İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK ETMENLİ ORTAMLAR İÇİN CNP TABANLI MÜZAKERE PROTOKOLÜ

YÜKSEK LİSANS TEZİ

Çiğdem ALBUZ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

OCAK 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK ETMENLİ ORTAMLAR İÇİN CNP TABANLI MÜZAKERE PROTOKOLÜ

YÜKSEK LİSANS TEZİ

Çiğdem ALBUZ
504081509

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Nadia ERDOĞAN

OCAK 2015

İTÜ, Fen Bilimleri Enstitüsü'nün **504081509** numaralı Yüksek Lisans Öğrencisi **Çiğdem ALBUZ**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"ÇOK ETMENLİ ORTAMLAR İÇİN CNP TABANLI MÜZAKERE PROTOKOLÜ"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Nadia ERDOĞAN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. Güray Yılmaz** Hava Harp Okulu	.....
	Yrd. Doç. Dr. Tolga Ovatman İstanbul Teknik Üniversitesi	.....

Teslim Tarihi : **15 Aralık 2014**
Savunma Tarihi : **21 Ocak 2015**

Anneme,

ÖNSÖZ

Tez çalışmam süresince hem teknik hem de manevi desteğinden ve göstermiş olduğu hoşgörüden dolayı danışman hocam sayın Prof. Dr. Nadia Erdoğan'a ve bu süreçte motivasyonumu yüksek tutan aileme sonsuz teşekkürlerimi sunarım.

Ocak 2015

Çiğdem Albuz
Bilgisayar Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

CNP **:** Contract Net Protocol
JADE **:** Java Agent Development Framework
FIPA **:** The Foundation of Intelligent Physical Agents
ACL **:** Agent Communication Language
CFP **:** Call For Proposal
TILAB **:** Telecom Italia Lab
AMS **:** Agent Management System
DF **:** Directory Facilitator
AID **:** Agent Identifier

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

ÇOK ETMENLİ ORTAMLAR İÇİN CNP TABANLI MÜZAKERE PROTOKOLÜ

ÖZET

Çok etmenli sistemlerde, etmenler arası görev dağıtımı önemli bir araştırma konusudur. Etmenler arası müzakereler için protokoller geliştirilmiştir. Bu protokoller arasında yer alan ve oldukça yaygın olan CNP, Smith ve Davis [1, 2] tarafından geliştirilmiştir. Bu protokol, ihale kavramına dayanan bir dağıtık görev atama müzakere modelidir. Yönetici ve adaylar vardır. Bu sistem dinamiktir ve herhangi bir anda yönetici ve adaylar katılabilir ya da ayrılabilirler. CNP'de müzakere adımları şu şekilde ilerler:

- Yönetici, elindeki işi duyurur. Buna teklif daveti denir (CFP).
- Adaylar, elindeki iş listesine göre, yeni duyurulan iş için teklif verirler.
- Yönetici tüm teklifleri toplar ve içlerinden en iyi olana işi atar.
- Onay alan aday, yöneticiye teyit mesajı gönderir.

Ancak CNP protokolünün bazı kısıtları vardır. Görüşmeler sıralı olarak yapılır. Müzakereleri sırasallaştırmak bazı anlaşmaları kaçırmaya neden olabilir. Adaylar tarafından verilen teklifler, daha iyi fırsatların oluşması durumunda tutulmayabilir. Etmenlerin çökmesi durumunda ihale kilitlenebilir.

Bu çalışmada çok etmenli bir ortamda taşımacılık problemi simüle edilmiş ve S.Aknine, S. Pinson, M.Shakun[3] tarafından geliştirilmiş CNP protokolü gerçeklenerek yürütme ortamı sağlanmıştır. Bu çalışma CNP'ye ek olarak şunları sağlar:

1) Bir etmenin birkaç görüşmeyi paralel yürütmesine olanak verir.
2) Görüşme süresini optimize eder.
3) Adayların sözlerinden caymasını azaltır.
4) Görüşmeye katılan etmenlerden çöken olursa anlaşılmasını ve görüşmenin bloklanan adaylar ile devam etmesini engeller.

Genişletilmiş protokolde, orjinal CNP'deki teklif verme ve atama temel adımları geliştirilerek, ÖnTeklif, KesinTeklif, ÖnAtama ve KesinAtama adımları eklenmiştir. Görüşmeler iki aşamada yapıldığı için ulaşılan sonuç, her iki taraf için de optimumdur.

Çok etmenli ortamın yaratılması için JADE platformu kullanılmıştır. JADE, FIPA standartlarında, çok etmenli sistemleri geliştirmeye olanak sağlayan bir ara katman yazılımıdır. Java tabanlıdır. Etmenlerin yaratılmasını, yönetilmesini ve birbileri ile haberleşmelerini sağlayan araçlar ve kütüphaneler sunar. Jade kütüphaneleri kullanılarak yönetici ve aday etmenlerden oluşan çok etmenli bir sistem kurulmuş ve birbirleri ile haberleştikleri bir ihale sistemi çalışması ortaya çıkartılmıştır.

Yönetici ve aday sayıları değiştirilerek çeşitli deneyler yapılmış ve sonuçları kıyaslanmıştır. Başarı kriteri olarak sonuca ulaşmak için atlatılan adım sayısı ve

müzakere sırasında geçen süre baz alınmıştır. Görev ve aday etmen sayısındaki artış ihale süresinin uzamasına neden olmaktadır. Ancak en iyi teklif bilgisinin iletildiği durumlarda, iletilmediği durumlara göre özellikle toplam etmen sayısının arttığı karmaşık sistemler için ihalelerin daha kısa sürede sonuçlandığı görülmüştür.

CNP BASED NEGOTIATION PROTOCOL FOR MULTI-AGENT SYSTEMS

SUMMARY

Multi-agent negotiation and task allocation is a fundamental research issue. Various negotiation protocols have been developed. One of them is the well known CNP protocol which is defined by Smith and Davis[1, 2]. It is used for decentralized task allocation and based on the notion of call for bids. There are manager and contractor agents. The system is dynamic and each agent on the network can join or leave the system at different times. CNP negotiation steps are as the following:

- Manager announces the task to the possible agents by sending a call for proposal (CFP) message.
- Contractors examine their current task lists and prepare proposals.
- Manager receives all of the proposals and evaluates them, chooses the best bidder and assignes the task.
- Chosen contractor sends a confirmation message.

The CNP protocol has some limitations. Firstly, in a distributed environment, several managers can concurrently call for bids, so an agent may have to manage several negotiation processes in parallel in order to reduce the length of its negotiation processes. Some applications of the CNP force the contractors to sequence their negotiation processes. Thus, when there are several managers that call for bids, contractors cannot give proposals in parallel. Thus negotiation processes take longer, and meanwhile, contractors may miss some contracts.

Secondly, CNP-based applications enable the agent to break its commitments when an agent receives an offer for a better task in comparison with those for which the agent is committed. If a manager comes with a better task, contractors may break their commitments. This is not a good solution because it makes managers call again for bids to find new contractors for their tasks. In case of agent failures, negotiation processes may be blocked.

This study describes the design and implementation of an agent based execution environment where agents follow the extended CNP negotiation protocol defined by S.Aknine, S. Pinson, M.Shakun[3] during negotiations. An application based on a transportation problem has been developed to assess the system.

The transition of negotiation states between a manager and the contractors are defined as below in extended CNP negotiation protocol:

- A manager announces a task and the negotiation process starts.
- Contractors prepare a proposal considering their current task list and send their temporarily bids.
- Manager receives these PreBids and sends PreAccept message, temporarily positive answer, to the best contractor and sends PreReject messages to the remaining contractors.

- Whenever the situation of the contractors changes positively, they can bid again with a better PreBid until they receive DefinitiveReject from the manager. At this level, negotiation process between manager and contractor can evolve to a PreAcccept state or stay in a PreReject state.
- The contractor which received PreAccept message sends DefinitiveBid, the final proposal for execution of the task.

Manager agent may question this DefinitiveBid either if, there is a better PreBid or if DefinitiniveBid of the potential contractor is much lower than its PreBid. If this is the case, manager agent sends DefinitiveReject message to the potential contractor and continues the negotiation process with the remaining contractors. Otherwise negotiation process result in signing the contract with a DefinitiveAccept message to the potential contractor and DefinitiveReject message is sent to all other contractors.

With the extended protocol, along with efficient task allocation, the following advantages are also obtained:

1) A contractor may attend several negotiation processes in parallel. MxN negotiations are enabled. A contractor can manage several negotiations concurrently with M managers and a manager can manage several negotiations concurrently with N contractors. Thus, global negotiation duration is reduced.
2) An agent gives a temporary bid called PreBid and it can change its offer until it gives a definitiveBid. This enables the contractor to make the best choice.
3) The protocol reduces the risk of decommitment.
4) Manager gets the best offer since it informs the remaining contractors to get a better bid until the potential contractor gives DefinitiveBid.
5) Decommitment cases are reduced since both the manager and the contractors choose the best choice, thus manager will not need to reinitiate the negotiation process.

In the extended CNP protocol, differently from the 2 phased original CNP, there are 2 additional phases; PreBidding, PreAssignment, DefinitiveBidding and DefinitiveAssignment. Since the negotiation is processed in 2 steps, the result has the maxsimum benefit for both sides.

In this study, JADE is used to implement the multi-agent system. Jade is a middleware that facilitates the development of multi-agent systems. It is FIPA compliant, an organization that defines the standarts for multi-agent systems. It is written in Java. Jade provides tools and libraries to create and manage agents which communicate with each other. Using Jade libraries, a multi agent system containing managers and contractors is created and negotiation processes are handled in this environment.

Two types of agents exist in the implemented task allocation environment:

- Manager agents
- Contractor agents

Each agent type has different behaviors and run different algorithms. These algorithms base on below assumptions:

- Agents can communicate, negotiate and make agreements.
- There exists no explicit dependency between tasks.

- Agents are self interested and aim to maximize their own benefits.

Various experimental studies have been performed for different numbers of manager and contractor agents and the results are compared. The success key is the negotiation time and the number of the steps applied until the negotiation process is completed.

In order to see the effect of varying numbers of contractor agents and task count on negotiation time, we have carried out three sets of experiment. The first set of experiments were executed with the number of contractor agents equal to 2, the second with the number of contractor agents equal to 4 and the third with the number of contractor agents equal to 6. Task count is increased from 2 to 20 and the time taken for negotiation process is recorded for both the manager agent inform the contractor agents about the best offer and does not.

We have also carried out experiments to observe how varying the number of agent and task counts affects negotiation completion step count.

Increased in the number of the contractor and manager agents cause the negotiation time take longer. Better results have been observed when the best offer is also sent by the PreReject message compared to not being sent, especially for more complex and crowded agent systems.

1. GİRİŞ

Etmenler, tasarım hedefleri doğrultusunda bir ortam içerisinde bağımsız hareket edebilen bilgisayar sistemleridir. 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 geldiklere ağa, çoklu etmen sistemi adı verilir.

Etmen tabanlı sistemler 1990'lardan itibaren bilgisayar dünyasının en canlı ve önemli araştırma alanlarından biri olmuştur. Etmen sistemleri kavramı; bilgisayar ağları, yazılım mühendisliği, nesneye dayalı programlama, yapay zekâ, insan bilgisayar etkileşimi, dağıtık ve paralel sistemler, kontrol sistemleri, karar verme ve bilgi çıkarımı gibi birçok bilgi teknolojisinde geniş kullanım alanı bulmaktadır. E-ticaret, grafik uygulamaları, koordine savunma sistemleri, taşımacılık, lojistik gibi çeşitli alanlarda çok geniş uygulanabilirliğe sahiptir. Özellikle ağ ve mobil teknolojilerde otomatik ve dinamik yük dengeleme ve yüksek ölçeklenebilirlik sağlamak için kullanılmaktadır. Etmen sistemlerinin otonom, zeki ve hedef tabanlı yapısı çeşitli alanlardaki yeni nesil yazılımlar için umut verici çözümler sunmaktadır. Bu nedenle günümüzde de oldukça önemli bir araştırma konusudur.

Çok etmenli sistemlerde her bir etmen belirli bir görev için özelleşmiş olabilir. Ortamdaki diğer etmenler bir göreve ihtiyaç duyduğunda o görevi gerçekleştiren etmenle iletişim kurup sonuç alabilir. Etmenlerin bu şekilde işbirlikçi çalışması hem yazılımsal iş yükünü azaltmakta hem de çevikliği arttırmaktadır. Bu nedenle çok etmenli sistemlerde, etmenler arası görev dağıtımı için çeşitli çalışmalar yapılmış ve protokoller geliştirilmiştir. Bunlar arasında çok bilinen ve oldukça yaygın olan CNP protokolü [1, 2] Smith ve Davis tarafından geliştirilmiş ve pek çok çalışmada temel alınmıştır. Bu çalışmalardan biri de S.Aknine, S. Pinson, M.Shakun tarafından orijinal CNP protokülünün bazı kısıtlarına çözüm üretmek için tasarlanan genişletilmiş-CNP (ECNP) protokolüdür [3].

FIPA, çok etmenli sistemler arasındaki birlikte çalışabilirliği en üst düzeye çıkartmak için evrensel standartlar ortaya koymak amacı ile kurulan, kar amacı gütmeyen bir topluluktur. Günümüzde ortaya konan etmen tabanlı yazılım sistemlerinin büyük bir kısmı bu topluluğa ait soyut mimariye uygun olarak tasarlanmıştır.

JADE, FIPA standartlarında, çok etmenli sistemleri geliştirmeye olanak sağlayan Java tabanlı bir ara katman yazılımıdır. Etmenlerin yaratılmasını, yönetilmesini ve birbileri ile haberleşmelerini sağlayan araçlar ve kütüphaneler sunar.

Bu çalışmada; çok etmenli sistemlerde, etmenler arası görev paylaşım problemi ele alınmıştır. Etmenlere, etkin ve verimli bir şekilde görev atanması amaçlanmıştır. Bu doğrultuda FIPA standartları ile uyumlu bir çoklu etmen ortamı, JADE kullanılarak tasarlanmıştır. Tasarım ve gerçekleme sırasında S.Aknine, S. Pinson, M.Shakun tarafından geliştirilen genişletilmiş-CNP(ECNP) protokolü [3] kullanılmıştır. Algoritma, taşımacılık şirketine ait bir ihale süreci simüle edilerek uygulanmış ve çeşitli deneysel çalışmalar yapılmıştır.

1.1 Tezin Amacı

Bu çalışmada, iş yapma yeteneği olan otonom etmenlere verimli ve etkin bir şekilde görev atanması amaçlanmıştır. Jade kullanılarak etmen tabanlı bir yürütme ortamı tasarlanmış ve ECNP protokolü ile etmenler arasında görev paylaşımı gerçeklenmiştir. Uygulamada ihale sistemi modellenmiş ve taşımacılık problemi üzerine uygulanmıştır. Yönetici ve aday etmen sayıları değiştirilerek çeşitli testler yapılmış ve sonuçları kıyaslanmıştır.

2. LİTERATÜR ARAŞTIRMASI

Çok etmenli sistemlerde, etmenler arası görev dağıtımı ile ilgili çeşitli çalışmalar yapılmış ve protokoller geliştirilmiştir. Bu konuyla ilgili en temel çalışmalardan biri Smith ve Davis tarafından geliştirilmiş olan Contract Net Protocol (CNP)'dir [1, 2]. Bu protokol, ihale kavramına dayanan bir dağıtık görev atama modelidir. Yönetici ve aday etmenlerden oluşan dinamik bir sistem vardır. Pek çok çalışmada temel alınmış ve üzerinde geliştirmeler yapılmıştır. Bugeliştirmelerden biri de S.Aknine, S. Pinson, M.Shakun tarafından geliştirilmiş olan genişletilmiş-CNP protokolüdür (ECNP) [3]. Bu çalışmada orjinal CNP'deki bazı kısıtlar için öneriler getirilmiştir. Bu önerileri *M-N-P* sistemlere uyguladıkları başka bir çalışmaları daha mevcuttur [4].

T. Bouron tarafından yapılan bir çalışmada aday etmenler, önerilen işin tamamını yapamayacaksa, işi parçalara bölerek başka adaylar ararlar [5]. Bu işlem için 2 yöntem kullanılabilir.

- Aday etmen öncelikle iş kabul eder. Ardından bu işi yapabilecek başka aday etmenler arar. Bu durumda uygun aday etmen bulunamaması halinde, anlaşılan yönetici etmene karşı verilen söz tutulamazya da ceza ödenir.
- Aday etmen, yönetici tarafından henüz kesin kabul almadan, işi yerine getirebilecek yeni aday etmenler ile anlaşır, ardından yönetici etmen ile görüşür. Eğer iş alınamazsa, sistemde kilitlenmeler oluşabilir.

