Kanı-İstek-Hedef Etmenlerinin Geliştirilmesi için bir Tersine Mühendislik Yöntemi

Hüseyin Küçük[1], Geylani Kardaş[2]

Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü, 35100 Bornova, İzmir, Türkiye
[1] huseyin.kcuk@gmail.com
[2] geylani.kardas@ege.edu.tr

Özet. Kanı-İstek-Hedef (BDI) etmenlerinin model-güdümlü mimari kullanılarak geliştirilmesinde, genellikle yukarıdan aşağıya bir yaklaşım izlenerek, etmenler önce tanımlanan bir üstmodele göre çeşitli bakışaçılarından modellenmektedir. Sonrasında, hazırlanan bu modeller bir dizi modelden modele ve modelden koda dönüşümlere girdi olarak verilmektedir. Bu dönüşümler işletilerek etmen yazılımlarının otomatik elde edilmesi sağlanmaktadır. Fakat bu süreçte kod üretiminin ardından geliştiricilerin yazılım kodlarında yapacakları değişiklikler etmen tasarım modellerine yansıtılamamaktadır. Mevcut çalışmalarda görülen bu eksiklik, özellikle hazırlanan etmen modellerinin ve yazılımlarının yeniden kullanılabilirliğini zorlaştırmaktadır. Bu bildiride, söz konusu eksikliği gidermek amacıyla etmen sistemlerinin yazılım kodlarından BDI modellerinin otomatik elde edilmesini sağlayacak bir tersine mühendislik yöntemi ve bunu destekleyen bir araç tanıtılmaktadır. Yeni yöntemin uygulanması ile yaygın kullanıma sahip JACK platformu üzerinde çalışan yazılımlardan etmenlerin BDI modellerinin geri elde edilmesi ve böylece kodlardaki değişikliklerin etmen yazılım modellerine yansıtılması mümkün hale getirilmiştir.

Anahtar Kelimeler: Model-güdümlü Mimari, Yazılım Etmeni, Kanı-İstek-Hedef Modeli, Tersine Mühendislik, Etmen-tabanlı Yazılım geliştirme.

A Reverse Engineering Methodology for the Development of Belief-Desire-Intention Agents

Abstract. Model-driven architectures, used for the development of Belief-Desire-Intention (BDI) agents, are mostly constructed by following a top-down approach in which software agents are modelled according to a metamodel and the prepared models are given into a series of model-to-model and model-to-text trasnformations to automatically generate the agent implementations. However, modifications made in these auto-generated artifacts can not be reflected back to the agent models. Hence the synchronization between the agent models and the corresponding software is ruined which also makes the

reusability of agent models and software difficult. In order to eliminate these deficiencies, we introduce a reverse engineering methodology and a supporting tool for the automatic generation of BDI models from existing agent software. With the use of the proposed tool, it is possible to both re-generate BDI agent models from the programs running on the well-known JACK platform and reflect the changes made in the programs to the corresponding models.

Keywords: Model-driven Architecture, Software Agent, Belief-Desire-Intention Model, Reverse Engineering, Agent-oriented Software Engineering.

1 Giriş

Yazılım etmenleri (ing. software agents) ve bunların oluşturduğu Çok-etmenli Sistemlerin (ing. Multi-agent Systems) (MAS) Kanı-İstek-Hedef (ing. Belief-Desire-Intention) (BDI) modeline [1] göre geliştirilmesi, bu modelin, özellikle etmenlerin iç yapısının gösterimini ve karşıt-eylemli ya da pro-aktif etmen davranışlarının kompozisyonunu kolaylaştırması nedeniyle Etmen-tabanlı Yazılım Mühendisliği (ing. Agent-oriented Software Engineering) (AOSE) [2] alanında yaygındır.

Bir BDI mimarisinde yazılım etmenleri sürekli ortamı izlemekte ve ortamdaki değişikliklere cevap vermektedir. Bu tepkileri etmenlerin akli tutumlarına dayanmaktadır. Bir etmenin bu mimaride *kanı*, *istek* ve *hedef* adı verilen üç tip akli tutumu vardır. *Kanılar*, etmenin çevre hakkındaki bilgi durumunu temsil eder. Çevreye kendisi ve diğer etmenler de dahildir. Bilgi yerine kanı kavramının kullanılmasının sebebi, etmenin kanılarının doğru olma zorunluluğu olmaması veya gelecekte değişebilecek olmasıdır. *İstekler*, etmenin motivasyon durumunu gösterir. Etmenin gerçekleştirmek istediği amaçları veya durumları temsil eder. Bu MAS'larda amaçlara karşılık gelir. Son olarak *Hedefler*, etmenin hangi işi gerçekleştireceğini seçme yönündeki bilincini temsil eder. Hedefler, etmenin kendini adadığı isteklerdir ve MAS'larda planlara denk gelir [3].

BDI4Jade [4], Jadex [5], JACK [6] gibi etmen programlama ortamları BDI tabanlı etmenlerin geliştirilmesine imkan vermektedir ancak yazılım geliştiriciler etmen kanılarının ve planlarının oluşturulması için gerekli mantık tabanlı kurguyu bu ortamlarda oluşturmakta zorlandığı için kodlama öncesinde BDI etmenlerinin daha soyut düzeyde modellenmesini ve model-güdümlü olarak geliştirilmesini sağlayan birçok yazılım mimarisi (örneğin [7, 8, 9]) ve alana-özgü modelleme dili (DSML) (örneğin [3, 10, 11]) AOSE araştırmacıları tarafından ortaya konulmuştur. Yazılım etmenlerinin ortaya konan bu yaklaşımlar ile model-güdümlü geliştirilmesinde [12] genellikle etmenlerin tanımlanan bir üstmodele göre çeşitli bakışaçılarından modellenmesi ve devamında hazırlanan bu modellerin bir dizi modelden modele ve modelden koda dönüşümlere girdi olarak verilmesinden sonra modellenen MAS'a ait yazılımların otomatik elde edilmesi söz konusudur. Fakat kod üretiminin ardından geliştiricilerin yazılım kodlarında yapacakları değişiklikler etmen tasarım modellerine yansıtılamamaktadır. Bu eksiklik, özellikle hazırlanan MAS modellerinin ve geliştirilen sistemlerin yeniden kullanılabilirliğini zorlaştırmaktadır.

BDI etmenlerinin model-güdümlü yazılım mimarilerine göre geliştirilmesinde yukarıda bahsedilen eksikliği gidermek amacıyla bu bildiride etmen sistemlerinin yazılım kodlarından BDI modellerinin otomatik elde edilmesini sağlayacak bir tersine mühendislik yöntemi tanıtılmaktadır. Yazılımların tersine mühendisliği yazılımların yönetilmesine ve zaman içerisindeki değişimlere göre evrimleşmesine yardımcı olmaktadır [13]. Kaynak kodlar incelenerek yazılım sistemlerinin ana elemanları belirlenebilir ve yazılımın önceden tanımlanmış bir düzeydeki soyutlaması elde edilebilir [14]. Bu çalışmada tersine mühendislik prensiplerinden yola çıkılarak yaygın kullanıma sahip JACK BDI etmen platformu [6] üzerinde çalışan MAS yazılımlarından etmenlerin BDI modellerinin geri elde edilmesi ve böylece kodlardaki değişikliklerin etmen yazılım modellerine yansıtılması mümkün hale getirilmiştir.

