Bu durumda, yeniden yapılandırma kaynak ve hedef modellerin aynı dil ile oluşturulduğu içten kaynaklanan model dönüşüm tiplerinden biridir.

Yatay veya dikey olması: Yatay dönüşümler, kaynnak ve hedef modelleri aynı soyutlama seviyesinde bulunan dönüşümlerdir. Dikey dönüşümler ise kaynak ve hedef modelleri farklı soyutlama seviyelerinde bulunan dönüşümlerdir. Örneğin, tasarım modellerinden program kodu üreten bir model dönüşümü dikey model dönüşümü olarak değerlendirilir.

Üst seviye soyutlamada bulunan bir kaynak modelden daha alt seviye soyutlamada bulunan bir modele dönüşüm yeniden yapılandırmanın amaçlarından biri değildir. Yeniden yapılandırma hedef ve kaynak modelleri aynı soyutlama düzeyinde olan bir yatay model dönüşümüdür.

Sözdizimsel veya anlamsal olması: Bu ölçüt sadece sözdizimsel dönüşümler yapan model dönüşümleri ile modellerin sözdizimi yanısıra anlamsallıklarını da dönüştüren daha karmaşık model dönüşümlerini birbirinden ayırmak için kullanılmaktadır.

Yeniden yapılandırma kaynak modelden yola çıkılarak anlamsal olarak daha düzgün yapıdaki bir modelin elde edilmesini amaçladığından bu ölçüte göre anlamsal model dönüşümü tipindedir.

(Porres, 2005)'de bir kural tabanlı model yeniden yapılandırma yaklaşımı tanıtılmaktadır. Bu çalışmada, model dönüşümleri kaynak ve hedef modellerin üst-modellerine göre eşleme ve güncelleme dönüşümü olarak iki gruba ayrılmıştır. Eşleme dönüşümü, modelleri farklı olan iki modelin elemanlarının birbirlerine eşlenmesi üzerine odaklanır. Güncelleme dönüşümü ise bir modelin elemanlarının yapısının güncellenmesi

üzerine odaklanmıştır. Bu yüzden, güncelleme dönüşümünde kaynak ve hedef modelin üst modelleri aynıdır. Diğer taraftan, bir yeniden yapılandırma operasyonu başlangıç modelin güçlendirilmiş modele güncellenmesi sürecidir. Bu sınıflandırmaya göre yeniden yapılandırmanın başlangıç ve hedef modeller arasındaki güncelleme dönüşümüdür.

6.1.1 Model Dönüşüm Kuralları

Her dönüşüm kuralı sağ taraf ve sol taraf olmak üzere iki bölümden oluşmaktadır. Kuralın sol tarafı kaynak modele erişir, sağ taraf ile hedef modeli genişletir (Czarnecki and Helsen, 2003). Her iki bölümde aşağıdakilerin bir karışımı olarak tanımlanabilir:

Değişkenler: Değişkenler, kaynak ve hedef modelden gelen elemanları tutarlar.

Desenler: Desenler, 0-n değişken içeren model parçalarıdır. Desenler kaynak ve hedef modellerin diline göre soyut veya somut sözdizimi kullanılarak belirtilir ve bu sözdizimi metinsel veya grafiksel olabilir.

Mantık: Mantık, model elemanları üzerindeki sabitlemeleri ve hesaplamaları gösterir. Mantık çalıştırılabilir veya çalıştırılamaz biçimde olabilir. Çalıştırılamayan mantık modeller arasındaki bir ilişkiyi göstermek için kullanılır. Çalıştırılabilir mantık, dekleratif (*"declarative"*) veya imperatif (*"imperative"*) biçimlerde olabilir. İmperatif mantık çoğunlukla programlama dili kod çağırma deposu biçimine sahiptir. Bu yüzden, imperatif mantık kullanılarak oluşturulan kuralların işletilmesi için temel alınan programlama diline özgün yorumlayıcılara gereksinim duyulur.

Dönüşüm mekanizmaları arasındaki temel farklılık hangi tip man-

tık üzerine kurulduğudur. Dekleratif yaklaşımlar "ne?" sorusu üzerine odaklanır. Bu tür yaklaşımlar kaynak ve hedef model arasında ilişkiler tanımlayarak "neye dönüşüm için ne gerekli?" sorusunu cevaplamaya çalışır. İmperatif yaklaşımlar ise "nasıl?" sorusu üzerine odaklanır ve kaynak modelden hedef modele götüren adımları tanımlayarak dönüştürümün nasıl işletilmesi gerektiği sorununa cevap arar.

(Akehurst and Kent, 2002) gibi dekleratif yaklaşımlar, bir çıkarsama motoru tarafından desteklenen otomatik iki yönlülük ve izlenebilirlik gibi servislerinden dolayı daha ilgi çekicidirler. Dekleratif yaklaşımın dönüşüm dilleri içerisinde kaynak model içerisinde gezinme, hedef modelin yaratılması ve kuralların sıralanması için kullanıldığı birçok çalışma bulunmaktadır. Bu nedenlerde dolayı, dekleratif dönüşümlerin oluşturulması ve anlaşılması daha kolaydır.

(Sprinkle et al., 2003) gibi imperatif yaklaşımlar ise dekleratif yaklaşımların servislerini garanti edemediği dönüşümlerin gerçekleştirilmesi için gerekli olurlar. Özellikle, bir dönüşüm setinin uygulama sırasının net olarak kontrol edilmesi gerektiği durumlarda, bir imperatif yaklaşım içerdiği sıralama, seçim ve yineleme fikirleri nedeniyle daha uygun olabilir.

Bir dekleratif yaklaşım için kullanılabilecek paradigmalardan biri fonksiyonel programlamadır. Böyle bir yaklaşımda, her dönüşüm bazı girdileri (kaynak model) çıktılara (hedef model) dönüştüren bir fonksiyondur. Dekleratif yaklaşım için kullanılabilecek diğer bir paradigma da mantıksal programlamadır. Bir mantıksal programlama dili model dönüşümünü doğrudan ilgilendiren geriye dönme (*"backtracking"*) ve birleştirme gibi

bazı özelliklere sahiptir.

ReSeagent aracının hedef modelleri olan SEAGENT plan ve rol-hedef modelleri tanımlama mantığı (*"description logic"*) tabanlı VOL dili kullanılarak oluşturulduğundan, bu modeller üzerinde işletilecek dönüşüm kurallarının dekleratif yaklaşım ile mantık tabanlı bir kural dili kullanılarak tanımlanması uygundur. Bir sonraki alt-bölümde, VOL modelleri üzerinde işletilmek üzere tasarlanan mantık tabanlı bir kural dili olan anlamsal veb kural dili" - AVKD (*"semantic web rule language" - SWRL*) detaylı olarak incelenmiştir.

6.1.2 Anlamsal Veb Kural Dili - AVKD

2004 yılında Dünya Geneli Veb Konsorsiyumuna - DGVK (*"World Wide Web Consortium" - W3C*)[3] sunulan anlamsal veb kural dili" - AVKD (*"semantic web rule language" - SWRL*)[4] son yıllarda en yaygın kullanılan kural dillerinden birisidir. AVKD, ontolojiler içerisinde tanımlanan özellik (*"property"*) ve örnekleri (*"individuals"*) kullanarak OWL ontolojilerini genişleten kurallar yazılması amacıyla tasarlanmıştır.

AVKD sözdizimi (Sean et al., 2004)'de tanımlanan VOL soyut sözdizimini genişletir. VOL kullanılarak oluşturulan bir ontoloji aksiyom (*"axiom"*) ve gerçeklerin (*"facts"*) bir sıralamasını içerir. Aksiyomlar *subClass* ve *equivalentClass* gibi çeşitli türlerde olabilir. AVKD, VOL aksiyomunu genişleten kural aksiyomunu kullanmaktadır.

$$aksiyom ::= kural$$

[3]http://www.w3.org/ , son erişim: 02-Ağustos-2008

[4]http://www.w3.org/Submission/SWRL/ , son erişim: 03-Ağustos-2008

AVKD sözdiziminde kuralın sol tarafı öncül (*"antecedent"*), sağ tarafı ise sonuç (*"consequent"*) olarak tanımlanmaktadır. Bir kural aksiyomu her biri bir cümle setini içeren öncül ve sonuç bölümlerinden oluşur.

$$kural ::=' belirtir\,([URIreferans]\,\{aciklama\}\,oncul\;izleyici')'$$
$$oncul ::=' oncul\,\left('\,\{cumle\}'\right)'$$
$$izleyici ::=' izleyici\,\left('\,\{cumle\}'\right)'$$

Bir kural, öncül bölümü içerisinde belirtilen koşullar sağlandığında, sonuç bölümü içerisinde belirtilen koşulların da sağlanması gerektiği şeklinde okunabilir. Öncül ve sonuç bölümleri bir veya daha fazla cümleden oluşabileceği gibi hiç bir cümle de içermeyebilir. Boş bir öncül, sonuç bölümü içerisindeki koşulların her durumda sağlandığını gösterir. Boş bir sonuç bölümü ise, öncüldeki durum ne olursa olsun herhangi bir yeni koşulun üretilmeyeceğini belirtir. Boş öncül bölümü içeren kurallar, koşula bağlı olmayan gerçekleri belirtmek için kullanılabilir. Boş olmayan öncül ve izleyici bölümleri içerdikleri cümlelerin kesişimi olarak ele alınırlar. Bununla birlikte, birleşim içeren bir sonuç bölümüne sahip kurallar, her biri atomik sonuç bölümlerine sahip çoklu kurallara dönüştürülmelidirler.

$$cumle ::= tanimlama'\,('i-nesne')'$$
$$|\; dataRange'\,('d-nesne')'$$
$$|\; individualvaluedPropertyID'\,('i-nesne')'$$
$$|\; datavaluedPropertyID'\,('i-nesne\;d-nesne')'$$
$$|\; sameAs'\,('i-nesne\;d-nesne')'$$
$$|\; differentFrom'\,('i-nesne\;i-nesne')'$$
$$|\; builtIn'\,\left('builtinID\,\{d-nesne\}'\right)'$$
$$builtinID ::= URIreferans$$

Kural cümleleri $C(x)$, $P(x,y)$, $sameAs(x,y)$ veya $differentFrom(x,y)$ biçiminde olabilir. Bu cümlelerde C bir VOL tanımlaması, P bir VOL özelliğidir. x ve y değişken, VOL örneği (*"individual"*) veya VOL veri değeri (*"data value"*) olabilir.

Bir $C\,(x)$cümlesi, x'in C sınıf tanımlaması veya veri aralığının bir örneği olduğu durumda geçerli olur. Bir $P\,(x,y)$ cümlesi, x'in P özelliği ile y'ye bağlı olduğu durumlarda geçerlidir. Bir $sameAs\,(x,y)$cümlesi x'in y ile aynı nesne olarak yorumlandığı durumda doğru sonucunu verir. Benzer olarak, bir $differentFrom\,(x,y)$cümlesi x ve y'nin farklı nesneler olarak yorumlandığı durumlarda geçerlidir.

$$i-nesne ::= i-degisken \mid individualID$$

$$d-nesne ::= d-degisken \mid dataLiteral$$

Cümleler örneklere, veri literalllerine (*"data literals"*), örnek değişkenlerine (*"individual variables"*) veya veri değişkenlerine (*"data variables"*) referans içerebilir. Değişkenler, kapsamları kural için sınırlanmış evrensel nicelikler (*"universally quantified"*) olarak ele alınırlar. Doğal olarak, sadece kuralın öncül bölümünde bulunan değişkenler sonuç kısmında bulunabilir.

$$i-degisken ::=' I-degisken\,('URIreferans')'$$

$$d-degisken ::=' D-degisken\,('URIreferans')'$$

6.1.3 ReSeagent ve AVKD

SEAGENT çerçevesi plan ve hedef modellerinin geliştirimi için VOL dilini kullandığından, yeniden yapılandırma aracının geliştiriminin başlarında bu platform için tanımlanacak yeniden yapılandırmaların VOL ontoloji-

leri için kural yazmayı sağlayan AVKD kullanılarak tanımlanan bir kural seti olarak tanımlanabileceği düşünülmüştür. Fakat, AVKD'nin yeniden yapılandırma mekaniklerinin tanımlamak için aşağıda sıralanan eksikleri vardır:

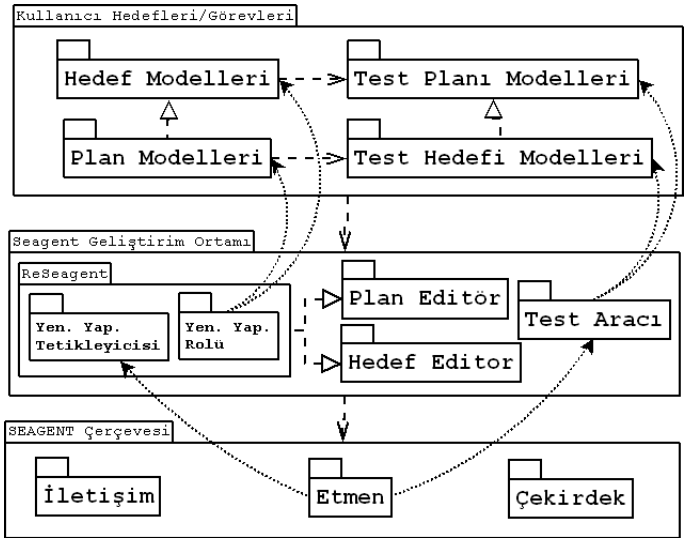
- AVKD sadece, ontolojide daha önceden tanımlanmış olan özellikler kullanılarak, var olan örnekler üzerinde yeni bildirimler yapılması yolu ile ontolojinin genişletilmesini desteklemektedir.
- AVKD, öncül kısmında tanımlanan durumun sağlanması halinde modeldeki bir sınıfın yeni bir örneğinin oluşturulabilmesini desteklememektedir.
- Modelde var olan bilgiler (örnekler ve aralarındaki özellikler) AVKD kuralları ile değiştirilemez ve silinemez.

Yeniden yapılandırma operasyonları modele yeni elemanların eklenmesinin yanı sıra, var olan elemanların değiştirilmesi ve modelden çıkarılmasına da gereksinim duyarlar. Bu yüzden, yeniden yapılandırma teknikleri sadece dekleratif yaklaşım kullanılarak AVKD ile oluşturulan kural setleri olarak tanımlanamamıştır. ReSeagent aracı yeniden yapılandırma tekniklerinin tanımlanması için dekleratif ve imperatif yaklaşımı birleştiren melez bir yaklaşım kullanmaktadır. İmperatif yaklaşım SGA planlama dili ile sağlanmaktadır. ReSeagent aracı içerisinde yeniden yapılandırma operasyonlarının tanımlanması için SGA planlarını kullanılmaktadır. Her plan bir yeniden yapılandırma operasyonu mekaniklerini gerçekleştirmek için eylemlere sahiptir. Modeli genişleten mekanikler AVKD kurallarını işleten eylemler sayesinde, var olan modelde değişiklik veya

silme gerektiren mekanikler ise modele doğrudan müdahale eden eylemler sayesinde gerçekleştirilir. Bir sonraki bölümde, bu melez yaklaşımı kullanan yazılım mimarisi detaylı olarak anlatılmaktadır.

6.2 Aracın Genel Mimarisi

ReSeagent yeniden yapılandırma aracı, Bölüm 5.2.1'de tanıtılan üç seviye içerisinde tanımlanan yeniden yapılandırma tekniklerini desteklemek amacıyla geliştirilmiştir. Geliştirilen yeniden yapılandırma aracının genel mimarisi Şekil 6.1'de görülmektedir.



Şekil 6.1. ReSeagent yeniden yapılandırma aracının genel mimarisi

Araç, bir yeniden yapılandırıcı rolü ve tetikleyici modülünden oluşmaktadır. Tetikleyici, SEAGENT geliştirim ortamındaki plan ve hedef edi-

törlerin üzerine bir eklenti olarak geliştirilmiştir ve bu ortamlardan aldığı her istek için uygun yeniden yapılandırma operasyonunu başlatır. Yeniden yapılandırıcı rolü ise yeniden yapılandırma teknikleri için tanımlanmış planları barındıran özel bir roldür.

Geliştiriciler plan veya rol-hedef editörden yeniden yapılandırma eklentisini kullanarak bir yeniden yapılandırma operasyonunu başlatabilirler. Yeniden yapılandırma operasyonları plan veya rol-hedef geliştirim ortamı içerisinde plan veya hedef modellerin oluşturulması sırasında yeni oluşturulan modeller üzerinde veya ilgili geliştirim ortamı içerisinde yeniden açılan daha önceden oluşturulmuş modeller üzerinde uygulanabilir.

6.2.1 Yeniden yapılandırma Tetikleyicisi

ReSeagent aracının *Yeniden Yapılandırma Tetikleyicisi* modülü karışık olmayan bir yapıya sahiptir. Bu modül, plan ve rol-hedef geliştirim ortamlarından yeniden yapılandırma isteklerinin ve bu istekler için gerekli kullanıcı tercihlerinin alınarak, bu isteklere uygun bir yeniden yapılandırma operasyonlarının başlatılmasından sorumludur.

Geliştirici SEAGENT plan ve rol-hedef editörü içerisindeki yeniden yapılandırma eklentilerinden birini kullanarak bir yeniden yapılandırma operasyonu başlatmak istediğinde, tetikleyici “yeniden yapılandırıcı” rolünü oynayan bir yeniden yapılandırıcı etmenini başlatır ve yeniden yapılandırma isteğini girdileriyle birlikte bu etmene iletir. Yeniden yapılandırıcı etmeni aldığı yeniden yapılandırma isteğini uygun bir yeniden yapılandırma planı ile eşler ve girdi değerlerini ihtiyaç

olarak geçirerek bu planı çalıştırır. İstek ile eşlenebilecek birden fazla yeniden yapılandırma planı bulunabilir. Böyle bir durumda, etmen kullanıcı arayüzünden kendisine gelen girdileri ve güncel durumunu değerlendirerek hangi planın çalıştırılacağına karar verir.

Tetikleyici modülünün diğer bir sorumluluğu ise yeniden yapılandırma operasyonu sonucunda elde edilen güncellenmiş model(ler)in uygun editöre aktarılmasıdır.

6.2.2 Yeniden yapılandırıcı Rolü

Yeniden yapılandırıcı rolü, yeniden yapılandırma hedeflerine ve bu hedefleri başaracak yeniden yapılandırma planlarına sahip, sistemdeki eylemlere, plan ve rol-hedef modellerine erişebilen özel bir roldür. Tüm yeniden yapılandırma seviyelerindeki yeniden yapılandırma tekniklerinin sistemde var olan eylemler, plan modelleri ve rol-hedef modelleri üzerinde uygulanmasından sorumludur.

6.3 Yeniden Yapılandırma Planları

Geliştirdiğimiz yeniden yapılandırma aracında yeniden yapılandırma kalıpları SGA planları olarak gerçekleştirilmektedir. Bu planlar içerisinde AVKD var olan modeli genişleten dönüşüm kurallarının tanımlanması için kullanılmaktadır. Yeniden yapılandırmaların tanımlanmasında SGA'nın kullanılmasının bazı avantajları bulunmaktadır;

- Test etme, gerçekleştirim ve yeniden yapılandırma için aynı programlama tekniğinin (SGA) kullanılması, yeniden yapılandırma

aracına yeni yeniden yapılandırmaların eklenmesini kolaylaştırmaktadır,

- Büyük yeniden yapılandırmalar daha önceden tanımlanmış yeniden yapılandırma planlarının üst düzey bir SGA yapısı içerisinde yeniden kullanılması ile kolayca gerçekleştirilebilir,

- Yeniden yapılandırma planları, çalışma zamanında etmenlerin özerk olarak yeniden yapılandırma yapabilmeleri fikri için bir zemin oluşturmaktadır.

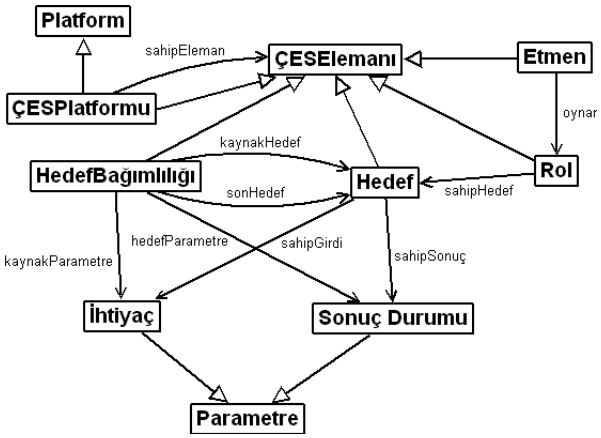
Her yeniden yapılandırma planı ilgili yeniden yapılandırmanın mekaniklerini gerçekleştiren bazı alt-görevler içerir. Planın girdileri bu alt-görevlere daha önceden plan yapısında tanımlandığı şekilde geçirilir ve bu alt-görevler önceden tanımlanmış SGA yapısına göre sırayla çalıştırılır. Planın sonunda, yeniden yapılandırma sonucunda elde edilen güncellenmiş model uygun editörde kullanıcıya gösterilmek üzere tetikleyiciye gönderilir.

ReSeagent güncel versiyonunda ikisi rol seviyesi, dördü plan seviyesi olmak üzere toplam altı adet yeniden yapılandırma planı gerçekleştirilmiş durumdadır.

6.3.1 ReSeagent Rol Seviyesi Yeniden yapılandırma Planları

ReSaegent içerisindeki rol seviyesi yeniden yapılandırmalar için tanımlanan etmen planları, geliştirilmekte olan çoklu etmen sisteminin

SEAGENT rol-hedef üst modelini temel alarak oluşturulmuş rol-hedef modelinin daha düzgün bir yapıya güncellenmesi için geliştirilmiştir. SEAGENT çerçevesinde güncel olarak kullanılan rol-hedef üst modeli[5] Şekil 6.2'de biçimsel olarak gösterilmiştir.



Şekil 6.2. SEAGENT rol-hedef üst modeli

Aracın güncel versiyonunda aşağıda listelenen rol seviyesi yeniden yapılandırmalar desteklenmektedir;

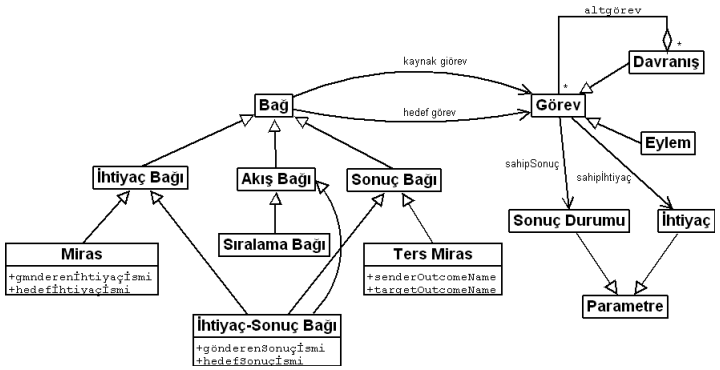
- *Rol Böl*
- *Sorumluluk Taşı*

Bu yeniden yapılandırmalar için ReSeagent içerisinde gerçekleştirilen yeniden yapılandırma planlarının detaylı anlatımları Ek E'de sunulmuştur.

[5]http://etmen.ege.edu.tr/etmen/ontologies/Platform.owl Son erişim:01-09-2008

6.3.2 ReSeagent Plan Seviyesi Yeniden yapılandırma Planları

ReSeagent içerisinde plan seviyesi yeniden yapılandırmalar için tanımlanan etmen planları, SEAGENT plan (SGA) üst modeli temel alınarak oluşturulmuş olan plan modellerinin yeniden yapılandırılarak daha düzgün yapılara güncellenmesi için geliştirilmiştir. Bu yüzden, bu seviyedeki yeniden yapılandırma planları içerisindeki eylemlerin gerçekleştirimi doğrudan SEAGENT plan üst modeline bağımlıdır. SEAGENT çerçevesi tarafından güncel olarak kullanılan plan üst modeli[6] Şekil 6.3'de biçimsel olarak gösterilmiştir.