Çok etmenli sistemlerde koordinasyon problem ile ilgili bir çalışma da Jennings tarafından yapılmıştır [6]. Bu çalışmada görev atama işlemini yapan merkezi bir dağıtıcı bulunur ve diğer etmenlerin görev yetenekleri hakkında sahip olduğu bilgiyi kullanarak görev atama işlemini gerçekleştirir. Bu çözüm, merkezi bir etmen gerektirir.

Dağıtık sistemlerdeki çoklu etmenlerin anlaşarak bir görevi işbirliği ile yerine getirmesinedayanan çalışmalar [7, 8] olduğu gibi her etmenin bireysel çalışıp, duyurulan bir görevin sadece belli bir parçası için teklif vermesine yönelik çalışmalar da bulunur [9]. Ancak bu sistem bağımlı bir yöntemdir. Sadece karmaşık ve

bölünebilir görevleri içeren sistemlere uygulanabilir. Ayrıca yönetici etmenin, görevin tamamına teklif veren etmenleri mi yoksa parçalarına teklif veren adaylarımı seçeceğinin belirlenmesi gerekir.

Aday etmenlerin sözünden caymasını engellemek için ceza uygulaması yapan çalışmalar bulunur [10]. Bu da gönüllülük esasına dayanan işleri aksatır.

Çok etmenli sistemlerdeki koordinasyon problemini ele alan çalışmalardan biri de Güncel-CNP çalışmasıdır [11]. Bu çalışmada da orjinal CNP deki bazı kısıtlamalar için çözüm önerilmiştir. Değiştirilebilirlik ve iletişim yükü sınırlamaları açısında CNP'yi geliştirmeye yönelik bir çalışmadır. Bir yönetici etmen tarafından, bir aday etmene atanan görevin içeriğinin değişmesi durumunda, orjinal CNP'de görev atama işlemi tekrar başlar. Güncel-CNP ile bu değişikliklerin yönetici etmenler tarafından işi yürüten aday etmene bildirilmesi önerilmiştir. Böylece iletişim yükünün azalacağı ve CNP sürecinin kısalacağı öne sürülmüştür. Bu işlem, orjinal CNP aşamalarına yeni bir adım daha eklenerek sağlanmıştır. Bu adımda, aday etmen tarafından henüz işin yürütülmesi devam ederken görevde değişiklik olması durumunda, bu değişiklik bir mesaj ile aday etmene iletilir. Böylece, özellikle sık sık görev tanımında değişiklik olan sistemler için, protokolün verimliliğinin artacağı belirtilmiştir.

Orjinal CNP üzerinde geliştirmeler yapan bir diğer çalışma ise etmenlerin güvenirliklerini ele almaktadır [12]. Bir görev atama sürecinin, iletişimde olan etmenlerin güvenilir olmadığı bir ortamda verimli olamayacağı düşüncesine dayanmaktadır. Bu nedenle orjinal CNP'ye güvenilirlik hesaplama bileşeni eklenmiştir. Yönetici etmen, teklifleri değerlendirirken bu hesaplamayı da dikkate alır. Güvenilirlik hesaplama bileşeni şu şekilde çalışmaktadır:

- Yönetici etmen "Etmen Güvenirlik Tablosu"na sahiptir. Bir görevi duyurup teklifleri toplamaya başladığında, karar vermek için bu tabloyu kullanır.
- Bir aday etmen bu yönetici etmene ilk kez teklif veriyorsa tabloda kaydı yoktur. Öncelikle "0" güvenirlik değeri ile bir kayıt atılır.
- Bu aday etmen bir görev için atanıp başarılı bir şekilde yerine getirirse bu değer 1 artırılır, tamamlayamazsa bu değer 1 eksiltilir.

- Güvenirlik değeri için bir eşik değeri seçilir (Bu çalışma için -5 seçilmiştir.) Eğer bir aday etmen bu eşik değerin altına düşerse güvenilmez olarak kabul edilir ve bir daha teklif veremez.
- Aday etmenler arasında görev atanırken, güvenirlik değeri en iyi olan adaya öncelik verilir.
- Eğer eksi değere düşmüş bir aday varsa, eşik değerini aşmamış olması koşulu ile, ancak kendisinden daha iyi bir aday yoksa görev atanır, böylelikle pozitif tarafa geçmesi için bir fırsat verilmiş olur.

Bu çalışma ile, orjinal CNP'ye göre, görev atama performansında ciddi bir iyileşme olduğu belirtilmektedir.

Yazılım etmenlerini geliştirmek için kullanılan çeşitli platformlar mevcuttur. Bu platformlar etmen iletişimi, kullanıcı ara yüzü, etmen davranış planlaması ve zamanlaması gibi bazı temel hizmetleri sağlar ve farklı problem alanlarına göre etmen gerçekleştirimine olanak tanırlar. Bu etmen geliştirme platformlarını inceleyerek benzerlik ve farklılıkları ele alan çalışmalar da bulunmaktadır [13].

Yazılım etmenleri teknolojisinin ne tür yazılım sistemlerinin geliştirilmesinde kullanılabileceğini araştıran çalışmalar [14], FIPA uyumlu çok etmenli sistem tasarımı içeren çalışmalar [15, 16] bulunmaktadır.

3. ETMEN SİSTEMLERİ

3.1 Etmen Tanımı

Etmenler, tasarım hedefleri doğrultusunda bir ortam içerisinde bağımsız hareket edebilen bilgisayar sistemleridir [17]. Bir etmen sistemi oluşturmak için, ortamda çalışan ve ortamla iletişim halinde olan tek bir etmen yeterli olsa da, genellikle birçok etmen aynı ortamda birlikte çalışır. Birden fazla etmenin; aynı ortamda, birbirleriyle çelişen veya uyum halinde çalıştıkları ve birbirleriyle mesajlaşma yoluyla iletişim kurabildikleri sistemlere ise çoklu etmen sistemleri adı verilir.

Etmen tabanlı sistemler 1990'lardan itibaren bilgisayar dünyasının en canlı ve önemli araştırma alanlarından biri olmuştur [18]. Etmen sistemleri kavramı; bilgisayar ağları, yazılım mühendisliği, nesneye dayalı programlama, yapay zekâ, insan bilgisayar etkileşimi, dağınık ve paralel sistemler, kontrol sistemleri, karar verme ve bilgi çıkarımı gibi birçok bilgi teknolojisinde geniş kullanım alanı bulmaktadır. Etmen sistemlerinin otonom, zeki ve hedef tabanlı yapısı çeşitli alanlardaki yeni nesil yazılımlar için umut verici çözümler sunmaktadır [18].

3.2 Etmenlerin Özellikleri

Bir yazılımın etmen olarak kabul edilebilmesi için bazı özellikleri sağlıyor olması gerekir. Bir etmene ilişkin özellikler iki grupta incelenmektedir. Her etmende olması gereken özellikler birincil özellikler olarak adlandırılmaktadır. Birincil özellikler, bir etmeni klasik donanım veya yazılım sistemlerinden ayıran özelliklerdir. Bir donanım veya yazılım sisteminin etmen olarak nitelendirilebilmesi için mutlaka birincil özellikleri barındırması gerekmektedir. Bunun dışında, bir sistemin etmen olma kavramını güçlendiren ve genellikle geliştirilen uygulamaya bağlı olan özellikler de bulunmaktadır. Bu özellikler ikincil özellikler olarak adlandırılmaktadır [19].

3.2.1 Birincil özellikler

Bir etmenin sahip olması gereken birincil özellikler şu şekilde sıralanabilir:

- **Özerklik (Autonomy):** Bir etmenin insan kullanıcıların, diğer etmenlerin veya sistemlerin doğrudan katılımı olmadan görev başlatabilme ve çalışabilme yeteneği olarak tanımlanmaktadır. Özerk bir etmen, kendi eylemleri ve içsel durumunu kontrol edebilmektedir. Bir etmenin özerk olması onun sınırsız bir özerklik çerçevesi içinde çalıştığı anlamına gelmemektedir. Bir etmenin özerkliği kullanıcısı tarafından denetlenebilmelidir. Bu bağlamda, etmenler gerektiğinde kullanıcılarına danışarak onlardan aldıkları geribildirimler doğrultusunda eylemlerini planlamaktadırlar.

  Oda sıcaklığını algılayan ve buna göre ısıtıcıyı çalıştıran veya durduran bir termostat ile bir nükleer reaktörü denetleyen bir sistemde özerklik özelliği bulunmaktadır. Belli bir yazılım ortamında arka plan süreçleri olarak çalışan bazı yazılım birimleri de özerk programlar olarak nitelendirilmektedir. Kullanıcısına gelen e-postaları izleyen ve yeni bir e-posta geldiğinde kullanıcısını uyaran bazı uygulamalar bu tür arka plan süreçlerine ilişkin örneklerdir [20].

- **Karşıt-eylemlilik (Reactivity):** Bir etmen sürekli olarak bulunduğu ortamı algılamalı, ortamdaki değişimlere göre bilgisini, amaçlarını, eylemlerini değiştirebilmelidir. Bununla birlikte gerektiğinde diğer etmenlere ileti göndererek söz konusu değişikliği onlara da bildirebilmelidir. Etmenin içinde bulunduğu ortam, fiziksel algılayıcılar ile algılanabilen fiziksel bir ortam, bir etmenler topluluğu veya İnternet olabilmektedir. Örneğin, İnternet ortamında herhangi bir konuda bilgi sunan bir sunucuyu izleyen bir etmen, izlediği sunucudaki bilgide güncelleme yapıldığını algıladığında bilgisini, amaçlarını ve eylemlerini gözden geçirmeli, ayrıca bu değişikliği ilgili kullanıcı ve etmenlere onlara ileti göndermek yolu ile bildirmelidir.

- **Amaç-yönelimlilik (Goal-orientedness):** Amaç-yönelimlilik, bir etmenin kendisinden beklenenleri yerine getirmek için planlama yaparak bu planların gerektirdiği eylemleri gerçekleştirmesi olarak tanımlanmaktadır. Bir etmenin eylemde bulunması bu tez kapsamında, o eyleme ilişkin yordam, yöntem veya fonksiyonların işletilmesi anlamında kullanılmaktadır. Etmenin bir eylemde bulunabilmesi için o eyleme ilişkin ön koşulların sağlanmış olması gerekmektedir. Etmenin bir eylemde bulunması sonucunda etmenin bilgisi,

varsayımları değişebilmekte veya başka etmenlere ileti göndermesi gerekebilmektedir.

Bir etmenin planlar oluşturabilmesi için plan kütüphanelerinde bulunan plan kalıplarından yararlanılmaktadır. Etmenin amaca ulaşması için yerine getirmesi gereken görevler alt görevlere ayrılarak, her bir alt göreve karşılık gelen plan parçası kütüphaneden alınmakta, diğer alt görevlere ilişkin plan parçaları ile birleştirilmekte ve sonuçta plan oluşturulmaktadır.

- Bir etmenin yalnızca karşıt-eylemli (reactive) veya yalnızca amaç-yönelimli (goal-oriented) olması yeterli olmamaktadır. Yalnız amaç-yönelimli bir etmen planlarının gerektirdiği eylemleri yaparken ortamda olabilecek herhangi bir değişikliği göz önüne almamaktadır. Bu durumda, planın işletimi sırasında plana ilişkin tüm görevlerin sonuçlanmasını engelleyecek bir ortam değişikliği olduğunda plan tamamlanamayacak ve amaca ulaşılamamış olacaktır. Örneğin, belli bir plan çerçevesinde belli bir bilgi kaynağına erişim gerekiyorsa ve söz konusu kaynak o anda servis dışı kalmışsa planın bu kaynağa erişim ile ilgili olan kısmının bitmesini beklemek bir sonuca götürmeyecektir. Bu nedenle, bir plana ilişkin görevlerin yapılması sırasında ortam sürekli algılanıp gerektiğinde planlar değiştirilebilmelidir. Yukarıdaki örnek göz önüne alınacak olursa, bilgi kaynağının servis dışı kaldığı algılandığında plan yeni bir bilgi kaynağı bulunması amacıyla değiştirilmelidir. İşletilmekte olan bir planın kesilip, onun yerine yeni bir planın işletilmeye başlamasına yeniden planlama (re-planning) adı verilmektedir.
- İdeal bir etmen, amaç-yönelimlilik ve karşıt-eylemlilik özelliklerini elden geldiğince dengeli kullanmaya çalışan bir etmendir. Böylece, etmenin bir sonuca ulaşmayacak eylemleri ısrarla sürdürmesi engellenmiş olacaktır.
- **Sosyal yetenek (social ability):** Etmenler diğer etmenlerle (bazen de insanlarla) etmen dili denilen dillerle ya da herhangi bir programlama diliyle iletişime geçebilirler ve hatta amaçlarına yönelik ortaklıklar kurabilirler.
- **Kalıcı süreklilik (Temporal continuity):** Bir etmen, belli bir görevi yapıp durmamalıdır. Etmenler belli bir ortamda sürekli olarak çalışan birimlerdir. Etmen, başlattığı görevleri tamamladıktan sonra da etkin olarak kalmalı ve

çevresini algılayıp gereken eylemleri gerçekleştirmek için hazır olarak beklemelidir.

3.2.2 İkincil özellikler

Bir etmenin ikincil özellikleri arasında gezicilik, öğrenme, akılcılık, dürüstlük ve olumluluk sayılabilmektedir. Bir etmenin ikincil özelliklerin tümünü içermesi gerekmemektedir. Geliştirilen uygulamaya bağlı olarak bazı özellikler çok önemli olabilirken, bazıları da hiç kullanılmayabilmektedir. Örneğin, kendini uyarlayabilen arabirim uygulamasında öğrenme özelliği mutlaka barındırılması gereken bir özellik iken gezicilik özelliğinin olmaması kendini uyarlayabilen arabirim uygulamasını etmen olarak nitelendirmeyi engellememektedir. İkincil özelliklerden bazıları şunlardır:

- **Gezicilik (Mobility):** Bir etmen eğer ağ üzerinde bir yerden başka bir yere gidebiliyor ise "gezici etmen" olarak adlandırılmaktadır. Örneğin, bir işlemi daha güçlü işlemcili bir bilgisayarda yapıp geri dönmek veya ağ üzerinde bilgi kaynaklarını gezerek bilgi toplamak gibi istekler etmenin gezici olmasını gerektirmektedir. Gezici etmenlerin ağ üzerinde gezinmesi güvenliğin sağlanması konusunu ön plana çıkarmaktadır.
- **Öğrenme (Learning):** Bir etmen kendisini ortama uyarlayabilmeli, kullanıcı eğilimlerini öğrenerek eylemlerini bu doğrultuda değiştirebilmelidir.
- **Akılcılık (Rationality):** Bir etmenin amaçları doğrultusunda planlama yapmak için çalışması "akılcılık" olarak adlandırılmaktadır. Akılcı bir etmen, kasıtlı olarak, amaçlarına ulaşmayı engellemek için plan yapmamalıdır.
- **Dürüstlük (Veracity):** Bir etmenin kasıtlı olarak yanlış bilgi iletişiminde bulunmaması "dürüstlük" olarak adlandırılmaktadır.
- **Olumluluk (Benevolence):** Bir etmenin, amaçlarına ters düşmedikçe kendisinden beklenen her şeyi yerine getirmek için çalışması "olumluluk" olarak adlandırılmaktadır.

3.3 Çoklu Etmen Sistemleri

Daha önce de bahsedildiği gibi etmenler sadece çevrelerini algılamak ve onlara tepki vermekte yetinmezler. Etmenler sosyal ilişkiler kuran aktif yazılım bileşenleridir.

Etmenler arasındaki ilişki bazen rekabet, bazen ortaklık temelli bazen de her iki temele birden dayanacak şekildedir. Çoklu etmen sistemleri, birbirleriyle iletişim kuran birden fazla akıllı etmenin bir araya gelerek oluşturduğu sistemlerdir [21]. Etmenler kendi yetenekleri ve bilgilerinin yeterli olmadığı problemleri bir araya gelerek oluşturdukları bu sistemlerle çözebilirler. Çoklu etmen sistemlerinin özellikleri [22] şu şekilde sıralanabilir:

- Her bir etmenin problemin tamamını çözmek için eksik bilgisi ve yeteneği vardır.
- Tüm sistemi kontrol edemezler.
- Veri merkezileşmemiştir. Bir tek noktadan tüm veriye erişim mümkün değildir.
- Hesaplamalar asenkron yapılır.