Bildirinin 2. bölümünde JACK etmen geliştirme çerçevesi hakkında bilgi verilmiştir. JACK BDI etmenlerin geliştirilmesi için önerilen tersine mühendislik yöntemi 3. bölümde anlatılmıştır. Tersine mühendislik yaklaşımının ve geliştirilen aracın kullanılmasını örnekleyen bir durum çalışması 4. bölümde yer almaktadır. Bölüm 5'te, ilgili literatürdeki önceki çalışmalar anlatılmış; önerilen yöntemin mevcut çalışmalara göre farkları belirtilmiştir. Son bölümde çalışmanın bir değerlendirmesi, elde edilen sonuçlar ve ileriye yönelik çalışma hedefleri yer almaktadır.

2 JACK Etmen Geliştirme Çerçevesi

JACK [6], Java üzerine kurulmuş ve BDI etmenlerinin Java sınıflarının birer uzantısı şeklinde oluşturulabildiği bir MAS çerçevesidir. JACK, etmen tanımlama süreçleri için bir grafiksel kullanıcı arayüzü içermektedir. Bu arayüz yardımıyla; etmenler, etmenlerin kabiliyetleri, inanç (kanı) kümeleri ve planlar BDI mimarisine göre tanımlanabilmektedir. JACK etmenleri, akıl yürütme yeteneğine sahip zeki etmenlerdir. Bu akıl yürütme yeteneği etmen tanımlama süreçlerini karmaşık hale getirmektedir. Bir JACK uygulaması çok sayıda etmen, olay, plan, inanç kümesi ve yetenek içerebilmektedir. JACK etmen geliştirme çerçevesinin programlama yapısı aşağıdaki bileşenleri içermektedir [15]:

Olaylar: Olaylar bir etmeni harekete geçmeye motive eden yapılardır. JACK'de, her biri farklı kullanımlara sahip bir dizi olay türü vardır. Olaylar, etmen odaklı bir sistemdeki tüm faaliyetlerin kaynağıdır. Olayların yokluğunda bir etmen boşta durur. Bir olay meydana geldiğinde, etmen bu olayı ele almak için bir görevi başlatmaktadır.

Plan: Plan yapısı, bir olayın meydana gelmesi veya dış ortamdan etmene bir etkide bulunulması durumunda bir etmenin alabileceği eylem dizisini açıklar. Bir etkinin ya da olayın oluşması ve bir yazılım etmeninin bu olayı göz önüne alan bir görevi kabul etmesi durumunda, bu görev dahilinde etmenin işleteceği eylemlerden oluşan bir planın etmen tarafından bulunması ve uygulanması gerçekleşir. Planlar ve bunların içerdiği alt görevler geleneksel programlama dillerindeki yöntemlere benzetilebilir.

Etmen: JACK etmenleri BDI mimarisine dayandığından inançlara, isteklere ve niyetlere sahiptirler. Bu bileşenler her etmenin kendi içsel durumuna aittir ve diğer etmenler tarafından erişilemezler.

İnanç Kümesi: JACK'de bir etmenin dünyayla ilgili inançlarını sürdürmesi için inanç kümeleri kullanılır. İnançlar etmenin yaşam süresi boyunca ortamdaki değişimlere göre ya da etmenin işlettiği planlar sonrasında değişim gösterebilirler.

Görüş: JACK'ın veri modelleme kabiliyetinin merkezinde yer almaktadır. JACK inanç kümeleri, Java veri yapıları ve mevcut sistemlerden gelen geniş bir veri kaynağı yelpazesini JACK çerçevesine entegre etme olanağı sağlamaktadır. Bir görüş, etmenlerin heterojen veri kaynaklarını kullanmasına olanak sağlayan bir veri soyutlama mekanizmasıdır.

Yetenek: Yetenek kavramı, seçilmiş akıl yürütme yeteneklerini uygulayan etmenlerin muhakeme unsurlarını yapılandırmaktır. Bu teknik, etmen sistem tasarımını basitleştirir. Ayrıca tanımlanmış mevcut planların ve etmen amaçlarının yer aldığı yeteneklerden yeni etmenlerin geliştirilmesi sırasında da yararlanılması kod yeniden kullanılabilirliğini sağlamakta ve etmen işlevselliğinin kapsüllenmesine izin vermektedir. Yetenekler, bir etmenin işlevsel yönlerini temsil etmektedir.

JACK görsel kullanıcı arayüzü (ing. graphical user interface) (GUI) aracılığı ile karmaşık akıl yürütme yeteneğine sahip etmen sistemleri modeller kullanılarak geliştirilebilmektedir. JACK ortamı söz konusu yazılım modelleri ile geliştirilmiş etmen sistemlerinin uygulanabilmesine / çalıştırılabilmesine olanak sağlamaktadır. Entegre derleme özelliği ile geliştirilmiş MAS'lar doğrudan JACK platformunda çalıştırılabilir Java uygulamalarına dönüştürülebilmektedir [6].

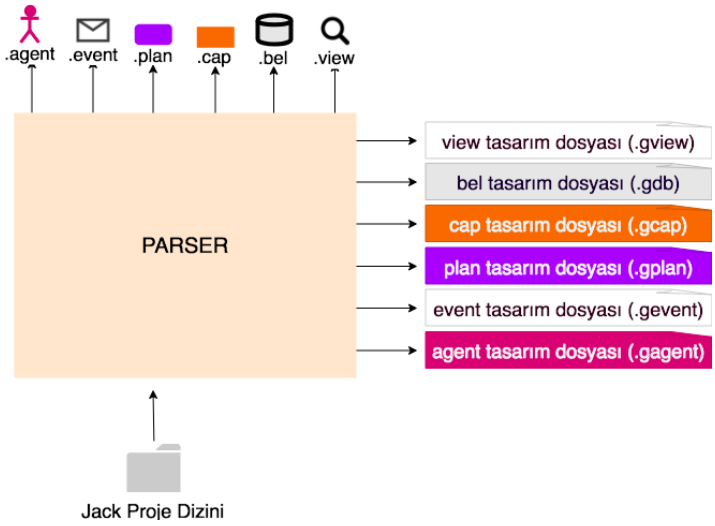
3 JACK Etmen Yazılımları için Tersine Mühendislik Mimarisi

JACK platformunda çalışan MAS'lara ait yazılımlardan MAS modellerinin otomatik oluşturulması için bir yöntem ve bu yöntemin uygulanmasına imkan veren bir yazılım mimarisi bu çalışmada geliştirilmiştir. Önerilen tersine mühendislik yaklaşımının uygulanması için geliştirilen yazılım mimarisi yine bu çalışma kapsamında geliştirilen bir "Parser" (ing. ayrıştırıcı) aracına dayanmaktadır (Şekil 1). Geliştirilen Parser aracı, JACK üzerinde tasarlanmış ve çalıştırılabilir durumda bir MAS yazılımının kaynak kodlarını analiz ederek, doğrudan yazılım kodlarında yapılmış olan değişikliklerin JACK tasarım modellerine yansıtılabilmesine imkan vermektedir.