Şekil 6.3. Güncel SEAGENT plan üst modeli

Aracın güncel versiyonunda aşağıdaki plan seviyesi yeniden yapılandırmalar desteklenmektedir;

[6]http://etmen.ege.edu.tr/etmen/ontologies/HTNOntology-2.1.1.owl Son erişim: 01-09-2008

- *Görevleri Bir Görev İle Yer Değiştir*
- *Davranış Çıkarımı*
- *Davranıştan Plana*
- *Plan Çıkarımı*

Bu yeniden yapılandırmalar için ReSeagent içerisinde gerçekleştirilen yeniden yapılandırma planlarının detaylı anlatımları Ek E'de sunulmuştur.

6.3.3 ReSeagent Eylem Seviyesi Yeniden Yapılandırma Planları

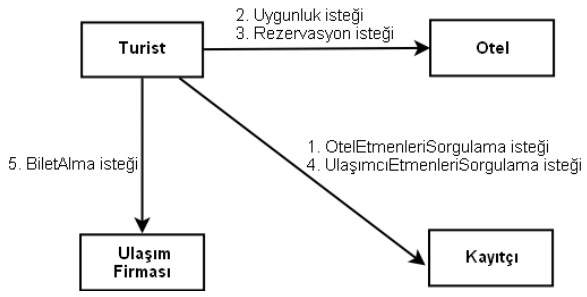
SEAGENT içerisinde çalışabilir kod parçalarını içeren eylemler Java sınıfları olarak tanımlanmaktadır. Bu yüzden, eylemler üzerindeki yeniden yapılandırmaların çoğu sınıf yapılarının güçlendirilmesini hedefleyen geleneksel yeniden yapılandırmaları temel alır. Bununla birlikte, yeniden yapılandırma operasyonları sırasında eylemlerin SGA paradigmasından gelen bazı özelliklerinin de ele alınması gerekebildiğinden dolayı eylemlerin yeniden yapılandırılması için geleneksel yeniden yapılandırma teknikleri yeterli değildir. Bu tekniklerin eylemlerin SGA planları içerisindeki ilişkileri değerlendirilerek genişletilmesi gerekmektedir. Örneğin *Eylem Böl* yeniden yapılandırması geleneksel *Sınıf Çıkarımı* yeniden yapılandırmasını kullanarak bir eylemden birden fazla eylem sınıfı oluşturur. Fakat bu yeterli değildir, çünkü bu yeni eylemler arasındaki akış bağlarının oluşturulması gerekmektedir. Aracın güncel versiyonunda herhangi bir eylem seviyesi yeniden yapılandırma desteği bulunmamaktadır.

7 ÖRNEK UYGULAMA

Bu bölümde tezde çoklu etmen sistemlerin evrimsel geliştirimi için önerilen yaklaşımların ve bu yaklaışmları desteklemek amacıyla geliştirilen SeaUnit test çerçevesi ve ReSeagent yeniden yapılandırma araçlarının gerçek bir ÇES uygulamasının geliştirimi sırasında kullanımını göstermek amacıyla gerçekleştirilen bir örnek uygulama sunulmaktadır. Örnek, turizm alanı için geliştirilen bir ÇES içerisindeki "tatil oluşturma" sistem hedefinin evrimsel bir yol içerisinde geliştirilmesi üzerine odaklanmaktadır. Bu sistem hedefi EYTGG yaklaşımının önerdiği geliştirim tarzı kullanılarak geliştirilmiştir.

"Tatil oluşturma" sistem hedefinin ilişkili olduğu ÇES senaryosunda turist kendi tercihlerine uygun bir tatil hazırlamak istemektedir. Tatil hazırlanması kalacak yerin kiralanması ve buraya ulaşım için uygun biletin alınması işlemlerini içermektedir. Senaryo içerisinde turistin kalacak yer ve ulaşım tercihlerinin daha önceden bilgi tabanına kaydedildiği varsayılmaktadır. Diğer taraftan, "tatil oluşturma" sistem hedefi turistin tatil hazırlarken otellere ve ulaşım firmalarına göndereceği isteklere cevap verebilecek rol hedeflerini de içermektedir.

Uç programlamanın basit tasarım prensibi doğrultusunda, "Tatil oluşturma" sistem hedefinin geliştiriminde ilk olarak hedefin başarımı için çalışacak rolleri ve bu roller arasındaki istekleri gösteren başlangıç bir rol etkileşim diyagramı - RED çizilmiştir. Başlangıç RED Şekil 7.1'de gösterilmektedir.

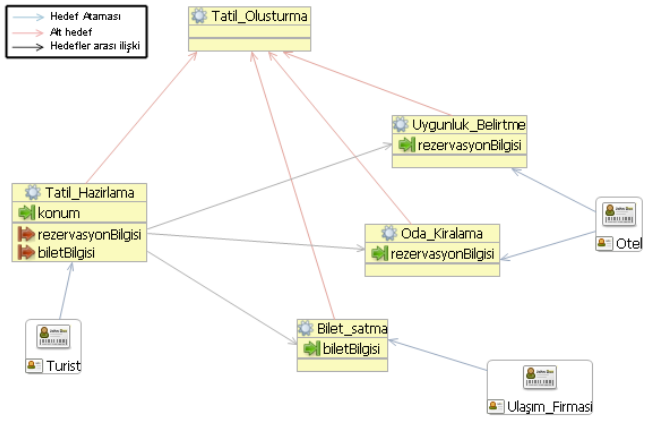


Şekil 7.1. "Tatil oluşturma" sistem hedefi için oluşturulan başlangıç rol etkileşim diyagramı

Rol etkileşim diyagramının tanımladığı senaryoda turist ilk olarak sistemde kayıtlı otel etmenlerini bulmak için kayıtçıya otel etmenlerini sorgulama isteği gönderir. Kayıtçıdan bu isteğe cevap olarak aldığı otel etmenleri ile etkileşime geçerek önce turistin kalacak yer tercihlerine uygun odası olup olmadığını sorar, daha sonra seçtiği uygun odalardan birisi için kiralama isteği gönderir. Turist kalacak yeri ayarladıktan sonra buraya ulaşım için bilet almak amacıyla tekrar kayıtçıyla etkileşime girerek ulaşımcı etmenleri sorgular ve kendi tercihlerine uygun olan bir ulaşım firmasına bilet satın alma isteği gönderir.

Başlangıç olarak modellenen RED ışığında SEAGENT rol-hedef editörü kullanılarak "tatil oluşturma" sistem hedefi için Şekil 7.2'de görülen hedef modeli tanımlanmıştır. Bu hedef model sadece alan bağımlı rolleri ve bu rollerin geliştirilmekte olan "tatil oluşturma" sistem hedefinin başarımı için çalışan hedeflerini göstermektedir. Rol etkileşim diyagramındaki kayıtçı ise zeki fiziksel etmenler kuruluşu - ZFEK (*"Foundation for Intelligent Physical Agents"* - *FIPA*) tarafından (FIPA, 2000)'da önerilen standart dizin yardımcısı (*"directory facilitator"* - DF) rolüdür. Bu

rol tüm sistem hedeflerinin başarımı içerisinde aynı şekilde çalışan tek bir sorumluluğa sahiptir: kendisine gönderilen kriterlere uygun etmenlerin bulunması ve cevap olarak gönderilmesi. Bu sorumluluk her sistem hedefinin geliştiriminde tekrar geliştirilmediğinden, "tatil oluşturma" sistem hedefi için oluşturulan hedef modeli içerisinde gösterilmesine gerek duyulmamıştır.



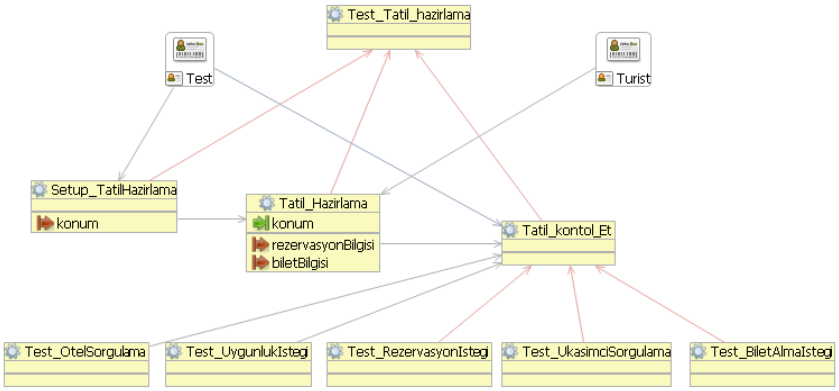
Şekil 7.2. "Tatil oluşturma" sistem hedefi için başlangıç hedef modeli

Sistem hedefinin geliştirim sırasında başlangıç hedef modeli temel alınarak ilk EYTGG döngüsü başlatılmıştır. Döngünün ilk adımında, diğer hedeflere gönderdiği istekler aracılığıyla sistem hedefinin başarım senaryosunun ilerleyişini belirleyen turist rolü en kritik rol olarak geliştirilmek üzere seçilmiştir.

Döngünün ikinci adımında, turist rolü için bu rolün "tatil oluşturma" sistem hedefinin işbirliği ile başarımı içerisindeki ana hedefi olan

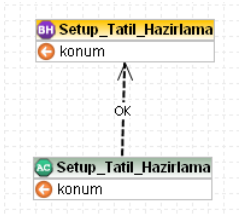
ve bu sistem hedefinin başarım senaryosunu başlatacak olan “tatil hazırlama” rol hedefi tanımlanmış ve geliştirilecek rol hedefi olarak seçilmiştir.

Üçüncü adımda tezde önerilen hedef yönelimli test etme yaklaşımı kullanılarak, bir önceki adımda seçilen “tatil hazırlama” hedefinin başarımını doğrulayan birim test hedefleri SeaUnit test çerçevesi kullanılarak gerçekleştirilmiştir. “Tatil oluşturma” sistem hedefinin geliştirimindeki ilk EYTGG döngüsünün test adımında “tatil hazırlama” hedefi için tanımlanan birim test hedefinin yapısı Şekil 7.3’de gösterilmektedir.



Şekil 7.3. “Tatil hazırlama” rol hedefi için tanımlanan test hedefi yapısı

Test_Tatil_Hazirlama test hedefi turist rolünün “tatil hazirlama” rol hedefinin konum ön koşulunu sağlayan *Setup_Tatil_Hazirlama* kurucu hedefini, “tatil hazirlama” hedefinin kendisini ve *Tatil_Kontrol_Et* bildirim hedefini kapsamaktadır. *Setup_Tatil_hazirlama* isimli kurucu hedefi gerçekleştirmek üzere geliştirilen basit SGA yapısı Şekil 7.4’de gösterilmektedir.



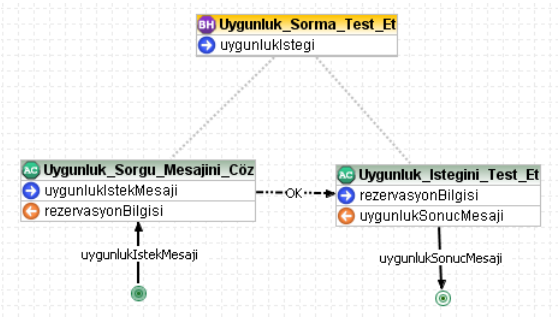
Şekil 7.4. *Setup_Tatil_Hazirlama* kurucu hedefi için SGA yapısı

Şekilde görüldüğü gibi, kurucu hedefi gerçekleştiren plan yapısı sadece tek bir eylem içermektedir. "Tatil hazırlama" hedefinin çalıştırılması için gerekli ön koşulları sağlamayı amaçlayan bu eylemin kaynak kodu Şekil 7.5'de sunulmuştur. Bu eylem "tatil hazırlama" hedefinin başlatılabilmesi için gerekli olan konum ihtiyacını sağlamak için bir *konum* nesnesi yaratır ve sonuç olarak dışarı gönderir.

```
package tr.edu.ege.seagent.turizm.test;

import tr.edu.ege.seagent.planner.htn.Action;

public class Setup_Tatil_Hazirlama extends Action {
    private Konum konum = null;

    public Konum getKonum() {

    public void setKonum(Konum konum) {

    public void execute() throws Exception {
        Konum tatilKonumu = new Konum();
        tatilKonumu.setUlke("Turkiye");
        tatilKonumu.setBolge("Akdeniz");
        this.konum = tatilKonumu;
    }
}
```

Şekil 7.5. *Setup_Tatil_Hazirlama* eyleminin kaynak kodu

"Tatil oluşturma" sistem hedefinin başarımı başlangıç rol etkileşim diyagramında görünen isteklerin "tatil hazırlama" hedefinin gerçekleştirilmesi sırasında düzgün olarak gönderilmesine bağlıdır. Bu yüzden, *Tatil_Kontrol_Et* bildirim hedefi bu istekleri doğrulayan alt-bildirim hedeflerine bölünmüştür. Bu bildirim hedeflerinden biri olan *Test_UygunlukIstegi* isimli hedefin başarımı için tanımlanan plan yapısı Şekil 7.6'de gösterilmektedir.



Şekil 7.6. *Test_UygunlukIstegi* bildirim hedefi için SGA plan yapısı

Bu plan yapısı içerisinde bulunan *Uygunluk_Istegi_Test_Et* eyleminin kaynak kodu örnek bildirim kodu olarak Şekil 7.7'de verilmiştir.

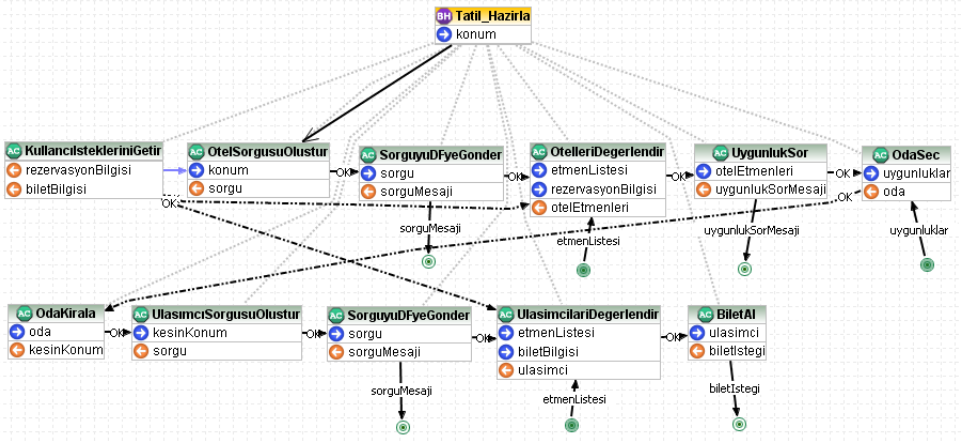
Başlanıç rol etkileşim diyagramında ve test hedefi yapısında görüldüğü gibi "tatil hazırlama" hedefi genel sistem hedefinin başarım senaryosu içerisinde otel ve ulaşım firması gibi diğer rollerdeki bazı rol hedefleriyle işbirliği içerisinde çalışmalıdır. Bu hedefin başarımı ancak diğer hedeflerle etkileşimleri sağlanarak doğrulanabilir. Bununla birlikte,

gerçekleştirimin başında diğer hedefler geliştirilmemiş olduğundan bu aşamada ”tatil hazırlama” hedefinin isteklerine cevap veren basit planlar oluşturulmuştur.

```
package tr.edu.ege.seagent.turizm.test;
import java.util.Vector;

public class Uygunluk_Istegini_Test_Et extends Action {

    private RezervasyonBilgi rezervasyonBilgisi;
    private @SendingMessage
    ACLMessage uygunlukSonucMesaji;

    // Returns rezervasyonBilgisi provision of this action.
    public RezervasyonBilgi getRezervasyonBilgisi() {

    // Returns uygunlukSonucMesaji outcome of this action.
    public ACLMessage getUygunlukSonucMesaji() {

    public void execute() throws Exception {
        Konum rezervasyonKonumu=this.rezervasyonBilgisi.getKonum();

        Assert.assertEquals(rezervasyonKonumu.getUlke(),"Turkiye");
        Assert.assertEquals(rezervasyonKonumu.getBolge(),"Akdeniz");
        Assert.assertTrue(rezervasyonBilgisi.HasKapaliHavuz);

        ACLMessage sonucMesaji = new ACLMessage();
        Oda uygunOda= new Oda ("Suit",2,50);
        Vector<Oda> uygunOdalar = new Vector<Oda>();
        uygunOdalar.add(uygunOda);
        sonucMesaji.setContent(uygunOdalar);
        this.uygunlukSonucMesaji=sonucMesaji;
    }
}
```

Şekil 7.7. *Uygunluk_Istegi_Test_Et* eyleminin kaynak kodu

Test hedefi içerisindeki kurucu ve bildirim hedeflerini tümü için SGA planları gerçekleştirildiğinde EYTGG döngüsünün bir sonraki adımı olan plan gerçekleştirme adımına geçilmiştir. Bu adımda, bir önceki adımda bildirim hedefleri içerisinde tanımlanan testleri geçecek SGA planı gerçekleştirilmiştir. “Tatil hazırlama” ismindeki rol hedefi için geliştirilen *Tatil_Hazirla* planın SGA yapısı Şekil 7.8’de gösterilmetekdir.



Şekil 7.8. *Tatil_Hazirla* plan yapısı

Tatil_Hazırla planında ilk olarak yerel bilgi tabanında tutulan kalacak yer ve ulaşım tercihleri çekilir. Plana ihtiyaç olarak gelen konum bilgisi kullanılarak bu konuma uygun otel etmenleri için sorgu hazırlanır ve kayıtçıya gönderilir. Kayıtçıdan cevap olarak gelen etmen listesi turistin kalacak yer tercihlerine göre değerlendirilerek uygun olan otel etmenlerine turistin rezervasyon tercihlerine uygun odası olup olmadığına dair bir uygunluk isteği mesajı gönderilir. Otel etmenleri cevap olarak rezervasyon tercihlerine uygun odalarının listesini gönderirler. Planda *OdaSec* eylemi bu uygunluk mesajlarını toplayarak uygun olan odalar içerisinden birini kiralamak için seçer. Daha sonraki *OdaKirala* eyleminde, seçilen oda için ilgili etmene oda kiralama istek mesajı gönderilerek kalacak yerin kiralanması işlemi tamamlanır. Bir sonraki eylemde, kalacak yerin kesin konumu ihtiyaç olarak alınarak buraya ulaşım sağlayan ulaşım firmalarının etmenleri için bir sorgu hazırlanır ve kayıtçıya gönderilir. Kayıtçıdan ce-

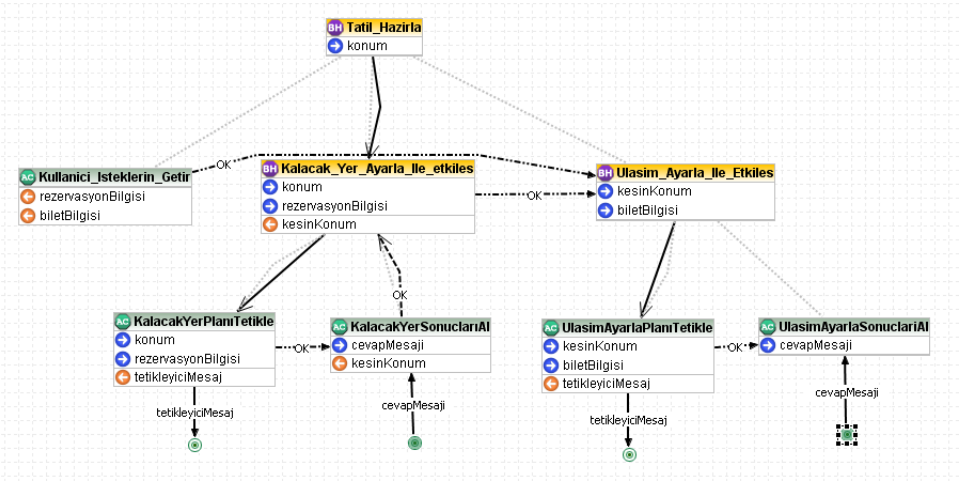
vap olarak gelen ulaşım firması etmenleri listesi turistin ulaşım tercihlerine göre değerlendirilerek içlerinden biri seçilir. Son eylemde, seçilen ulaşım firması etmenine bilet alma istek mesajı gönderilerek ulaşım bileti alınır.

Plan bir önceki adımda tanımlanan tüm testleri geçtiğinde EYTGG döngüsünün bir sonraki adımına geçilmiştir.

EYTGG döngüsünün son adımı olan yeniden yapılandırma adımında, bir önceki adımda gerçekleştirilen *Tatil_Hazirla* planının diğer görevlerden bağımsız olarak kendi koşulları ile test edilmesi gereken bazı alt-hedefler içerdiği fark edilmiştir. Tatil için kalacak yerin ayarlanması görevi turist tercihlerinin bilgi tabanından alınması ve ulaşım bileti ayarlanması ile ilgili görevlerden bağımsız olarak test edilebilmelidir. Aynı zamanda, bu alt-hedefin çalışma kararı etmenler tarafından özerk ve diğer planlardan bağımsız olarak verilmelidir. Aynı şekilde, ulaşım ayarlama işlemlerini kotaran görevler için de geçerli olan bu durum *Plan İçerisinde Çalışma Kararı* kötü kokusunu göstermektedir.

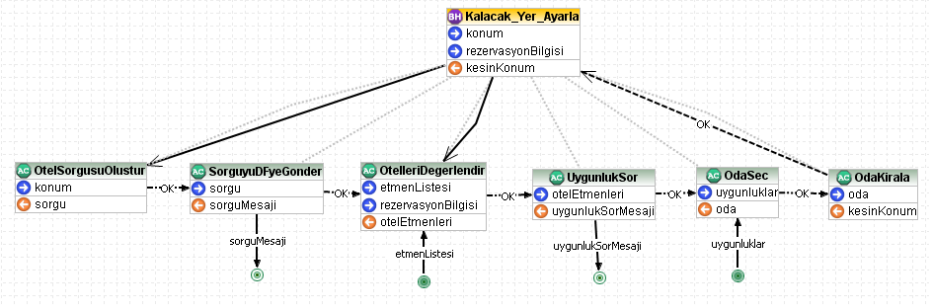
Bununla birlikte, plan yapısı içerisinde çok sayıda alt-görevin bulunması planın okunabilirliğini ve yönetilebilirliğini azaltmaktadır. Bu da *Büyük Davranış* kötü kokusunu gösteren bir durumdur.

Tatil_Hazirla plan yapısı içerisinde yakalanan kötü kokuların ortadan kaldırılması için hem kalacak yer ayarlamakla ilgili görevler hem de ulaşım ayarlamakla ilgili görevler üzerinde *Plan Çıkarımı* yeniden yapılandırması uygulanmıştır. Yeniden yapılandırma operasyonu sonucunda oluşan *Tatil_Hazirla* planının yeni yapısı Şekil 7.9'de gösterilmektedir.



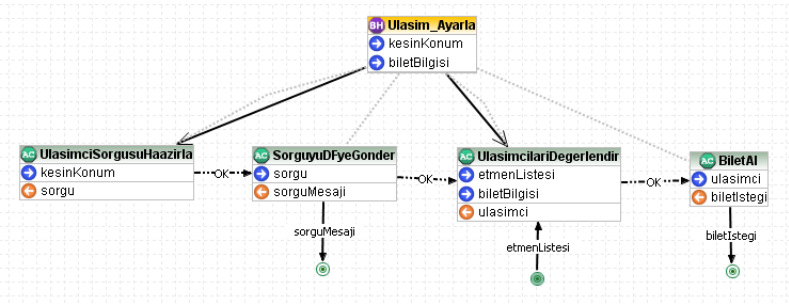
Şekil 7.9. *Plan Çıkarımı* sonrası *Tatil_Hazirla* plan yapısı

Kalacak yerin ayarlanması ile ilgili görevler üzerinde uygulanan *Plan Çıkarımı* yeniden yapılandırması sırasında tanımlanan "kalacak yer ayarlama" isimli yeni rol hedefini başarmak üzere orijinal plan yapısından dışarı çekilen görevler ile elde edilen *Kalacak_Yer_Ayarla* isimli yeni plan yapısı Şekil 7.10'de gösterilmektedir.