4. JAVA AGENT DEVELOPMENT FRAMEWORK (JADE)

JADE, TILAB tarafından geliştirilen, noktadan noktaya iletişim mimarisini temel alan, FIPA standartlarına uygun olarak tasarlanmış bir dağıtık çoklu etmen geliştirme platformudur [23]. Yürütme anında bir etmen platformu, ağdaki aynı işletim sistemine sahip olmayan farklı makineler üzerine dağıtılabilir ve uzaktan bir grafik arabirimi ile yönetilebilir. İletişim mimarisi, etmenler arasında esnek ve verimli bir şekilde, eş zamanlı ve eş zamanlı olmayan haberleşmeye izin verir. FIPA etmen ve iletişim modelini tamamen destekleyecek şekilde gerçeklenmiştir ve sistemle bütünleştirilmiştir. JADE, FIPA üyesi olan veya olmayan birçok şirket ve akademik grup tarafından kullanılmaktadır. JADE, tamamıyla Java dilinde yazılarak bu dilin esnek yapısını ve hazır kütüphanelerini kullanarak kolaylıkla etmen geliştirme yapısı sunmayı hedeflemiştir. JADE'in kullanıcılarına kullanımı ve özelleştirmeyi kolaylaştırmak için hazır olarak sunduğu özellikler aşağıdaki şekilde sıralanabilir [24]:

- Etmenler, tam bir dağıtık sistem oluştururlar. Her etmen, farklı bir thread olarak çalışır ve farklı makineler üzerinde çalışabilir. Etmenler bulundukları makinenin ve işletim sistemlerinin özelliklerinden bağımsız olarak JADE'in haberleşme altyapısını kullanabilir.
- AMS (Agent Management System – Etmen Yönetim Sistemi), DF (Directory Facilitator – Sarı Sayfalar Servisi), ACC (Agent Communication Channel – Etmen İletişim Kanalı) başta olmak üzere FIPA standartlarına uyumludur. Böylece aynı standardı kullanan farklı etmenler, birbirleriyle kolayca haberleşebilirler. FIPA iletişim şartnamesinin kullanılması için gerekli olan kütüphaneler kullanıma hazır şekilde geliştiricilere sunulmuştur [25].
- Grafik arabirimi ile etmenler kontrol edilebilir.
- Etmenlere ilişkin kod ve verinin farklı platformlardaki katılımcı bilgisayarlar arasında taşınabilmesini sağlar.
- Davranış modelleri ile çoklu, paralel ve eş zamanlı etmen aktivitesini düzenler.

- Etmenlerin, dağıtık etmen platformunda benzersiz bir şekilde isimlendirilmesi JADE tarafından otomatik olarak yapılmaktadır. Her etmen başlatıldığında ona benzersiz bir isim atanır ve arama servisine kaydedilir. Etmenler, sağlanan API'ler ve görsel araçlar sayesinde yerel ortamdan veya uzaktan başlatılabilir.
- Etmen olmayan ve JADE ortamına bağlanmamış dış uygulamalar tarafından da etmenlerin başlatılmasını sağlayan bir arabirim mevcuttur.
- Yaygın, güçlü ve esnek bir programlama dili olan Java ile gerçeklenmiştir ve açık kaynak kodludur.
- Uygulamaya özel içerik dilleri ve ontolojiler tanımlamaya izin verir.
- Java dilinin Serializable özelliğini kullanarak mesajların içeriğine istenilen bir nesne eklenebilmektedir.

4.1 JADE Platformunun Genel Yapısı

JADE platformu aşağıdaki Java paketlerinden oluşur [24].

- *jade.core* paketi sistemin çekirdeğini oluşturur. Agent isimli sınıfı içerir. Yazılacak sistemdeki tüm etmenler bu sınıftan türetilirler. *jade.core.behaviours* isimli alt paketindeki Behaviour sınıfı ise etmenler tarafından yürütülecek görev sınıfları için ana sınıfıdır. Bu pakette çeşitli amaçlara yönelik hazırlanmış davranış-görev sınıfları bulunur.
- *jade.lang.acl* paketi ise FIPA ACL standartlarına uygun olarak etmen haberleşmesini sağlayan sınıfları barındırır.
- *jade.content* paketi uygulamalar tarafından tanımlanan ontolojileri ve içerik dillerini desteklemek için kullanılan sınıflardan ve alt paketlerden oluşur.
- *jade.domain* paketi ve alt paketleri FIPA standartları tarafından AMS ve DF özelliklerini yani yaşam döngüleri ve sarı sayfalar servislerini sağlayan etmen sınıfları tutarlar.
- *jade.gui* paketi oluşturulacak uygulamalar için kullanıcı arabirimi geliştirmekte kullanılan sınıfları barındırır.

- *jade.mtp* paketi sistemlerin JADE platformuna entegre çalışması için gerçeklemeleri gereken Mesaj İletim Protokolüne dair ara yüzleri ve bu ara yüzlerinin gerçeklenmiş bazı örneklerinden oluşur.
- *jade.proto* paketinin içerdiği sınıflar standart etkileşim protokollerini gerçeklerler. Aynı zamanda kullanıcılar tarafından protokol tanımlanması için gereken ana sınıflar da bu pakettedir.
- *jade.wrapper* paketi JADE platformuna üst seviye özellikler eklemelerini ve başka uygulamaların JADE etmenlerini yürütmelerini sağlayan sınıflardan oluşur.

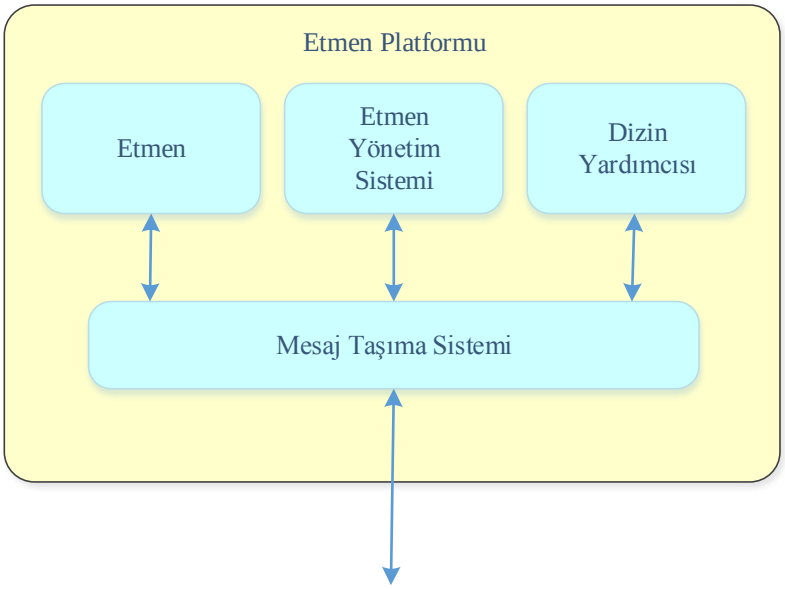
4.2 FIPA

FIPA, etmenler ve etmen tabanlı sistemlerin birlikte çalışabilmesi için gerekli standartları belirleyerek etmen sistemler endüstrisini teşvik etmeyi amaçlayan uluslararası bir organizasyondur [26]. FIPA'nın belirlediği standartlar, soyut bir etmen sistemi mimarisi belirler ve geliştirilecek mimariye en az sayıda kısıt getirmeyi hedefler. FIPA'nın belirlediği standartlarda en çok, ticari ve endüstriyel kullanım amaçlı etmen ortamları hedeflenmiştir [27]. JADE, FIPA tarafından belirlenen soyut etmen mimarisine göre tasarlanmıştır ve bileşenleri bu mimariye göre belirlenmiştir.

4.3 Etmen Platformu

JADE çalışma ortamının her çalışan örneğine Taşıyıcı (Container) adı verilir. Her taşıyıcı birden fazla etmeni aynı anda bulundurabilir. Aktif taşıyıcılar topluluğuna Platform adı verilir. Bir platformda her zaman bir ana taşıyıcı aktif halde bulunur ve diğer taşıyıcılar çalışmaya başladıkları anda platforma kayıt olurlar. Bir platformda açılan ilk taşıyıcı, ana taşıyıcıdır ve daha sonra başlatılan taşıyıcılar normal taşıyıcılardır Normal taşıyıcılar başlatılırken, kayıt olacağı ana taşıyıcının adresinin ona bildirilmesi gereklidir [26].

Etmen platformu, FIPA mimarisinde tanımlanan ve Şekil 4.1'de [24] görülen bileşenlerden oluşur.



Şekil 4.1 : FIPA etmen platformu mimarisi.

AMS (Agent Management System – Etmen Yönetim Sistemi) etmeni tüm etmen platformunun yönetiminden sorumludur. Her sistemde yalnız bir tane AMS bulunur. AMS etmenler için yaşam döngüsü servisleri sunar, etmenlere kimlik atanması (AID) ve durumlarının takip işlemlerini yapar. Platforma bağlanacak her etmen AMS'den geçerli bir kimlik almak zorundadır.

DF (Directory Facilitator – Dizin Yardımcısı) etmenler arası sarı sayfalar servisi sunar.

ACC (Agent Communication Channel – Mesaj İletim Sistemi) platform içinde mesaj iletimi yapısını kuran bileşendir. Bu mesajlaşmaya, uzak platformlarla yapılan mesajlaşma da dâhildir.

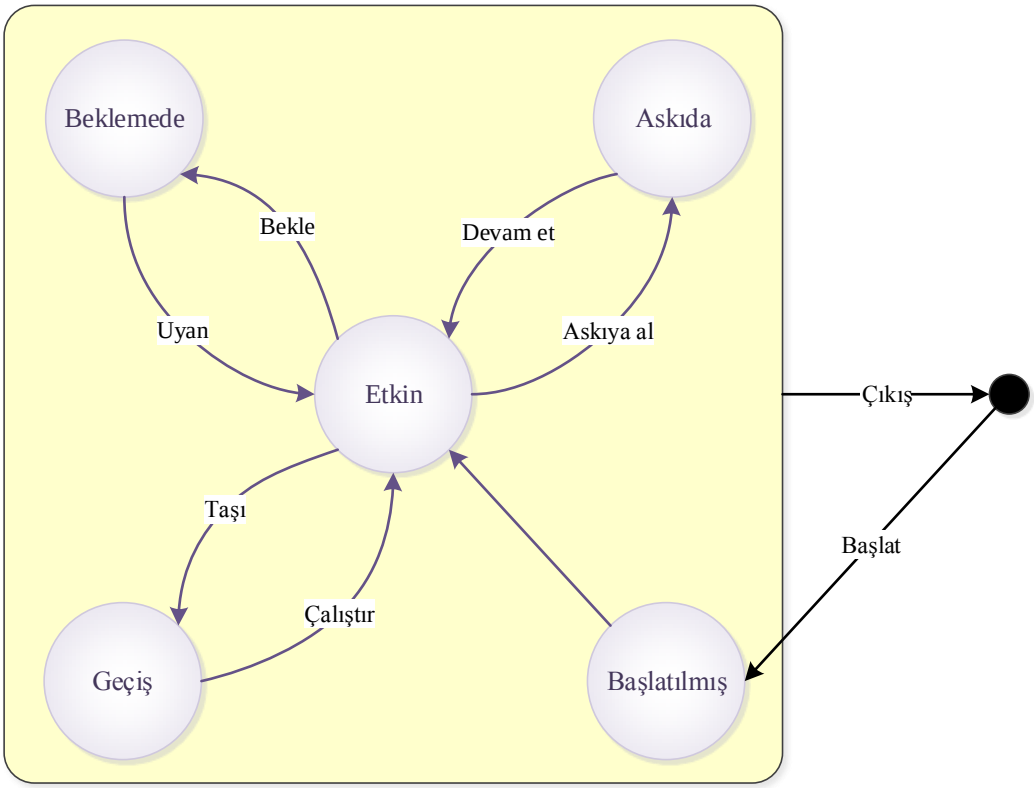
JADE FIPA standartlarıyla ve yukarıda tanıtılan mimariyle tam uyumludur. JADE platformu çalıştırıldığında AMS, DF ve ACC bileşenleri başlatılır ve çalıştırılır. Aynı zamanda bu bileşenlerin her biri farklı bilgisayarlarda çalıştırılabilecek yapıya sahiptirler [24].

4.4 Etmen Yönetimi

JADE platformunda etmenler, *jade.core.Agent* sınıfı kullanılarak başlatılırlar. Bu sınıftan kalıtımla türetilen sınıflar ile özelleştirilmiş etmenler başlatılır. Etmen ilk çalıştırıldığında bu sınıfın *setup()* metodu çağrılır. Bu metot ile etmen, kullanacakları

kaynakları oluşturur, eğer bir servis sunacak ise kendisini DF'ye kayıt ettirir ve diğer başlangıç koşulları ile ilgili işlemlerini yapar. Etmen yok edilirken *takeDown()* metodu çağırılır. Bu metot ile etmen kullandığı kaynakları iade eder ve DF'ye kayıt olduysa bu kaydını siler [24].

FIPA standartlarına göre bir etmen çalışma süresi boyunca çeşitli durumlarda bulunabilir. Durum geçiş diyagramı Şekil 4.2'de [24] verilmiştir.



Şekil 4.2 : Etmenin yaşam döngüsü.

- **Başlatılmış (Initiated):** Etmen oluşturulmuş, fakat AMS'ye kayıt olmamıştır. Henüz bir ismi ve adresi yoktur ve diğer etmenler ile haberleşemez.
- **Etkin (Active):** Etmen oluşturulmuş ve AMS'ye kayıt olmuştur. İsmi ve adresi vardır. Tüm JADE özelliklerine erişebilir durumdadır.
- **Askıda (Suspended):** Etmenin çalıştığı thread ve etmenin davranışları durdurulmuştur.
- **Beklemede (Waiting):** Etmen bloke olmuş durumdadır ve çalışmaya devam etmek için bir koşulun gerçekleşmesini beklemektedir.
- **Silinmiş (Deleted):** Etmenin çalışması tamamen sonlanmıştır başka bir deyişle etmenin AMS kaydı silinmiş durumdadır.

- **Geçiş (Transit):** Bir hareketli etmen başka bir konuma taşınırken bu duruma geçer. Sistem bu etmene gönderilen mesajları tampon belleğe alır ve geçiş tamamlandığında mesajları etmene iletir.

4.4.1 Etmen başlatma

JADE platformunda, yeni bir etmen başlatılırken şu adımlar uygulanır [24]: etmen kurucu metodu çalıştırılır, etmene benzersiz bir isim verilir, yeni etmen AMS'ye kayıt edilir, etmenin durumu etkin yapılır ve son olarak etmenin *setup()* metodu çağırılır. FIPA standartlarına göre bir etmenin evrensel benzersiz bir isime ve başka etmenlerin yeni oluşturulan etmenle haberleşmesi için bir adres kümesine ihtiyacı vardır.

Geliştirici, etmeni başlatmak için *setup()* metodunu kullanır. Bu metodun içerisinde şu işlemler yapılabilir:

- Gerekli görüldüğü durumlarda AMS'ye kayıt edilmiş veriler değiştirilir.
- Gerekli görülen durumlarda, etmenin açıklaması ve ortama sunduğu servisler ayarlanır. Ayrıca ihtiyaç duyulursa etmen birden çok DF'ye kayıt ettirilir.
- Görev kuyruğuna *addBehaviour()* metodu kullanılarak yeni davranışlar eklenir.

Etmeni başlatmak için kullanılan *setup()* metodunda en az bir adet görev tanımlanmalıdır. Bu metodun sonlanması ile otomatik olarak hazır davranış kuyruğundan alınan ilk davranış çalıştırılır ve davranışlar sonlandıkça kuyruktan alınan sıradaki davranış çalıştırılır. Bir davranış çalışırken başka bir davranış onun çalışmasını kesemez. Etmene davranış ekleme ve etmenden davranış çıkartma metotları etmenin görev kuyruğunu yönetmek için kullanılır.

4.4.2 Etmen sonlandırma

Etmenin *doDelete()* metodu çağrıldığında etmen sonlandırılır. Etmenin kullandığı kaynakları temizlemesi ve kayıt olduğu servislerden kaydını silmesi işlemlerinin yapılabilmesi için otomatik olarak *takeDown()* metodu çağırılır, *takeDown()* metodunun çağrılması ile etmen yok edilir. Bu metot çağrıldığı sırada etmenin AMS kaydı silinmemiş haldedir, başka etmenlere mesaj gönderebilir ve sonlandırılmakta olduğunu haber verebilir. Metot sonlandıktan sonra etmen için başlatılmış olan thread sonlandırılır [24].

4.4.3 Etmen davranışları

Bir etmenin görevleri yerine getirmesi, davranış (behaviour) tanımı aracılığıyla olur. Bir davranış *jade.core.Behaviours.Behaviour* sınıfının genişletilmesi ile tanımlanır. Bu davranışın etmen tarafından yürütülebilmesi için etmen sınıfı içerisinde *addBehaviour()* metoduyla etmene eklenmesi gerekir. Davranışlar, etmenin açılışında çalıştırılan *setup()* metodunda veya daha sonra başka bir davranış içerisinden de eklenebilir.

Behaviour sınıfını genişleten her sınıf, soyut bir metot olan *action()* metodunu gerçeklemek zorundadır. *action()* metodunda, davranışın çalışması sırasında yapılacak işler tanımlanır. Bir davranışın *action()* metodu sonlanmadan başka bir davranışın *action()* metodu çalıştırılmaz.

JADE platformu, davranışlarla çalışmayı kolaylaştırmak için farklı özelliklere sahip çeşitli davranış sınıfları bulundurur. Bu sınıflar kullanım amaçlarına göre doğrudan kullanılabilir veya özelleştirilmiş bir çalışma şekli için ilgili soyut metotları gerçeklenerek genişletilebilir.