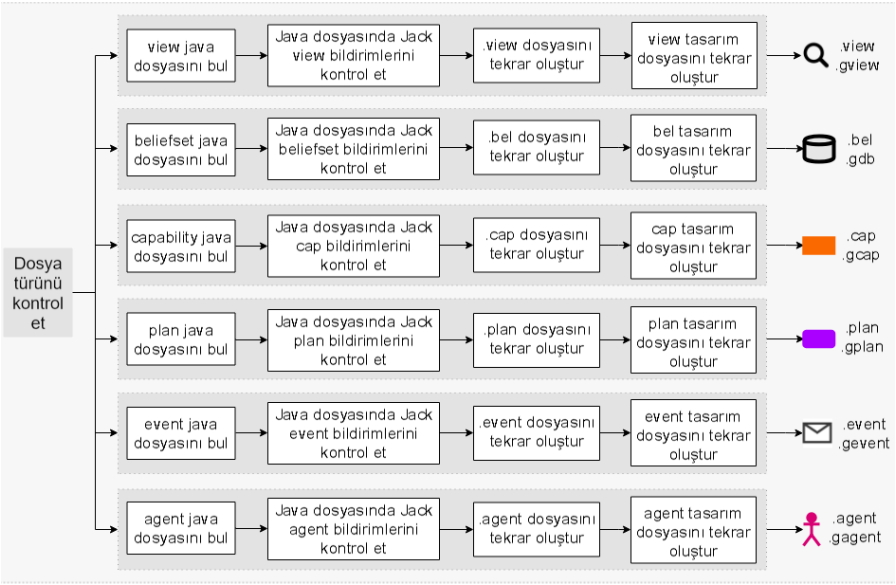
Parser aracı, JACK tarafından üretilmiş olan Java kodları üzerinde bir statik analiz uygulamaktadır. Statik analiz, yazılım etmenlerine ait kaynak kodların içeriğinin satır satır okunarak söz dizimsel analizlerin yapılmasına olanak tanımaktadır. JACK kaynak kodları üzerinde yapılan statik analizlerde desen eşleme tekniği kullanılarak, JACK tarafında tasarım modellerine dayanarak üretilmiş olan JACK dosyaları ve JACK tasarım dosyaları tekrar oluşturulabilmektedir.

JACK, bir MAS yazılımının bileşenlerini temsil eden altı adet dosya türü üretebilmektedir. Bu dosyalar *agent (.agent), event (.event), plan (.plan), capability (.cap), view (.view)* ve *beliefset (.bel)*'dir. Bu dosyaların yanında, tasarım modellerinin ayarlarının yer aldığı altı adet tasarım dosyası (*.gagent, .gevent, .gplan, .gcap, .gview, .gdb*) da JACK tarafından geliştirilmekte olan MAS proje klasörüne yerleştirilmektedir. Her bir dosya içerisinde farklı tanımlamalar yer almaktadır. Bu

sebeple Parser aracı, her dosya için farklı kontrol ve dosya üretim süreçleri izlemektedir.



(a) Parser aracının genel görünümü



(b) Parser aracının iç yapısı

Şekil 1. JACK etmen yazılımları için geliştirilen tersine mühendislik aracı

JACK, içerisinde yer alan bir özellik ile bileşen dosyalarındaki değişiklikleri algılayabilmektedir; fakat bu değişiklikleri direkt olarak MAS modellerine yansıtamamaktadır. Bir MAS projesi JACK geliştirme ortamının arayüzünde açıkken değişikliklerle ilgili uyarı vermekte; ancak modelleri güncellememektedir. Bunun için geliştiricilerin arayüzden tüm projeyi, değişiklikleri yükledikten sonra kaydetmeleri gerekmektedir. Bununla birlikte JACK, bir MAS'ın son çalıştırılabilir Java kodlarındaki değişikliklerinin algılanmasını ve ilgili MAS'a ait modellere bu değişikliklerin yansıtılmasını da sağlayamamaktadır. Bu bildiride tanıtılan Parser aracı söz konusu eksikliği gidermekte ve hem JACK kütüphanesine göre geliştirilmiş kaynak dosyalarındaki değişiklikleri algılayıp MAS modellerine aktarabilmekte hem de değişiklikleri doğrudan etmen modellerinin grafiksel görünümlerine yansıtabilmektedir. Etmen yazılımı geliştiricilerin bu özellikleri kullanabilmek için yalnızca Parser aracına ilgili JACK proje dizinini tanıtmaları yeterli olacaktır.

Şekil 1'de Parser aracının bu çalışmada önerilen tersine mühendislik sürecinde kullanımı gösterilmiştir. Parser aracı, JACK çerçevesinde yer alan ve yukarıda bahsedilen altı dosya türü içerisinde yer alabilecek tanımlamalar için daha önce de belirtildiği gibi bir desen eşleme tekniğini uygulamaktadır. Şekil 1.a.'da Parser'in bir MAS'a ait tüm kodları girdi olarak aldıktan sonra hangi JACK dosya türlerinde çıktı verdiği resmedilmiştir. Parser'in iç yapısı ise Şekil 1.b.'de verilmiştir. Şekil 2, 3 ve 4'te bazı MAS dosya türleri üzerinde Parser'ın nasıl çalıştığı ve bu dosya türlerinden etmen modellerinin nasıl elde edildiği sözde kodlar (ing. pseudocode) ve bunların açıklanması üzerinden verilmiştir.

Şekil 2'de, ortaya çıkması halinde bir BDI etmeninin cevap vermeye çalışacağı olayları belirten "handles event" JACK tanımlamasının MAS'a ait mevcut Java kodunda olup olmadığının belirlenmesi sürecinin sözde kod hali yer almaktadır. Java kodunda anahtar kelime olarak "addEvent" kelimesi aranmakta ve bulunduğu takdirde de olayın türü için belirleme süreci işletilmektedir.

Aranacak olan JACK Java kodu örneği

```
addEvent("DebitAccountRequest", aos.jack.jak.agent.Agent.HANDLED_EVENT);
```

Parse İşlemi

```
find pattern(start("addEvent("), regex("(.*?)"), end(")")) in .agent
if pattern exist
    loop for all patterns found
        split found string with ,
        if second string is HANDLED_EVENT
            generate JACK code with divided strings
        end if
    end loop
end if
```

Parse işlemi sonucu (JACK MAS modeli bileşeni)

```
#handles event DebitAccountRequest;
```

Şekil 2. "handles event" tanımlamalarından yazılım modelinin elde edilmesi

JACK dosyası oluşturulurken ihtiyaç duyulacak bilgiler "addEvent(" ile ")" arasında yer almaktadır (Şekil 2). Bu aradaki kod bloğunu "," ile ayırdığımızda ilk "string" event sınıfını, ikinci "string" ise "event"'in türünü ifade etmektedir. "handles

event" için olay türünün HANDLED_EVENT olması gerekmektedir. Eğer HANDLED_EVENT kontrolü de başarılı olursa, gerekli bilgiler ile JACK tanımlaması oluşturulmaktadır.