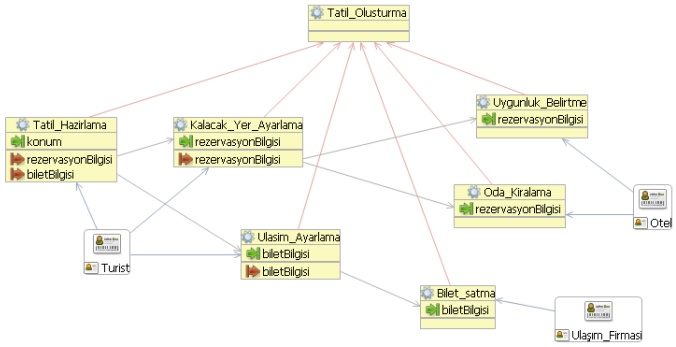
Şekil 7.10. *Kalacak_Yer_Ayarla* plan yapısı

Ulaşımın ayarlanmasıyla ilgili görevler üzerinde uygulanan *Plan Çıkarımı* yeniden yapılandırması ile elde edilen *Ulasim_Ayarla* isimli yeni plan Şekil 7.11'de gösterilmektedir.



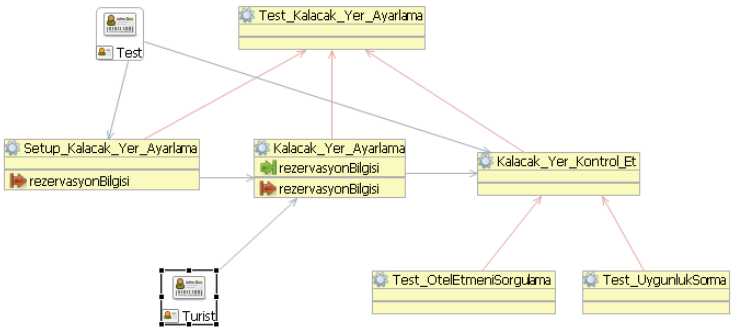
Şekil 7.11. *Ulaşım_Ayarla* plan yapısı

Yeniden yapılandırma operasyonları sonucunda "tatil oluşturma" sistem hedefinin hedef modeli Şekil 7.12'deki hale gelecek şekilde değişmiştir.



Şekil 7.12. "Tatil oluşturma" sistem hedefinin güçlendirilmiş hedef modeli

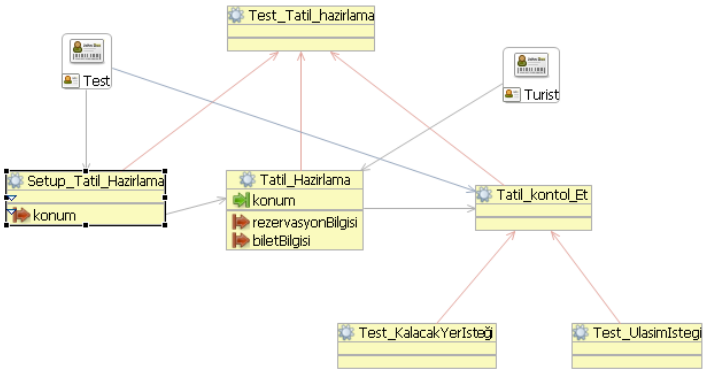
Hedef modele yeni rol hedeflerinin eklenmesi hedeflerin başarımını doğrulamak üzere yeni testlerin de yazılabilmesine olanak sağlamıştır. Daha önceden *Tatil_Hazirla* planının akışı doğrultusunda bu planın ön koşulları ile test edilebilen kalacak yer ayarlama ve ulaşım ayarlama işlemleri için yeni rol hedeflerinin tanımlanması sayesinde bu işlemleri kendi ön koşulları ile başka bir plan akışına bağlı kalmadan test eden yeni test hedefleri tanımlanabilmiştir. "Kalacak yer ayarlama" rol hedefi için tanımlanan test hedefinin yapısı Şekil 7.13'de gösterilmektedir.



Şekil 7.13. "Kalacak yer ayarlama" rol hedefi için oluşturulan test hedefi

Yeniden yapılandırma operasyonu ile kalacak yer ve ulaşım ayarlama sorumlulukları *Tatil_Hazirla* planından alınarak yeni planlara eklenmesi ile bu planın yapısı sadeleşmiştir. Bu sorumlulukları yerine getiren görevler plandan çıkarılmış, yerine sadece yeni planlarla etkileşim kuran sade görevler eklenmiştir. Plan yapısının sadeleşmesi bu plan için

daha önceden yazılan test hedefinin de sadeleşmesini sağlamıştır. Test hedefi yapısı içerisinde bulunan ve ayrı plana çekilen görevlerin sorumluluklarını doğrulayan bildirim hedefleri çıkarılmış yerine planın yeniden yapılandırma operasyonu sırasında oluşturulan iki yeni plan ile etkileşimlerini doğrulamayı amaçlayan bildirim hedefleri eklenmiştir. "Tatil hazırlama" rol hedefini test eden *Test_Tatil_Hazirlama* isimli test hedefinin *Plan Çıkarımı* yeniden yapılandırmasından sonra düzenlenen yeni hali Şekil 7.14'de gösterilmektedir.

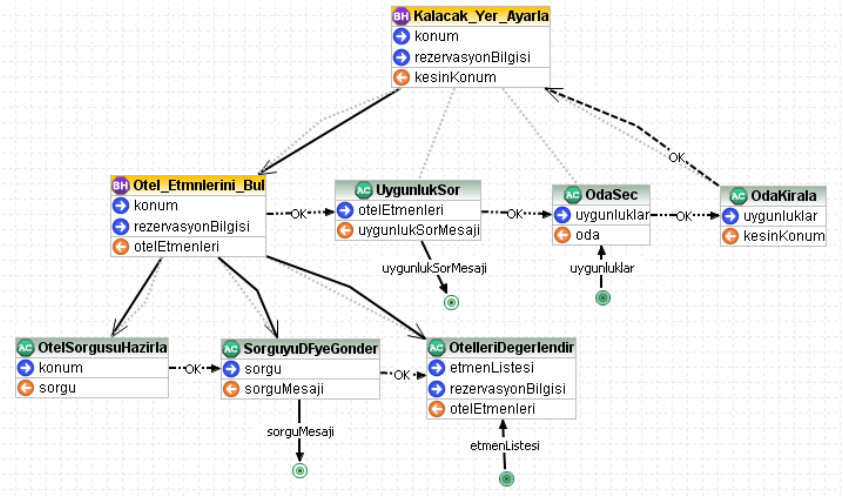


Şekil 7.14. Yeniden düzenlenen *Test_Tatil_Hazirlama* test hedefi

Otel ve ulaşım firması etmenlerinin bulunması gereksinimleri geliştirilen turizm sisteminin farklı senaryoları içerisindeki diğer etmen planlarında da bulunmaktadır. Bu durum, *Kalacak_Yer_Ayarla* planı içerisinde otel etmenlerinin bulunması, *Ulasim_Ayarla* planı içerisinde ulaşımcı etmenlerin bulunması için kullanılan görevlerin farklı plan yapıları içerisinde tekrar etmesine neden olmaktadır. Bu yüzden,

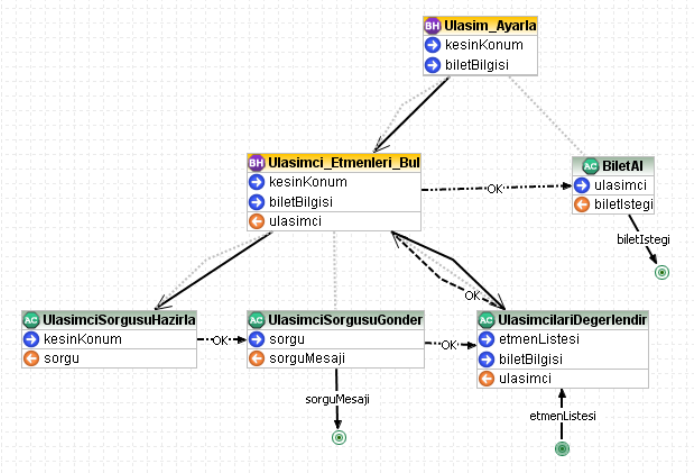
bahsedilen iki plan yapısı içerisinde *Tekrarlanan Davranış Yapısı* kötü kokusu fark edilmiş ve bu kötü kokulardan kurtulmak için iki plan yapısında *Davranış Çıkarımı* yeniden yapılandırması operasyonları uygulanmıştır.

Kalacak_Yer_Ayarla planı içerisindeki otel etmenlerinin bulunması ile ilgili görevler üzerinde uygulanan *Davranış Çıkarımı* yeniden yapılandırma operasyonu sonunda bu planın yeni yapısı Şekil 7.15'de gösterilmektedir.



Şekil 7.15. Güçlendirilmiş *Kalacak_Yer_Ayarla* plan yapısı

Ulasim_Ayarla planı içerisindeki ulaşımcı etmenlerinin bulunması ile ilgili görevler üzerinde uygulanan *Davranış Çıkarımı* yeniden yapılandırma operasyonu sonunda bu planın yeni yapısı ise Şekil 7.16'de gösterilmektedir.



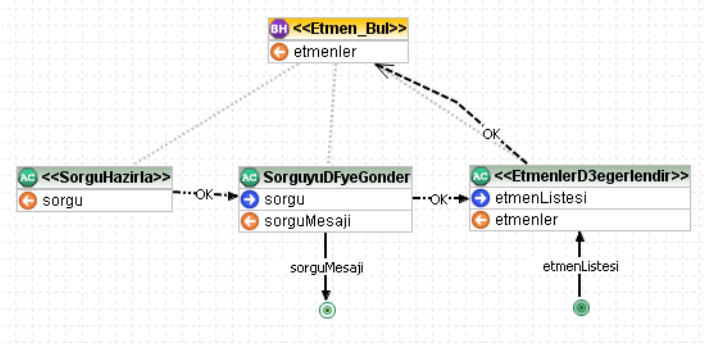
Şekil 7.16. Güçlendirilmiş *Ulasim_Ayarla* plan yapısı

Davranış Çıkarımı yeniden yapılandırması operasyonları ile sistemdeki diğer plan yapıları içerisinde yeniden kullanılabilecek *Otel_Etmenlerini_Bul* ve *Ulasimci_Etmenleri_Bul* isimlerinde iki yeni davranış elde edilmiştir. Yeniden yapılandırma operasyonu aynı zamanda *Kalacak_Yer_Ayarla* ve *Ulasim_Ayarla* plan yapılarının sadeleşmesini sağlayarak planların okunabilirliğini arttırmıştır.

Davranış Çıkarımı yeniden yapılandırması hedef model içerisinde herhenagi bir değişikliğe neden olmadığından bu model içerisindeki hedeflerin test edilmesi için daha önceden tanımlanmış test hedefi yapıları ve bu yapılar içerisindeki bildirim hedefleri yeniden yapılaıdırma operasyonu sonrasında herhangi bir değişiklik gerektirmeden kullanılmaya devam etmiştir.

Daha sonra *Davranış Çıkarımı* yeniden yapılandırması sonucu

oluşan iki yeni davranış yapısı arasında da *Tekrarlanan Davranış Yapısı* kötü kokusu sezilmiştir. Bu davranışlardan biri otel diğeri ise ulaşım firması etmenlerini bulma sorumluluğuna sahiptir ve bu benzer sorumlulukları yerine getirmek için benzer bir görev yapısı kullanmaktadırlar. Bununla birlikte, kayıtçı ile etkileşime geçerek istenen tipte etmenlerin bulunması diğer etmenlerle etkileşim halinde çalışan tüm plan yapılarında bulunan bir gereksinimdir. Bu yüzden, soyut bir "etmen bulma" davranışı oluşturmak üzere *Otel_Etmenlerini_Bul* ve *Ulasimci_Etmenleri_Bul* davranışları üzerinde *Süper-Davranış Çıkarımı* yeniden yapılandırması uygulanmıştır. Bu yeniden yapılandırma sonucunda elde edilen *Etmen_Bul* isimli soyut davranışın SGA yapısı Şekil 7.17'de gösterilmektedir.



Şekil 7.17. *Etmen_Bul* soyut davranışı

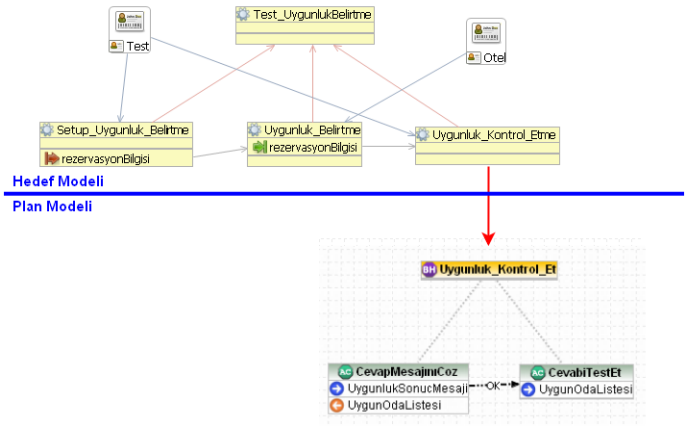
Süper-Davranış çıkarımı yeniden yapılandırması üzerinde uygulandığı somut davranışların işleveselliğini değiştirmediğinden ve dolayısıyla geliştirilmekte olan sistem hedefinin hedef modelini

etkilemediğinden sistemde var olan testler bu yeniden yapılandırma operasyonundan sonra da herhangi bir yeniden düzenlemeye gerek duyulmadan kullanılmaya devam etmiştir.

Tatil_Hazirla plan yapısı üzerinde uygulanan yeniden yapılandırmalar ile bu yapıda fark edilen kötü kokular çözümlendiğinde "tatil oluşturma" sistem hedefinin geliştirimindeki ilk EYTGG döngüsü tamamlanmıştır. Bu noktada sistem hedefi içerisindeki rol hedeflerinden biri olan"tatil hazırlama" hedefini başaran esnek ve yönetilebilir SGA planları ve bu planların çalışmasını kontrol ederek hedefin başarımını doğrulayan test hedefleri gerçekleştirilmiş durumdadır.

Bununla birlikte, daha önce de bahsedildiği gibi sadece "tatil hazırlama" rol hedefinin başarımı tüm "tatil oluşturma" sistem hedefinin başarımı için yeterli değildir. Bu rol hedefi, diğer rollerdeki bazı rol hedefleriyle etkileşimli olarak çalışmaktadır. "Tatil hazırlama" rol hedefinin geliştirildiği EYTGG döngüsünün test adımı sırasında bu etkileşimler belirli isteklere basit cevaplar veren basit planlar ile sağlanmıştır. İlk EYTGG döngüsünün sonunda "tatil hazırlama" hedefinin düzgün çalışması garanti altına alındıktan sonra sıra bu hedefin etkileşim içerisinde çalıştığı rol hedeflerinin gerçekleştirimine gelmiştir.

Devam eden birkaç EYTGG döngüsü ile "tatil oluşturma" sistem hedefininin işbirliği ile başarımı için gerekli olan otel rolüne ait "uygunluk belirtme", "oda kiralama" ve ulaşım firması rolüne ait "bilet satma" isimli rol hedefleri geliştirilmiştir. "Uygunluk belirtme" rol hedefi için tanımlanan test hedefinin yapısı Şekil 7.18'de gösterilmektedir.



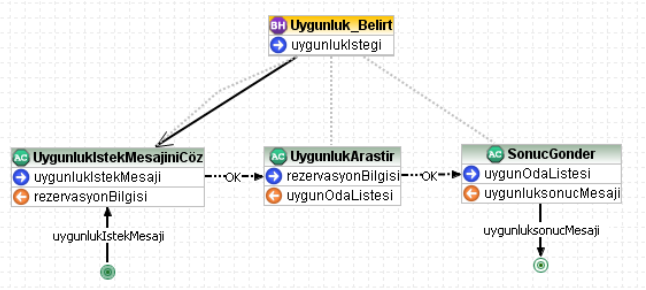
Şekil 7.18. *Test_Uygunluk_Belirtme* test hedefi yapısı

Bu yapıdaki *Uygunluk_Kontrol_Etme* bildirim hedefini gerçekleştiren plan içerisindeki *CevabiTestEt* isimli test eyleminin kaynak kodu sekil 7.19'deki gibidir.

```
package tr.edu.ege.seagent.turizm.test;

import java.util.Vector;

public class CevabiTestEt extends Action {

    private Vector<Oda> UygunOdaListesi;

    public Vector<Oda> getUygunOdaListesi() {
        return UygunOdaListesi;
    }
    public void setUygunOdaListesi(Vector<Oda> UygunOdaListesi) {
        this.UygunOdaListesi = UygunOdaListesi;
    }

    @ExecutionMethod
    public void execute() throws Exception {
        for (Oda uygunOda:this.UygunOdaListesi){
            Assert.assertTrue(uygunOda.uygunMu());
        }
    }
}
```

Şekil 7.19. *UygunlukKontrolEt* test eyleminin kaynak kodu

Bu test hedefinin içerdiği testleri geçmek üzere geliştirilen *Uygunluk_Belirt* isimli etmen planının SGA yapısı ise Şekil 7.20'de gösterilmektedir.

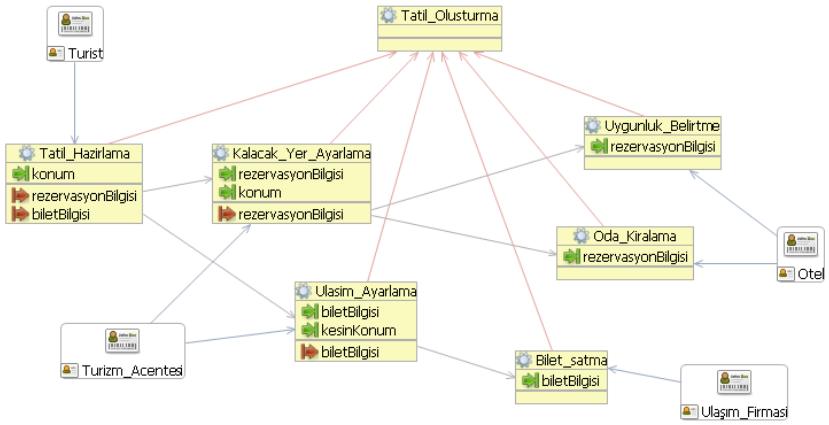


Şekil 7.20. *Uygunluk_Belirt* plan yapısı

"Oda kiralama" ve "bilet satma" rol hedefleri için tanımlanan test hedefleri ve etmen planları turist rolünün "tatil hazırlama" hedefinin başarımı sırasında gönderilecek isteklere cevap vermek amacıyla "uygunluk belirtme" rol hedefinin test hedefi ve planına benzer yapıya sahiptirler. Bu yüzden, bu hedefler için tanımlanan test hedefleri ve etmen planları bu bölümde gösterilmemiştir.

"Tatil hazırlama" rol hedefinin etkileşimde bulunduğu tüm rol hedeflerinin geliştirimi tamamlandığında geliştirilmekte olan "tatil oluşturma" sistem hedefi düzgün şekilde başarılır hale gelmiştir. Fakat, turist rolünün turizm sistemi içerisindeki diğer senaryolarda çok sayıda sorumluluğa sahip olabileceği göz önüne alındığında, bu rolün sahip

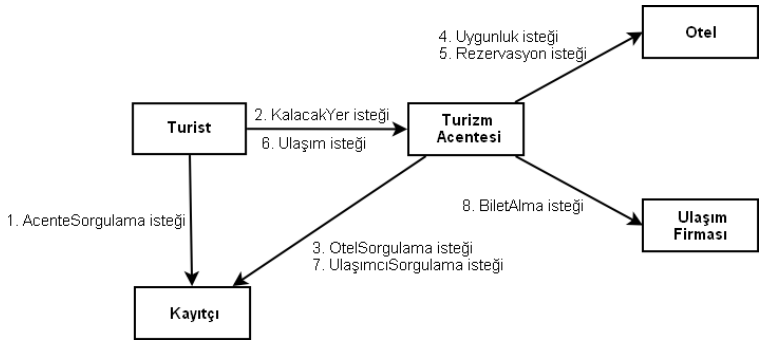
olduğu “kalacak yer ayarlama” ve “ulaşım ayarlama” hedeflerinin diğer rollerdeki hedeflerle oldukça fazla sayıdaki etkileşim gereksiniminin bu rolün dolayısıyla da tüm sistemin performansını olumsuz şekilde etkileyeceği fark edilmiştir. Bu durum Ek-A.2’de tanıtılan *Aşırı Yüklenmiş Rol* kötü kokusunu göstermektedir. Bu kötü konunun ortadan kaldırılması için “kalacak yer ayarlama” ve “ulaşım ayarlama” rol hedefleri üzerinde *Sorumluluk Taşı* yeniden yapılandırması operasyonları uygulanarak bu iki rol hedefi “turizm acentesi” adındaki yeni bir role taşınmıştır. Bu bölüm içerisinde geliştirim detayları sunulan “tatil oluşturma” sistem hedefinin bu yeniden yapılandırma operasyonları sonucunda elde edilen hedef modeli Şekil 7.21’de gösterilmektedir.



Şekil 7.21. “Tatil oluşturma” sistem hedefinin güçlendirilmiş hedef modeli

“Tatil oluşturma” sistem hedefinin geliştiriminin başlangıcında modellenen RED bu sistem hedefinin evrimsel geliştirimi sırasında uygu-

lanan yeniden yapılandırma operasyonları sonucunda Şekil 7.22'deki gibi değişmiştir.



Şekil 7.22. "Tatil oluşturma" sistem hedefinin rol etkileşim diyagramının son hali

Turizm alanı için geliştirilmekte olan çoklu etmen sistemi içerisindeki "tatil oluşturma" sistem hedefi tezde önerilen yeniden yapılandırma yaklaşımı ve bu yaklaşımı destekleyen ReSeagent yeniden yapılandırma aracı kullanılarak evrimsel bir yol içerisinde geliştirilmiştir.

8 SONUÇ ve TARTIŞMA

8.1 Gözden Geçirme ve Tartışma

Bu tezde, çoklu etmen sistem tasarımımın evrimleşmesini yöneterek bu sistemlerin evrimsel geliştirimini mümkün hale getiren bir yeniden yapılandırma yaklaşımı ortaya konmaktadır. Bu temel amaç doğrultusunda, en yaygın çevik süreçlerden biri olan uç programlamada tasarımın evrimleşmesini yönetmek için önerilen yeniden yapılandırma pratiğinin ÇES geliştirimine aktarılması için gereksinimler tanımlanmış ve gerçekleştirilmiştir.

İlk olarak, etmen sistemlerin evrimsel bir yol içerisinde geliştirimini sağlayarak yeniden yapılandırma için test altyapısı oluşturan etmen yönelimli test güdümlü geliştirim - EYTGG yaklaşımı tanıtılmıştır. Daha sonra, çoklu etmen sistemleri için yeniden yapılandırma seviyelerini, ortak tasarım problemlerini ve yeniden yapılandırma tekniklerini tanımlayan bir yeniden yapılandırma yaklaşımı tanıtılmıştır. Ayrıca, EYTGG yaklaşımını destekleyen SeUnit adında bir test çerçevesi ve yeniden yapılandırma yaklaşımını destekleyen ReSeagent adında bir yeniden yapılandırma aracı SEAGENT etmen çerçevesi üzerine gerçekleştirilmiştir.

SeaUnit test çerçevesinin güncel versiyonu birim ve bütünleştirme test seviyeleri için test hedeflerinin oluşturulmasını ve işletilmesini desteklemektedir. ReSeagent yeniden yapılandırma aracını güncel versiyonu plan seviyesinde dört, rol seviyesinde iki olmak üzere toplam altı adet yeniden yapılandırma tekniğini desteklemektedir.

Tezde tanıtılan yaklaşımlar ve bu yaklaşımları destekleyen araçlar SEAGENT tabanlı farklı ÇES uygulamalarının geliştirimi sırasında kullanılmıştır. Bu deneyimler sırasında elde ettiğimiz gözlemler ve geribildirimnler tezde tanıtılan yaklaşımların ve araç desteklerinin çoklu etmen sistemlerinin evrimsel geliştiriminde faydalı olduğunu göstermektedir.