- **Behaviour:** Behaviour sınıfının genişletilmesiyle, genel amaçlı davranış sınıfları oluşturulur. Oluşturulan sınıf, içerisinde bir durum değişkeni tutar ve bu değişkenin değerine göre yapacağı işleme veya davranışın sonlanmasına karar verir [26].
- **OneShotBehaviour:** Bir defa çalışan ve hemen sonlanan davranışlar oluşturulur [26]. Bu davranışlar tüm işlemlerini *action()* metodunun sadece bir defa çağrılacağını göz önünde bulundurarak yapmalıdır.
- **CyclicBehaviour:** Bu tanımdaki davranışlar, hiç bir zaman sonlanmazlar. Davranışın her çağrılmasında *action()* metodundaki aynı işlemler tekrarlanır [26]. Davranış ancak *removeBehaviour()* metodu ile sonlandırılır.
- **TickerBehaviour:** Belirli zaman aralıklarıyla gerçekleştirilmesi gereken görevler için kullanılır [26]. Davranışa parametre olarak verilen zaman aralığı her dolduğunda, *onTick()* metodu çağırılır. Bu metot sonlandığında, davranış zaman aralığı tekrar dolduğunda çalışacak şekilde görev kuyruğuna eklenir.
- **WakerBehaviour:** Belirli bir süre geçtikten sonra veya belirli bir tarih ve saatte gerçekleştirilmesi istenen görevler için kullanılır [26]. Davranışın

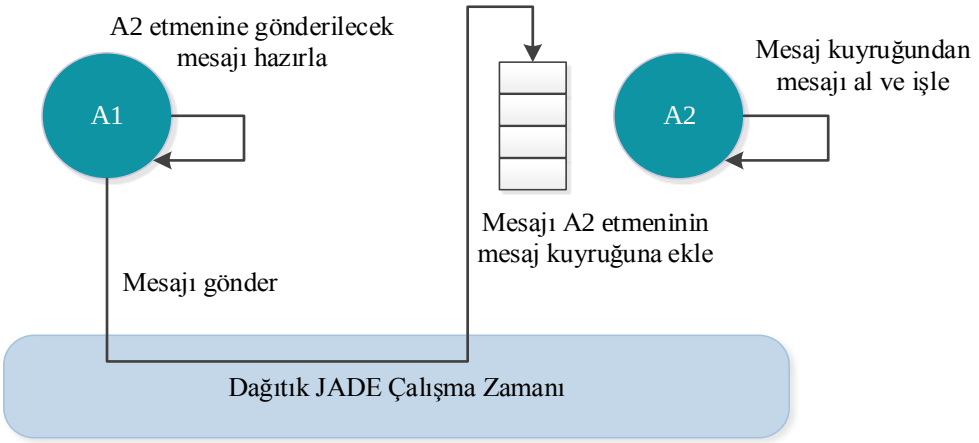
çalışması istenen tarih ve saat geldiğinde, davranışın *onWake()* metodu çalıştırılır ve bu metodun çalışması bittikten sonra etmen sonlandırılır.

- **SequentialBehaviour:** Birden fazla davranışın sırayla çalıştırılmasını sağlayan davranıştır. Bu sınıfın *addSubBehaviour()* metodu kullanılarak çalıştırılması istenen davranışlar sırayla eklenir. Davranış çalışmaya başladığında tüm alt davranışlarını sırayla çağırır. Bir alt davranış sonlandığında sıradaki alt davranışa geçilir [26].
- **ParallelBehaviour:** Birden fazla davranışın paralel olarak çalıştırılmasını sağlayan davranıştır. Bu sınıfın *addSubBehaviour()* metodu ile çalıştırılması istenen davranışlar eklenir.
- **FSMBehaviour:** Birden fazla davranışın, sonlu durumlu makine modeline göre çalıştırılmasını sağlayan davranıştır. Sonlu durumlu makine modeline uygun çalışmasını sağlamak için bir durum geçiş tablosu tutulur. Bu durum geçiş tablosundaki her düğüm bir alt davranışı temsil eder. Alt davranış sonlandığında çalıştırılacak yeni davranışı belirlemek için durum tablosundaki değerlerle karşılaştırılır. Karşılaştırma sonucunda çalıştırılacak yeni alt davranış belirlenir veya eğer sonlanma durumuna ulaşıldıysa ana davranış sonlandırılır [26].

Yapılan çalışmada OneShotBehaviour, CyclicBehaviour, TickerBehaviour, SequentialBehaviour ve ParallelBehaviour sınıfları kullanılmıştır.

4.5 Etmen Haberleşmesi

Etmen haberleşmesi JADE'in en temel özelliğidir ve FIPA iletişim modeline uygun olarak tasarlanmıştır. İletişimin temelinde eş zamanlı olmayan mesaj taşıma işlemi vardır. Her etmenin, başka etmenlerin gönderdiği ve JADE çalışma zamanı tarafından iletilen mesajların ulaştığı etmen mesaj kuyruğu vardır. JADE haberleşme mimarisi, ana hatlarıyla Şekil 4.3'te [26] verilmiştir. Diğer etmenlerden gelen mesajlar dağıtık JADE çalışma zamanı üzerinden geçerek mesajın gönderildiği etmenin mesaj kuyruğuna gelir. İlk gönderilen mesaj ilk önce işlenmek üzere gelen mesajlar mesaj kuyruğundan alınarak sırayla işlenir.



Şekil 4.3 : JADE haberleşme mimarisi.

4.5.1 FIPA iletişim modeli

Etmen sistemleri için oluşturulmuş FIPA modelinin merkezinde, etmenlerin uygulama tarafından talep edilen işlemleri yapabilmeleri için birbirlerine anlamlı mesajlar aktarabilmesini sağlayan etmen haberleşmesi bulunur [27]. Bu nedenle etmenler arasında mesaj aktarım yöntemi, mesajların temsil edilme yöntemi (XML, ikili nesneler, gibi) ve diğer özellikleri tanımlayan etmen mesajlaşma dili (ACL) oluşturulmuştur. FIPA standartlarına göre bir ACL mesajında bulunan alanlar aşağıda verilmiştir [28]:

- performative: Mesajın iletişimsel eylemini bir başka deyişle mesajın türünü belirtir.
- sender: Mesajı gönderen etmeni belirtir.
- receiver: Mesajın gönderileceği etmeni veya etmenler kümesini belirtir.
- reply-to: Mesaja verilecek cevabın gönderilmesi gereken etmeni belirtir.
- content: Mesajın içeriğini bulundurur. İçerik bilgisi alıcı etmen tarafından yorumlanır.
- language: Mesajın içeriğini bulunduran content alanının biçimsel dilini (SQL, FIPA-SL, Prolog, vb.) belirtir.
- encoding: content alanının karakter kodlamasını belirtir.
- ontology: Mesajdaki content alanındaki sembollere anlam verebilmek için hangi kelimeler kümesinin kullanıldığını belirtir.
- protocol: Eğer mesaj FIPA tarafından tanımlanmış bir protokole göre gönderilirsa, bu protokolü belirten alandır.

- conversation-id: Etmenin, yaptığı farklı mesajlaşmaları birbirinden ayırt etmek için kullandığı alandır.
- reply-with: Alıcı etmenin cevap verirken kullanması gereken ifadeyi belirtir.
- in-reply-to: Mesajın, hangi tipten bir mesaja cevap olarak gönderildiğini belirtir.
- reply-by: Alıcı etmenin aldığı mesaja son cevap verme tarihini belirtir.

FIPA standartlarına göre bir mesajın iletişimsel eylem tipi başka bir deyişle türü aşağıdakilerden birisi olabilir [29]:

- accept-proposal: Daha önce propose ile yapılan bir önerinin kabul edildiğini belirtir.
- agree: Daha önce request ile yapılması istenen bir eylemin yapılmasının kabul edildiğini belirtir.
- cancel: Bir etmenin, diğer etmene daha önce gönderdiği bir işlemin yapılması isteğinin artık geçerli olmadığını belirtmesini sağlar.
- CFP: Bir etmenin, mesajı alan etmenlerden bir öneri beklediğini belirtir.
- confirm: Göndericinin, alıcı etmene bir önermenin doğru olduğunu bildirdiğini belirtir.
- disconfirm: Göndericinin, alıcı etmene bir önermenin yanlış olduğunu bildirdiğini belirtir.
- failure: Yapılması istenen bir eylemin başarısız olduğunu belirtir.
- inform: Göndericinin, alıcı etmene bir önermenin doğru olduğunu bildirdiğini belirtir.
- inform-if: Göndericinin, alıcı etmene bir önermenin doğru veya yanlış olduğunu bildirdiğini belirtir.
- not-understood: Göndericinin, alıcının yaptığı bir eylemi anlamadığını belirtir.
- propagate: Alıcı etmenin, mesajın içeriğine gömülü bir mesajı, belirtilen etmenlere göndermesi gerektiğini belirtir.
- propose: Göndericinin, alıcı etmenin belirli koşullar altında bir eylem yapmasını önerdiğini belirtir.
- proxy: Alıcı etmenin, verilen bir açıklamaya göre gerekli etmenleri seçerek gömülü olan mesajı seçtiği etmenlere göndermesi gerektiğini belirtir.

- query-if: Göndericinin, alıcı etmene bir önermenin doğru olup olmadığını sorduğunu belirtir.
- query-ref: Göndericinin, nesnenin kaynağını belirten bir ifade ile alıcı etmenden bir nesneyi istediğini belirtir.
- refuse: Yapılması istenen bir eylemin reddedildiğini belirtir.
- reject-proposal: Daha önce yapılan bir önerinin kabul edilmediğini belirtir.
- request: Göndericinin, alıcı etmenden bir eylem veya başka bir iletişimsel eylem yapmasını istediğini belirtir.
- request-when: Göndericinin, alıcı etmenden bir koşul gerçekleştiğinde bir eylem yapmasını istediğini belirtir.
- request-whenever: Göndericinin, alıcı etmenden bir koşul her gerçekleştiğinde bir eylemi tekrar yapmasını istediğini belirtir.
- subscribe: Göndericinin, alıcıdan belirli bir değer her değiştiğinde haberdar edilmesini istediğini belirtir.

4.5.2 ACLMessage sınıfı

JADE kütüphanesinde bir ACL mesajı, jade.lang.acl.ACLMessage sınıfı ile tanımlanır. Bu sınıf, FIPA modeline göre gerekli olan tüm alanları bulundurur. Bu alanların her biri için *get()* ve *set()* metotları gerçeklenmiştir. FIPA tarafından tanımlanan tüm iletişimsel eylem tipleri yani mesaj türleri, ACLMessage sınıfı içerisinde birer sabit değişken olarak tanımlanmıştır [24].

4.5.3 Mesaj gönderme ve alma

Mesaj gönderme işlemi, ACLMessage sınıfının *send()* metoduyla, mesaj alma işlemi ise *receive()* ve *blockingReceive()* metotlarıyla yapılır. *blockingReceive()* metodu adından da anlaşılacağı üzere mesaj gelene kadar etmenin ana thread yapısını bloke eder ve diğer davranışlarını yapmasını engeller, bu nedenle kullanımında dikkatli olunmalıdır. *receive()* metodu ise mesaj gelene kadar null değerini döndürür fakat diğer davranışların çalışmasını engellemez.

5. ÇOKLU ETMEN SİSTEMLERİNDE GÖREV ATAMA PROTOKOLLERİ

5.1 Contract Net Protocol (CNP)

CNP, çok etmenli sistemlerde görev paylaşımı protokolüdür. Bir etmen tamamlanması gereken bir görev için diğer etmenlerden yardım alabilir. Bu durumda yönetici rolündedir. Bu görevi yerine getirebilecek niteliğe sahip etmenler de görevin tamamlanması için teklif verirler. Görev atama işlemi bir nevi ihale sistemi gibidir ve 4 aşamadan oluşur:

1) Yönetici görevi, işi yapabilecek nitelikteki aday etmenlere duyurur.
2) Adaylar, önerilen işin tanımına göre teklif hazırlarlar.
3) Yönetici, tüm teklifleri alır ve inceler. En iyi teklif veren adaya görevi atar.
4) İşi alan Aday, yöneticiye teyit mesajı gönderir.

Ancak CNP protokolünün bazı kısıtları vardır. Görüşmeler sıralı olarak yapılır. Müzakereleri sırasallaştırmak bazı anlaşmaları kaçırmaya neden olabilir. Adaylar tarafından verilen teklifler, daha iyi fırsatların oluşması durumunda tutulmayabilir. Etmenlerin çökmesi durumunda ihale kilitlenebilir.

Bu çalışmada orjinal CNP protokolündeki kısıtlamalar için geliştirmeler yapılan ve S.Aknine, S. Pinson, M.Shakun [3] tarafından geliştirilen CNP algoritması temel alınmış ve algoritmanın üzerinde gerçekleneceği çoklu etmen tabanlı bir yürütme ortamı tasarlanıp gerçeklenmiştir.

5.2 Geliştirilmiş CNP Protokolü

CNP üzerinde geliştirmeler yapılmış ve daha verimli bir müzakere süreci amaçlanmıştır. CNP protokolüne ek olarak aşağıdaki özellikler geliştirilmiştir:

1) Bir aday birkaç görüşmeyi paralel yürütebilir.
2) Görüşme süresini optimize eder.
3) Adayların sözlerinden cayma durumlarını azaltır.

4) Görüşmeye katılan etmenlerden çökenler olursa, bu etmenlerin anlaşılmasını sağlar ve görüşmenin bloklanan etmenler ile devam etmesini engeller.

5.3 Görev Atama Algoritması

Bu bölümde, gerçeklenen S.Aknine, S. Pinson, M.Shakun [3] algoritması ayrıntılı olarak ele alınacaktır. Sistemin çalışacağı ortam için bazı tanımlar ve kabuller yapılmıştır.

Kabuller:

- Etmenler birbirleri ile iletişim kurabilir, anlaşma yapabilir.
- Görevler basittir, tek bir etmen tarafından gerçekleştirilebilirler.
- Görevler arası bağımlılık olmadığı kabul edilir.

Tanımlar:

Etmen kümesi *n* tane etmenden oluşur, $A = \{A_1, A_2, \dots, A_n\}$. Görev kümesi *m* tane birbirinden bağımsız görevden oluşur, $T = \{t_1, t_2, \dots, t_m\}$.

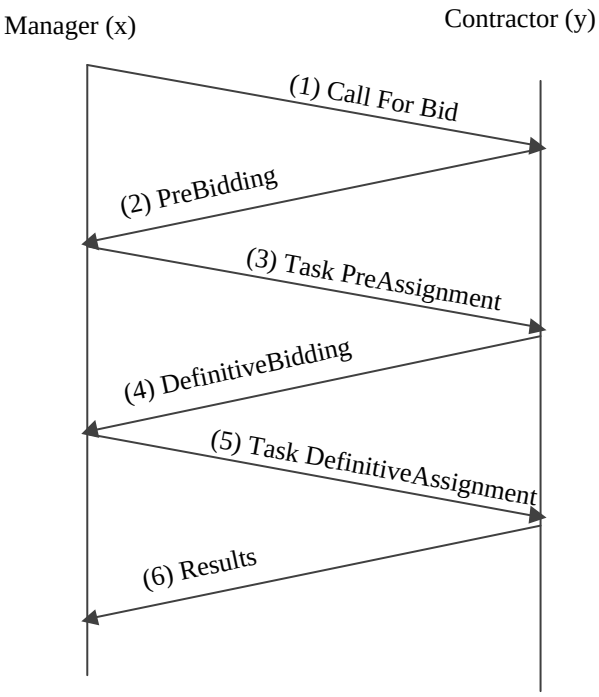
Çok etmenli bir sistemde yönetici *M* etmeni ve bir aday etmen a_i arasındaki T_k görevi için yapılan müzakere protokolü 6 bileşenlidir:

$$Neg_{ik} = (a_i, M, T_k, P, s, v)$$

- $a_i \in A$; a_i, *M* yöneticisi tarafından Neg_{ik} için işe alınmış bir aday etmendir.
- $M \in A$ *M*; *M*, Neg_{ik} müzakeresini başlatan yönetici etmendir.
- T_k; Neg_{ik} müzakeresinin sunulan görevidir.
- P; görüşme ilkelidir. PReBid, PreReject, PreAccept, DefinitiveBid, DefinitiveAccept ve DefinitiveReject değerlerini içerir.
- $s \in P$; s Neg_{ik} müzakeresinin o anki durumudur.
- v; a_i aday etmenin, *M* yöneticisine, T_k görevi için verdiği tekliftir.

5.4 Aknine-Pinson-Shakun Görev Atama Protokolü

Bir etmenin eşzamanlı olarak birkaç ihaleye katılabilmesi için CNP'deki 2 aşamalı (Bid ve Assignment) protokol 4 aşamaya çıkarılmıştır: PreBidding, Definitive Bidding, PreAssignment, DefinitiveAssignment. Yeni protokolün aşamaları Şekil 5.1'de gösterilmiştir.