Şekil 3'te veri ya da inanç kümesi (ing. belief set) bildirimi içeren "uses data" JACK tanımlamalarının mevcut MAS yazılımına ait Java kodunda olup olmadığının belirlenmesi sürecinin sözde kod hali yer almaktadır. Java kodunda anahtar kelime olarak "getNamedObject" kelimesi aranmakta ve bulunduğu takdirde "getNamedObject(" ile ")" arasındaki kod bloğu "," ile ayrılarak gerekli bilgiler tespit edilmektedir. Son olarak da bu bilgiler yardımıyla karşılık gelen JACK model tanımlaması oluşturulmaktadır.

Aranacak olan JACK Java kodu örneği
schedule = (Schedule) agent.getNamedObject("schedule","Schedule");
Parse İşlemi

find pattern(start("getNamedObject("), regex("(.*?)"), end(")")) in .event
if pattern exist
  loop for all patterns found
  split found string with ,
  generate JACK code with divided strings
  end loop
end if

Parse işlemi sonucu (JACK MAS modeli bileşeni)
#uses data Schedule schedule;

Şekil 3. "event uses data" tanımlamalarından yazılım modelinin elde edilmesi

Şekil 4'te bir etmen yeteneğinin kapsadığı ya da bir etmenin uyguladığı planın kullandığı arayüzleri belirten "plan uses interface" tanımlamalarının mevcut Java kodunda bulunup bulunmadığının belirlenmesi sürecinin sözde kod hali yer almaktadır. Java kodunda anahtar kelime olarak "getIF" aranmakta ve bulunduğu takdirde de "interface" tanımında kullanılan "reference_name"'in belirlenmesi sürecine geçilmektedir. Bu süreçte "getIF" ile üretilen nesnenin atandığı değişken ismi "reference_name" olarak aranmaktadır. Bulunan "reference_name" ve "getIF" içerisindeki "interface" sınıf ismiyle birlikte JACK tanımlaması oluşturulmaktadır.

Aranacak olan JACK Java kodu örneği
services = (ex.accounts.AccountServices) __ns.getIF(ex.accounts.AccountServices.class);
Parse İşlemi

find pattern(start("getIF("), regex("(.*?)"), end(")")) in .plan
if pattern exist
  loop for all patterns found
  clear found string and set InterfaceType
  find pattern(start(";"), regex("(.*?)"), end("(" + InterfaceType)) in .plan for reference
  generate JACK code with divided strings
  end loop
end if

Parse işlemi sonucu (JACK MAS modeli bileşeni)
#uses interface AccountServices services;

Şekil 4. ."plan uses interface" tanımlamalarından yazılım modelinin elde edilmesi

Yukarıdaki örneklerde sözde kod ile ifade edilmeye çalışılan Parser süreçlerinin gerçek uygulaması çok daha fazla kontrol ve tanımlayıcı içermektedir. Ancak hem Parser mekanizmasının daha sade anlatılması amacıyla hem de yer kısıtları nedeniyle kodlanan tüm kontrol ve tanımlamalar bu bildiride yer almamaktadır.

Desen eşleme tekniği ile tüm dosya türleri ayrıştırıldıktan sonra etmenlere ait Java kodlarından JACK dosyaları tekrardan üretilmektedir. Parser aracı aynı zamanda MAS'a ait görsel tasarım modellerini de doğrudan bu sistemlere ait Java kaynak kodlarından oluşturabilmektedir.

JACK dosya türleri analiz edilirken, JACK dosyalarındaki tanımlamalar için analiz yapmak dışında bir de dosyalara eklenebilen harici metotlar ve değişkenler için de analizlerin yapılması zorunludur. Burada Java kodundan okunan değişken ya da metodun gerçekten JACK kodunda olması gekiyor mu yoksa JACK'in ürettiği ön tanımlı bir metot/değişken mi sorgulamasının yapılması önemlidir. Bu sorgulamalar aracılığı ile JACK dosyalarına ve dolayısıyla da tasarım modellerine yansıtılacak harici değişken ve metotlar belirlenebilmektedir.

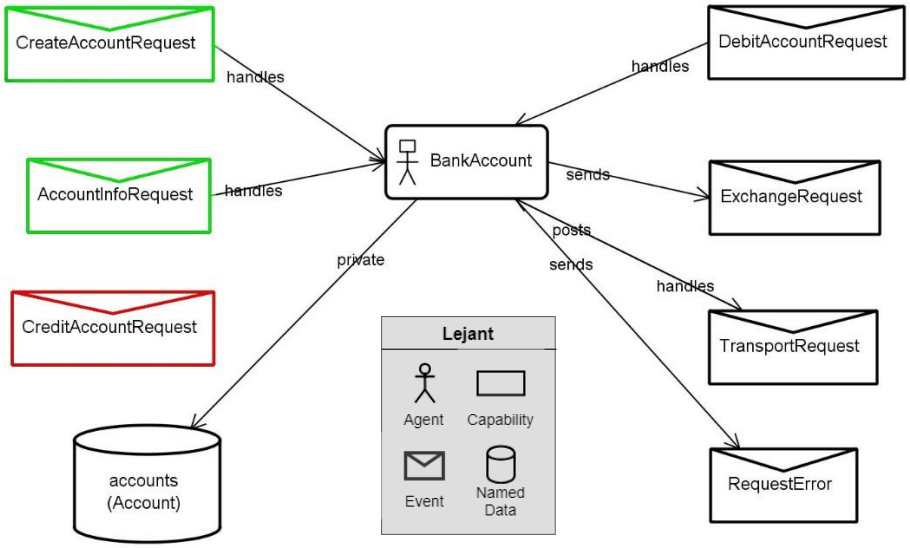
4 Durum Çalışması

Geliştirilen tersine mühendislik yönteminin uygulamasını ve Parser aracına ait yazılım mimarisinin çalışma mantığını göstermek amacıyla bu bölümde JACK platformunda çalışan bir MAS'ta yer alan yazılım etmenlerinden birinin BDI mimarisindeki bir olay güncellemesi örneklenmiştir. İlgili MAS aynı zamanda JACK'in dağıtım sitesinde [6] yer alan ve JACK platformunda yazılım geliştirme için yaygın olarak kullanılan örneklerden biridir.

Sistemdeki JACK etmenleri bankacılık uygulamalarının içerdiği işlemlerin nispeten küçük bir bölümünü yerine getirmektedir. Bu sistem içerisindeki yazılım etmenleri, banka hesaplarını belirli bir para biriminde saklamakta ve bu hesaplar ile yapılacak işlemlerin herhangi bir para biriminde gerçekleştirilebilmesi için para dönüşümü görevini yerine getirmektedir.

Yer kısıtları nedeniyle bu durum çalışmasında sadece bu MAS'taki "BankAccount" isimli tek bir etmenin farklı sistem olaylarını ele alması örneklenmiştir. Şekil 5'te bu BDI etmeninin JACK ortamındaki tasarım modeli görülmektedir. Yine örneği basit tutmak amacıyla bu etmenin mimarisi içerisinde yer alan tüm BDI olaylarından sadece "Hesap Oluşturma İsteği" (ing. "CreateAccountRequest"), "Hesap Bilgisi İsteği" (ing. "AccountInfoRequest") ve "Kredi Hesabı İsteği" (ing. "CreditAccountRequest") isimli 3 adet olayın kapsandığı bir değişiklik göz önüne alınacaktır.