EYTGG yaklaşımı, önerdiği artışlı ve yinelemeli yaşam döngüsü sayesinde tüm geliştirim süreci boyunca hataların erken yakalanmasını sağlamıştır. Bu yaklaşım tarafından önerilen hedef yönelimli test yaklaşımı ve SeaUnit test çerçevesi testlerin gerçekleştirim ile aynı programlama dili kullanılarak oluşturulmasını desteklediğinden, geliştiricilerin ayrı bir teknoloji öğrenmelerine gerek kalmadan testleri kolaylıkla yazabildikleri gözlemlenmiştir. Bu durum, test yönelimli geliştirimin önerdiği "koddan önce test" tarzının geliştiriciler tarafından kabul edilmesini kolaylaştırmıştır.

Baştan itibaren EYTGG yaklaşımı ile geliştirilen sistemlerde, geliştirim sonunda tüm sistem birimleri için elde edilen test hedefleri tüm sistem için bir dökümantasyon ürünü olarak kullanılmış ve geliştirici grubuna yeni katılan geliştiricilerin sistemi anlamasında faydalı olmuştur.

Tezde önerilen test yaklaşımı, sistemdeki hedefleri başaracak etmen planlarının hatasız çalışmasını doğrulamayı amaçlamaktadır. Bu testlerde amaç etmenin özerk çalışmasının test edilmesi değildir. Özerk çalışma, etmenin içsel mekanizmaları tarafından etmen mimarisine (KİN gibi) bağlı olarak etmenin sahip olduğu bilgi tabanındaki bilgilere göre verilen kararlar ile sağlanır. Bu yüzden, etmen mimarisine bağlı olarak oluşturulan bu bilgi tabanlarının hatasız oluşturulması, etmenlerin bekle-

nen durumlarda beklenen hedefleri başarması için önemlidir. Etmenlerin özerk çalışmasının test edilebilmesi için etmen mimarisine bağlı oluşturulan bilgi tabanının hatasızlığını doğrulayan test yaklaşımlarına gereksinim vardır.

Araştırma grubumuzun geliştirim etkinlikleri sırasında, tezde tanıtılan yeniden yapılandırma tekniklerinin evrimsel ÇES geliştirimi içerisinde sık sık gerekli olduğu gözlenmiştir. ÇES geliştirimindeki ortak problemlerin ve bu problemeleri çözecek yeniden yapılandırma tekniklerinin net olarak tanımlanması evrimsel geliştirim sırasında tasarımdaki kötüye gidişlerin daha erken fark edilmesini ve çözümlenmesini sağlamıştır. ÇES geliştirimi için tanıtılan yeniden yapılandırma yaklaşımı ReSeagent araç desteği ile birlikte ÇES bakım etkinliklerini kolaylaştırmıştır.

Tezde hedef yönelimli ÇES geliştirim yöntemleri için ortak soyutlamaları içeren bir üst-model tanımlanarak EYTGG ve yeniden yapılandırma yaklaşımları bu üst-modele dayandırılmıştır. Tanıtılan yeniden yapılandırma yaklaşımı içerisinde bu üst-modeldeki çalışma zamanı ürünleri yeniden yapılandırma açısından incelenmiştir. Bununla birlikte, etmen yönelimli yöntemler bu çalışma zamanı ürünlerini elde edebilmek için kendi geliştirim adımları içerisinde kendilerine özgü çeşitli geliştirim ürünleri (protokol, bilgi tabanı ve rol kuralı gibi) modellemektedirler. Yeniden yapılandırma yaklaşımının bir yöntem ile birlikte kullanılabilmesi için bu yaklaşım içerisindeki yeniden yapılandırma tekniklerinin yönteme özgü geliştirim ürünlerine etkileri tanımlanmalıdır.

8.2 Katkılar

Bu tezin katkıları aşağıda sıralanmıştır:

- Çoklu etmen sistemlerinin evrimsel geliştirimi için artışlı ve yinelemeli bir yol sunan etmen yönelimli test güdümlü geliştirim yaklaşımı tanımlanmıştır.

- Yazılım mühendisliği alanı içerisinde tanıtılan dört temel test seviyesi ÇES geliştirimi sırasında elde edilen ürünler ile eşleştirilerek, diğerlerinden farklı olarak etmen yerine rol hedefini en küçük birim olarak kabul eden hedef yönelimli bir test süreci tanımlanmıştır.

- Tanıtılan hedef yönelimli test yaklaşımı içerisindeki test hedeflerinin oluşturulmasını kolaylaştırarak bu test yaklaşımını destekleyen SeaUnit adında bir test çerçevesi gerçekleştirilmiştir.

- Geliştirim sırasında sürekli olarak değişiklikler karşısında sistemin tasarımını iyileştirerek çoklu etmen sistemlerinin evrimsel geliştirimini mümkün hale getiren yeniden yapılandırma yaklaşımı tanımlanmıştır.

- ÇES geliştirim sırasında sık sık karşılaşılan problemlerin (kötü kokular) katalogu oluşturulmuştur. Bu katalogda ikisi rol seviyesi, beşi plan seviyesi ve üçü eylem seviyesinde olmak üzere toplam on adet kötü koku tanımlanmıştır.

- ÇES geliştirim için tanımlanan kötü kokuların üstesinden gelebilmek üzere tanımlanmış yeniden yapılandırmaları içeren

bir yeniden yapılandırma katalogu hazırlanmıştır. Bu katalogun güncel durumunda iki rol seviyesi, sekiz plan seviyesi ve iki eylem seviyesi olmak üzere on iki adet yeniden yapılandırmanın detaylı tanımlaması bulunmaktadır.

- ÇES geliştiriminde kullanımak üzere önerilen yeniden yapılandırma yaklaşımını destekleyen ReSeagent adında bir yeniden yapılandırma aracı geliştirilmiştir.

8.3 İleriye Dönük Araştırma Olanakları

Bu tezde sunulan çalışmayı geliştirebilecek ve genişletebilecek farklı araştırma olanakları mevcuttur:

- **Diğer UP pratiklerinin ÇES geliştirimine aktarılması:** Tezde ÇES geliştirimine aktarılmaları üzerine odaklanılan test güdümlü geliştirim ve yeniden yapılandırma pratikleri UP içerisinde evrimsel geliştirim için en önemli pratikler olmakla birlikte, bu pratiklerin ÇES geliştirimi sırasında başarılı bir şekilde uygulanabilmesi için UP'nin sürekli bütünleştirme, kolektif sahiplik gibi diğer pratiklerinin de bu sistemlerin geliştirimine aktarılması faydalı olacaktır

- **Çalışma zamanından otomatik yeniden yapılandırma:** Çoklu etmen sistemleri çalışma zamanında sisteme yeni gereksinimlerin, yeni rollerin eklenmesine izin veren açık sistemlerdir. Bu yüzden, bu sistemlerin tasarımlarında çalışma zamanında da değişimler olabilir. Çalışma zamanındaki bu değişikliklerin yönetimi etmenlere bırakılması kendinden çalışan sistemin tasarımını sürekli olarak

iyileştiren, kendinden uyarlanabilir sistemlerin oluşturulabilmesine imkan sağlayacaktır. Böyle bir sistemde, etmenler sistem tasarımındaki kötüye gidişi fark ederek, özerk olarak bir yeniden yapılandırma operasyonu uygulayabilirler. Bu tür sistemlerin geliştirilmesi için temel gereksinimler aşağıda sıralanmıştır;

- Kötü kokuların, üzerinde çıkarsama yapılabilecek mantık tabanlı bir biçimde tanımlanması
- Yeniden yapılandırma tekniklerinin etmen planları olarak tanımlanması (tanıtılan yeniden yapılandırma aracı ile gerçekleştirilmiştir)
- Etmen çerçevesinin rol atama, etmen başlatma, bilgi tabanı gönderme/güncelleme gibi yükleme işlemlerinin çalışma zamanında yapılmasını desteklenmesi.
- Çalışan sistemi sürekli izleyerek sistem tasarımında bir kötü koku sezdiğinde önceden tanımlanmış yeniden yapılandırma tekniklerini kullanarak sistem tasarımını güçlendirebilen yeniden yapılandırma rolünün gerçekleştirilmesi.

Çoklu etmen sistemlerinde çalışma zamanında otomatik yeniden yapılandırma fikri için örnek bir mimari sunan yeniden yapılandırma tabanlı yük paylaştırma yaklaşımı (Bora et al., 2008)'de tanıtılmıştır.

KAYNAKLAR DİZİNİ

Akehurst, D. H. and Kent, S. J. H. (2002). A Relational Approach to Defining Transformations in a Metamodel. In Jezequel, J.-M. and Hussmann, H., editors, *The Unified Modeling Language: Model Engineeing, Concepts, and Tools*, volume 2460 of *Lecture notes in computer science*. Springer.

Ambler, S. W. (2007). Test-driven development of relational databases. *IEEE Software*, 24(3):37–43.

Ambler, S. W. and Sadalage, P. J. (2006). *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley Signature Series.

Baumeister, J. and Seipel, D. (2006). Verification and refactoring of ontologies with rules. In Staab, S. and Svatek, V., editors, *EKAW*, volume 4248 of *Lecture Notes in Computer Science*, pages 82–95. Springer.

Baumeister, J., Seipel, D., and Puppe, F. (2004). Refactoring Methods for Knowledge Bases. In *Engineering Knowledge in the Age of the Semantic Web: 14th International Conference, EKAW 2004, LNAI 3257*, pages 157–171. Springer Verlag.

Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional.

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*. Wiley.

Bézivin, J. and Gerbé, O. (2001). Towards a precise definition of the omg/mda framework. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 273,

Washington, DC, USA. IEEE Computer Society.

Bora, S., Tiryaki, A. M., and Dikenelli, O. (2008). A load sharing approach based on refactoring of roles in multi-agent systems. In *The 9th Annual International Workshop Engineering Societies in the Agents World (ESAW 08)*.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Caire, G., Cossentino, M., Negri, A., Poggi, A., and Turci, P. (2004). Multi-agent systems implementation and testing. In *From Agent Theory to Agent Implementation, Fourth International Symposium (AT2AI-4)*.

Cernuzzi, L., Cossentino, M., and Zambonell, F. (2005). Process models for agent-based development. *Journal of Engineering Applications of Artificial Intelligence*, 18 (2).

Chella, A., Cossentino, M., Sabatucci, L., and Seidita, V. (2004). From passi to agile passi: Tailoring a design process to meet new needs. In *IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology (IAT-04)*.

Chella, A., Cossentino, M., Sabatucci, L., and Seidita, V. (2006). Agile passi: An agile process for designing agents. *Comput. Syst. Sci. Eng.*, 21(2).

Clynch, N. and Collier, R. (2007). Sadaam: Software agent development an agile methodology. In *Proceedings of the Workshop of Languages, methodologies, and Development tools for multi-agent systems*.

Cockburn, A. (2004). *Crystal Clear*. Addison-Wesley, Boston.

Coelho, R., Kulesza, U., von Staa, A., and Lucena, C. (2006). Unit testing in multi-agent systems using mock agents and aspects. In *SELMAS '06: Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, pages 83–90, New York, NY, USA. ACM Press.

Cossentino, M., Sabatucci, L., and Chella, A. (2003). Patterns reuse in the passi methodology. In *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World (ESAW'03*, pages 29–31. Springer-Verlag.

Cossentino, M. and Seidita, V. (2004). Composition of a new process to meet agile needs using method engineering. In *SELMAS*, pages 36–51.

Czarnecki, K. and Helsen, S. (2003). Classification of model transformation approaches. In *OOPSLA-03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*.

Daconta, M. C., Smith, K. T., and Obrst, L. J. (2003). *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. John Wiley & Sons, Inc., New York, NY, USA.

Dam, K. H. and Winikoff, M. (2003). Comparing agent-oriented methodologies. In *AOIS*, pages 78–93.

Dastani, M. and Gomez-sanz, J. J. (2005). Programming multi-agent systems. *Knowl. Eng. Rev.*, 20(2):151–164.

DeLoach S. A. (1999). Multiagent Systems Engineering A Methodology and Language for Designing Agent Systems. In *Proc. of Agent Oriented Information Systems*, pages 45–57.

Dikenelli, O., Erdur, R. C., Gumus, O., Ekinci, E. E., Gurcan, O., Kar-

das, G., Seylan, I., and Tiryaki, A. M. (2005a). Seagent: a platform for developing semantic web based multi agent systems. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1271–1272, New York, NY, USA. ACM Press.

Dikenelli, O., Erdur, R. C., Kardas, G., Gumus, O., Seylan, I., Gurcan, O., Tiryaki, A. M., and Ekinci, E. E. (2005b). Developing multi agent systems on semantic web environment using seagent platform. In Dikenelli, O., Gleizes, M. P., and Ricci, A., editors, *ESAW*, volume 3963 of *Lecture Notes in Computer Science*, pages 1–13. Springer.

Durfee, E. and Lesser, V. (1989). Negotiating Task Decomposition and Allocation Using Partial Global Planning. *Distributed Artificial Intelligence*, 2:229–244.

Ekinci, E. E., Tiryaki, A. M., and Dikenelli, O. (2008). Goal oriented agent testing revisited. In *Agent Oriented Software Engineering 2008*. Springer Verlag.

Ekinci, E. E., Tiryaki, A. M., Gurcan, O., and Dikenelli, O. (2007). A planner infrastructure for semantic web enabled agents. In *OTM Workshops*, volume 4805 of *Lecture Notes in Computer Science*, pages 95–104, Vilamoura, Algarve, Portugal. Springer.

Engelfriet, J., Jonker, C. M., and Treur, J. (2002). Compositional verification of multi-agent systems in temporal multi-epistemic logic. *J. of Logic, Lang. and Inf.*, 11(2):195–225.

Erdur, R. C. (2001). Yazılım etmeni teknolojisinin internet tabanlı yazılım yeniden kullanımına uygulanması. *Ege Universitesi Doktora tezi*.

Erol, K., Hendler, J. A., Nau, D. S., and Tsuneto, R. (1995). A critical look at critics in htn planning. In *IJCAI*, pages 1592–1598.

FIPA (2000). Fipa agent management specification. Specification, FIPA.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA.

George, B. and Williams, L. (2003). An initial investigation of test driven development in industry. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 1135–1139, New York, NY, USA. ACM.

Gomez-sanz, J. and Pavon, J. (2004). Methodologies for developing multi-agent systems. *Journal of Universal Computer Science*, 10:359–374.

Gomez-Sanz, J. and Pavon, J. (2005). Agent oriented software engineering with ingenias. In *Proceedings of the 3rd Central and Eastern Europe Conference on Multiagent Systems*, pages 394–403. Springer Verlag, LNCS.

Graham, J. R., Decker, K. S., and Mersic, M. (2003). Decaf - a flexible multi agent system architecture. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):7–27.

Haftmann, F., Kossmann, D., and Lo, E. (2007). A framework for efficient regression tests on database applications. *The VLDB Journal*, 16(1):145–164.

Hamill, P. (2004). *Unit Test Frameworks*. O'Reilly Media.

Hanks, S., Pollack, M., and Cohen, P. (1993). Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. *AI Magazine*, 14(4):17–42. Also published as TR 93-06-05, University of

Washington.

Henderson-Sellers, B. (2005). Agent-oriented methodologies: method engineering and metamodelling. *SIGSOFT Softw. Eng. Notes*, 30(4):1–1.

Huber, M. J. (1999). Jam: a bdi-theoretic mobile agent architecture. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 236–243, New York, NY, USA. ACM.

Huget, M.-P. (2004). Agent uml notation for multiagent system design. *IEEE Internet Computing*, 8(4):63–71.

Iglesias, C. A., Garijo, M., Centeno-GonzÃ¡lez, J., and Velasco, J. R. (1997). Analysis and design of multiagent systems using mas-common kads. In Singh, M. P., Rao, A. S., and Wooldridge, M., editors, *ATAL*, volume 1365 of *Lecture Notes in Computer Science*, pages 313–327. Springer.

Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture–Practice and Promise*. Addison-Wesley Professional.

Knublauch, H. (2002). Extreme programming of multi-agent systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 704–711, New York, NY, USA. ACM Press.

Knublauch, H. and Rose, T. (2002). Tool-supported process analysis and design for the development of multi-agent systems. In *AOSE*, pages 186–197.

Kruchten, P. (1998). *The Rational Unified Process, An Introduction*. Addison Wesley.

Li, H. and Thompson, S. (2008). Tool Support for Refactoring Functional Programs. In *Partial Evaluation and Program Manipulation*, San Francisco, California, USA.

Lind, J. (2001). *Iterative Software Engineering for Multiagent Systems: The MASSIVE Method*, volume 1994 of *Lecture Notes in Computer Science*. Springer.

Link, J. and Frolich, P. (2003). *Unit Testing in Java: How Tests Drive the Code*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Massol, V. and Husted, T. (2003). *JUnit in Action*. Manning Publications Co., Greenwich, CT, USA.

Mens, T. and Gorp, P. V. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142.

Meszaros, G. (2007). *XUnit Test Patterns: Refactoring Test Code*. Addison-Wesley.

Morreale, V., Bonura, S., Francaviglia, G., Centineo, F., Cossentino, M., and Gaglio, S. (2006). Goal-oriented development of bdi agents: The practionist approach. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 66–72, Washington, DC, USA. IEEE Computer Society.

Muller-Ettrich, G. (1997). System development with v-model and uml. In *UML Workshop*, pages 238–249.

Nguyen, D. C., Perini, A., and Tonella, P. (2007). A goal-oriented software testing methodology. In Luck, M. and Padgham, L., editors, *AOSE*, volume 4951 of *Lecture Notes in Computer Science*, pages 58–72. Springer.

Nilsson, N. J., Russell, S. J., and Norvig, P. (1996). Artificial intelligence: A modern approach. *Artif. Intell.*, 82(1-2):369–380.

Norling, E. and Ritter, F. E. (2001). Embodying the jack agent architecture. In *AI 2001: Advances in Artificial Intelligence, volume 2256 of Springer Lecture Notes in Artificial Intelligence*, pages 368–377. Springer.

Omicini, A. (2001). Soda: societies and infrastructures in the analysis and design of agent-based systems. In *First international workshop, AOSE 2000 on Agent-oriented software engineering*, pages 185–193, Secaucus, NJ, USA. Springer-Verlag New York, Inc.

Padgham, L. and Winikoff, M. (2002). Prometheus: a methodology for developing intelligent agents. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 37–38, New York, NY, USA. ACM Press.

Paolucci, M., Kalp, D., Pannu, A. S., Shehory, O., and Sycara, K. (1999). A planning component for retsina agents. In *Lecture Notes in Artificial Intelligence, Intelligent Agents VI*.

Porres, I. (2005). Rule-based update transformations and their application to model refactorings. *Software and System Modeling*, 4(4):368–385.

Rao, A. S. and Georgeff, M. P. (1995). Bdi-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Sean, Frank, Hendler, J., Horrocks, I., Mcguinness, D. L., Patel-Schneider,

P. F., and Stein, L. A. (2004). Owl web ontology language reference. Technical report.

Seidewitz, E. (2003). What models mean. *IEEE Softw.*, 20(5):26–32.

Sendall, S. and Kozaczynski, W. (2003). Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20:42–45.

Shoham, Y. (1997). An overview of agent-oriented programming. *Software agents*, pages 271–290.

Sprinkle, J., Agrawal, A., Levendovszky, T., Shi, F., and Karsai, G. (2003). Domain model translation using graph transformations. In *ECBS*, pages 159–167. IEEE Computer Society.

Stollberg, M. and Rhomberg, F. (2006). Survey on goal-driven architectures. Technical Report DERI-TR-2006-06-04, DERI, University of Innsbruck, Austria.

Sycara, K., Paolucci, M., Velsen, M. V., and Giampapa, J. (2003). The retsina mas infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48.

Sycara, K. P. (1998). Multiagent systems. *AI Magazine*, 19(2):79–92.

Tappenden, A., Beatty, P., and Miller, J. (2005). Agile security testing of web-based systems via httpunit. In *AGILE*, pages 29–38.

Thompson, S. and Reinke, C. (2001). Refactoring functional programs. Technical report, Computing Laboratory, University of Kent.

Tiryaki, A. M., Ekinci, E. E., and Dikenelli, O. (2008). Refactoring in multi agent system development. *Lecture Notes in Artificial Intelligence*, 5244:183–194.

Tiryaki, A. M., Oztuna, S., Dikenelli, O., and Erdur, R. C. (2006). Sunit: A unit testing framework for test driven development of multi-agent systems. In Padgham, L. and Zambonelli, F., editors, *AOSE*, volume 4405 of *Lecture Notes in Computer Science*, pages 156–173. Springer.

Tiryaki, A. M., Tamer, S. O., Ekinci, E. E., Dikenelli, O., and Erdur, R. C. (2007). Etmen tabanlı test yönelimli geliştirim. In *Ulusal Yazılım Mühendisliği Sempozyumu-UYMS 2007*, pages 33–40.

Triebig, C. and Klugl, F. (2008). Refactoring of agent-based simulation models. In *Multikonferenz Wirtschaftsinformatik*.

van Deursen, A., Moonen, L., van den Bergh, A., and Kok, G. (2001). Refactoring test code. In Marchesi, M. and Succi, G., editors, *International Conference on Extreme Programming and Flexible Processes*.

Williamson, M., Decker, K., and Sycara, K. (1996). Unified information and control flow in hierarchical task networks. In *Theories of Action, Planning, and Robot Control: Bridging the Gap: Proceedings of the 1996 AAAI Workshop*, pages 142–150, Menlo Park, California. AAAI Press.

Wooldridge, M. and Ciancarini, P. (2001). Agent-oriented software engineering: the state of the art. In *First international workshop, AOSE 2000 on Agent-oriented software engineering*, pages 1–28, Secaucus, NJ, USA. Springer-Verlag New York, Inc.

Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312.

Zambonelli, F., Jennings, N. R., Omicini, A., and Wooldridge, M. (2001).

Agent-oriented software engineering for internet applications. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 326–346.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370.

Zambonelli, F. and Omicini, A. (2004). Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283.

EkA ÇES GELİŞTİRİMİNDE KÖTÜ KOKULAR

A.1 Yanlış Sorumluluk

Etmen sistemlerdeki roller genel hedeflere ulaşabilmek için sistemdeki sorumlulukları paylaşırlar. Etmen sistemler gibi karmaşık ve her zaman yeni hedeflerin ve/veya rollerin sisteme eklemesine açık yazılım sistemlerinin geliştirim sürecinin erken safhalarında sorumlulukları rollere tam olarak dağıtabilmek neredeyse imkansızdır.

Her rol sistemde pek çok sorumluluğa sahip olabilir. Fakat, bu sorumlulukları bir bütün olarak ifade eden genel bir karakteristiğe sahiptir (bir arabulucu rolünün sistemdeki etmenler arasında arabuluculuk yapma işlemi ile ilgili sorumluluklara sahip olması gibi). Benzer sorumlulukları yerine getirecek etmen planlarının birbirleriyle etkileşimli bir şekilde çalışması büyük bir olasılıktır. Bu sorumlulukları tek bir rol altında toplamak sorumluluklar arasındaki yoğun etkileşimleri rol içerisinde tutmayı sağlar. Bir rol içerisindeki etkileşim roller arası etkileşime göre çok daha az iletişim yükü gerektirir. Bu yüzden, benzer sorumlulukları tek bir rol altında toplamak tüm sistemin performansı için önemlidir.

Rollerin kendi genel karakteristiklerine uygun olmayan sorumluluklara sahip olmalarının ortaya çıkardığı diğer bir sorun ise rollerin başka sistemler tarafından yeniden kullanılabilirliğini düşürmesidir. Geliştirilen her rolü sistem içerisinde bir alt-sistem olarak ele aldığımızda diğerlerinden farklı olan bir sorumluluk sistemin modülerliğini olumsuz yönde

etkileyecektir.

Yanlış Sorumluluk kötü kokusu çoklu etmen sistemindeki bir rolün kendi genel karakteristiğine uygun olmayan bir sorumluluğa sahip olduğu durumu tanımlamaktadır. Sistemin karmaşıklığı ve genellikle çok sayıda rol ve sorumluluk içermesi nedeniyle bu kötü kokuyla ÇES geliştirimi sırasında sıkça karşılaşılır. Bunun yanı sıra çalışma zamanında da yeni sorumlulukların ve rollerin eklenmesi mümkün olabileceğinden *Yanlış Sorumluluk* kötü kokusu ÇES'in çalışmasının ileyleyen zamanlarında da oluşabilir.