Şekil 5.1 : Protokol aşamaları.

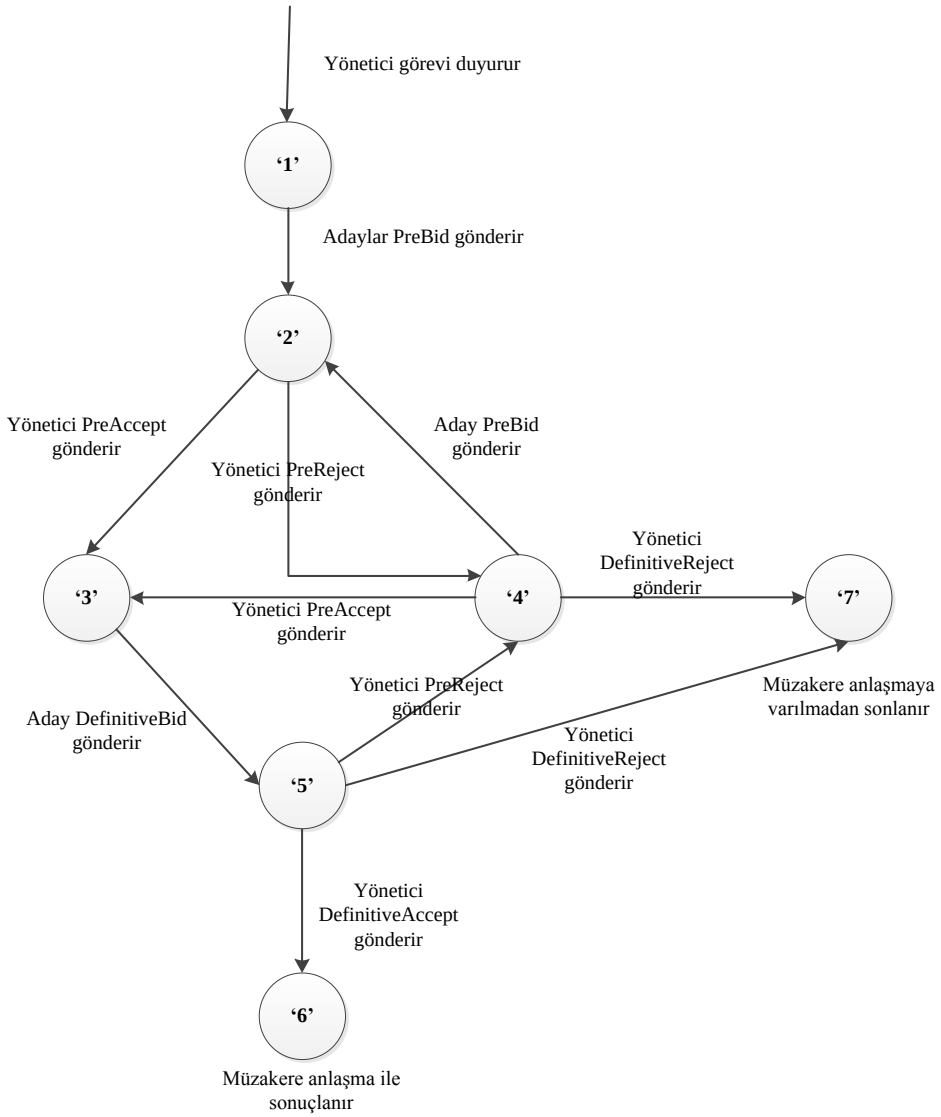
Yönetici ve adaylar arasında kullanılan iletişim ilkelleri Announce, PreBid, PreAccept, PreReject, DefinitiveBid, DefinitiveAccept ve DefinitiveReject'dir. Çizelge 5.1'de her operatörün anlamı açıklamıştır.

Çizelge 5.1 : İletişim ilkellerinin anlamları.

İlkel	Anlamı
Announce	Bir etmen bir görevi duyurur.
PreBid	Bir etmen, bir görev için geçici teklif verir.
PreAccept	Bir etmen, görev için geçici kabul aldı.
PreReject	Bir etmen, görev için geçici red aldı.
DefinitiveBid	Bir etmen, bir görev için son(kesin) teklifini verir.
DefinitiveAccept	Bir etmen, görev için kesin kabul aldı.
DefinitiveReject	Bir etmen, görev için kesin olarak red aldı.

Yönetici ve adaylar arasındaki müzakere durum geçişleri Şekil 5.2'deki gibidir. Yönetici adaylara bir görev duyurduğunda (durum 1), adaylardan geçici teklif olan PreBid'leri toplar (durum 2). Bu geçici teklifler, adayların mevcut iş listeleri dikkate alınarak hesaplanır. Yönetici, en iyi teklifi veren adaya geçici kabul olan PreAccept gönderirken (durum 3), teklif veren diğer adaylara da geçici red olan PreReject mesajı gönderir (durum 4). Eğer geçici red alan adayların durumu değişirse, kesin red almadıkları sürece, tekrar daha iyi bir teklif sunabilirler (durum 2). Bu durumda adayların durumu PreAccept alma (durum 3) veya PreReject durumunda kalma (durum 4) şeklinde değişebilir. PreAccept alan bir aday kesin teklifini vererek

DefinitiveBid aşamasına geçebilir(durum 5). Bu aşamada yönetici, aldığı kesin teklifi inceler. Sistemdeki etmenler iş birliği içinde olabileceği gibi bireysel de hareket edebilirler. Bu durumda sahtekarlık durumlarını önlemek için algoritmaya bir strateji eklenmiştir. Amaç, ilk teklif aşamasında çok yüksek değer verip diğer adayların çekilmesine neden olan, kesin teklif aşamasında ise ilk teklifinden çok düşük bir teklif verebilecek etmenleri engellemektir. Böyle bir durumda bu etmen DefinitiveReject alır (durum 7). Kesin teklif anlaşma ile de sonuçlanabilir (durum 6). Bu durumda diğer bütün adaylara, yönetici tarafından DefinitiveReject gönderilir (durum 7). Görüşme, seçilen adayın görevi yerine getirmesi ile sonlanır.



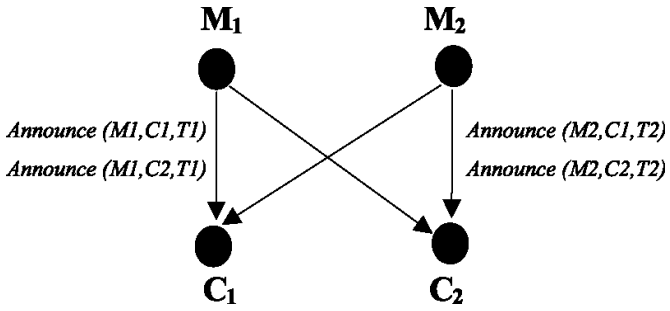
Şekil 5.2 : Müzakere Durum Geçiş Diyagramı

5.5 Protokollerin Problemlere Uygulanması ve Performans Kıyaslaması

Bu bölümde her iki protokol bir problem üzerinde uygulanıp performans kıyaslaması yapılmıştır. Örnekte, elindeki aynı nitelikteki görevleri duyuran M1 ve M2 yöneticileri olsun. Bu görevleri yerine getirebilecek 2 aday C1 ve C2'dir. M1, 30 mns uzunluğundaki T1 görevini, M2 ise 40 mns uzunluğundaki T2 görevini duyurur. Her iki adayın elinde daha önceden anlaştıkları iş listeleri vardır ve bu görevleri yürütmektedirler. C1 20 mns sonra, C2 ise 95 mns sonra serbest kalacaktır.

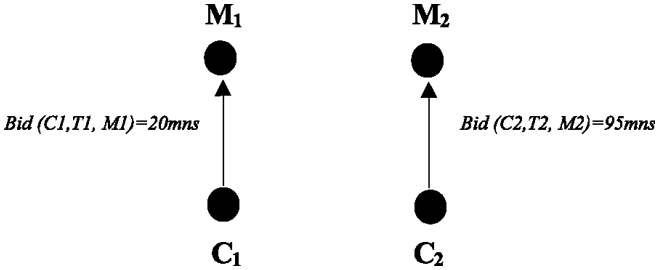
CNP protokolü uygulandığında müzakere şu şekilde ilerler:

- M1 ve M2, T1 ve T2 görevlerini, C1 ve C2 adaylarına duyurur (Şekil 5.3).



Şekil 5.3 : Görev duyuru aşaması.

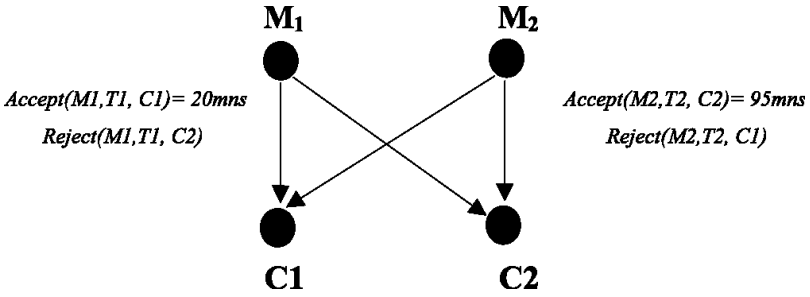
- CNP'de paralel görüşme (ihale) olmadığı için adaylar ihalelere ilgilendikleri sıra ile teklif verirler, cevap aldıktan sonra diğer yöneticiye teklif verirler. C1'in T1 ile, C2'nin de T2 ile başlamak istemesi durumunda C1, T1 görevi için M1'e serbest kalacağı 20 mns'i, C2 de T2 görevi için M2'ye 95 mns'i gönderir. (Şekil 5.4)



Şekil 5.4 : Teklif aşaması.

- Bu aşamada CNP protokolünde C1, M1 ile olan görüşmesini tamamlamadan M2 ye teklif veremediği için bekler. Aynı şekilde C2 de M1'e teklif gönderemez. Dolayısıyla her iki aday, teklif verdikleri yöneticiler ile olan görüşmelerini sıralı yapmak durumundadırlar. M1 ve M2 de sırasıyla henüz teklif iletmeyen C2 ve C1'i beklerler. Bloklanma durumlarının oluşmaması

için belli bir zaman aşımından sonra M1 ve M2, diğer adayın teklif vermeyeceğini düşünerek C1 ve C2 nin tekliflerini kabul ederler ve diğer adaya gönderdiği teklifi iptal ederler. (Şekil 5.5)

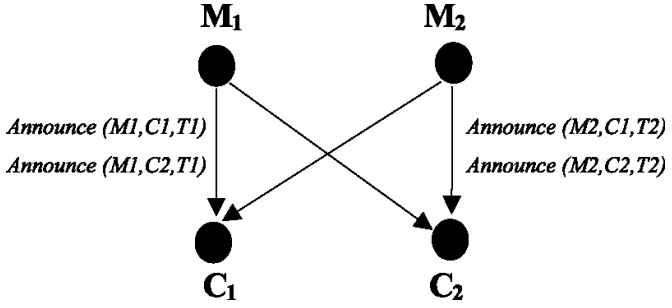


Şekil 5.5 : Atama aşaması.

Ancak görev atama işlemi verimli tamamlanmamıştır çünkü C1, T1 görevini bitirip 50 mns'de serbest kalmasına rağmen, T2 görevine C2 tarafından 95 mns'e kadar başlanmayacak. Oysa C1 her iki görevi sıralı olarak yapsaydı, T2 görevi 90 mns'de tamamlanmış olurdu.

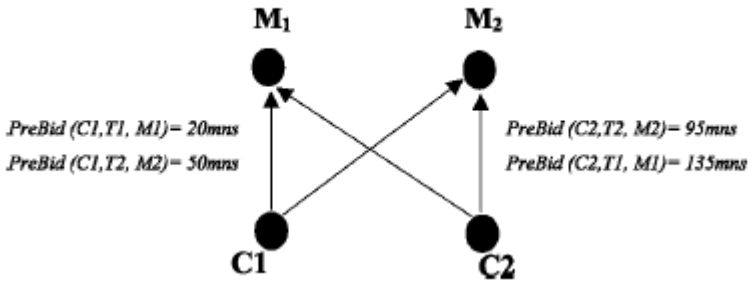
Aynı problem için geliştirilmiş CNP protokolü uygulandığında ise müzakere şu şekilde ilerler:

- M1 ve M2, T1 ve T2 görevlerini, C1 ve C2 adaylarına duyurur (Şekil 5.6).



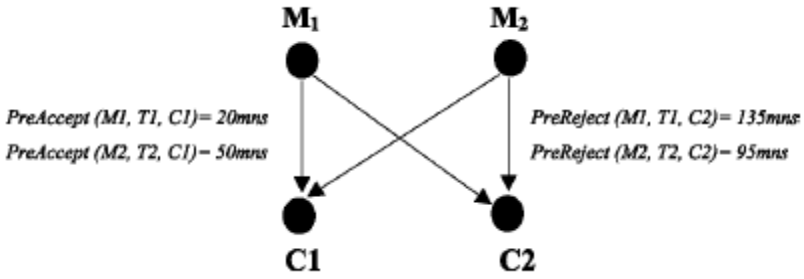
Şekil 5.6 : Görev duyuru aşaması.

- C1 ve C2 elindeki görevleri sıralayıp tahmini tekliflerini verirler. C1, T1 görevi için M1'e serbest kalacağı 20 mns'i gönderir. Aynı zamanda T2 görevi için M2'e 50 mns teklifini gönderir, bu teklifi hazırlarken T1 görevinin yürütülme süresini dikkate almıştır. C2 de T2 görevi için M2'ye 95 mns, T1 görevi için M1'e 135 mns teklifini gönderir. Böylece her iki aday da görüşme sürecini paralelleştirmiş olur(Şekil 5.7).



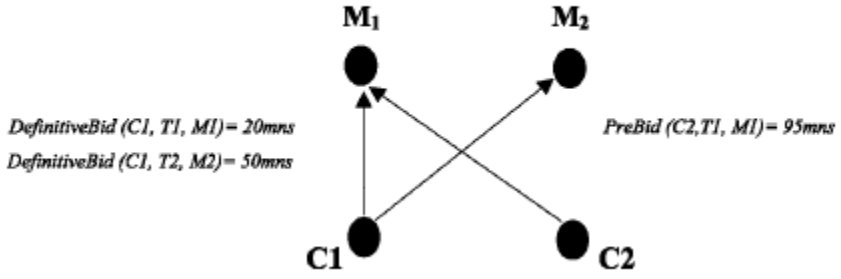
Şekil 5.7 : Geçici teklif aşaması.

- M1 ve M2, T1 ve T2 görevleri için C1 adayına geçici kabul gönderirken C2'ye de geçici red gönderirler (Şekil 5.8)



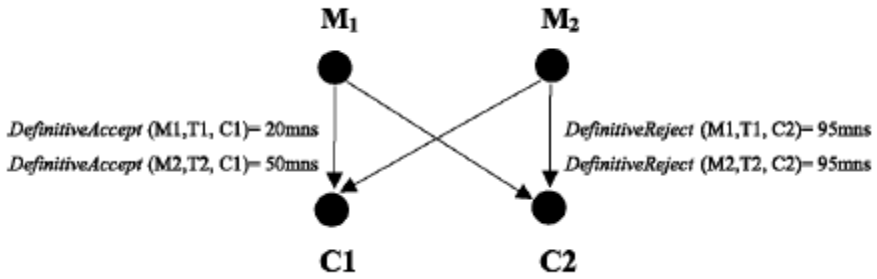
Şekil 5.8 : Geçici cevap aşaması.

- Eğer yönetici red bilgisiyle birlikte aldığı en iyi teklif bilgisini de gönderirse C2 zaten ihaleden çekilir çünkü hiçbir şekilde ne M1'e ne de M2'e daha iyi teklif veremez. Ama bu bilgiyi alamazsa tekliflerini tekrar düzenleyerek bu sefer T2'den önce T1'e teklif vermeyi dener. T1 görevi için M1'e öncekinden daha iyi teklif olan 95 mns'i gönderir. C1 de her iki manager'a DefinitiveBid gönderir (Şekil 5.9)



Şekild 5.9 : Kesin teklif aşaması.

- M1 ve M2, C1'e DefinitiveAccept, C2'e DefinitiveReject gönderir.



Şekil 4.10 : Kesin atama aşaması.

Böylece CNP protokolüyle her iki görevin tamamlanması 135 mns sürerken, yeni protokol ile 90 mns sürer. (45 mns daha verimli).