"BankAccount etmeni banka hesaplarının bir tablosunu tutmakta ve bu hesaplar üzerinde işlemler gerçekleştirmektedir. İlgili etmenin şu anki modeli etmenin "CreateAccountRequest" ve "AccountInfoRequest" olaylarına yanıt verebilecek fakat CreditAccountRequest olayına yanıt vermeyecek şekilde tasarlanmıştır (Şekil 5).



Şekil 5. "BankAccount" etmeninin JACK GUI'deki tasarım modeli

Şekil 6'daki kod bloğu, Şekil 5'teki modelde görülen "BankAccount" etmeninin JACK kod bloğundan bir kısmı göstermektedir. Modelde "BankAccount" etmeni için tanımlanan "CreateAccountRequest" olayına yanıt verme işlemi "#handles event CreateAccountRequest" koduna, "AccountInfoRequest" olayına yanıt verme işlemi ise "#handles event AccountInfoRequest" koduna karşılık gelmektedir. Etmen, "CreditAccountRequest" olayına yanıt vermeyeceği için de kodda şu an herhangi bir tanımlaması yer almamaktadır.

```
public agent BankAccount extends Agent implements AccountServices {
    #handles event AccountInfoRequest;
    #handles event CreateAccountRequest;
    #handles event DebitAccountRequest;
    #handles event TransportRequest;
    #posts event TransportRequest tev;
    #sends event ExchangeRequest exchangeRequest;
    #sends event RequestError rev;
```

Şekil 6. "BankAccount" etmeninin JACK ".agent" kodundan bir bölüm

İlgili JACK kod bloklarından JACK geliştirme ortamı içerisinde yer alan derleme özelliği kullanılarak çalıştırılabilir Java kodları elde edilmiştir. Şekil 7'deki kod bloğu model aracılığı ile tasarlanan "BankAccount" etmeninin çalıştırılabilir Java kodunun bir kısmını göstermektedir. "CreateAccountRequest" ve "AccountInfoRequest" olaylarına etmen yanıt vereceği için, bu olayların kodları 2. ve 3. satırlara yerleştirilmiştir. Mevcut tasarıma göre "CreditAccountRequest" olayına yazılım etmeni cevap vermeyeceğinden ilgili Java kodunda bu BDI olayı için herhangi bir tanım yer almamaktadır. Buna göre Şekil 7'deki kod çalıştırıldığında "BankAccount"

etmeni yeni hesap oluşturma ve hesap bilgisi görme isteklerine yanıt verecek fakat kredi hesabı isteğine yanıt vermeyecektir.

```
addNamedObject("accounts","ex.accounts.Account",aos.jack.jak.agent.Agent.WRITEABLE);
addEvent("ex.accounts.AccountInfoRequest",aos.jack.jak.agent.Agent.HANDLED_EVENT);
addEvent("ex.accounts.CreateAccountRequest",aos.jack.jak.agent.Agent.HANDLED_EVENT);
addEvent("ex.accounts.DebitAccountRequest",aos.jack.jak.agent.Agent.HANDLED_EVENT);
addEvent("ex.comms.TransportRequest",aos.jack.jak.agent.Agent.HANDLED_EVENT);
addEvent("ex.comms.TransportRequest",aos.jack.jak.agent.Agent.POSTED_EVENT);
addEvent("ex.exchange.ExchangeRequest",aos.jack.jak.agent.Agent.SENT_EVENT);
addEvent("ex.comms.RequestError",aos.jack.jak.agent.Agent.SENT_EVENT);
```

Şekil 7. BankAccount etmeninin çalıştırılabilir Java kodundan bir bölüm

Kodda yapılacak değişikliklerin modele otomatik olarak yansıtılmasını görebilmek için bu aşamada son kullanıcı konumundaki bir yazılım geliştiricinin Java kodu üzerinde değişiklik yapması göz önüne alınmıştır. Bu yazılım geliştirici MAS'ın bir süre için yeni hesap oluşturma ve hesap bilgisi görme olaylarına yanıt vermemesini fakat kredi hesabı isteği olayına cevap vermesini istemektedir.

Yazılım geliştirici bu işlemi gerçekleştirebilmek adına, çalıştırılabilir Java kodunda Şekil 8'deki değişiklikleri yapmıştır. 2. satırdaki ***addEvent("ex.accounts. AccountInfoRequest", ...)*** ve 3. satırdaki ***addEvent("ex.accounts. CreateAccountRequest", ...)*** satırlarını kaldırmış, onların yerine ise ***addEvent("ex.accounts.CreditAccountRequest", ...)*** satırını eklemiştir.

```
addNamedObject("accounts","ex.accounts.Account",aos.jack.jak.agent.Agent.WRITEABLE);
addEvent("ex.accounts.CreditAccountRequest",aos.jack.jak.agent.Agent.HANDLED_EVENT);
addEvent("ex.accounts.DebitAccountRequest",aos.jack.jak.agent.Agent.HANDLED_EVENT);
addEvent("ex.comms.TransportRequest",aos.jack.jak.agent.Agent.HANDLED_EVENT);
addEvent("ex.comms.TransportRequest",aos.jack.jak.agent.Agent.POSTED_EVENT);
addEvent("ex.exchange.ExchangeRequest",aos.jack.jak.agent.Agent.SENT_EVENT);
addEvent("ex.comms.RequestError",aos.jack.jak.agent.Agent.SENT_EVENT);
```

Şekil 8. "BankAccount" etmeninin değiştirilmiş Java kodu

Etmenin kaynak kodundaki bu değişiklik için tasarım modeli ve JACK ".agent" kodu üzerinde herhangi bir değişiklik yapılmamıştır. Bu nedenle JACK de MAS'a ait proje üzerinde herhangi bir değişiklik yapıldığını henüz anlayamamaktadır. Bu aşamada tersine mühendisliği sağlayacak Parser aracı çalıştırılmaktadır. Bölüm 4'te anlatılan mekanizmaya bağlı olarak Parser, ".agent" dosyasını ve etmen tasarım modelini, Şekil 8'deki değişikliklere göre tekrar oluşturmaktadır.

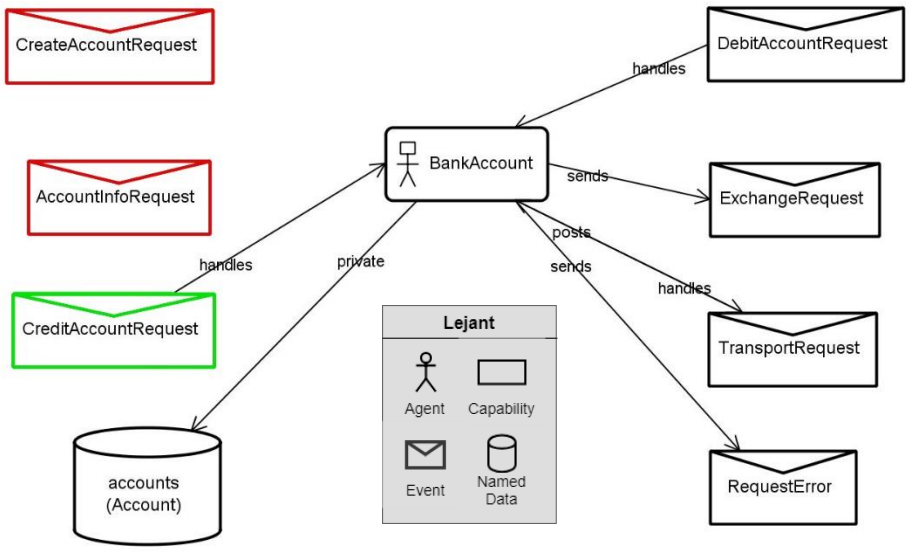
Parser çalıştırıldıktan sonra "AccountInfoRequest" olayı kaldırıldığı için "#handles event AccountInfoRequest" satırı ve "CreateAccountRequest" olayı kaldırıldığı için de "#handles event CreateAccountRequest" satırı yenilenmiş ".agent" dosyasında yer almamaktadır. "CreditAccountRequest" olayı eklendiği için ise "#handles event CreditAccountRequest" satırı ".agent" dosyasına eklenmiştir (Şekil 9).

```
public agent BankAccount extends Agent implements AccountServices {
    #handles event CreditAccountRequest;
    #handles event DebitAccountRequest;
    #handles event TransportRequest;
    #posts event TransportRequest tev;
    #sends event ExchangeRequest exchangeRequest;
    #sends event RequestError rev;
```