Bir rol üzerinde *Yanlış Sorumluluk* kötü kokusu fark edildiğinde rolün genel karakteristiğine uymayan veya bir başka role daha uygun olan sorumluluk *Sorumluluk Taşı* yeniden yapılandırması kullanılarak başka bir role taşınmalıdır.

A.2 Aşırı Yüklenmiş Rol

Tezde önerilen hedef yönelimli geliştirim yaklaşımı etmenleri sadece geliştirilen rollerin çalıştırılması için kullanılan bir çalışma zamanı desteği olarak kabul etmektedir. Çoklu etmen sistemleri içerisinde çalışan etmenlerin iş yükleri de diğer tüm özellikleri gibi kendisine atanan rollerden gelmektedir. Bu yüzden, çoklu etmen sistemlerinde etmenlerin değil rollerin iş yüklerinden bahsedilmelidir. Bir rolün iş yükünü sistem içerisindeki sorumluluklarının diğer rollerle olan etkileşimlerinin sayısı belirler.

Etmen sistem geliştirimi sırasında sistem içerisindeki bazı kritik rollere daha fazla sorumluluk düşer. Rollerin fazla sorumluluğa sahip olması bu rolleri oynayacak etmenlerin çalışma performansını doğrudan et-

kileyecektir. Bu yüzden, sorumlulukları ve buna bağlı olarak diğer rollerle olan iletişimleri çok fazla olan bir rol tüm sistemin güvenilirliği için önemli bir sorundur.

Aşırı Yüklenmiş Rol kötü kokusu geliştirilmekte veya çalışmakta olan ÇES içerisinde sorumlulukları çok fazla artan bir rolün bulunduğu durumu tanımlar. Bu kötü koku fark edildiğinde *Sorumluluk Taşı* yeniden yapılandırması ile sorumluluklarından bazılarının sistemde var olan veya yeni yaratılan diğer bir role taşınması rolün yükünün hafiflemesini sağlayacaktır. Kötü kokudan kurtulmanın diğer bir yolu ise *Rol Böl* yeniden yapılandırması kullanılarak aşırı yüklenmiş rolün daha küçük rollere bölünmesidir.

A.3 Büyük Davranış

Büyük Davranış kötü kokusu etmen planları geliştirirken sıklıkla karşılaşılan, içerisindeki yapıda çok sayıda görev ve bu görevler arasındaki bağları tutan bir davranışın bulunduğu durumu tanımlamaktadır.

Büyük davranışların sistemde modülerlik ve yeniden kullanılabilirlik açılarından sorunlar yaratır. Davranışlar gerçekleştiriminde kullanılan programlama tekniği izin verdiği takdirde daha büyük diğer davranışlar içerisinde yeniden kullanılabilirler. Büyük davranışlar modülerliği azaltarak sahip oldukları görevlerin bir grubu tarafından gerçekleştirilebilecek bir işlevselliğin yeniden kullanılabilme fırsatının kaçmasına neden olurlar. Böyle bir durumda sadece davranışın tamamını yeniden kullanabilirsiniz.

Büyük davranışların getireceği diğer bir sorun ise çok sayıda görevden oluşan karmaşık bir yapıdaki bu davranışların değiştirilmesi ve

bakımının zor olabilmesidir. Bu tür davranışların okunabilirliği düşüktür ve içerdiği görevler arasındaki bağlantılar nedeniyle yapılacak ufak bir değişikliğin zaman ve efor maliyeti küçük davranışlara göre daha fazla olur.

Planların değiştirilmesi ve bakımı küçük davranış yapıları ile daha kolay yapılabilir. Küçük davranışlar daha anlaşılır ve daha yönetilebilirdir.

Büyük Davranış kötü kokusu ile karşılaşıldığında *Davranış Çıkarımı* yeniden yapılandırması kullanılarak davranış içerisinden daha küçük davranışlar çekilebilir. Bu şekilde, hem davranışın bakımı kolaylaşır hem de yeniden kullanılabilir daha küçük bir davranış elde edilir.

A.4 Plan İçerisinde Çalışma Kararı

Geliştirilen davranışlar sistemdeki rol hedeflerinden biri ile eşleştirilerek plan haline getirilirler. Başka bir deyişle, planlar rol hedefleri ile eşleştirilmiş davranışlardır. Etmenler belirli bir hedefi başarabilmek için planları çalıştırırlar. Planın çalışma kararı ortamın o andaki durumu ve eldeki bilgiler değerlendirilerek etmenin iç yapısındaki mekanizmalar tarafından verilir. Bu mekanizmalar etmenin belirli bir durum karşısında en uygun planı çalıştırarak akılcı davranmasını sağlayan, yapay zeka teknikleriyle desteklenen mekanizmalardır.

Etmen, başarmak istediği hedefle eşleştirdiği planın çalışmasını tetikledikten sonra görevlerin çalışma sırasına karışmaz. Bu sıra planın tuttuğu yapı ile belirlenir. Plan yapısında bulunan görevlerin çalışmasına daha önce çalışan görevlerin sonuçlarına ve görevler arasındaki bağlantılara göre çalışma zamanında karar verilir. Bu yapı doğrultusunda, girdi-

leri hazırlanan görevler başka bir karar mekanizmasına gerek duyulmadan çalıştırılır.

Plan İçerisinde Çalışma Kararı kötü kokusu bir plan yapısı içerisinde ne zaman işletilip işletilmeyeceği kararı plan akışı ile verilemeyen, etmen tarafından özerk olarak verilmesi gereken bir görev veya görev grubu bulunduğu durumu tanımlamaktadır.

Bu kötü koku fark edildiğinde ayrı bir plan olarak çalışması istenen görev veya görev grubu *Plan Çıkarımı* veya *Davranıştan Plana* yeniden yapılandırmaları kullanılarak bir rol hedefi ile eşleşmiş yeni bir plan olarak tanımlanır.

A.5 İş Yavaşlatan Görev Zinciri

Hedefleri başaracak planların gerçekleştirim sırasında sistemdeki rollerin her biri için pek çok görev (davranış ve eylem) elde edilebilir. Özellikle büyük bir sistem geliştirilirken aynı işlevselliği farklı yollarla gerçekleştiren görevlerin oluşması olağan bir durumdur. Böyle bir durumda, bir işlevselliği gerçekleştirmek için daha kısa bir yol bulunabilir. Çok daha az sayıda görev ile başarılabilecek bir işin gereksiz yere büyük bir görev grubu tarafından gerçekleştirilmesi etmen planlarının karmaşıklığını arttırır ve bakımını zorlaştırır.

Etmen sistem geliştirim deneyimlerimiz göstermiştir ki; görevleri mümkün olduğunca birbirinden ayırarak küçük geliştirmek modülerliği arttırarak bu görevlerin yeniden kullanımını önemli ölçüde arttırmaktadır. Fakat, bazen yeniden yapılandırmalar sonucunda gereksiz yere çok küçülebilen görevler ufak bir işin yapılması içini bile plan içerisine büyük

bir görev sıralaması eklenmesine neden olmaktadır. Bu durum da planların gereksiz yere büyümesini ve karmaşıklaşmasına neden olarak plan geliştirimini zorlaştırabilmektedir.

İş Yavaşlatan Görev Zinciri kötü kokusu bir işin daha az sayıda görev tarafından yerine getirilebilme fırsatı varken çok sayıda görevin işbirliği içerisinde çalışması ile gerçekleştirildiği durumu tanımlar. Bu kötü koku fark edildiğinde *Görevleri Bir Görev İle Yer Değiştir* yeniden yapılandırması gruptaki görevler için bir veya daha fazla uygulanarak büyük görev grubunun aynı işlevselliği yerine getiren ve daha az sayıda görev içeren diğer görev grubu ile değiştirilmesi bu işlevselliğe gereksinim duyulan plan yapılarının sadeleşmesine neden olarak bakımını kolaylaştıracaktır. Bu değişimin yapılabilmesi için iki görev grubunun da grup dışındaki görevlerden aynı girdileri alması ve grup dışarısına aynı sonuçları göndermesi gerekmektedir. Bu şekilde büyük görev grubu aynı girdileri alıp aynı çıktıları veren daha küçük bir görev grubu ile değiştirilmiş olur.

A.6 Tekrarlanan Davranış Yapısı

SEAGENT platformu ile ÇES geliştirim deneyimlerimiz sırasında en sık karşılaştığımız sorunlardan bir tanesi benzer davranış yapılarının bir çok plan içerisinde tekrar etmesi olmuştur. Örneğin çoklu etmen sistemleri için tanımlanmış ortak iletişim protokollerinin gerçekleştirilmesi sırasında aynı protokolü kullanan tüm planlarda benzer yapıların oluşması doğaldır. Böyle bir durumda benzer davranış yapısının her plan içerisinde tekrar tekrar gerçekleştirilmek zorunda kalınması sistemin geliştirim maliyetini arttırmaktadır.

Tekrarlanan Davranış Yapısı kötü kokusu benzer davranış yapılarının birden fazla davranış içerisinde tekrar etmesini tanımlar. Böyle bir durumun oluşması davranıl yapısının yeniden kullanılabilir bir yol içerisinde geliştirilmediğini gösterir. Tekrar eden yapının farklı davranışlar içerisinden yeniden kullanımını sağlamak için bu yapı üzerinde soyutlama yapılmalıdır. *Süper-Davranış Çıkarımı* yeniden yapılandırması kullanılarak tekrar eden yapının soyut bir davranışa taşınması ve somut davranışların sadece kendilerine özel yapıyı tutarak sadeleşmesi sağlanabilir.

Bir davranış yapısı sistem içerisindeki pek çok plan içerisinde tekrarlanıyorsa bu yapı üzerinde soyutlama yapılarak elde edilen bir davranış deseni sisteme daha sonra eklenecek benzer davranışların bu davranış deseninin genişleterek daha kolay geliştirilebilmesine olanak sağlamaktadır..

A.7 Anlamsız Davranış

Her davranışın genel bir amacı vardır. Davranışın alt-görevleri ise bu ortak amaç için birlikte çalışırlar. Bir davranış içerisindeki alt-görevler ortak bir amaç için çalışmıyorsa davranışın amacını belirlemek mümkün olmayacaktır. Ortak bir amaca sahip olmayan görevlerin oluşturduğu bir davranışın yeniden kullanımının da düşünülmeyeceğinden dolayı bu görevlerin gereksiz yere bir davranış altında tutulduğu söylenebilir.

Anlamsız Davranış kötü kokusu ortak bir amacı olmayan görevlerden oluşan bir görev grubunun gereksiz yere bir davranışın alt-görevleri olarak tutulduğu durumu tanımlamaktadır. Planlar arasında yeniden kul-

lanılamayan bir davranış geliştiricilerin kafasını karıştırmakla beraber aynı zamanda bu görevlerin daha düzgün şekilde başka davranışlar altında toplanarak yeniden kullanılabilmesi fırsatlarının da kaçmasına neden olabilir.

Anlamsız Davranış kötü kokusu fark edildiğinde davranışın altgörevleri üzerinde *Davranıştan Kurtar* yeniden yapılandırması uygulanarak bu görevler davranış yapısından çıkarılabilir.

A.8 Eylem Kodu ve Parametresi Tekrarı

Davranışlar görevlerin birbirine bağlanmasından oluşan yapıyı tutarken eylemler çalışabilir koda sahip en küçük görevlerdir. Tıpkı davranış yapılarının tekrar etmesi gibi eylemler içerisindeki çalışabilir kod ve bu eylemlerin giriş çıkış parametreleri de bir çok eylem içerisinde tekrar edebilir. Eylemler davranışlara göre daha küçük ve daha çok sayıda olan görevler olduklarından dolayı eylemler arası kod ve parametre tekrarı daha fazla karşılaşılan bir durumdur.

Karmaşık etmen sistemleri çok sayıda rol, bu roller çok sayıda plan ve bu planlar da çok sayıda eylem içerebilirler. Aynı zamanda bir eylem bir çok davranış içerisinde aynı işlevselliği yerine getirmek üzere kullanılabilir. Gerçekleştirilen eylemlerin bazıları aynı veya benzer girdi-çıktı parametrelerine ve\veya çalıştırılabilir kod parçasına sahip olabilirler. Bu karmaşa içerisinde eylemlerin yönetilebilmesi için yeniden kullanıma izin verecek şekilde geliştirilmeleri gerekir.

Eylem Kodu ve Parametresi Tekrarı kötü kokusu benzer çalışabilir kod parçalarının ve giriş-çıkış parametrelerinin birden fazla eylem

içerisinde tekrar etmesiyle oluşur. Bu kötü koku ile karşılaşıldığında *Süper-Eylem Çıkarımı* yeniden yapılandırması uygulanarak eylemler içerisinde tekrar eden kodu ve parametreleri tutan soyut bir eylem yaratılabilir. Bu soyutlama sonucunda daha kolay yönetilebilir eylemler elde edilir.

A.9 Büyük Eylem

Büyük işlevsellikleri yerine getirmek üzere oluşturulan büyük eylemler eylemlerin farklı planlar içerisinde yeniden kullanılabilirliğini düşüren faktörlerden biridir. Büyük bir işlevselliği tek bir eylem ile gerçekleştirmek, bu büyük işlevsellik içerisindeki daha küçük işlevselliklerin başka davranış yapıları içerisinde yeniden kullanılabilmesi fırsatlarının kaçmasına neden olur.

Eylemler mümkün olduğunca yeniden kullanılabilir, küçük eylemler şeklinde geliştirilmelidir. Bu, sistemin modülerliğini arttırarak daha sade, anlaşılabilir ve yönetilebilir eylemlerin oluşmasını sağlayacaktır. Bununla birlikte, tek başına bir amacı olmayacak kadar küçük eylemler de sistemde karmaşa yaratır. Bu yüzden, eylemlerin anlamlı kalacak seviyeye kadar küçültülmesi faydalı olacaktır.

Büyük Eylem kötü kokusu fark edildiğinde bu eylem *Eylem Böl* yeniden yapılandırması kullanılarak daha küçük eylemlere bölünebilir. Oluşan küçük eylemler küçük işlevselliklerin de diğer davranış yapıları içerisinden yeniden kullanımına izin verir.

A.10 Eylem İçerisinde Akış Kararı

Etmen planları içerisindeki görevlerin çalışma sırasına planın yapısına ve çalışan görevlerin sonuçlarına göre çalışma zamanında dinamik olarak karar verilir. Görevler birbirlerine bilgi akış bağları ile bağlanırlar. Bir görevin çalışabilmesi için girdi olarak beklediği tüm bilgilerin hazır olması gerekir. Bu bilgi gereksinimi de daha önce çalışan eylemlerin çıktılarından karşılanır. Burada görevlerin çalışma sırasını dinamik hale getiren bir görevin çalışması sonucunda farklı sonuç durumları oluşturabilmesidir. Bu sonuç durumlarının çıktıları farklı görevlerin girdilerini sağlayacaklarından bu görev sonucunda hangi görevin çalışmasının tetikleneceğine ancak çalışma zamanında görevin çalışması sonucunda ortaya çıkan sonuç durumuyla karar verilebilir. Bu da, tasarım sırasında elde edilen plan yapılarının çalışma zamanında dinamik olarak farklı görev sıralamaları ile ilerleyebilmesine olanak sağlar.

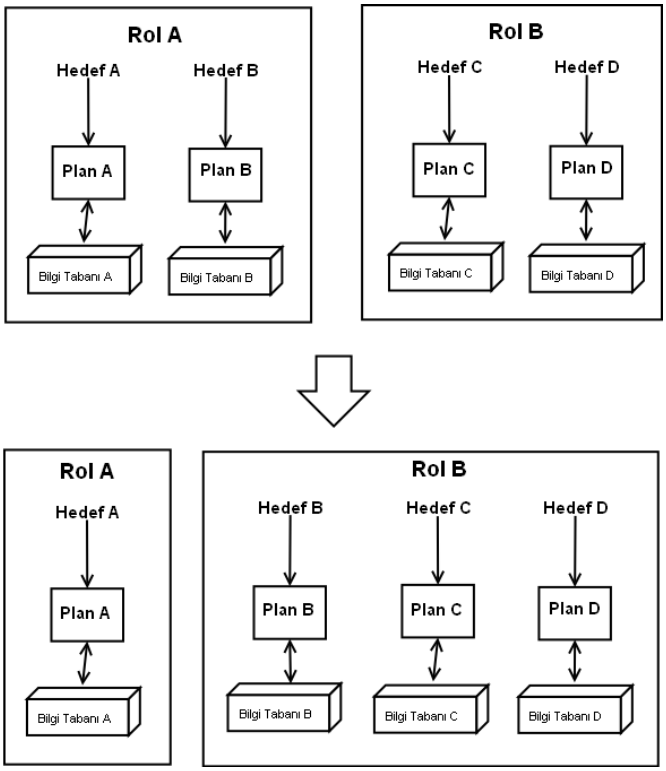
Eylem İçerisinde Akış Kararı kötü kokusu çalışabilir kodu içerisinde çalışma kararının böyle bir akış kararı ile dinamik olarak karar verilmesi gereken bir kod parçası olan bir eylemi göstermektedir. Böyle bir durumda, dinamik karar gerektiren kod parçasının çalışması sadece eylem içerisindeki durağan kod sıralamasıyla verilmektedir.

ÇES geliştirimi sırasında bu kötü koku ile karşılaşıldığında akış kararı gerektiren kod parçasını içeren eylem üzerinde *Eylem Böl* yeniden yapılandırması uygulanarak dinamik karar gerektiren kod parçası yeni bir eyleme çekilir. Böylelikle, yeni eyleme alınan kod parçasının çalışmasına plandaki diğer görevlerin sonuçlarına göre karar verilebilir.

EkB ROL SEVİYESİNDEKİ YENİDEN YAPILANDIRMA DESENLERİ

B.1 Sorumluluk Taşı

Farklı bir rol tarafından başarılması gereken bir rol hedefiniz olduğu durumlarda kullanılır. Böyle bir durumda, hedefi eşleştirildiği planlar ve bu planlar tarafından kullanılan bilgi tabanları ile birlikte başka bir role taşı



Şekil B.1. *Sorumluluk Taşı* yeniden yapılandırması

Motivasyon

Sorumluluk Taşı yeniden yapılandırması rol seviyesinde en çok başvurulan yeniden yapılandırmalardan biridir. Etmen sistemler dağıtık ve karmaşık sistemler olduklarında dolayı sistem içerisindeki sorumlulukların geliştirimin başlarında rollere düzgün bir şekilde paylaştırılması zordur. Bununla birlikte, bu tür sistemlerin dinamik doğası nedeniyle rollerin sorumlulukları sistemin çalışması sırasında bile değişebilir.

Etmen sistemlerde roller arasındaki yapı ve rollerin sorumlulukları sürekli olarak değişebilir. Bir rol üzerinde *Yanlış Sorumluluk* kötü kokusu sezildiğinde rolün genel amaçları dışında kalan sorumluluğu sistemde var olan diğer bir role veya yeni oluşturulacak bir role taşınmalıdır. Rollerin sorumlulukları hedef olarak tanımlandığı için sorumluluk taşınması bir hedefin bir rolden diğer bir role aktarılması işlemidir. Bu süreç içerisinde taşınmakta olan hedefi başarmak üzere bu hedefle eşleştirilmiş olan etmen planları ve bu planların kullandığı bilgi tabanları da hedef role taşınmalıdır.

Sorumluluk Taşı yeniden yapılandırmasının uygulanmasını gerektirebilecek diğer bir kötü koku da *Aşırı Yüklenmiş Rol* kötü kokusudur. Bir rol kendisine çok fazla sorumluluk atanmasından dolayı aşırı yüklenmiş ise bu sorumlulukların bazılarının başka rollere taşınması rolün yükünü hafifletebilir.

Bu yeniden yapılandırma sonucunda bir roldeki bir sorumluluk kendisini başaran planlar ve bu planların kullandığı bilgi tabanları ile birlikte başka bir role taşınmış olur. Eğer roller etmenlere atanmış durumda ise kaynak ve hedef rolü oynayan etmenler bu değişiklikle ilgili haberdar edilmelidir.

Mekanikler

1. Taşınmak istenen sorumluluk yeni bir role eklenmek isteniyorsa yeni bir rol yarat.

2. Sorumluluğu tanımlayan rol hedefini hedef role taşı

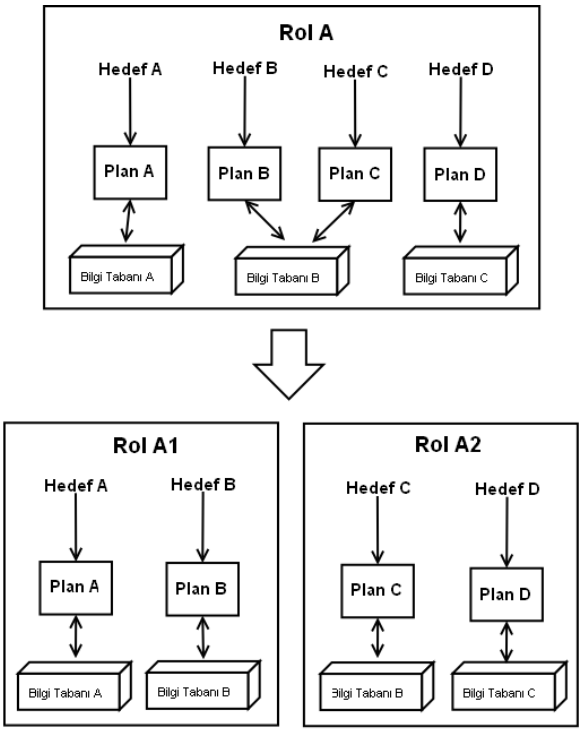
3. Taşınan hedeflerin kullandığı bilgi tabanlarını tara. Bu bilgi tabanlarının her biri için bilgi tabanı başka hedefler tarafından da kullanılıyorsa bilgi tabanını hedef role kopyala. Sadece taşınan hedefler tarafından kullanılan bilgi tabanlarını hedef role taşı

4. Taşınan hedefin yeni rolde başarımını test et.

5. Yeni role taşınan hedefin etkileşimde olduğu kaynak roldeki diğer hedeflerle birlikte çalışmasını test et.

B.2 Rol Böl

Sorumlulukları alt-rollere dağıtılması gereken bir rol belirlendiği durumlarda uygulanır. Böyle bir durumda alt-rolleri oluştur, hedefleri rol bölme stratejisine göre bu hedefleri başaran planlar ve bu planların kullandığı bilgi tabanları ile birlikte alt-rollere dağıt.



Şekil B.2. *Rol Böl* yeniden yapılandırması

Motivasyon

Rol Böl yeniden yapılandırması mekanikleri içerisinde *Sorumluluk Taşı* yeniden yapılandırmasını içeren üst düzey bir yeniden yapılandırmadır. Bu yeniden yapılandırmanın amacı çoklu etmen sistemindeki bir rolün daha küçük rollere bölünmesini sağlamaktır.Böylelikle orijinal rolün sorumlulukları daha küçük rollere diğer bir deyişle orijinal rolün alt-rollerine dağıtılmış olur.

Aşırı Yüklenmiş Rol kötü kokusu fark edildiğinde aşırı yüklenmiş

rolün sorumluluklarının birden fazla role dağıtılması ile aşırı yüklemenin üstesinden gelinebilir. Geliştirici bir rol bölme stratejisi belirleyerek *Rol Böl* yeniden yapılandırmasını uygular. Rol bölme stratejisi rolün hangi rollere bölüneceğini ve rolün hedeflerinden hangisinin hangi alt-role taşınacağını belirler.

Yeniden yapılandırma operasyonu sonucunda orijinal rol sistemden silinse de bu rolğn sorumlulukları yeni oluşturulan roller tarafından aynen yerine getirildiği için silinen bu rolün dışsal davranışında bir değişim olmaz, sadece bu dışsal davranış farklı roller tarafından işbirliği ile sağlanmış olur. Yani roller arasındaki iç yapı değişmesine rağmen rollerin işbirliği ile başardıkları sistem hedefinin başarımında bir değişiklik olmaz.