Verimli görev tahsisi dışında aşağıda belirtilen faydaları da vardır:

- M-N müzakerelere olanak verir. Bir aday eş zamanlı olarak M yönetici ile görüşebilir ve bir yönetici birden fazla müzakereyi N aday ile yürütebilir. Böylece global müzakere süresi azalmış olur.
- Bir aday DefinitiveBid vermediği sürece, değişen duruma göre teklifini yenileyebilir.
- İlk Teklif aşamasının uzunluğu etmenlerin kendileri için en iyi seçimi, DefinitiveBid öncesinde bulmalarını sağlar. Dolayısıyla bu da kabul aldıktan sonra cayma durumlarını azaltır.
- Paralel görüşmeler yöneticilerin ihaleleri yönetmesini kolaylaştırır. Yöneticiler, potansiyel adaydan DefinitiveBid alıp anlaşıncaya kadar diğer adaylara DefinitiveReject vermez ancak böyle bir teklif olduğundan haberdar eder.
- Adayların sözünden cayması durumlarını azaltmak demek, yöneticinin aynı ihaleyi tekrar başlatması ve tüm işlemleri yinelemesi yükünün ortadan kalkması demektir.

5.6 Aknine- Pinson-Shakun Görev Atama Algoritması

Bu bölümde hem yöneticilerin hem de adayların davranışını açıklayan algoritmaların detayı verilmektedir.

Adayların davranışını içeren algoritma adımları aşağıdaki gibidir:

1) C_i bir aday etmen ve T_k anons edilen bir görev olsun. C_i mevcut işleri arasında T_k görevi için uygun sırayı hesaplar, iş listesine ekler ve T_k için PreBid gönderir.

2) C_i aday etmeni, T_k görevi için M_j yöneticisinden PreReject alırsa tekrar sıralama yapar ve bu hesaplama sonucundaki değişimlere göre PreBid ve DefinitiveBid mesajlarını gönderir. M_j yöneticisinden kesin cevap bekler.

3) C_i aday etmeni, M_j yöneticisinden DefinitiveReject alırsa T_k görevini listesinden çıkararak yeni bir sıralama yaparak PreBid ve DefinitiveBid mesajlarını gönderir.

4) C_i aday etmeni, M_j yöneticisinden PreAccept alırsa, T_k görevi için DefinitiveBid hesaplar ve gönderir.

5) C_i aday etmeni, M_j yöneticisinden DefinitiveAccept alırsa, T_k görevini yürütmeye başlayabilir.

6) C_i aday etmeni, yeni bir $T_{k'}$ görevi için anons alırsa, mevcut işleri arasında $T_{k'}$ görevi için uygun sırayı hesaplar, iş listesine ekler ve $T_{k'}$ için PreBid gönderir.

Yöneticilerin davranışını içeren algoritma adımları aşağıdaki gibidir:

1) Elindeki T_k görevini C_i aday etmenlerine duyuran M_j yönetici etmeni, teklifler için bekler.

2) M_j yönetici etmeni, ilk atama aşamasında bir C_i aday etmeninden PreBid alırsa, aldığı teklifleri inceler ve potansiyel adayının teklifi C_i aday etmeninden daha iyiyse C_i aday etmenine PreReject gönderir. Değilse C_i aday etmenini potansiyel aday olarak belirler. Tüm adaylar teklif verdiğinde potansiyel adaya PreAccept gönderir.

3) M_j yönetici etmeni, kesin atama aşamasında bir C_i aday etmeninden PreBid alırsa, aldığı teklifi potansiyel adayın PreBid teklifi ile kıyaslar. Eğer daha iyi değilse PreReject gönderir, daha iyiyse potansiyel adaya PreReject, C_i 'ye PreAccept gönderir.

4) M_j yönetici etmeni, T_k görevi için DefinitiveBid alırsa ve bu teklif potansiyel adayın ilk teklifinden ya da bütün PreBid'ler DefinitiveBid'den kötüyse, M_j yönetici etmeni potansiyel etmene DefinitiveAccept ve diğer bütün adaylara da DefinitiveReject gönderir. Diğer durumlarda M_j, yeni bir potansiyel aday seçer ve PreAccept mesajı gönderir. Eski potansiyel adaya da PreReject gönderir.

Eğer M_j yönetici etmeni, aday etmenlerden birinin çöktüğünden şüphelenirse bu adayın ihale ile ilgilenmediğine karar verir.

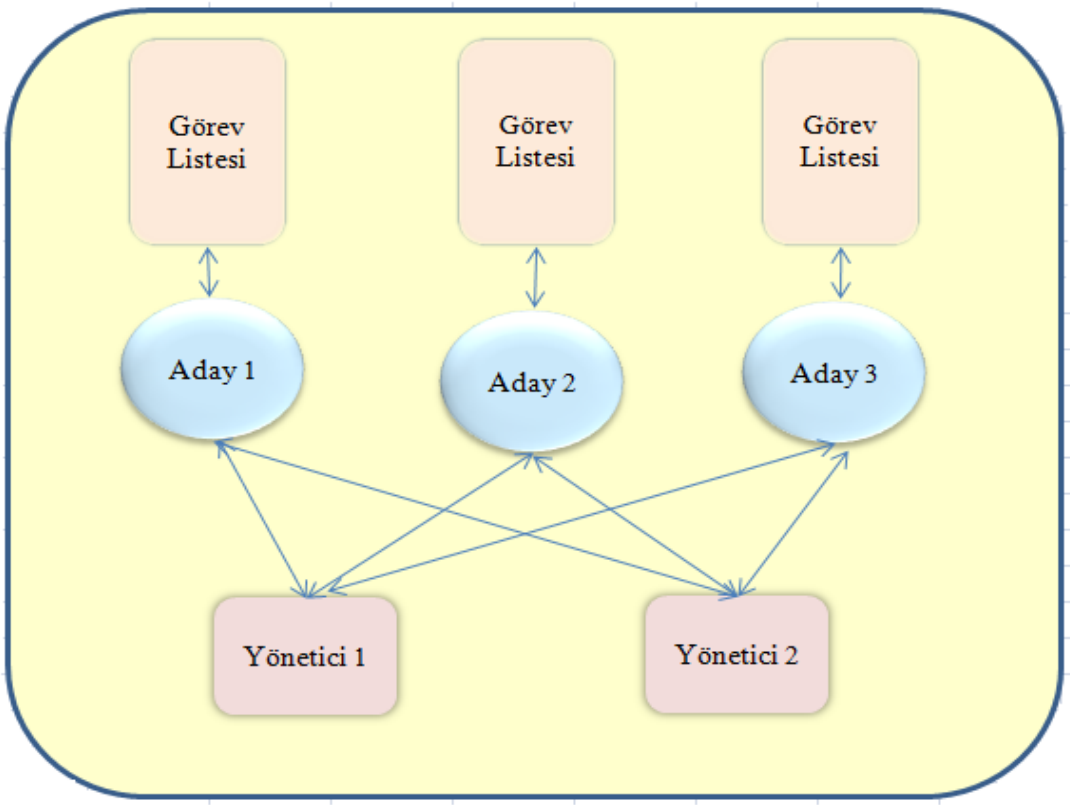
6. SİSTEMİN MİMARİ TASARIM

6.1 Sistemin Mimari Tasarımı

Gerçekleştirilen görev atama ve yürütme ortamının tasarımında 2 çeşit etmen yer almaktadır.

- Yönetici etmenler
- Aday etmenler

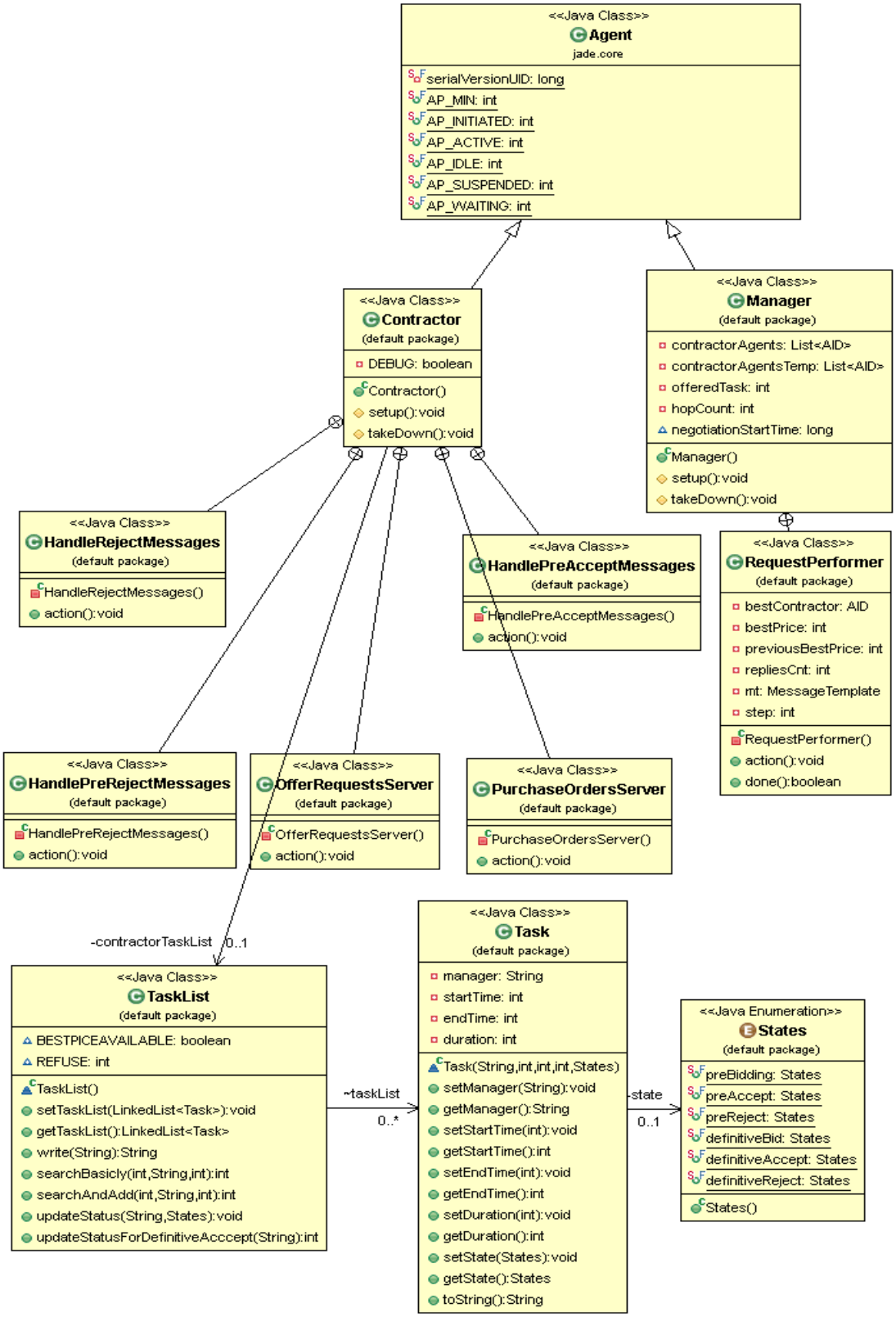
Her iki etmenin görev tanımları ve yürüttükleri algoritma farklıdır. Detayları aşağıda ayrıntılı olarak anlatılmıştır. Uygulamada ihale sistemi modellenmiş ve taşımacılık problemi üzerine uygulanmıştır.



Şekil 6.1 : Etmenler arası etkileşim.

6.1.1 Sınıf tasarımı

Yapılan çalışmada, tüm sınıflar Transport paketi içinde yer alır ve Şekil 6.2'deki gibi tasarlanmıştır.



Şekil 6.2 : Sınıf Tasarım Diyagramı.

Sınıf yapısı aşağıdaki şekilde düzenlenmiştir:

States.java : İhale aşamaları tutulur. Toplam 6 aşama bulunmaktadır. Her bir aşama, ihaledeki görev için verilen teklifin durumunu belirtir, aday etmen bu bilgiye göre tekliflerini düzenler. Anlamları önceki bölümde detaylandırılan aşamalar şöyledir:

- preBidding
- preAccept
- preReject
- definitiveBid
- definitiveAccept
- definitiveReject

Task.java:

Görev modelini tutar. Göreve ait meta dataları içerir. Çizelge 6.1'de detaylandırılmıştır, Ek A1'de sınıf içeriği verilmiştir.

Çizelge 6.1 : Görev bilgileri.

Alan	Açıklama	Örnek
Manager	Yönetici etmen	manager1
startTime	Görev başlangıç zamanı	30
endTime	Görev bitiş zamanı	60
Duration	Görev süresi	30
State	Etmenin durumu	PreAccept

Sınıf için getter ve setter metodlar bulunur. Ayrıca nesne içeriğini göstermek amacıyla toString() override edilmiştir.

TaskList.java:

Görev listesinin tutulduğu sınıftır. İçinde, Task tipinde elemanlar içeren bir bağlı liste bulunur. Her aday etmenin kendisine ait bir görev listesi vardır ve iş planlarını bu listeye göre yapar. Ek A2'de sınıf içeriği verilmiştir.

Bu listeye görev eklenmesini, silinmesini, mevcut bir görev üzerinde değişiklik yapılmasını sağlayan metodları vardır.

- **searchAndAdd** metodu sınıfın en önemli metodudur. İncelenen görevin aday etmene ait iş listesindeki en uygun aralığa yerleştirilmesini sağlar. Algoritması şu şekildedir:
    - Liste tek tek dolaşılır.
    - Mevcut kaydın statüsü red ise onun başlangıcından bir sonraki görevin başlangıcına kadar olan süreye bakılır, önerilen iş görev için yeterliyse güncellenir, yeterli değilse ve bir sonraki görev null ise yine güncellenir.
    - Bir sonraki görev null değil ama red ise onun süresini de ekleyerek sayaç tututulur.
    - Bu şekilde yeterli süre bulunana kadar devam edilir.
    - Bulunursa güncellenir, bulunmazsa red ya da boş aralık aramaya devam edilir.
    - Mevcut kaydın statüsü red değilse, bitişi ile sonraki görevin başlangıcı arasındaki süreye bakılır.
    - En kötü ihtimalle listenin sonuna eklenir.
- **updateStatus** metodu ile liste içindeki bir görevin durumu güncellenir.
- **updateStatusForDefinitiveAcccept** metodu ile kesin kabul alınmış bir görevin durumu gücellenir. Bu işlem sırasında, eğer daha iyi teklif verilebiliyorsa iyileştirme de yapılır.
- **write** metodu ile liste içereği yazdırılır.

Contractor.java:

Aday etmenlerin davranışı gerçeklenmiştir. Sürekli(döngüsel) olarak iş duyurularını dinler ve uygun aksiyonu alır. Bir görev listesine sahiptir. Yaşam döngüsüne başlarken yürüttüğü iş ile ilgili olarak kendisini sarı sayfalara kaydeder. Böylece, aynı nitelikte iş yaptırmak isteyen yöneticiler varlığından haberdar olur. Bir ihale sırasında oluşabilecek ihtiyaçlara cevap verebilmek için aşağıdaki davranışlara sahiptir:

- **OfferRequestsServer** davranışı ile yeni gelen ihale teklifleri karşılanır. Eğer önerilen görev için teklif verilebiliyorsa PROPOSE, verilemiyorsa REFUSE mesajı gönderilir.
- **PurchaseOrdersServer** davranışı ile ACCEPT_PROPOSAL mesajları işlenir. Kesin kabul alınmıştır ve görevin yerine getirileceğini teyit etmek için INFORM mesajı gönderilir.
- **HandlePreRejectMessages** davranışı ile DISCONFIRM mesajları işlenir, PreReject mesajı alınmıştır ve mevcut görevin durumu güncellenir.
- **HandleRejectMessages** davranışı ile REJECT_PROPOSAL mesajları işlenir. Kesin red mesajı alınmıştır ve mevcut görevin durumu güncellenir.
- **HandlePreAcceptMessages** davranışı ile CONFIRM mesajları işlenir, PreAccept mesajı alınmıştır ve mevcut görevin durumu güncellenir.

Ek A3'te sınıf içeriği verilmiştir.

Manager.java:

Yönetici etmenlerin davranışı gerçeklenmiştir. Öncelikle duyurmak istediği görevi yerine getirebilecek nitelikteki adayların listesini sarı sayfalardan alır. RequestPerformer isminde tek bir davranışı vardır ve amacı elindeki iş için en uygun adayı bulmaktır. Aşama aşama gerçekleşen bir algoritmaya sahiptir, bir aşama tamamlandığında diğer aşamaya geçilir. Bunun için bekirli kurallar gerçeklenmelidir.

- **1. aşamada** yönetici adaylara işi duyurur, bunun için her aday etmene CFP mesajları gönderir. 2. aşamaya geçilir.
- **2. aşamada** etmen adaylardan PROPOSE/REFUSE mesajlarını toplar. Bloklanarak sonsuza kadar beklememesi için bir süre aşım zamanı belirlenmiştir. 2 mns bekledikten sonra aldığı teklifler ile bir sonraki 3. aşamaya devam eder.
- **3. aşamada** alınan bütün teklifler incelenerek içinden en iyisi seçilir, bu teklif en iyi teklif olarak, teklifi veren aday etmen de potansiyel aday olarak belirlenir. Potansiyel adaya CONFIRM mesajı ile PreAccept, diğer tüm adaylara da DISCONFIRM mesajı ile PreReject cevabı gönderilir, tekrar 1. aşamaya dönülür. 1. ve 2. aşama tekrarlanır. Daha iyi teklif aldıkça bu işlem devam eder, eğer bir öncekinden daha iyi teklif verilememiş ise 4. aşamaya geçilir.