Şekil 9. Parser aracının otomatik oluşturduğu "BankAccount" etmeninin JACK ".agent" kodundan bir bölüm

Geliştirilen Parser, son olarak BDI etmeninin Java kodundaki değişikliği ".agent" dosyası ile birlikte etmenin model tasarımına da yansıtmıştır. "CreateAccountRequest" ve "AccountInfoRequest" olaylarına etmenin yanıt vermemesi istendiğinden etmenin görsel modelinde bu olayları temsil eden model objelerinden etmen objesine bağlı olan ilişki okları kalkmıştır (Şekil 10). Bunun yanında "CreditAccountRequest" olayına artık etmen cevap vereceği için bu olayın model objesinden etmen objesine bir bağlantı oku modelde otomatik olarak oluşturulmuştur.

Eğer Parser aracı çalıştırılmasaydı, yazılım etmeninin Java kodunda yazılım geliştiricinin yaptığı değişiklikler etmen tasarım modeline doğrudan yansıtılamayacaktı ve MAS'a ait model güncelliğini yitirecekti. Modelin yazılım geliştirici tarafından manuel olarak değiştirilmemesi varsayıldığında, mevcut model değişmediğinden dolayı JACK ortamında ilgili MAS için bir sonraki kod üretiminde, son kullanıcı konumundaki yazılım geliştiricinin yaptığı değişiklikler kaybedilecek ve MAS ilk halindeki gibi çalışmaya devam edecekti.



Şekil 10. Parser aracının oluşturduğu yeni "BankAccount" etmeni tasarım modelinin JACK GUI içerisindeki görünümü

5 İlgili Çalışmalar

MAS'ların tasarım ve uygulamasının etmenlerin içsel davranış yapısının karmaşıklığı, etmenlerin özerkliği, etmen etkileşimlerinin dağıtıklığı, vb. nedenlerle zor olması nedeniyle bu sistemlerin geliştirilmesinde soyutlama düzeyini arttıran model-güdümlü yaklaşımların kullanılması AOSE araştırma alanında güncelliğini korumaktadır [16]. [12]'de de belirtildiği gibi MAS'ların model-güdümlü geliştirilmesi için etmen sistemlerinin etmen, MAS organizasyonu, plan, etkileşim, vb. farklı bakışaçılarından modellenmesini amaçlayan üstmodel tanımlarının literatürde sunulduğu (örneğin [9, 17-19]) görülmektedir. Başka bir grup araştırma da (örneğin [20-25]) sadece etmen üstmodellerinin tanımlanması ile kalmamakta; OMG'nin Model-Güdümlü Mimarisi'ni (ing. Model-driven Architecture) (MDA) [26] doğrudan benimseyerek farklı soyutlama düzeylerindeki etmen modelleri arasındaki dönüşümler üzerinden MAS'ların model-güdümlü geliştirilmesini önermektedirler.

Son yıllarda etmenlerin modellenmesi ve bu modeller üzerinden etmen yazılımlarının otomatik üretilmesi için gerçekleştirilen çalışmaların daha çok birer DSML üretmeye yönelik olduğu görülmektedir [27]. Genellikle bu dillerin soyutsözdizimleri de yine etmen üstmodellerine dayanmaktadır. Dillerin somut sözdizimleri ise çoğunlukla etmen yazılımı geliştiricilerin sürükle-bırak mantığı ile etmen modellerini oluşturmaya imkan veren grafiksel yapılardan ibarettir. Etmenlerin modellenmesi sonrasında yine bu DSML'lerin bünyesindeki işletimsel semantikler uygulanarak modellenmiş MAS'ların JADE [28] ve JACK [6] gibi etmen platformları üzerinde çalıştırılmasını sağlayan kaynak kodlar otomatik olarak üretilmektedir. Bu DSML'lerin bir kısmı (örneğin [29-32]) MAS'ların farklı etmen mimarileri ve davranış modelleri için çeşitli iş alanlarına özgü olarak geliştirilmesini sağlarken diğer bir grup çalışma (örneğin [3, 10, 11]) doğrudan BDI mimarisindeki etmenlerin model-güdümlü geliştirilmesine yoğunlaşmıştır.

Yukarıda değinilen tüm bu model-güdümlü MAS geliştirme ve MAS DSML çalışmalarında sadece etmen sistemlerinin yukarıdan aşağıya bir mantıkla farklı soyutlama düzeylerine göre modellenmesi ve bu modellerden etmen yazılımlarının ve diğer etmen işletim dosyalarının otomatik olarak üretilmesinin mümkün olduğu görülmektedir. Bu çalışmalarda, üretilen MAS yazılımlarındaki değişikliğin ilgili sisteme ait modellere yansıtılması ve yazılımlar ile modellerin senkronizasyonuna ait herhangi bir yaklaşım ve bunu destekleyecek bir yazılım mimarisi yer almamaktadır. Bu bildiride önerilen yöntem ve geliştirilen Parser mimarisi ile AOSE alanındaki bu eksikliğe bir çözüm getirilmeye çalışılmıştır.

Tersine mühendislik yaklaşımının MAS yazılımlarının geliştirilmesinde kullanılmasına yönelik az sayıda çalışma bulunmaktadır. Hirst [33] bilişsel etmen sistemlerinin geliştirilmesi amacıyla kullanılan Soar mimarisinin eski sürümleri ile üretilmiş etmen yazılımlarını bu mimarinin stabil yeni versiyonuna dönüştürmek için kullanılabilecek ViSoar isimli bir ortamı tanıtmıştır. ViSoar ile eski Soar etmen yazılımlarının analiz edilmesi ve mevcut etmenin özelliklerini kaybetmeden etmenin yeni Soar mimarisine uygun olarak inşasının bir tersine mühendislikle mümkün olabileceği kavramsal olarak anlatılmakla birlikte buradaki tersine mühendislik etmen yazılımının modelini elde etmekten ziyade sadece eski kod yapısını yeni Soar kod