Mekanikler

1. Rol bölme stratejisi ile belirlenen alt-rollerin her biri için bu alt-rollere karışılık belirlenen rol hedeflerini bu role taşımak üzere *Sorumluluk Taşı* yeniden yapılandırmasını uygula

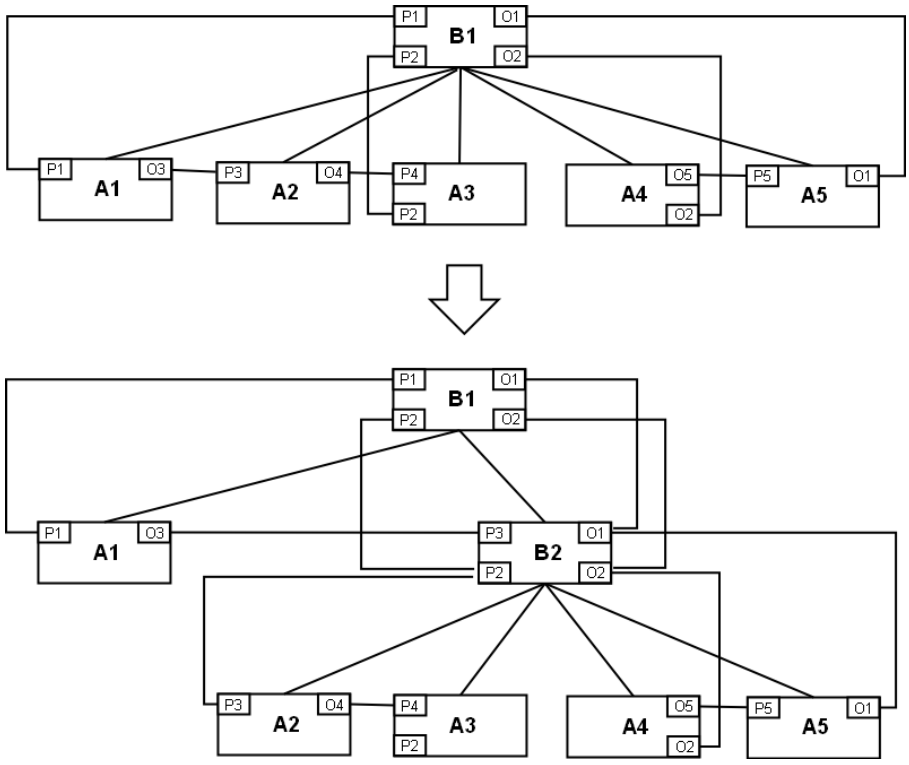
2. Orijinal rolün atandığı etmenlere yeni oluşturulan rolleri ata.

3. Orijinal rolü sil

EkC PLAN SEVİYESİNDEKİ YENİDEN YAPILANDIRMA DESENLERİ

C.1 Davranış Çıkarımı

Bir arada gruplandırılabilecek bir görev topluluğunuz olduğu durumlarda kullanılır Görev topluluğunu ismi tüm bu görevlerin ortak amacını tanımlayan bir davranışın içerisinde topla.



Şekil C.1. *Davranış Çıkarımı* yeniden yapılandırması

Motivasyon

Davranış Çıkarımı etmen planlarının geliştirimindeki en çok kullanılan yeniden yapılandırma türlerinden biridir. Bu yeniden yapılandırma bir davranış çok fazla sayıda görevi barındırdığı zaman uygulanmalıdır. Böyle bir durumda, bu görevlerin bir grubu yeni bir davranış içerisinde toplanır.

Küçük davranışlar ÇES senaryolarının geliştiriminde çeşitli avantajlar sağlarlar. İlk olarak, görevlerin diğer görevler içerisinde yeniden kullanılabilirliğini arttırır. Grup içerisindeki görevlerin tümü sadece bir davranış yoluyla yeniden kullanılabilirler. İkincisi, planların değiştirilmesi ve bakımı küçük davranış yapıları ile daha kolay yapılabilir. Diğer bir avantaj küçük davranışlar tarafından oluşturulmuş planların daha anlaşılır ve daha yönetilebilir olmasıdır.

Plan içerisindeki bir davranış için *Büyük Davranış* kötü kokusu fark edildiğinde bu davranışı *Davranış Çıkarımı* yeniden yapılandırması ile daha küçük davranışlara bölmek davranışların bakımını kolaylaştırır ve okunabilirliğini arttırır.

Davranış Çıkarımı yeniden yapılandırmasının kullanılabileceği diğer bir durum ise *Tekrarlanan Davranış Yapısı* kötü kokusunda tanıtılmaktadır. Bu kötü kokuya göre bir davranış yapısı birden fazla davranış içerisinde tekrarlanmaktadır. Böyle bir durumda tekrar eden davranış yapısının içerdiği görevler üzerinde *Davranış Çıkarımı* yeniden yapılandırması uygulanarak yeniden kullanılabilir bir davranış elde edilebilir.

Davranış Çıkarımı yeniden yapılandırması sonucunda kendi sorumluluklarını yerine getiren ve diğer görevler tarafından kullanılabilen bir davranış elde edilir.

Mekanikler

1. Yeni bir davranış oluştur ve bu davranışı amacına uygun olarak isimlendir (davranış içerisindeki bütün görevlerin işlevsellikleri göz önüne alınarak isim verilmelidir) ,

2. Dışarı çekilmek istenen görevleri yeni davranışın içine kopyala,

3. Orijinal plan yapısı içerisindeki akış bağlarını kaynak ve hedef görevleri plandan çekilecek görevlere referans olanlar için tara. Böyle akış bağı tanımlamaları varsa bu bağları yeni davranış yapısı içerisine kopyala.

4. Yeni karmaşık görev yapısını ihtiyaç tanımlamaları içinde gönderici olarak ana görev yapısında bulunan görevlere referanslar için tara. Eğer böyle ihtiyaç tanımlamaları varsa, bu ihtiyaçların her birini yeni göreve ekle

5. Yeni karmaşık görev yapısını çekilen görevlerden hiç birisi ile bağlı olmayan sonuç durumları için tara. Eğer böyle sonuç durumu tanımlamaları varsa bu sonuç durumlarının her birini yeni göreve ekle

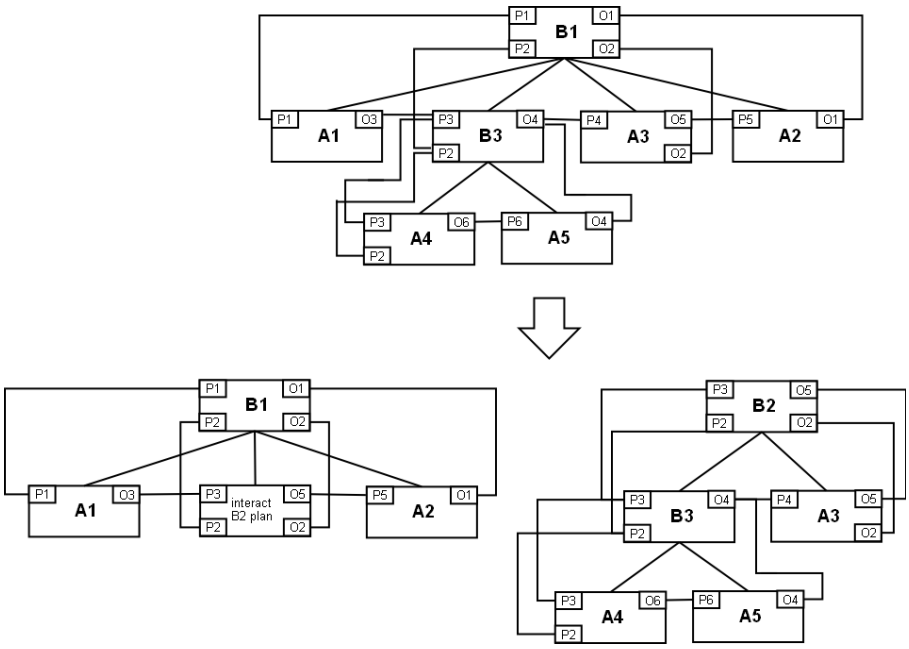
6. Yeni davranışın ihtiyaçlarının her biri için bu ihtiyaç ile alt-görevlerdeki aynı isme sahip ihtiyaç arasında kalıtım bağı oluştur.

7. Yeni davranışın sonuç durumlarının her biri için bu sonuç durumu ile alt-görevlerdeki aynı isme sahip sonuç durumu arasında ters kalıtım bağı oluştur.

8. Plandan çekilecek görevler ile oluşturulan yeni davranış üzerinde Görevleri Bir *Görev İle Yer Değiştir* yeniden yapılandırmasını uygula.

C.2 Plan Çıkarımı

Bir plan yapısı içerisinde ayrı bir plan olarak çalıştırılması gereken bir görev grubu bulunduğunda uygulanabilir. Böyle bir durumda görev grubu içerisindeki görevleri tek bir davranış içerisinde topla ve bu davranışı plan yap.



Şekil C.2. *Plan Çıkarımı* yeniden yapılandırması

Motivasyon

Plan Çıkarımı yeniden yapılandırması plan seviyesindeki iki önemli yeniden yapılandırmayı kapsayarak bir plan yapısı içerisindeki görev grubunun bir rol hedefi ile eşleştirilmiş ayrı bir plan olarak tanımlanmasını amaçlar. Bu amaç doğrultusunda önce *Davranış Çıkarımı* yeniden yapılandırması ile görevleri bir davranış içerisinde toplar daha sonra da yeni oluşmuş bu davranış üzerinde *Davranıştan Plana* yeniden yapılandırmasını uygulayarak yeni bir plan elde eder.

Plan içerisindeki görevlerin çalışmasına planın çalışması sırasında görevler arasındaki bilgi akışıyla karar verilirken planların çalışması etmenlerin iç mekanizması tarafından karar verilir. Etmen belirli bir durum karşısında hangi hedefi başarması gerektiğine ve bu hedefi başarmak için hangi planı kullanacağına özerk olarak karar verir. Bir planda *Plan İçerisinde Çalışma Kararı* kötü kokusu fark edildiğinde çalışma kararı plan akışı doğrultusunda değil de etmen tarafından özerk olarak verilmesi gereken görev grubu üzerinde *Plan Çıkarımı* yeniden yapılandırması uygulanarak bu görevleri içeren ayrı bir plan elde edilmelidir.

Bu yeniden yapılandırma sonucunda ayrı bir rol hedefi ile eşleştirilmiş yeni bir plan ve orijinal planın bu yeni plan ile etkileşim halinde çalışan yeni yapısı elde edilir.

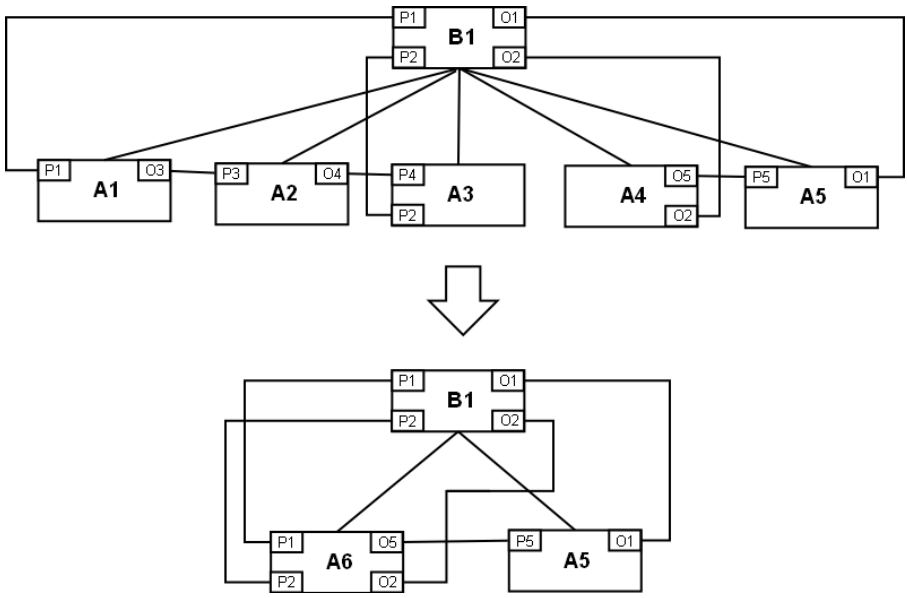
Mekanikler

1. Plan yapısından çekilmek istenen görevler üzerinde *Davranış Çıkarımı* yeniden yapılandırmasını uygula.

2. Orijinal plan yapısı içerisinde davranış olarak kullanılan yeni davranış üzerinde *Davranıştan Plana* yeniden yapılandırmasını uygula.

C.3 Görevleri Bir Görev İle Yer Değiştir

Plan içerisindeki bir görev grubunun yerine getirdiği işin tek bir görev tarafından yerine getirilebildiği durumlarda kullanılır. Böyle bir durumda görevleri yer değiştir ve görev setindeki görevlere bağlı olan bağları yeni göreve bağla.



Şekil C.3. *Görevleri Bir Görev İle Yer Değiştir* yeniden yapılandırması

Motivasyon

Görevleri Bir Görev İle Yer Değiştir yeniden yapılandırması plan içerisindeki birden fazla görev ile gerçekleştirilen bir işlevselliğin tek bir görev ile gerçekleştirilebileceği durumlarda kullanılır. Bu görevlerin plandan çıkarılarak yeni görevin plandan çıkarılan görevlerin yerine eklenmesinden sorumludur.

Çoklu etmen sistemlerinin karmaşık ve açık doğası nedeniyle, bu sistemlerin geliştirimi ve çalışması sırasında sürekli olarak yeni planlar ve yeni görevler sisteme eklenebilir. Özellikle, evrimsel bir geliştirim sürecinde her geliştirim döngüsünde sisteme yeni görevler eklenerek ilerlenir. Etmen sistemlerin bu açık doğası aynı işlevselliğin farklı yollar ile gerçekleştirilebileceği bir ortam sunar. Bu yüzden daha önceden birden fazla görevden oluşan bir görev gurubunun gerçekleştirdiği işlevselliği tek başına gerçekleştirebilen görevler elde edilebilir. Bir plan içerisinde *İş Yavaşlatan Görev Zinciri* kötü kokusu fark edildiğinde aynı işlevselliğe sahip görev grubu ile görev yer değiştirilerek planın daha sade ve okunabilir olması sağlanabilir.

Bu yer değiştirmenin yapılabilmesi için hedef görevin görev grubunun grup dışındaki diğer görevlerden beklediği ihtiyaçların ve diğer görevlere gönderdiği sonuç durumlarının aynılarına sahip olması gerekmektedir. Bu şekilde aynı girdileri alıp aynı çıktıları elde eden tek bir görev bir görev grubu yerine plana eklenmiş olur.

Mekanikleri

1. Plandan çıkarılacak görevleri sil ve yerine yeni görevi ekle.

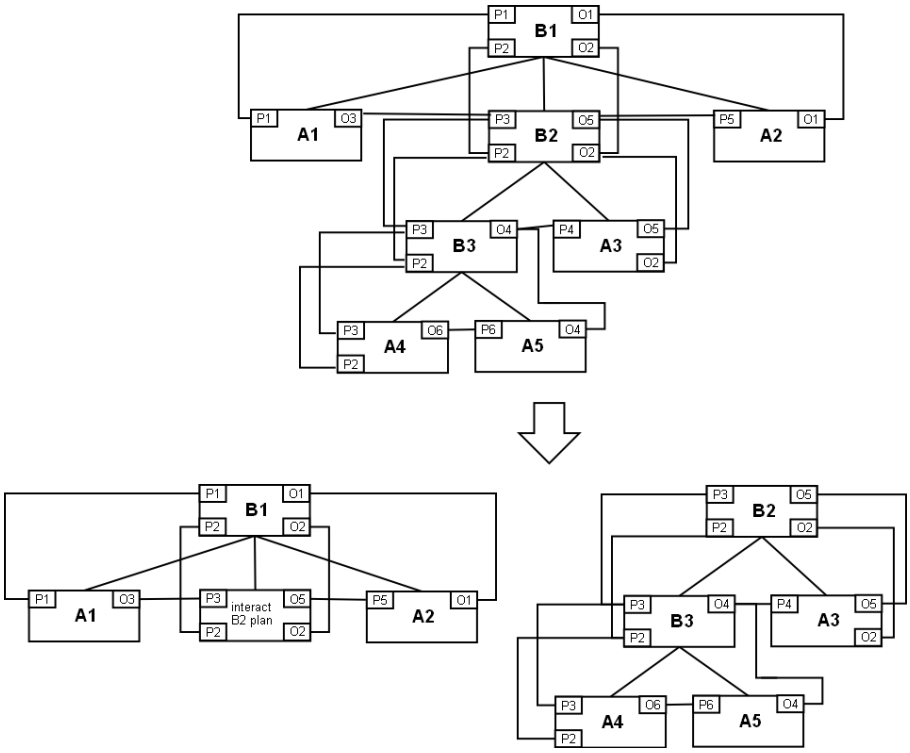
2. Plandan çıkarılacak görevlerin ihtiyaçlarına bağlı olan kalıtım ve ihtiyaç-sonuç durumu bağlarını bul ve bu bağları yeni görevin uygun ihtiyacına bağla.

3. Plandan çıkarılacak görevlerin sonuç durumlarına bağlı olan ters kalıtım ve ihtiyaç-sonuç durumu bağlarını bul ve bu bağları yeni görevin uygun sonuç durumuna bağla.

4. Plan yapısı içerisindeki ihtiyaç-sonuç durumu bağlarını kaynak ve hedef görevleri çekilen görevler olanlara referanslar için tara. Böyle bağlar varsa bu bağları plan yapısından sil.

C.4 Davranıştan Plana

Elinizde farklı bir plan olarak çalıştırılması gereken bir hedef olduğu durumlarda kullanılır. Böyle bir durumda davranışı bir rol hedefi ile eşle ve orijinal planı bu yeni planla etkileşir hale getir.



Şekil C.4. *Davranıştan Plana* yeniden yapılandırması

Motivasyon

Planlar kendi kendine çalışabilen davranışlardır. Plan yapısı içerisindeki davranışların çalışıp çalışmayacağına plan akışına göre karar verilir. Bununla birlikte etmenler hangi planın ne zaman çalıştırılacağına karar verirler. Bu karar etmenin dâhili modülleri tarafından etmenin o anki durumu (çevre, dâhili bilgi ve diğer kriterler) değerlendirilerek verilir.

Plan İçerisinde Çalışma Kararı kötü kokusu plan yapısı içerisinde

çalışmasının diğer planlardan bağımsız ve özerk olarak etmen tarafından karar verilmesi gereken görev grubu olması olarak tanımlanmaktadır. Bu kötü kokunun işaret ettiği görev grubu tek bir davranıştan oluşuyorsa bu davranış üzerinde *Davranıştan Plana* yeniden yapılandırması uygulanarak davranış bir rol hedefiyle eşleştirilmiş bağımsız bir plana dönüştürülür. ayrı bir plan haline getirilen davranış genellikle orijinal plan yapısı içerisindeki diğer görevlerle bağlantılara sahiptir. Bu yüzden, bu bağlantılar yeniden yapılandırma operasyonu sırasında ana plan ile dışarı çekilen yeni plan arasındaki etkileşimlere dönüştürülür.

Planların daha küçük planlara bölünmesi aynı zamanda görevlerin başarılması için etmenlerin birlikte çalışmasını kolaylaştırır. *Davranıştan Plana* kalıbı kullanıldığında plan daha küçük ve aynı roldeki diğer etmenler tarafından işletilebilecek yeni planlara bölünür.

Davranıştan Plana yeniden yapılandırmasının sonucunda birbirleriyle etkileşim halinde iki plan elde edilir.

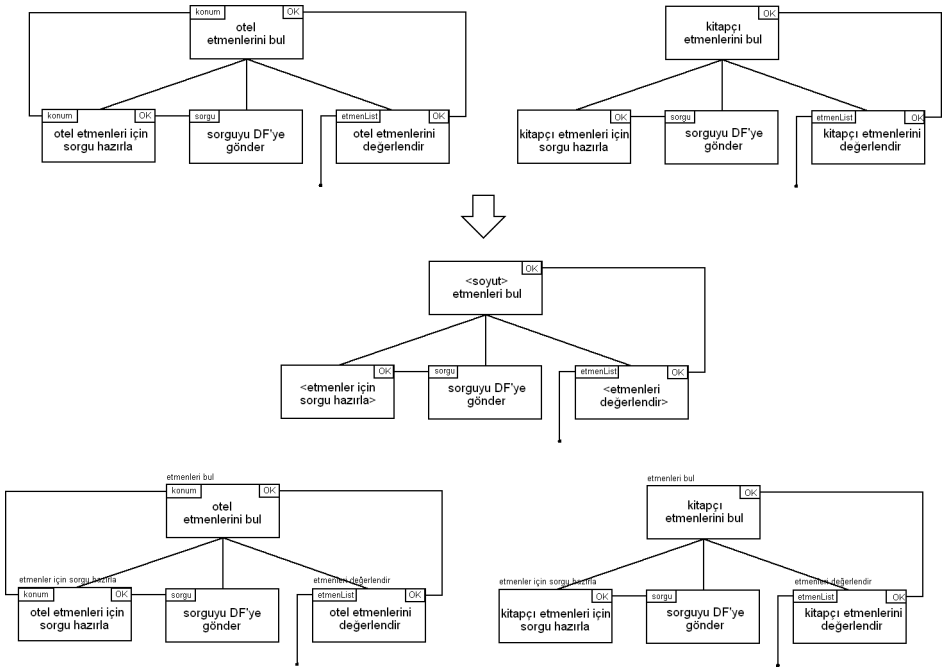
Mekanikler

1. Davranışı diğer planlardan bağımsız çalışabilir şekilde depola
2. Davranışı bir rol hedefi ile eşleştir.
3. Orijinal plan yapısını yeni planla etkileşir hale getir.

C.5 Süper-Davranış Çıkarımı

Benzer yapıya sahip davranışların olduğu durumda kullanılır. Böyle bir durumda tekrar eden yapıyı tutan soyut bir davranış oluştur ve orijinal

davranışları bu soyut davranışı genişletir hale getir.



Şekil C.5. *Süper-Davranış Çıkarımı* yeniden yapılandırması

Motivasyon

Tekrarlanan Davranış Yapısı plan seviyesi içerisindeki en fazla karşılaşılan kötü kokulardan biridir. Tekrarlanan yapının bir şekli benzer işleri tamamen aynı yapılar ile gerçekleştiren davranışlardır. Diğer bir şekli ise farklı davranış yapıları içerisinde benzer işlerin yapılmasıdır. Böyle durumlarda benzer şekilde veya aynen tekrar edilen davranış yapılarının tekrar tekrar gerçekleştirilmesi gerekir.

Süper-Davranış Çıkarımı yeniden yapılandırması kullanılarak bir kalıtım yapısı oluşturulup bu kötü kokunun üstesinden gelinebilir. Bu yeniden yapılandırma ortak davranış yapısını tutan yeni bir soyut davranış oluşturur. Yeni soyut davranışı genişleten orijinal davranışlar sadece kendileri için özelleşmiş yapılarını tutacaklarından bu planların okunabilirliği artacaktır. Soyut bir davranış oluşturmanın diğer bir önemli amacı benzer davranışlar için yeniden kullanılabilir bir davranış yapısı sunmaktır. Bu davranışlar yeniden kullanılabilir soyut davranışı genişleterek kolayca gerçekleştirilebilir.

Süper-Davranış Çıkarımı" yeniden yapılandırması sonucunda ortak yapıyı tutan ve spoer-davranış olarak adlandırılan yeni bir soyut davranış ve bu soyut davranışı genişleten somut davranışlar elde edilir.