- **4. aşamada** en iyi teklifi veren potansiyel adaya ACCEPT_PROPOSAL ile kesin kabul (definitiveAccept) cevabı, diğer etmen adaylara da REJECT_PROPOSAL ile kesin red (definitiveReject) mesajı gönderilir. 5. aşamaya geçilir.
- **5. aşamada** kesin kabul gönderilen aday etmenden, görevi yerine getireceğine dair teyit olan INFORM mesajı beklenir, alındığında 6. aşamaya geçilir. Bloklanarak sonsuza kadar beklememesi için bir süre aşım zamanı belirlenmiştir. 2 mns bekledikten sonra hala cevap gelmemiş ise 1. aşamaya dönerek ihaleyi tekrar başlatır.

Ek A4'te sınıf içeriği verilmiştir.

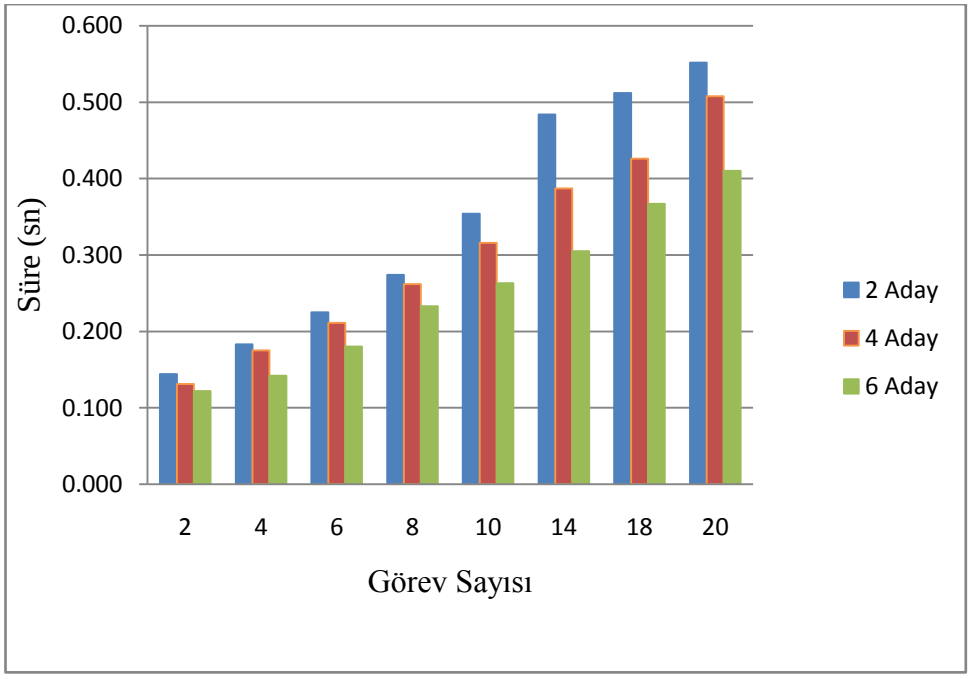
7. BAŞARIM DEĞERLENDİRMESİ

İhale sistemi simüle edilerek görev paylaşımına olanak sağlayan çok etmenli bir yürütme ortamı hazırlanmış ve çeşitli testler yapılmıştır. Testler 2 GHz Genuine Intel(R) işlemcili, 2 GB belleğe sahip, 32-bit Windows 7 Professional işletim sistemi kurulu kişisel bilgisayarda yapılmıştır. JADE 4.3.0 [23] sürümü kullanılarak geliştirilen etmenler üzerinde testler gerçekleştirilmiştir.

Testlerde aday etmen sayısı, görev sayısı ve en iyi teklifin iletilme durumuna göre ölçümler yapılarak sonuçlar incelenmiştir. Deneyler 100'er kez tekrarlanmış ve elde edilen sürelerin saniye cinsinden ortalamaları sunulmuştur. Performans ihale süresi ve ihalenin tamamlanma adımı olmak üzere 2 açıdan değerlendirilmiştir.

7.1 Görev ve Aday Etmen Sayısının İhale Süresine Etkisi

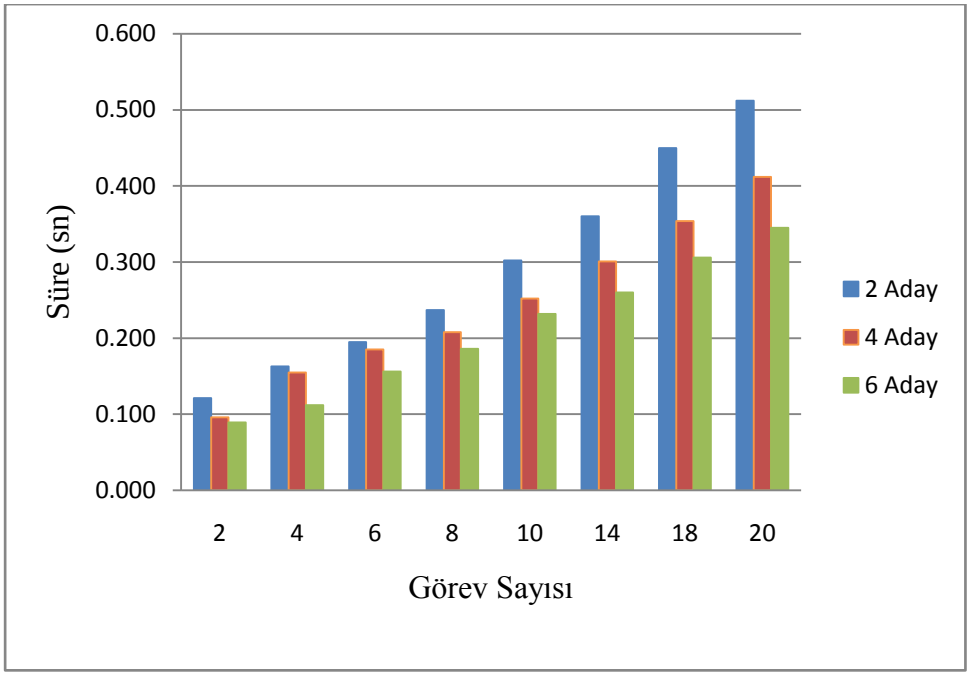
Şekil 7.1'de görev sayısındaki artışın ihale tamamlanma süresindeki etkisi gösterilmektedir.



Şekil 7.1 : Farklı görev ve aday etmen sayılarının en iyi teklif iletilmediğinde ihalenin tamamlanma süresine etkisi.

Bu testlerde, yönetici etmen, aday etmenlere elindeki en iyi teklifi iletmez. Aday etmen sayısının 2, 4 ve 6 olarak değiştiği üç farklı sistemde görev sayısı artırılarak testler gerçekleştirilmiştir. Görev ve aday etmen sayısındaki artış ihale süresinin de uzamasına neden olmaktadır. Ancak aynı görev sayısında, aday etmen sayısı arttıkça müzakereler daha kısa sürede sonuçlanmaktadır. Bu süre farkı, etmen sayısının arttığı karmaşık sistemler için daha belirgindir.

Şekil 7.2'de aynı test tekrarlanmıştır ancak bu kez yönetici etmen, aday etmenlere elindeki en iyi teklifi iletir ve bu bilgiye göre daha iyi bir teklif vermelerine ya da ihaleden çekilerek başka görevlerle devam etmelerine olanak sağlar.



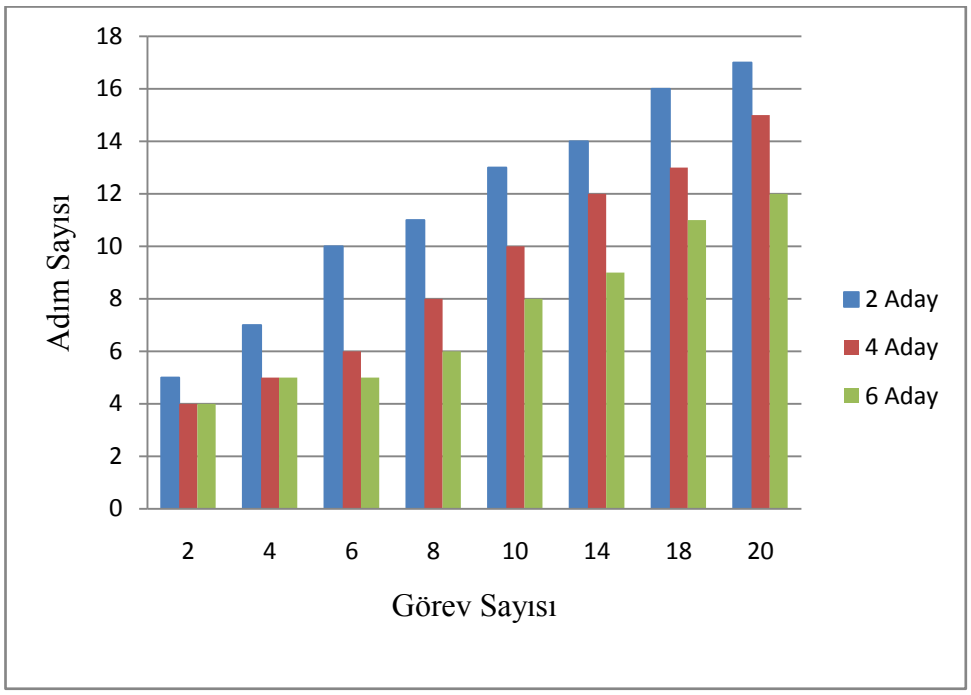
Şekil 7.2 : Farklı görev ve aday etmen sayılarının en iyi teklif iletildiğinde ihalenin tamamlanma süresine etkisi.

Görev ve aday etmen sayısındaki artış yine ihale süresinin uzamasına neden olmaktadır. Ancak en iyi teklif bilgisinin iletilmediği durum ile kıyaslanırsa, özellikle toplam etmen sayısının arttığı karmaşık sistemler için ihalelerin daha kısa sürede sonuçlanmasını sağlar. Çünkü aday etmenler olumlu cevap alamayacağı ihalelerden cevap bekleyip vakit kaybetmek yerine iş listelerini tekrar düzenleyerek diğer adaylar için daha iyi teklif hazırlarlar. Bu deney sonucunda da aynı görev sayısı için aday etmen sayısı arttıkça müzakerelerin daha kısa sürede sonuçlandığı gözlemlenmiştir. Bu süre farkı, etmen sayısının arttığı karmaşık sistemler için daha belirgindir. Örneğin 2 görevin duyurulduğu bir sistemde 2 ve 6 aday etmenin dahil

olduğu ihalelerin tamamlanma süreleri arasında 0,032 sn varken, 20 görevin duyurulduğu bir sistem için ihalelerin tamamlanma süreleri arasında 0,167 sn bulunmaktadır.

7.2 Görev ve Aday Etmen Sayısının İhale Tamamlanma Adımına Etkisi

Şekil 7.3'de ise görev ve aday etmen sayısındaki artışın ihalenin tamamlanma adımına etkisi gösterilmektedir.

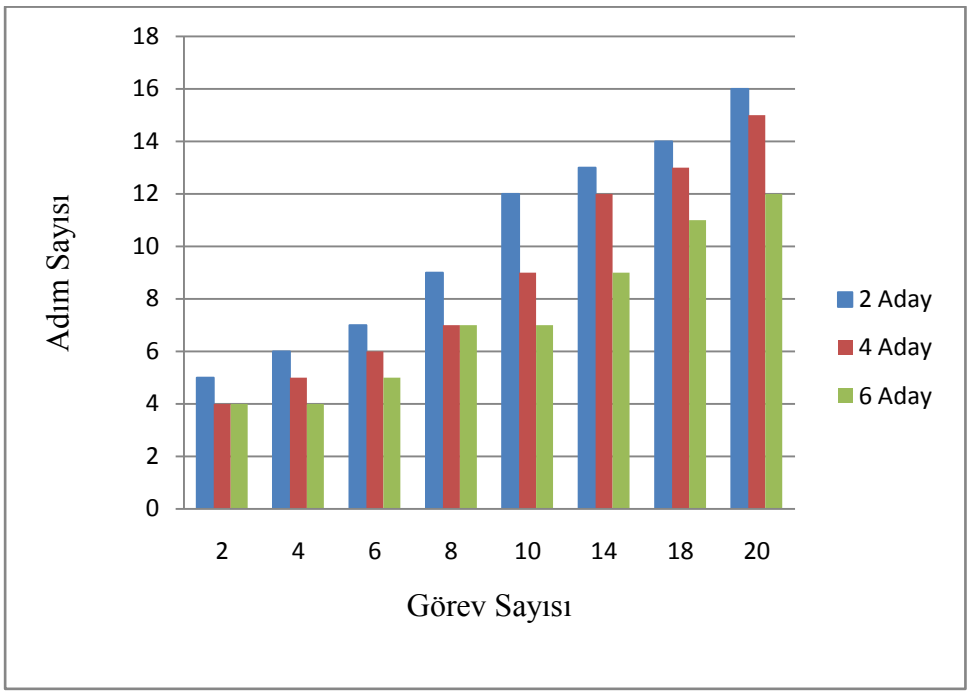


Şekil 7.3 : Farklı görev ve aday etmen sayılarının en iyi teklif iletilmediğinde ihalenin tamamlanma adımına etkisi.

Bu testlerde, yönetici etmen, aday etmenlere elindeki en iyi teklifi iletmez. Aday etmen sayısının 2, 4 ve 6 olarak değiştiği üç farklı sistemde görev sayısı artırılarak testler gerçekleştirilmiştir. Görev ve aday etmen sayısındaki artış ihale tamamlanma adımlarında hızlı bir artışa neden olmaktadır.

Şekil 7.4'de aynı test tekrarlanmıştır ancak bu kez yönetici etmen, aday etmenlere elindeki en iyi teklifi iletir. Görev ve aday etmen sayısındaki artış ihale tamamlanma adımlarında artışa neden olmaktadır, ancak en iyi teklifin iletilmediği duruma göre daha iyi sonuçlar elde edilmiştir. Ayrıca görev sayısı arttıkça, ihalelerin tamamlanma adımları arasındaki fark da artmaktadır. Örneğin 2 görevin duyurulduğu bir sistemde 2 ve 6 aday etmenin dahil olduğu ihalelerin tamamlanma adımları arasında 1 adım

fark varken, 20 görevin duyurulduğu bir sistem için ihalelerin tamamlanma adımları arasında 5 adım fark bulunmaktadır.



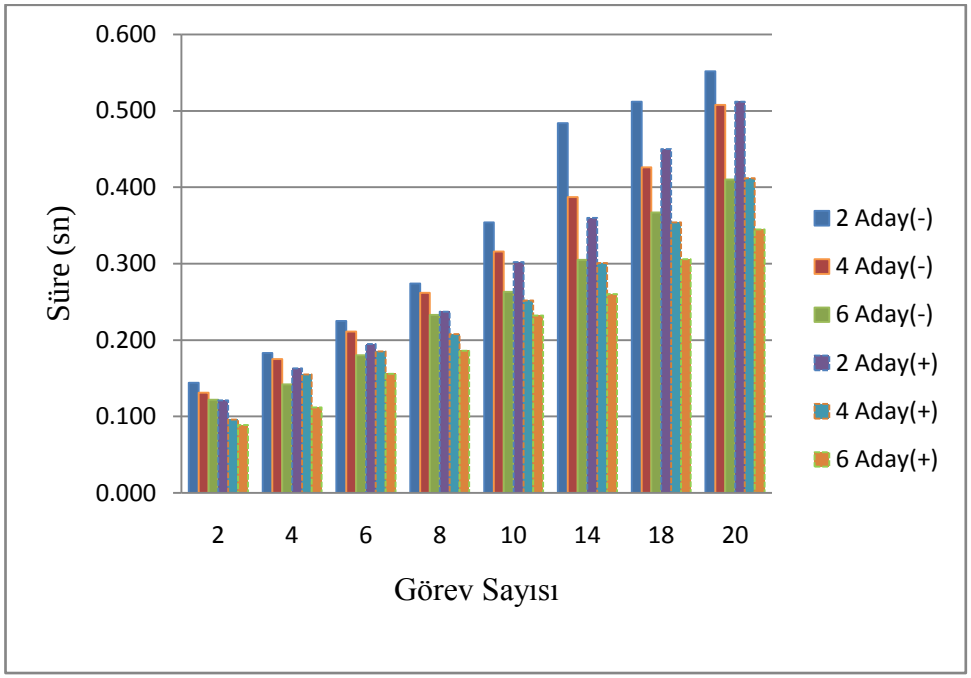
Şekil 7.4 : Farklı görev ve aday etmen sayılarının en iyi teklif iletildiğinde ihalenin tamamlanma adımına etkisi.