yapısına dönüştürmeyi sağlamaktadır. Moreno ve Lopez [34] MAS geliştirme yaşam döngüsünde MAS yazılımlarının detaylı bir tasarımını elde etmek için bu yazılımlara tersine mühendisliğin uygulanması gerekliliğini vurgulamıştır. Ancak bunu sağlamaya yönelik somut bir yaklaşımın getirilmesinin kendi mevcut çalışmalarının sonraki aşaması olduğunu belirtmiş; bunun yerine çalışmada etmen kullanıcılarının doğal dil ile belirtilmiş ihtiyaçlarının yazılım geliştirmeye aktarılması için bir öneri sunmuşlardır. Bosse ve ark. [35] etmen sistemlerinin davranışlarının analizi için bir tersine mühendislik çözümü önermiştir. Çalışmada bir izleyici (ing. tracer) araç kullanılarak etmen yazılımlarının anlaşılması ve "TTL Checker" adlı bir araç üzerinden de etmen sistemlerinin dinamik özelliklerinin kontrolü bütünleşik bir biçimde sağlanabilmektedir. Çalışma özellikle mevcut yazılımın analizi ile dinamik etmen özelliklerinin formalizasyonu ve etmen özelliklerinin mevcut işletim kayıtlarına göre doğrulanmasını sağlama açısından önem taşımakla birlikte içerdiği tersine mühendislik yöntemi bizim çalışmamızda önerdiğimiz kod üzerinden yazılım modellerinin geri elde edilmesini kapsamamaktadır. Perez-Castillo ve ark.'ın [36]'da tanıttığı REAGENT tekniği bu bildiride önerdiğimiz tersine mühendislik yaklaşımına benzer bir şekilde etmen yazılımı kaynak kodlarından MAS tasarım modellerinin elde edilmesini amaçlamaktadır. Ancak REAGENT sadece davranış tabanlı JADE [28] etmenlerinin soyut tasarım modellerini INGENIAS [37] metodolojisine uygun olarak elde etmeyi sağlamaktadır ve BDI etmenlerini göz önüne almamaktadır. Ayrıca elde edilen modeller JADE çerçevesinin platform modelleri yerine sadece INGENIAS modellerine dönüştürülmektedir. Her ne kadar INGENIAS'tan JADE platform modellerine dönüşüm mümkün olmakla birlikte bu dönüşüm sırasında INGENIAS ve JADE'in MDA'ye göre farklı soyutlama düzeylerinde bulunması nedeniyle model detaylarında kayıplar yaşanabilecektir. Bizim önerdiğimiz yaklaşım ise etmen yazılımının yine kendi platform modeline dönüşümünü sağlayacak bir tersine mühendislik içermektedir ve JACK BDI etmen yazılımları ile JACK platform modelleri arasında bir senkronizasyon sunmaktadır.

6 Değerlendirme ve Sonuç

BDI etmenlerinin model-güdümlü yazılım mimarilerine göre geliştirilmesi sırasında etmen sistemlerinin yazılım kodlarından BDI modellerinin otomatik olarak geri elde edilmesini sağlayacak bir tersine mühendislik yöntemi ve bu yöntemin uygulanmasını sağlayan bir Parser aracı tanıtılmıştır. Araç kullanılarak JACK BDI etmenlerinin kaynak kodlarının analizi gerçekleştirilmekte ve bu kodlardan ".agent", ".event", ".plan" gibi JACK dosyaları ve ilgili bileşenlerin görsel modellerine ait tasarım dosyaları otomatik olarak üretilebilmektedir. Böylece JACK platformunda çalışan MAS'ların yazılımları ile bu yazılımların modelleri arasındaki senkronizasyon sağlanabilmektedir. Bu aracın kullanımı ile ilgili bir demoya https://youtu.be/2B_MN9MhFN8 adresinden erişilebilinir.

Geliştirilen aracın, birkaçı JACK dağıtım sitesinde de [6] yer alan ve farklı iş alanları için üretilmiş çeşitli MAS yazılımları üzerinde uygulanması sonucunda bu yazılımlardaki dört JACK bileşeni (".agent", ".event", ".plan", ".capability") ve

bunların tasarım dosyaları için kaynak kodlarda yapılan değişiklikleri JACK dosyalarına ve ilgili tasarım modellerine başarılı bir şekilde yansıtabildiği görülmüştür. Fakat geliştirilen Parser, JACK ".plan" dosyaları içerisindeki "Akıl Yürütme" süreç tanımlama kısımlarını henüz tam olarak desteklememektedir. Bir başka deyişle bu bileşene ait akıl yürütme için yazılan Java kodlarındaki değişiklikler modellere bütünüyle yansıtılamamaktadır. Öte yandan JACK platformunun kendi içerisinde de bu dosyalardan akıl yürütme süreçlerine ait tasarım modellerinin oluşturulmasının desteklenmediği göz önüne alındığında Parser bu hali ile bile JACK içerisindeki mevcut duruma göre avantaj sağlamaktadır.

JACK ortamında geliştirilen MAS yazılımlarında genellikle yazılım geliştiriciler en çok BDI etmeninin içi, plan yapısı ve olay ve yetenek tanımlarında sonradan güncellemelerde bulunmaktadır. Bu bildiride tanıtılan Parser söz konusu bileşenlere ait kodlar için tersine mühendisliği sağlamaktadır. Her ne kadar yazılım geliştiricilerin geriye kalan iki JACK bileşeni (".view" ve ".beliefset") için otomatik üretilen kaynak kodlara, bu kodların karmaşıklığı, okunmalarının zorluğu ve MAS'ın gerçekleştirilmesinden çok JACK konfigürasyonuna yönelik olmalarından dolayı çok daha az müdahele etme ihtiyacı duyuyor olmalarına rağmen Parser kullanılarak bu kodlardan modellerin geri elde edilmesini sağlamaya yönelik çalışmalarımız devam etmektedir. Söz konusu bu iki bileşene ait JACK dosyalarının ve modellerin Java kodlarından elde edilmesi, JACK'in iki dosya türü içerisindeki farklı kod tanımlamaları için aynı Java kaynak dosyasını üretmesi nedeniyle zorlayıcı görünmektedir. Parser'in yazılım mimarisindeki mevcut ayrıştırma öncesinde bir ön işletim sağlanarak bu iki bileşen için üretilen tek kaynak dosyasının ayrıca analizinin ve ayrıştırılmasının sağlanması hedeflenmektedir.

Bu çalışmada geliştirilen tersine mühendislik yöntemi ve Parser aracı şu an BDI etmen mimarisini destekliyor olsa da ilerde BDI'ya göre çok daha basit olan davranış tabanlı ya da etki-tepki prensibi ile çalışan yazılım etmenlerinin oluşturulmasını destekleyecek şekilde genişletilebilir. Parser aracı, JACK özelinde altı farklı dosya türü için bir çok farklı ve zorlayıcı kural bütününü analiz ederek modelleri oluşturmaktadır. Örnek olarak davranış tabanlı etmen mimarisini destekleyen JADE çatısını [28] düşündüğümüzde, üç farklı dosya türü ve çok daha az kural için tersine dönüşüm tanımlamak yeterli olacaktır. Üstelik JADE'de, JACK'ten farklı olarak doğrudan Java ile programlanabilen sınıfların yazılması MAS geliştirilmesi için yeterli olmaktadır. Bununla birlikte BDI mimarisini kapsamadığından ve JACK'in gerektirdiği gibi etmen iç yapısının mantık tabanlı kurgusunu ayrı bir betik dili ile yazma ihtiyacı da bulunmadığından JADE ile hazırlanmış MAS'ların kaynak kodlarından yazılım modellerine dönüşümün basit bir şekilde Java sınıf dosyalarının ayrıştırması ile sağlanabilineceğine inanılmaktadır.