Mekanikler

1. Yeni bir soyut davranış oluştur ve orijinal davranışlardan bu soyut davranışı genişlet.

2. Orijinal davranışlarda tekrar eden ihtiyaçları ve sonuç durumlarını yeni oluşturulan soyut davranış içerisine taşı.

3. Orijinal plan yapılarında tekrarlanan alt-görevleri soyut davranış içerisine taşı

4. Eğer ortak olmayan bir alt-görev ile bağlantılı ortak bir bağ varsa; bu ortak olmayan görev için *Süper-Eylem Çıkarımı* veya *Süper-Davranış Çıkarımı* yeniden yapılandırmasını uygulayarak yeni bir

soyut görev oluştur ve bu soyut görevi soyut davranış yapısı içerisine ekle.

5. Orijinal davranış yapıları içerisinde tekrarlanan bağları soyut davranış içerisine taşı ve ortak olmayan görevlerle bağlantılı bağları bir önceki adımda elde edilen soyut görevlerle bağlantılı olacak şekilde yeniden düzenle.

C.6 Davranıştan Kurtar

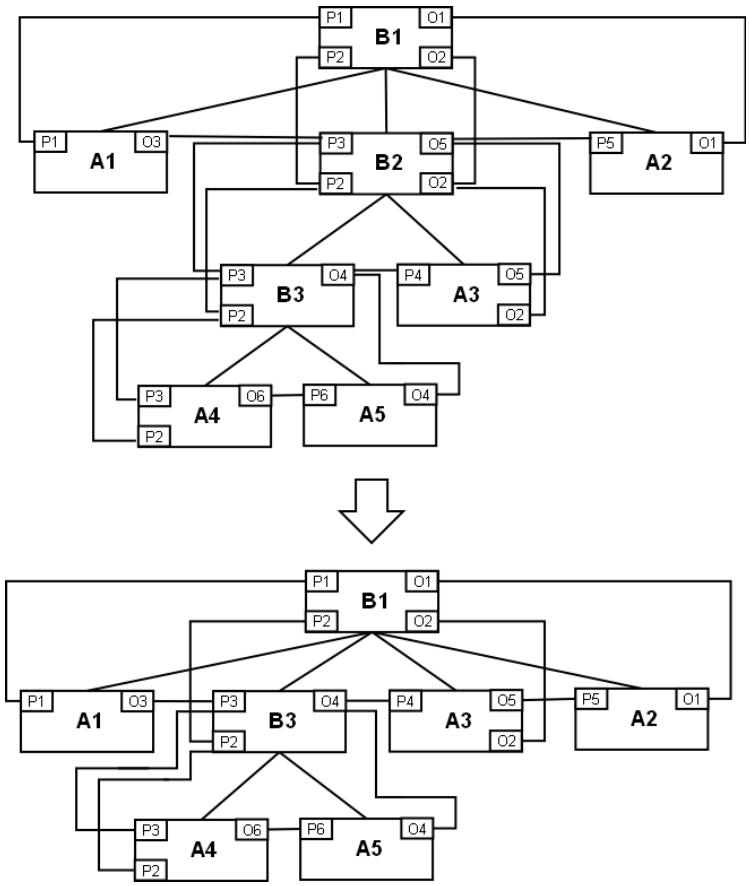
Ortak bir amaç için çalışmamalarına rağmen bir davranış içerisinde toplanmış bir görev grubu olduğu durumlarda kullanılır. Böyle bir durumda görevleri davranıştan çıkart ve plandaki bağları bu değişikliğe göre yeniden düzenle.

Motivasyon

Davranıştan Kurtar yeniden yapılandırması ortak bir amaçları olmamasına rağmen gereksiz yere bir davranış içerisinde tutulan görevlerin davranıştan kurtarılmasını sağlar ve davranışın bulunduğu plan yapılarındaki bu davranışa bağlı olan bağları davranıştan kurtarılan görevlere bağlı olacak şekilde yeniden düzenler.

Anlamsız Davranış kötü kokusunun fark edildiği durumlarda davranışın ortak bir amacı olmadığı için yeniden kullanılması da düşünülemez. Planlar arasında yeniden kullanılamayan bir davranış ise geliştiricilerin kafasını karıştıracaktır. Bu yanlış toplama aynı zamanda görevlerin daha düzgün bir şekilde bir davranış altında toplanması fırsatlarının

da kaçmasına neden olabilir. Bu yüzden *Anlamsız Davranış* kötü kokusu fark edildiğinde görevler *Davranıştan Kurtar* yeniden yapılandırılması ile davranıştan çıkarılır.



Şekil C.6. *Davranıştan Kurtar* yeniden yapılandırması

Mekanikler

1. Plan yapısı içerisinde anlamsız görevin sahip olduğu her ihtiyaç için bu ihtiyacı sağlayan bağları bul. Bu bağların hedef ihtiyacını alt-görevlerin sahip olduğu ve davranışın ihtiyacına kalıtım bağı ile bağlı olan ihtiyaç olarak değiştir.

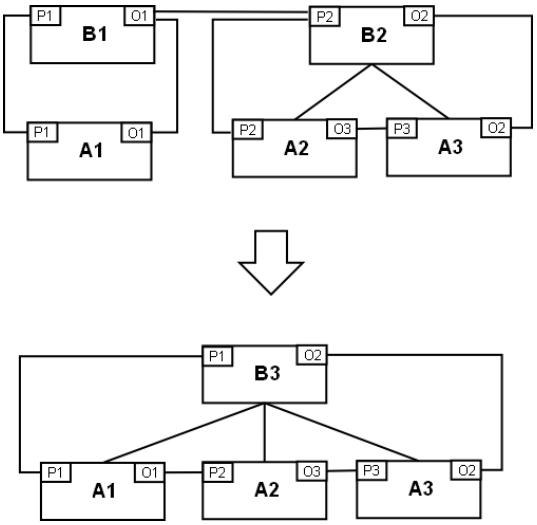
2. Plan yapısı içerisinde anlamsız görevin sahip olduğu her sonuç durumu için kaynak sonuç durumu bu sonuç durumu olan bağları bul. Bu bağların kaynak sonuç durumunu alt-görevlerin sahip olduğu ve davranışın sonuç durumuna ters kalıtım bağı ile bağlı olan sonuç durumu olarak değiştir.

3. Kaynak ve hedef parametreleri davranışın alt-görevleri olan bağları davranışın bulunduğu plan yapısına taşı

4. Plan yapısından davranışı çıkart yerine davranıştan kurtarılan görevleri ekle.

C.7 Davranışları Birleştir

Ortak bir hedef için çalışan bazı anlamsız davranışlar olduğu durumlarda kullanılır. Ortak hedefi göz önüne alarak isimlendirilen yeni bir davranış oluştur ve orijinal davranışlardaki görevleri yeni davranış yapısına taşı.



Şekil C.7. *Davranışları Birleştir* yeniden yapılandırması

Motivasyon

Bu yeniden yapılandırmanın ana amacı davranış yapılarının okunabilirliğinin arttırılmasıdır. SGA planlama paradigmasında aynı işlemin parçaları olan bazı eylemler farklı birden fazla plan yapısı içerisine dağılmış olabilirler. Veya yeniden yapılandırma operasyonları sonucunda bazı küçük ve anlamsız davranışlar oluşabilir.

Anlamsız Davranış kötü kokusu fark edildiğinde uygulanan *Davranışları Birleştir* yeniden yapılandırması daha anlamlı ve tutarlı bir davranış oluşturarak orijinal davranışları bu yeni oluşturulan davranış yapısı içerisinde toplar.

Mekanikler

1. Birleştirilecek davranışları *Davranış Çıkarımı* yeniden yapılandırmasını uygulayarak ortak bir davranış içerisinde topla.

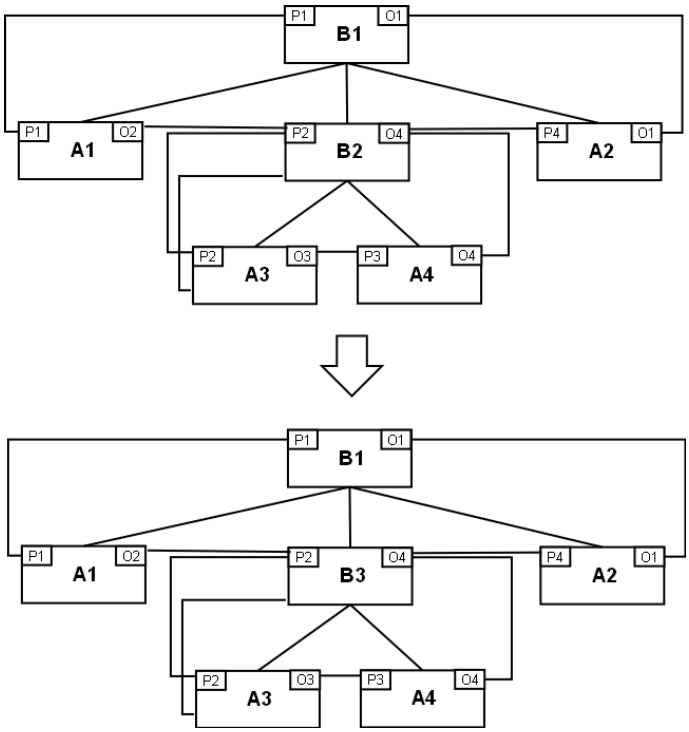
2. Yeni davranışın her alt-görevi için *Davranıştan Kurtar* yeniden yapılandırmasını uygulayarak anlamsız davranışları aradan çıkart.

C.8 Davranış İsmini Değiştir

Bir davranışın ismi davranışın genel amacını tanımlamadığı durumlarda kullanılır. Böyle bir durumda davranışa genel amacına uygun bir isim ata, davranışın tuttuğu yapıdaki ve davranışın bulunduğu daha üst seviye davranış yapılarındaki ilişkili bağları yeniden düzenle.

Motivasyon

Davranışlar belirli bir ortak amaca ulaşabilmek için davranış yapısının belirlediği sırada çalışan görevleri içerirler. Davranış isminin bu ortak amacı tanımlaması gerekir. Eğer bir davranışın isminin alt-görevlerinin ortak amacını göstermediğini düşünüyorsanız *Davranış İsmini Değiştir* yeniden yapılandırmasını kullanarak davranışın ismini ortak amacı tanımlayan yeni bir isim ile değiştirebilirsiniz. Bir davranışın isminin değiştirilmesi bu davranışa bağlı olan bağlardaki referansların da değiştirilmesini gerektirir.



Şekil C.8. *Davranış İsmini Değiştir* yeniden yapılandırması

Mekanikler

1. Davranışın genel amacını göz önüne alarak davranışa yeni bir isim ata

2. Davranışın tuttuğu yapı içerisindeki davranışa referans içeren bağların referanslarını yeni isme göre değiştir

3. Davranışın alt-görev olarak bulunduğu daha üst seviye plan yapıları varsa bu yapılar içerisinde davranışa referans içeren bağları yeni

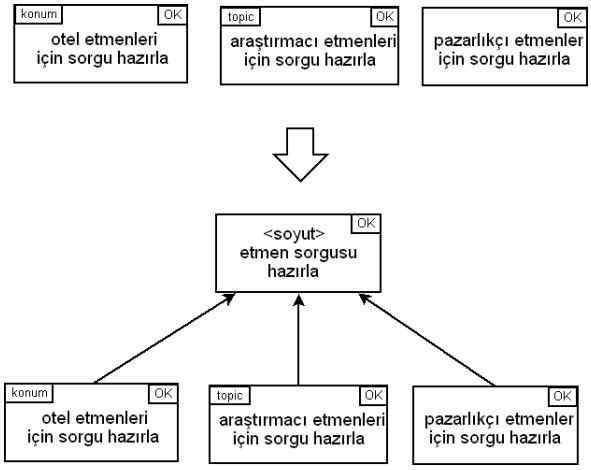
isme göre yeniden düzenle

4. Davranış bir rol hedefi ile eşleştirilmiş bir plan ise sistemin rol-hedef modelinde ilgili planın ismini değiştir.

EkD EYLEM SEVİYESİNDEKİ YENİDEN YAPILANDIRMA DESENLERİ

D.1 Süper-Eylem Çıkarımı

Aynı çalıştırılabilir kod parçasının ve\veya parametrelerin bir çok eylem içerisinde tekrarlandığı durumlarda kullanılır. Böyle bir durumda tekrarlanan kodu ve\veya parametreleri tutan soyut bir eylem oluştur ve orijinal eylemlerden bu soyut eylemi genişlet.



Şekil D.1. *Süper-Eylem Çıkarımı* yeniden yapılandırması

Motivasyon

Planlar çok sayıda eylem içerebilirler ve bir eylem bir çok plan yapısı içerisinde kullanılabilir. Bu yüzden ÇES geliştirim sırasında çok sayıda

eylem elde edilir. Bu eylemlerin bazıları aynı veya benzer girdi-çıktı parametrelerine ve\veya çalıştırılabilir kod parçasına sahip olabilirler. Bu yüzden eylemler benzer eylemler arasında yeniden kullanılabilirliğe izin verecek bir yol içerisinde geliştirilmelidirler.

Eylemler arasında *Eylem Kodu ve Parametresi Tekrarı* kötü kokusu fark edildiğinde bu eylemler üzerinde *Süper-Eylem Çıkarımı* yeniden yapılandırması uygulanarak benzer eylemlerde tekrar eden kod parçası ve parametreler soyut bir eylem içerisinde toplanır. Orijinal eylemler bu soyut görevi genişletir ve sadece kendileri için özelleşmiş kodu ve parametreleri tutarlar. Bu sayede benzer eylemler yeni oluşturulan soyut eylemi genişleterek kolayca gerçekleştirilebilir.

Mekanikler

1. Soyut bir eylem oluştur.

2. Orijinal eylemlerde yeni oluşturulan soyut eylemi genişlet

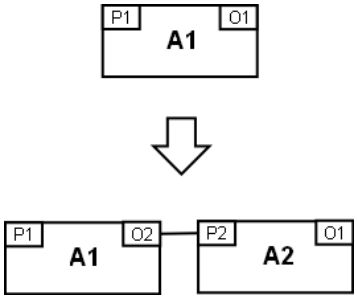
3. Eylem grubu içerisinde tekrar eden ihtiyaç ve sonuç durumlarını yeni soyut eyleme taşı

4. Orijinal eylemlerde tekrar eden kod parçası varsa bu kod parçasını soyut eyleme taşı

5. Her somut eylemi ayrı ayrı test et.

D.2 Eylem Böl

Bir eylemin birden fazla eylem tarafından gerçekleştirilmesi gereken işlemler içerdiği durumlarda kullanılır. Böyle bir durumda yeni bir eylem oluştur, farklı eylemde olması gereken kodu yeni eyleme taşı ve eylemler arası bağlantıları oluştur.



Şekil D.2. *Eylem Böl* yeniden yapılandırması

Motivasyon

Etmen planları içerisindeki görevlerin çalışma sırasına plan yapısındaki bağlara göre karar verilir. Bu şekilde plan içerisindeki işlemlerin akış kararı çalışma zamanında dinamik olarak karar verilir. Bazen içerdiği çalışabilir kod ile gerçekleştirdiği işlemler arasında dinamik olarak karar verilmesi gereken bir akış kararı bulunan eylemler oluşabilir. Çoklu etmen sistemindeki bir eylem üzerinde *Eylem İçerisinde Akış Kararı* kötü kokusu sezildiğinde *Eylem Böl* yeniden yapılandırması kullanılarak yeni bir eylem oluşturulur ve akış kararı gerektiren kod parçası oluşturulan bu yeni eyleme

taşınır. Eylemler arasında oluşturulan ihtiyaç-sonuç durumu bağları ile akış kararının dışarı alınması sağlanmış olur.

Eylem Böl yeniden yapılandırmasının uygulanabileceği diğer bir durum ise *Büyük Eylem* kötü kokusunda tanımlanmaktadır. Bir eylem gerçekleştirdiği işlevsellik için çok sayıda plan yapısı içerisinde yeniden kullanılabilir. Büyük eylemler yeniden kullanılabilirliği azaltır. Eylemleri mümkün olduğunca parçalayarak daha küçük işlemler için yeniden kullanılabilecek eylemler elde edebilirsiniz. Eylemleri daha küçük eylemlere bölmek aynı zamanda eylemlerin okunabilirliğini de arttırır.

Eylem Böl yeniden yapılandırması sonucunda birbirlerine ihtiyaç-sonuç durumu bağları ile bağlanmış orijinal eylemin işlevselliğini paylaşan iki eylem elde edilir.

Mekanikler

1. Yeni bir eylem oluştur ve diğer eylemden çekilecek kodun işlevselliğini göz önünde bulundurarak adlandır
2. Orijinal eylemin sonuç durumlarını yeni eyleme taşı
3. Geleneksel *Metot Çıkarımı* yeniden yapılandırmasını uygulayarak eylem içerisinden çekilmek istenen kodu yeni bir metotun içine taşı
4. Oluşan metotun her bir parametresi için
   (a) Orijinal eyleme bu parametreleri gönderecek yeni bir sonuç durumu ekle
   (b) Yeni eyleme bu parametreleri alacak yeni bir ihtiyaç ekle

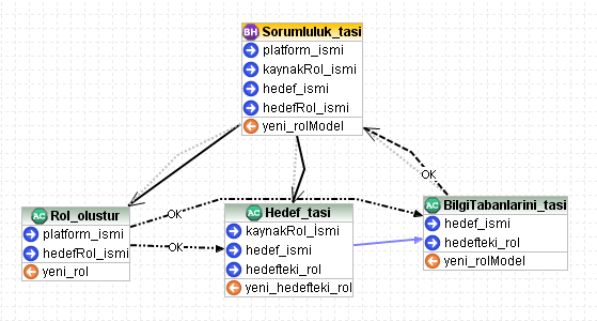
(c) Oluşturulan ihtiyaç ve sonuç durumunu birbirine bağla

5. Metotun içindeki kod parçasını yeni eyleme taşı ve metot tanımlamasını sil.

EkE RESEAGENT ARACI İÇERİSİNDEKİ YENİDEN YAPILANDIRMA PLANLARI

E.1 Sorumluluk Taşı Yeniden Yapılandırma Planı

Sorumluluk Taşı yeniden yapılandırma planı ek-B.1'de tanıtılan *Sorumluluk Taşı* yeniden yapılandırmasının amacı olan "belirli bir roldeki bir hedefin başka bir role aktarılması" hedefini başarmak üzere geliştirilmiştir. *Sorumluluk Taşı* yeniden yapıolandırmasının mekaniklerini gerçekleştiren görevleri içeren planın SGA yapısı şekil E.1'de gösterilmektedir.



Şekil E.1. ReSeagent *Sorumluluk Taşı* yeniden yapılandırma planı

Plan ihtiyaç olarak *Sorumluluk Taşı* yeniden yapılandırmasının uygulanacağı etmen platformunun ismini, kaynak rolün ismini, kaynak rolden taşınacak hedefin ismini ve bu hedefin taşınacağı rolün ismini alır. Aşağıda listelenen üç eylem ile etmen platformundaki rol hedefini kaynak

rolden hedef role taşır. Çalışması sonucunda elde edilen güncellenmiş rol-hedef modelini bir sonuç durumu aracılığıyla dışarı gönderir.

Alt-rolleri oluştur: Bu eylem ihtiyaç olarak ismini aldığı etmen platformunun rol-hedef modeli içerisinde yine ihtiyaç olarak aldığı rol isminde bir rol olup olmadığını araştırır. Eğer bu isimde bir rol yok ise yeni bir rol yaratır. AVKD yeni rol örneklerinin oluşturulmasını desteklemediğinden dolayı yeni rollerin oluşturulması işi doğrudan ontolojiye müdahale edilerek başarılır.

Hedefleri yerleştir: Bölünmek istenen roldeki hedeflerin, ihtiyaç olarak alınan rol bölme stratejisine göre yine ihtiyaç olarak alınan yeni rollere kopyalanmasından sorumludur. Bu işlem için aşağıda verilen AVKD kuralını oluşturur;

Rule(antecedent(Role(?x) hasGoal(?x ?y) MASElement-Name(?x rolName) MASElementName(?y goalName)) consequent(hasGoal(altRole ?y)))

Bu kural rol bölme stratejisi içerisindeki alt-rollere karşılık tutulan her hedef ismi için bu veriler parametre olarak geçirilerek oluşturulur ve ontolojiye eklenir. Daha sonra kurallar ile genişletilen rol-hedef ontolojisi Pellet[1] çıkarsama motoruna gönderilir. Son olarak çıkarsama sonucu oluşan ve uygun hedeflerin alt-rollere dağıtılmış olduğu yeni genişletilmiş ontoloji kaydedilir.

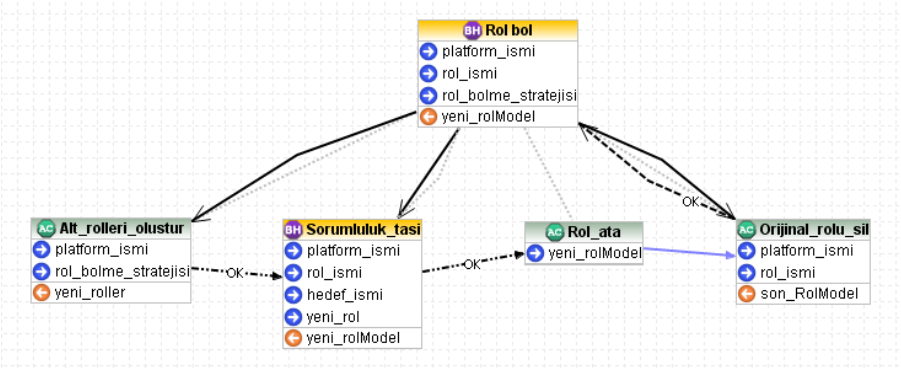
Bilgi tabanlarını taşı: Güncel SEAGENT rol-hedef üst modelinde bilgi tabanı kavramının yer almaması ve hedef-bilgi tabanı ilişkisinin

[1] http://pellet.owldl.com/ , son erişim: 12-Ağustos-2008

gösterilmemesi nedeniyle yeniden yapılandırmanın bu mekaniği henüz gerçekleştirilmemiştir.

E.2 Rol Böl Yeniden Yapılandırma Planı

Rol Böl planı rol-hedef modeldeki bir rolün kullanıcı tarafından tercih edilen bir rol bölme stratejisi kullanılarak alt-rollere bölünmesi hedefini başarmak için geliştirilmiştir. Plan, *Rol Böl* yeniden yapılandırmasının içerdiği mekanikleri gerçekleştiren alt-görevlerden oluşmaktadır. Bu planda tutulan SGA yapısı Şekil E.2'de görülmektedir.



Şekil E.2. Reseagent *Rol Böl* yeniden yapılandırma planı

Plan ihtiyaç olarak rol modelinde değişiklik yapılacak etmen platformunun ismini, bölünecek rolün ismini ve bir rol bölme stratejisini almaktadır. Rol bölme stratejisi etmen sistem geliştiricisi tarafından rolü bölmeye karar verildiği zaman oluşturulur. Strateji yeniden yapılandırma

tetikleyicisi tarafından rol-hedef editörden alınarak rolün bölüneceği alt-rollerin isimlerini ve bu alt-rollerin her birine eklenecek hedeflerin setini tutan bir *"Java Map"* nesnesi olarak plana gönderilmektedir.

Rol bölme planı her biri *Rol Böl* yeniden yapılandırmasının içerdiği mekaniklerden birini gerçekleştiren dört adet görevden oluşmaktadır. Bu görevlerin gerçekleştirim detayları aşağıda kısaca verilmiştir.

Alt-rolleri oluştur: Bu eylem ihtiyaç olarak aldığı platform isminin belirttiği etmen platformunun rol-hedef modeli içerisinde yine ihtiyaç olarak aldığı rol bölme stratejisi içerisinde bulunan alt-rol isimlerini kullanarak yeni rollerin oluşturulmasından sorumludur. AVKD yeni rol örneklerinin oluşturulmasını desteklemediğinden dolayı yeni rollerin oluşturulması işi doğrudan ontolojiye müdahale ederek başarılır.

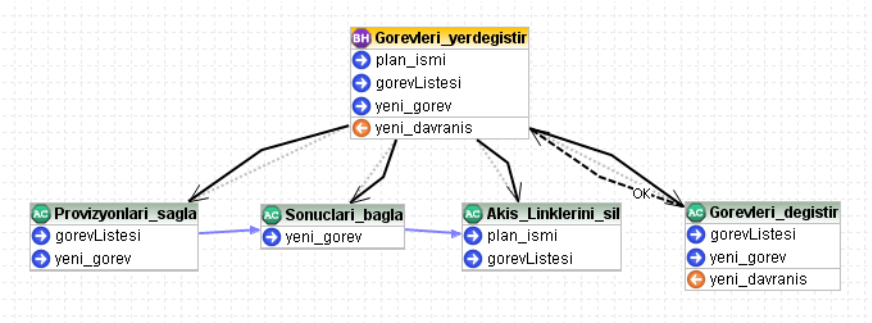
Hedefleri yerleştir: Bu görev *Sorumluluk Taşı* yeniden yapılandırma planının kök davranışı yeniden kullanılarak elde edilmiştir. İhtiyaç olarak ismini aldığı etmen platformunun rol-hedef modelinde yine ihtiyaç olarak aldığı rol hedeflerinin uygun rollere taşınmasından sorumludur.

Rolleri ata: Bu eylem çalışma zamanında gerçekleştirilen rol bölme operasyonu sırasında sistemde çalışmakta olan etmenlere yeni oluşturulan rollerin atanması işlemini yürütür. Bunula birlite, ReSeagent sadece tasarım zamanı yeniden yapılandırmayı desteklemektedir. Ayrıca, bu işlemin gerçekleştirilebilmesi için kullanılan etmen platformunun çalışma zamanında rol yükleme desteğini sunması gereklidir. SEAGENT platformu henüz böyle bir destek sunmamaktadır. Bu nedenlerden dolayı, ReSegent güncel versiyonunuda bu eylem gerçekleştirilmemiştir.

Orijinal rolü sil: Bu eylem ise yeni roller hedefleriyle birlikte kullanıma hazır olduktan sonra orijinal rolün modelden çıkarılması işini yürütmektedir. İhtiyaç olarak aldığı rol isminin gösterdiği rolü modelden siler ve oluşan yeni modeli sonuç durumu olarak dışarı gönderir.

E.3 Görevleri Bir Görev ile Yer Değiştir Yeniden Yapılandırma Planı

Görevleri Bir Görev İle Yer Değiştir yeniden yapılandırma planı ek-C.3'de tanıtılan *Görevleri Bir Görev İle Yer Değiştir* yeniden yapılandırmasının amacı olan "bir görev sıralmasının aynı işlevselliği yerine getiren tek bir görev ile değiştirilmesi" hedefini başarmak üzere geliştirilmiştir. *Görevleri Bir Görev İle Yer Değiştir* yeniden yapıolandırmasının mekaniklerini gerçekleştiren görevleri içeren planın SGA yapısı şekil E.3'de gösterilmektedir.



Şekil E.3. ReSeagent *Görevleri Bir Görev İle Yer Değiştir* yeniden yapılandırma planı

Görevleri Bir Görev İle Yer Değiştir yeniden yapılandırmasının uygulanabilmesi için, plana eklenecek yeni görevin plandan çıkarılacak görev grubundaki görevlerin grup dışındaki görevlerden beklediği ihtiyaçların ve grup dışına gönderdiği sonuç durumlarının tümüne sahip olması gerekmektedir.

Plan ihtiyaç olarak içerisinde *Görevleri Bir Görev İle Yer Değiştir* yeniden yapılandırmasının uygulanacağı planın ismini, plan yapısından çıkarılacak görevlerin listesini ve çıkarılan görevlerin yerine plana eklenecek görevi alır ve bu ihtiyaçları kullanılmak üzere alt görevlerine iletir. Planın çalışması sonucunda elde edilen güncellenmiş davranış yapısı bir sonuç durumu aracılığıyla dışarıya gönderilir. Plan aşağıdaki eylemleri içermektedir;

İhtiyaçları sağla: Bu eylem plan yapısından çıkarılacak görevlerin sahip olduğu ihtiyaçlara bağlı olan bağların silinip bu bağlar tarafından taşınan bilgiyi plana yeni eklenecek görevin ihtiyaçlarına aktaracak yeni bağların oluşturulmasından sorumludur. Bu amaçlar doğrultusunda ilk olarak, yeni bağların oluşturulması için aşağıdaki AVKD kuralını işletir:

```
Rule( antecedent(Task(?b) Task(?x) ProvisionLink(?z) name(?b
    ?a) isReplacingTask(?b "false"^^boolean) name(?x ?y)
    isReplacingTask(?x "true"^^boolean) senderTaskClass-
    Name(?z ?a) targetTaskClassName(?z ?y)) conse-
    quent(targetTaskClassName(?z FindHotels)) )
```

Daha sonra, plan yapısından çıkarılacak bağlar doğrudan müdahale ile plan modelinden silinir.

Sonuçları bağla: Bu eylem plan yapısından çıkarılacak görevlerin sahip olduğu sonuç durumlarına bağlı olan bağların silinip bu bağlar tarafından gönderilen bilgiyi plana yeni eklenecek görevin sonuç durumlarından alacak yeni bağların oluşturulmasından sorumludur. Bu amaçlar doğrultusunda ilk olarak, yeni bağların oluşturulması için aşağıdaki AVKD kuralını işletir:

Rule(antecedent(Task(?b) Task(?x) OutcomeLink(?z) name(?b ?a) isReplacingTask(?b "false"^^boolean) name(?x ?y) isReplacingTask(?x "true"^^boolean) targetTaskClassName(?z ?a) senderTaskClassName(?z ?y)) consequent(senderTaskClassName(?z FindHotels)))

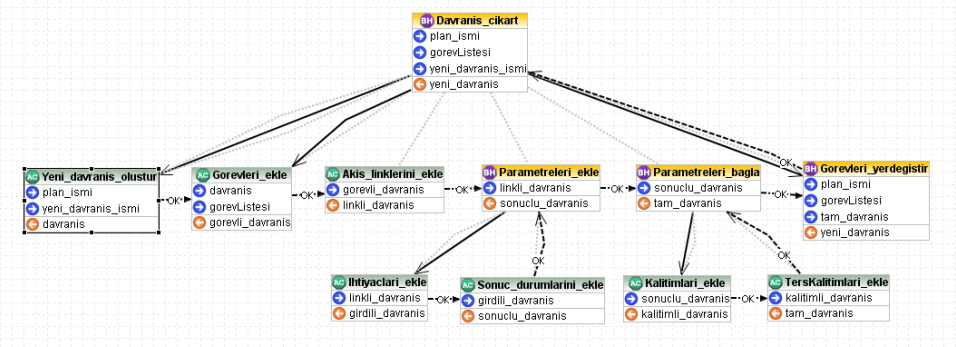
Daha sonra, plan yapısından çıkarılacak bağlar doğrudan müdahale ile plan modelinden silinir.

Akış linklerini sil: Bu eylem plan yapısından çıkarılacak görevlere bağlı akış bağlarının silinmesinden sorumludur. İhtiyaç olarak aldığı plan yapısı içerisinde yine ihtiyaç olarak aldığı görev listesindeki görevlere bağlı tüm akış linklerimi siler. Böylelikle, bu görevlerin plan akışından etkileri kalmaz.

Görevleri değiştir: İhtiyaç olarak aldığı görev listesindeki tüm görevlerin ilgili örneklerinin plan modelinden silinmesi ve yine ihtiyaç olarak aldığı yeni görevin plan yapısına alt-görev olarak eklenmesi işlerini yürütür. Çalışmasının sonunda, planın güncellenmiş halini döndürür.

E.4 Davranış Çıkarımı Yeniden Yapılandırma Planı

Davranış Çıkarımı yeniden yapılandırma planı Ek-C.1'de tanıtılan *Davranış Çıkarımı* yeniden yapılandırmanın amacı olan "bir davranış yapısındaki görev gurubunun yeni bir davranış altında toplanması" hedefini başarmak için geliştirilmiştir. Dolayısıyla bu yeniden yapılandırmanın mekaniklerini gerçekleştiren görevlerden oluşan bir yapı tutmaktadır. Bu planını SGA yapısı Şekil E.4'de gösterilmektedir.



Şekil E.4. ReSeagent *Davranış Çıkarımı* yeniden yapılandırma planı

Davranış Çıkarımı mekanikleri içerisinde diğer yeniden yapılandırmalardan birini içeren üst seviye bir yeniden yapılandırmadır. Plan içerisindeki *görevleri yer değiştir* görevi *Görevleri Bir Görev İle Yer Değiştir* yeniden yapılandırması için daha önceden gerçekleştirilmiş bir planın yeniden kullanımı ile elde edilmiştir. Burada görüldüğü gibi yeniden yapılandırmaların SGA planları şeklinde tanımlanması, mekanikleri içerisinde daha küçük yeniden yapılandırmaların birini veya

daha fazlasını içeren büyük yeniden yapılandırmaların tanımlanmasını önemli ölçüde kolaylaştırmıştır.

Davranış Çıkarımı planı ihtiyaç olarak yeniden yapılandırmanın uygulanacağı planın ismini, yeni davranışa çekilecek görevlerin listesini ve oluşturulacak yeni davranışın ismini almaktadır. Plan *Davranış Çıkarımı* yeniden yapılandırmasının mekaniklerinden gelen yedi eyleme ve diğer bir yeniden yapılandırma planlarının yeniden kullanımıyla elde edilen bir davranışa sahiptir. Aşağıda bu görevlerin gerçekleştirim detayları kısaca verilmiştir.

Yeni davranış oluştur: Bu eylem ismini ihtiyaç olarak aldığı plan yapısı içerisinde, yine ismini ihtiyaç olarak aldığı yeni bir davranış oluşturur. SGA üst modelindeki davranış sınıfının bir örneğinin yaratılabilmesi için plan ontolojisine müdahale eden kod parçasını içermektedir. Çalışmasının sonucu olarak oluşturulan yeni davranış örneğini sonuç durumu olarak dışarı gönderir.

Görevleri ekle: İhtiyaç olarak aldığı davranış örneğine yine ihtiyaç olarak aldığı plandan çekilecek görevlerin alt-görev olarak eklenmesinden sorumludur. Bu işlem için yeni davranışın ismini parametre olarak geçirerek aşağıdaki AVKD kuralını oluşturur ve Pellet çıkarsama motoru aracılığıyla işletir.

```
Rule( antecedent(Task(?x) isExtractingTask(?x "true"^^boolean))
    consequent(subTask(yeniDavranis ?x) ) )
```

Çalışması sonucunda alt-görevleri eklenmiş yeni davranış örneğini dışarı gönderir.

Akış bağlarını ekle: SEAGENT plan üst modelinde içsel ihtiyaç bağı ve sıra bağı adında iki çeşit akış bağı bulunmaktadır. Bu eylem yeni davranışa çekilen görevler arasındaki bağların, yeni ihtiyaç olarak aldığı yeni davranış yapısına eklenmesinden sorumludur. Bu işlem için akış bağının tipi ve yeni davranışın ismi parametre olarak geçirilerek aşağıdaki AVKD kuralını çalıştırması yeterli olur.

Rule(antecedent(Task(?a) Task(?b) FlowLinkType(?x) name(?a ?y) isExtractingTask(?a "true"^^boolean) name(?b ?z) isExtractingTask(?b "true"^^boolean) senderTaskClassName(?x ?y) targetTaskClassName(?x ?z)) consequent(FlowLinkType(BHName ?x)))

Bu kural iki tip akış bağı için ayrı ayrı ontolojiye eklenerek çalıştırılır. Sonuç olarak elde edilen görevleri arasındaki akış bağları eklenmiş yeni davranış örneği dışarı gönderilir.

İhtiyaçları ekle: İhtiyaç olarak aldığı yeni davranışa ihtiyaç tanımlamalarının eklenmesinden sorumludur. Göreve eklenecek ihtiyaçları belirlemek için plandan çekilen görevlerin diğer görevlerle bağlantılı ihtiyaçlarını arar ve bulduklarını yeni davranışa ekler. Bu işlem için aşağıdaki AVKD kuralını yeni davranışın ismini *newBHName* parametresi olarak geçirerek işletmektedir.

Rule(antecedent(ProvisionLink(?b) Task(?d) Task(?x) hasProvision(?x ?y) targetTaskClassName(?b ?a) senderTaskClassName(?b ?c) targetProvisionName(?b ?z) name(?d ?c) isExtractingTask(?d "false"^^boolean) name(?x ?a) isEx-

tractingTask(?x "true"^^boolean) name(?y ?z)) consequent(hasProvision(newBHName ?y)))

Kuralın işletimi sonucunda elde edilen modeldeki yeni davranış örneği bir sonuç durumu aracılığı ile dışarı gönderilir.

Sonuç durumlarını ekle: Bir önceki eyleme benzer sorumluluğa ve kod yapısına sahiptir. Bu eylemin sorumluluğu, ihtiyaç olarak aldığı davranış örneğine sonuç durumlarının eklenmesidir. Göreve eklenecek sonuç durumlarını belirlemek için plandan çekilen görevlerin diğer görevlerle bağlantılı sonuç durumlarını arar ve bulduklarını yeni davranışa ekler. Bu işlem için aşağıdaki kuralı yeni davranışın ismini *newBHName* parametresi olarak geçirerek işletmektedir.

Rule(antecedent(OutcomeLink(?b) Task(?d) Task(?x) hasOutcome(?x ?y) senderTaskClassName(?b ?a) targetTaskClassName(?b ?c) senderOutcomeName(?b ?z) name(?d ?c) isExtractingTask(?d "false"^^boolean) name(?x ?a) isExtractingTask(?x "true"^^boolean) name(?y ?z)) consequent(hasOutcome(newBHName ?y)))

Çalışmasının sonucunda elde ettiği yeni davranış örneğini bir sonuç durumu yardımıyla dışarı gönderir.

Kalıtımları ekle: Bu eylem, ihtiyaç olarak aldığı davranış örneğinin ihtiyaçlarının alt-görevlerinin uygun ihtiyaçları ile bağlayacak miras bağlarının, bu davranışın yapısına eklenmesinden sorumludur. Bu işlem için plan üst modelindeki miras sınıfının bir örneği oluşturulmalıdır. AVKD ile yeni bir örnek oluşturulamadığından bu eylem herhangi bir ku-

ral işletmez. Uygun özelliklerde yeni bir miras bağı oluşturan ve bu bağı plan ontolojisine ekleyen kodu içermektedir.

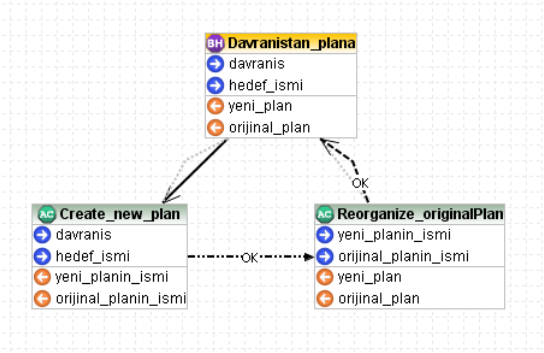
Ters kalıtımları ekle: Bir önceki eyleme benzer şekilde bu sefer ters kalıtım bağları ile yeni davranışın sonuç durumlarının, altgörevlerdeki sonuç durumlarına bağlanmasını amaçlamaktadır. Bu doğrultuda bir önceki eylem ile benzer kod parçası, bu sefer ters kalıtım bağları için gerçekleştirilmiştir.

Görevleri yer değiştir: Mekanikleri ayrıca tanımlanmış farklı bir yeniden yapılandırma planının kök davranışıdır. Bu yeniden yapılandırma parametre olarak aldığı görev listesindeki görevler tümünün tek bir görev ile yer değiştirmesini amaçlamaktadır.

E.5 Davranıştan Plana Yeniden Yapılandırma Planı

Davranıştan Plana yeniden yapılandırma planı Ek-C.4'de tanıtılan *Davranıştan Plana* yeniden yapılandırmanın amacı olan "bir davranışın kendi başına çalışabilen bir plan haline getirilmesi" hedefini başarmak için geliştirilmiştir. Dolayısıyla bu yeniden yapılandırmanın mekaniklerini gerçekleştiren görevlerden oluşan bir yapı tutmaktadır. Bu planını SGA yapısı Şekil E.5'de gösterilmektedir.

Davranıştan Plana yeniden yapılandırma planı ihtiyaç olarak plana dönüştürülecek davranışın ve bu davranışın bulunduğu planın isimlerini ve yeni plan ile eşleştirilecek hedef ismini alır. Çalışması sonunda ise, yeniden yapılandırma operasyonundan önce davranışı içeren planın son halini ve oluşan yeni plan yapısını dışarı gönderir. Plan *Davranıştan Plana* yeniden yapılandırmasının mekaniklerini gerçekleştiren üç eyleme sahiptir:



Şekil E.5. ReSeagent *Davranıştan Plana* yeniden yapılandırma planı

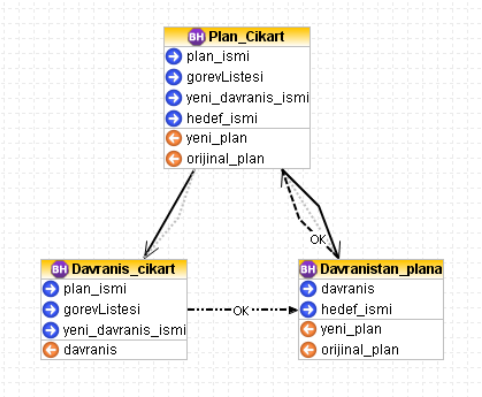
Yeni planı oluştur: Bu eylem başka bir plan modeli içerisinde tutulan davranış yapısının diğer planlardan bağımsız şekilde çalıştırılabilecek ayrı bir model olarak tutulmasını sağlamaktan sorumludur. Seagent içerisinde her planın kendine ait bir ontoloji modeli bulunmaktadır. Bu planların diğer planlardan bağımsız çalıştırılabilmesini sağlar. Davranışlar ise plan modelleri içerisinde tanımlanabildiğinden planlardan bağımsız çalıştırılamazlar. Bu eylem davranışı plan modeli dışına çekip sadece ona özgü bir ontoloji model oluşturur.

Hedef Eşle: Planların etmenler tarafından özerk olarak çalıştırılabilmesi için sistemdeki bir hedef ile eşleştirilmeleri gerekir. Bu eylem dışarı çekilen davranış yapısının bir hedef ile eşleştirilerek özerk çalıştırılabilecek bir plan haline getirilmesinden sorumludur.

Orijinal Planı düzenle: Bu eylem, orijinal plan yapısını, yeni plan ile etkileşimli olarak çalışacak şekilde yeniden düzenler.

E.6 Plan Çıkarımı Yeniden Yapılandırma Planı

Plan Çıkarımı yeniden yapılandırma planı Ek-C.2'de tanıtılan *Plan Çıkarımı* yeniden yapılandırmanın amacı olan "plan içerisindeki bir görev gurubunun yeni bir plan olrak dışarı çekilmesi" hedefini başarmak için geliştirilmiştir. Dolayısıyla bu yeniden yapılandırmanın mekaniklerini gerçekleştiren görevlerden oluşan bir yapı tutmaktadır. Bu planını SGA yapısı Şekil E.6'de gösterilmektedir.



Şekil E.6. ReSeagent *Plan Çıkarımı* yeniden yapılandırma planı

Plan Çıkarımı mekanikleri içerisinde diğer yeniden yapılandırmalardan birini içeren üst seviye bir yeniden yapılandırmadır. Plan içerisindeki *Davranış Çıkarımı* görevi *Davranıştan Plana* yeniden yapılandırması için daha önceden gerçekleştirilmiş bir planın yeniden kullanımı ile elde edilmiştir. Plan bu iki yeniden yapılandırma planının yeniden kullanımı işle elde edilen iki davranışa sahiptir:

Davranış çıkart: Davranış çıkarımı yeniden yapılandırma planının kök davranışıdır. Bu yeniden yapılandırma ihtiyaç olarak aldığı görev listesindeki görevler tümünün tek bir görev ile yer değiştirmesini amaçlamaktadır. Sonuç olarak elde edilen davranış yapısı bir sonuç durumu ile dışarı gönderilir.

Davranıştan plana: Davranıştan Plana yeniden yapılandırma planının kök davranışıdır. Bu yeniden yapılandırma ihtiyaç olarak aldığı davranışı bağımsız çalışabilen bir plana dönüştürür. Çalışması sonucunda, yeni plan yapısı bir sonuç durumu aracılığıyla dışarı gönderilir.

EkF TÜRKÇE-İNGİLİZCE TERİMLER SÖZLÜĞÜ

aksiyom *i.*: axiom
aktarım *i.*: transfer
aktarmak *e.:* to transfer
alan *i.*: domain
anlamsal *i.*: semantic
arayüz *i.*: interface
arıtma *i.*: refining
artışlı *s.:* incremental
bağ *i.*: link
bağlam *i.*: context
biçimsel *s.:* formal
bildirim *i.*: assertion
bileşen *i.*: component
bütünleştirme *i.*: integration
çerçeve *i.*: framework
çıktı *i.*: output
dağıtık *s.:* distributed
davranış *i.*: behaviour
dekleratif *s.:* declarative
depo *i.*: repository
desen *i.*: pattern
dışsal *s.:* external
eklenti *i.*: plug-in
eş zamanlı olmayan *s.:* asynchronous
eşgüdüm *i.*: coordination
eşleme *i.*: matching
eşlemek *e.:* to map
eşzamanlı *i.*: synchronous
etmen *i.*: agent
eylem *i.*: action
girdi *i.*: input
görev *i.*: task
görüşme *i.*: conversation
güdümlü *s.:* driven
güvenilirlik *i.*: reliability
hafif *s.:* light-weight
hazır bilgi *i.*: literal
herşeyi bilen *s.:* omniscient
içsel *s.:* internal
ihtiyaç *i.*: provision
ilgi *i.*: aspect
ilkel *s.:* primitive

imperatif *s.:* imperative
indirgeme *i.*: reduction
işlevsel *s.:* fonctional
istek *i.*: desire
iteratif *s.:* iterative
kabul *i.*: acceptance
kalıtım *i.*: inheritance
kanı *i.*: belief
kötü koku *i.*: bad smell
metafor *i.*: metaphor
niyet *i.*: intention
öğrenme *i.*: learning
öncül *i.*: antecedent
örnek *i.*: individual
özerk *s.:* autonomous
program hatası *i.*: bug
prototipleme *e.:* prototyping
şelale *i.*: waterfall
sıradüzensel *s.:* hierarchical
sonuç *i.*: consequent
sözdizim *i.*: syntax
süreç *i.*: process
tabanlı *s.:* based
taksonomi *i.*: taxonomy
tekbiçimli *s.:* uniform
ters kalıtım *i.*: disinheritance
tetiklemek *e.:* triggering
tetikleyici *i.*: trigger
topluluk *i.*: communitee
uç *s.:* extreme
veb *i.*: web
yeniden yapılandırma *i.*: refactoring
yeniden yapılandırmak *e.:* to refactor
yineleme *i.*: iteration
yinelemeli *s.:* iterative
yönelimli *s.:* oriented
yöntem *i.*: methodology
yordam *i.*: procedure
zemin *i.*: grounding

ÖZGEÇMİŞ

Ali Murat Tiryaki 18 Mart 1978 tarihinde İzmir'de doğdu. Lisans derecesini Çanakkale Onsekiz Mart Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 1999 yılında, yüksek lisans derecesini yine aynı üniversitenin Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği ABD'dan 2001 yılında almıştır. 2002 yılından beri Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'nde "Etmen Sistem Geliştiriminde Yeniden Yapılandırma" konusu üzerine doktora eğitimini sürdürmektedir 1999 ve 2002 yılları arasında Çanakkale Onsekiz Mart Üniversitesi Bilgisayar Mühendisliği Bölümü'nde araştırma görevlisi olarak çalışmıştır. 2002 yılından beri Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'de araştırma görevlisi olarak çalışmaya devam etmektedir. "Çoklu etmen sistem geliştirimi" konusu ile ilgili olarak, dergi ve konferans kitapçıklarında yayınlanan toplam on iki adet bilimsel yayını vardır Bir adet uluslararası çalıştayda organizasyon komitesinde yer almıştır (ESAW'05). Çoklu etmen sistemleri, yazılım geliştirmede çevik yaklaşımlar, etmen yönelimli yazılım geliştirme konularında araştırma çalışmaları yapmaktadır.