Referanslar

1. Rao, A.S., Georgeff, M.P.: Decision procedures for BDI logics. Journal of Logic and Computation 8(3), 293-343 (1998).
2. Shehory, O., Sturm, A.: Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks. Springer, Heidelberg (2014).

3. Kardas, G., Tezel, B. T., Challenger, M.: Domain-specific modelling language for belief-desire-intention software agents. IET Software 12(4), 356-364 (2018).
4. BDI4Jade, http://www.inf.ufrgs.br/prosoft/bdi4jade, son erişim: 12/11/2018.
5. Jadex, https://www.activecomponents.org/#/project/news, son erişim: 12/11/2018.
6. JACK, http://aosgrp.com/products/jack/, son erişim: 12/11/2018.
7. Fuentes-Fernandez, R., Garcia-Magarino, L., Gomez-Rodriguez, A.M., Gonzalez-Moreno, J.C.: A technique for defining agent-oriented engineering processes with tool support. Engineering Applications of Artificial Intelligence 23(3), 432-444 (2010).
8. Cossentino, M., Chella, A., Lodato, C., Lopes, S., Ribino, P., Seidita, V.: A notation for modeling Jason-like BDI agents. In: 6th International Conference on Complex, Intelligent and Software Intensive Systems, pp.12-19. IEEE, Palermo, Italy (2012).
9. Tezel, B. T., Challenger, M., Kardas, G.: A Metamodel for Jason BDI Agents. In: 5th Symposium on Languages, Applications and Technologies, pp. 8:1-8:9. Maribor, Slovenia (2016).
10. Wautelet, Y., Kolp, M.: Business and model-driven development of BDI multi-agent systems. Neurocomputing 182, 304-321 (2016).
11. Faccin, J., Nunes, I.: A Tool-Supported Development Method for Improved BDI Plan Selection. Engineering Applications of Artificial Intelligence 62, 195-213 (2017).
12. Kardas, G.: Model-driven development of multiagent systems: a survey and evaluation. Knowledge Engineering Review 28(4), 479-503 (2013).
13. Canfora, G., Di Penta, M., Cerulo, L.: Achievements and challenges in software reverse engineering. Communications of the ACM 54(4), 142-151 (2011).
14. Washizaki, H., Gueheneuc, Y.-G., Khomh, F.: ProMeTA: a taxonomy for program metamodels in program reverse engineering. Empirical Software Engineering 23(4), 2323-2358 (2018).
15. Howden, N., Ronnquista, R., Hodgson, A., Lucas, A.: Jack intelligent agents: Summary of an agent infrastructure. In:2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS at the 5th International Conference on Autonomous Agents, Montreal, Canada (2001).
16. Kardas, G., Gomez-Sanz, J.J.: Special issue on model-driven engineering of multi-agent systems in theory and practice. Computer Languages, Systems & Structures 50, 140-141 (2017).
17. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems 17(3), 432-456 (2008).
18. Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J.J., Pavon, J., Gonzalez-Perez, C.: FAML: A Generic Metamodel for MAS Development. IEEE Transactions on Software Engineering 35(6), 841-863 (2009).
19. Garcia-Magarino, I.: Towards the integration of the agent-oriented modeling diversity with a powertype-based language. Computer Standards & Interfaces 36, 941-952 (2014).
20. Gracanin, D., Singh, H. L., Bohner, S. A., Hinchey, M. G.: Model-driven architecture for agent-based systems. Lecture Notes in Artificial Intelligence 3228, 249-261 (2005).
21. Amor, M., Fuentes, L., Vallecillo, A.: Bridging the gap between agent-oriented design and implementation using MDA. Lecture Notes in Computer Science 3382, 93-108 (2005).
22. Xiao, L., Greer, D.: Towards agent-oriented model-driven architecture. European Journal of Information Systems 16(4), 390-406 (2007).
23. Hahn, C., Madrigal-Mora, C., Fischer, K.: A Platform-Independent Metamodel for Multiagent Systems. Autonomous Agents and Multi-Agent Systems 18(2), 239-266 (2009).
24. Kardas, G., Goknil, A., Dikenelli, O., Topaloglu, N. Y.: Model driven development of Semantic Web enabled multi-agent systems. International Journal of Cooperative Information Systems 18(2), 261-308 (2009).

25. Fuentes-Fernandez, R., Garcia-Magarino, L., Gomez-Rodriguez, A.M., Gonzalez-Moreno, J.C.: A technique for defining agent-oriented engineering processes with tool support. Engineering Applications of Artificial Intelligence 23(3), 432-444 (2010).
26. OMG Model-driven Architecture, http://www.omg.org/mda/, son erişim: 12/11/2018.
27. Challenger, M., Kardas, G., Tekinerdogan, B.: A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. Software Quality Journal 24(3), 755-795 (2016).
28. JADE, http://jade.tilab.com/, son erişim: 12/11/2018.
29. Hahn, C.: A Domain Specific Modeling Language for Multiagent Systems. In: 7th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 233-240, ACM, Estoril, Portugal (2008).
30. Gascuena, J.M., Navarro, E., Fernandez-Caballero, A.: Model-Driven Engineering Techniques for the Development of Multi-agent Systems. Engineering Applications of Artificial Intelligence 25(1), 159-173 (2012).
31. Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G., Kosar, T.: On the use of a domain-specific modeling language in the development of multiagent systems. Engineering Applications of Artificial Intelligence 28, 111-141 (2014).
32. Goncalves, E.J.T., Cortes, M.I., Campos, G.A.L., Lopes, Y.S., Freire, E.S.S., daSilva, V.T., deOliveira, K.S.F., deOliveira, M.A.: MAS-ML2.0: Supporting the modelling of multi-agent systems with different agent architectures. Journal of Systems and Software 108, 77-109 (2015).
33. Hirts, A.J.: Reverse engineering of Soar agents. In: 4th International Conference on Autonomous Agents, pp. 72-73, ACM, Barcelona, Spain (2000).
34. Moreno, J.C.G., Lopez, L.V.: Using techniques based on Natural Language in the Development Process of Multiagent Systems. In: International Symposium on Distributed Computing and Artificial Intelligence 2008, pp. 269-273, Springer, Salamanca, Spain (2008).
35. Bosse, T., Lam, D.N., Barber, K.S.: Tools for analyzing intelligent agent systems. Web Intelligence and Agent Systems: An International Journal, 6(4), 355-371 (2008).
36. Perez-Castillo, R., Pavon, J., Gomez-Sanz, J.J., Piattini, M.: REAGENT: Reverse Engineering of Multi-Agent Systems. In: 11th International Conference on Practical Applications of Agents and Multi-Agent Systems, pp. 228-238, Springer, Salamanca, Spain (2013).
37. Pavon J, Gomez-Sanz J.J., Fuentes R.: The INGENIAS methodology and tools. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-oriented methodologies, Idea Group Inc., Hershey, PA (2005).