7.3 En İyi Teklifin İletilmesinin İhale Tamamlanma Süresine ve Adımına Etkisi

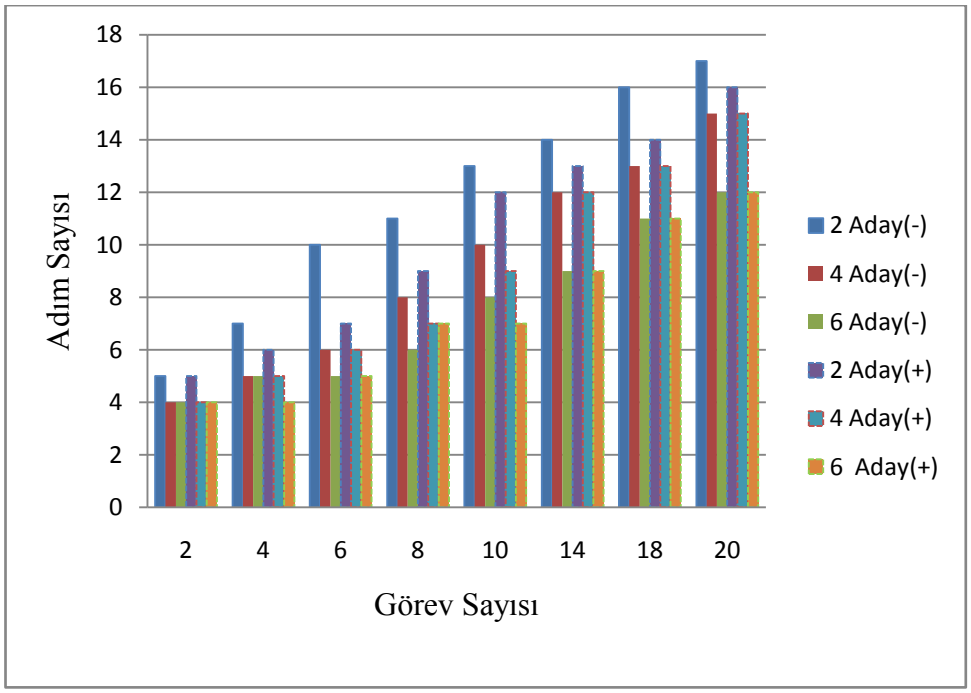
Şekil 7.5'de görev ve aday etmen sayısındaki artışın ihalenin tamamlanma süresine etkisi, yönetici etmenin en iyi teklifi ilettiği ve iletmediği durumlar için ortak bir grafikte gösterilmektedir. Aday etmen sayısının yanındaki (-) işareti, en iyi teklifin iletilmediğini, (+) işareti ise bu bilginin iletildiği durumu göstermektedir. Şekil 6.5'te 2 görev duyurulduğunda, en iyi teklif bilgisinin iletildiği ve iletilmediği durumlar için ihalenin tamamlanma süreleri arasındaki fark çok az olmakla birlikte 20 görevin duyurulduğu ihale için bu süreler arasındaki fark 0,1 sn'ye kadar çıkmaktadır. Özellikle aday etmen sayısı arttıkça, görevi yerine getirebilecek daha çok aday olduğu için daha iyi teklifler verilebildiği ve bu durumda da ihale sürelerinin kısaldığı gözlemlenmiştir.

Şekil 7.6'da görev ve aday etmen sayısındaki artışın ihalenin tamamlanma adımına etkisi, yönetici etmenin en iyi teklifi ilettiği ve iletmediği durumlar için ortak bir grafikte gösterilmektedir. Aday etmen sayısının yanındaki (-) işareti, en iyi teklifin iletilmediğini, (+) işareti ise bu bilginin iletildiği durumu göstermektedir. İhale süresi

ile ilgili sonuçlara benzer şekilde, en iyi teklif bilgisinin iletildiği ve iletilmediği durumlarda, etmen sayısının arttığı durumlar için ihalenin tamamlanma adımı arasındaki fark atmaktadır.



Şekil 7.5 : Farklı görev ve aday etmen sayılarının en iyi teklif iletilme durumuna göre ihalenin tamamlanma süresine etkisi.



Şekil 7.6 : Farklı görev ve aday etmen sayılarının en iyi teklif iletilme durumuna göre ihalenin tamamlanma adımına etkisi.

8. SONUÇ VE ÖNERİLER

Bu çalışmada iş yapma yeteneği olan otonom etmenlere görev atanması sağlanmıştır. İhale sistemi gerçeklenmiş ve bu aşamada yönetici ve aday etmenler için gerçeklenen algoritmalar ile etkili bir şekilde görev paylaşımı gerçekleşmiştir. Yapılan çalışmada tüm bu işlemlerin yapılabilmesi için etmen tabanlı bir yürütme ortamı hazırlanmış ve testler bu ortamda gerçekleştirilmiştir.

Bu çalışma ile görevlerin duyurulması ve tekliflerin alınması, ardından en hızlı ve etkili şekilde etmenlere görev atanması sağlanmıştır. Yönetici ve aday sayıları değiştirilerek çeşitli deneyler yapılmış ve sonuçları kıyaslanmıştır. Başarı kriteri olarak sonuca ulaşmak için atlatılan adım sayısı ve müzakere sırasında geçen süre baz alınmıştır. Görev ve aday etmen sayısındaki artış müzakere süresinin uzamasına neden olmaktadır. Ancak en iyi teklif bilgisinin iletildiği durumlarda, iletilmediği durumlara göre özellikle toplam etmen sayısının arttığı karmaşık sistemler için müzakerelerin daha kısa sürede sonuçlandığı görülmüştür. Daha yüksek etmen/görev sayılarında fark daha da belirgin olabilir.

Bu çalışmanın devamında görevlerin öncelik sırasına göre çalıştırılması sağlanabilir. Maliyet bilgisi eklenebilir. Etmenler arasında iş birliği kurularak görevleri beraber yerine getirmeleri sağlanabilir.

KAYNAKLAR

[1] **R. G. Smith** (1980). ‘‘The Contract net protocol: High-level communication and control in a distributed problem- solver,’’ IEEE Transactions on Computers, vol. 12.

[2] **R. G. Smith and R. Davis** (1981). ‘‘Frameworks for co-operation in distributed problem solving,’’ IEEE Transaction on System, Man and Cybernetics, vo. 11, no. 1.

[3] **S. Aknine, S. Pinson, and M. F. Shakun** (2004). ‘‘An Extended Multi-Agent Negotiation Protocol’, Autonomous Agents and Multi-Agent Systems, 8, 5 – 45.

[4] **S. Aknine** (2002). ‘‘Strategies and behaviors of agents in multi-phased negotiations,’’ Third International Confer- ence on Electronic Commerce and Web Technologies, EC-WEB with DEXA, LNCS, Springer Verlag, Sep- tember 2 – 6, Aix-en-Provence, France.

[5] **T. Bouron** (1992). ‘‘Structures de Communication et d’Organisation pour la Coope´ration dans un Univers Multi- agent,’’ Ph.D. Thesis, Universite´ Paris 6.

[6] **N. R. Jennings** (1995). “Controlling cooperative problem solving in industrial multi-agent systems using joint intentions,” Artificial Intelligence, vol. 75, no. 2.

[7] **El-Fallah Seghrouchni, A. et Haddad** (1996). ‘‘A coordination algorithm for multi-agent systems’’, Agents Break- ing Away, European Workshop on Modelling Autonomous Agents in a Multi-agent World, Maamaw’96, The Netherlands.

[8] **O. Shehory and S. Kraus** (1998). ‘‘Methods for task allocation via coalition formation,’’ Artificial Intelligence, Elsevier Science.

[9] **K. Fischer, J. P. Muller, and M. Pischel** (1996). ‘‘Scheduling an application domain for DAI,’’ Applied Artificial Intelligence, An International Journal, vol. 10, pp. 1 – 33.

[10] **T. Sandholm, S. Sikka, and S. Norden** (1999). ‘‘Algorithms for optimizing leveled commitment contracts,’’ Interna- tional Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden, pp. 535 – 540.

[11] **S. Kaur, H. Kaur, S. K. Sehra** (2013). “Modification of Contract Net Protocol(CNP) : A Rule-Updation Approach”, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 4, No. 11.

[12] **A. Singh, D. Juneja** (2013). “An Improved Design of Contract Net Trust Establishment Protocol”, ACEEE Int. J. on Communications, Vol. 4, No. 1.

[13] **Sonmez, D.** (2014), Akademik Bilisim Konferansi, JADE, JADEX, RETSINA, DECAF Etmen Geliştirim Platformlarının Karşılaştırılması.

[14] **Alaybeyoğlu A., Erdur R.** (2006), "Yazılım Etmenleri: Açık, Dinamik ve Heterojen Ortamlarda Yazılım Geliştirme İçin Bir Teknoloji", Akademik Bilişim 2006, Denizli, s. 468-471.

[15] **Alaybeyoğlu A, Kardaş G, Erdur R, Dikenelli O.** (2007). "SABPO Metodolojisi Kullanılarak FIPA Uyumlu Çok-Etmenli Bir Otel Rezervasyon Sisteminin Tasarımı ve Gerçekleştirilmesi", Akademik Bilişim 2007, Kütahya.

[16] **Ergun. S, Aydoğan, T** (2013). "JADE Etmen Çerçevesinde Çok Etmenli Bir Ders Yönetim Sisteminin SABRO Metodolojisi Kullanılarak Geliştirilmesi."Journal of Natural & Applied Sciences. 2013, Vol. 17 Issue 3, p51-55. 5p.

[17] **Weiss, G. ve Wooldridge, M.** (1999). Multiagent Systems A Modern Approach to Distributed Artificial Intelligence, The MIT Press, Cambridge, Massachusetts.

[18] **Luck, M., McBurney, P. ve Preist, C.** (2003). Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing). AgentLink. ISBN 0854 327886.

[18] **Erdur, R** (2001). Application of Software Agent Tecnology to Internet Based Software Reuse, Ph.D Thesis, Ege University

[20] **Wooldridge, M.** (1999). Intelligent agents, 27-77, Multiagent Systems:A Modern Approach to Distributed Artificial Intelligence, Weiss, G. (Ed.), The MIT Press, London, England, 619p.

[21] **Durfee E. H., Lesser V. R., Corkill D. D.** (1989). Trends in cooperative distributed problem solving, IEEE Transactions on Knowledge and Data Engineering, KDE-1 sf 63 – 83.

[22] **Wooldridge, M.** (1997). Agent-Based Software Engineering. IEE Proceedings on Software Engineering 26-37.

[23] **JADE Ana Sayfa** <http://jade.tilab.com>, alındığı tarih: 08.12.2012.

[24] **Bellifemine, F., Caire, G., Trucco, T. ve Rimassa, G.** (2010). JADE Programmer's Guide, Telecom Italia Lab.

[25] **Bellifemine, F., Caire, G., Poggi, A. ve Rimassa, G.** (2003). JADE A White Paper, Telecom Italia EXP magazine, Vol. 3, No.3, 6-19.

[26] **Caire, G.** (2009). JADE Programming for Beginners, Telecom Italia Lab.

[27] **SC0001L.** (2002). FIPA Abstract Architecture Specification, Geneva, Switzerland.

[28] **SC00061G.** (2002). FIPA ACL Message Structure Specification, Geneva, Switzerland.

[29] **SC00037J.** (2002). FIPA Communicative Act Library Specification Geneva, Switzerland.

EKLER

EK A: Sınıf Tasarımı Dosyaları
- **EK A1:** Task.java
- **EK A2:** TaskList.java
- **EK A3:** Contractor.java
- **EK A4:** Manager.java

EK A

EK A1: Task.java

```
public class Task {

	private String manager; //Manager ID
	private int startTime; //Task start time
	private int endTime; //Task end time
	private int duration; //Task duration
	private States state; //Task status

	Task(String manager, int startTime, int endTime, int duration, States state) {
		this.manager = manager;
		this.startTime = startTime;
		this.endTime = endTime;
		this.duration = duration;
		this.state = state;
	}

	//Attribute'lara ait get ve set metotlari
	public void setManager (String manager) { this.manager = manager; }
	public String getManager() { return this.manager; }

	public void setStartTime (int startTime) { this.startTime= startTime; }
	public int getStartTime() { return this.startTime; }

	public void setEndTime (int endTime) { this.endTime = endTime; }
	public int getEndTime() { return this.endTime; }

	public void setDuration(int duration) { this.duration = duration; }
	public int getDuration() { return this.duration; }

	public void setState(States state) { this.state = state; }
	public States getState() { return this.state; }

	//Nesne icerigini gostermek amaciyla superclass'a ait toString() metodu
override ediliyor.
	public String toString() {

		return "Manager: " + this.manager + " Task Baslangic: " +
this.startTime +
			" Task Bitis: " + this.endTime + " Task Suresi: " +
this.duration +
			" Durum :" + this.state;
	}
}
```

EK A2: TaskList.java

```
import java.util.LinkedList;

public class TaskList {

	LinkedList<Task> taskList = new LinkedList<Task>();
	boolean BESTPICEAVAILABLE = false;
	int REFUSE = -1;

	TaskList() {
	}

	//Attribute set ve get metotlari
	public void setTaskList(LinkedList<Task> tasklist) {
		this.taskList = tasklist;
	}
	public LinkedList<Task> getTaskList() {
		return this.taskList;
	}

	// Nesne içeriğini gösterir
	public String write(String owner) {
		int i;
		Task task = null;
		StringBuffer sb = new StringBuffer();
		int taskListSize = this.taskList.size();
		if (taskListSize > 0) {
			System.out.println(owner);

			for (i=0; i < this.taskList.size() ; i++) {
				task = taskList.get(i);
				sb.append("\n Index : "+ i);
				sb.append(task.toString());
				sb.append("\n
**********************************************************************
********************** \n");
			}
			sb.append("\n\n");
		}
		return sb.toString();
	}


	public int searchBasicly(int bestPrice, String manager, int offeredTask) {
		int firstAvailableTime = 0;
		// Liste boşsa ekle
		if (taskList.isEmpty()) {
			Task task = new Task(manager, 0, offeredTask, offeredTask,
					States.preBidding);
```

```
                        taskList.add(task);
                        return firstAvailableTime;
                } else {
                        int taskListSize = taskList.size();
                        boolean found = false;
                        for (int i = 0; i < taskListSize; i++) {
                                Task task = taskList.get(i);
                                States state = task.getState();
                                int startTime = task.getStartTime();
                                int endTime = task.getEndTime();
                                Task nextTask = null;
                                int availableInterval = 0;
                                int firstAvailableInterval = 0;
                                int counter = 0;
                                if (!(state.equals(States.preReject) ||
state.equals(States.definitiveReject))) {
                                        // listede tek eleman varsa, sonuna ekle.
                                        if (i + 1 == taskListSize) {
                                                Task task1 = new Task(manager,
endTime, endTime+ offeredTask, offeredTask, States.preBidding);
                                                taskList.add(task1);
                                                return endTime; // önceki taskın bitişi.
                                        } else {
                                                // Bir sonraki task ile arasında yeterli
boşluk varsa araya ekle
                                                nextTask = taskList.get(i + 1);
                                                States nextState = nextTask.getState();
                                                int nextStartTime =
nextTask.getStartTime();
                                                firstAvailableInterval = nextStartTime -
endTime;
                                                if (firstAvailableInterval >=
offeredTask) {
                                                        Task task1 = new Task(manager,
endTime, endTime + offeredTask, offeredTask, States.preBidding);
                                                        taskList.add(i + 1, task1);
                                                        return endTime;

                                                } else { // Yoksa ve sonraki task Reject
değilse döngüye devam et
                                                        if
(!(nextState.equals(States.preReject) || nextState.equals(States.definitiveReject))) {
                                                                firstAvailableInterval = 0;
                                                                continue;
                                                        } else { // sonraki task Reject ise
availableTime ı set edip gönder

        nextTask.setManager(manager);

        nextTask.setState(States.preBidding);
```

```
                                                    return
nextTask.getStartTime();
                                                }
                                            }
                                        }
                                    } else {
                                        task.setManager(manager);
                                        task.setState(States.preBidding);
                                        return task.getStartTime();
                                    }
                                }
                                // Clean the multiple instances
                                boolean shouldDelete = false;
                                for (int j = 0; j < taskListSize; j++) {
                            Task task1 = taskList.get(j);
                            String managerInfo = task1.getManager();
                                    if (!shouldDelete &&
managerInfo.equalsIgnoreCase(manager)) {
                                        shouldDelete = true;
                                continue;
                            } else if (shouldDelete && managerInfo.equalsIgnoreCase(manager))
{
                                taskList.remove(j);
                                break;
                            }
                                }
                                return firstAvailableTime;
                            }
                    }


                    public int searchAndAdd(int bestPrice, String manager, int offeredTask) {
                            int firstAvailableTime = 0;
                            // Liste boşsa ekle
                            if (taskList.isEmpty()) {
                                Task task = new Task(manager, 0, offeredTask, offeredTask,
States.preBidding);
                                taskList.add(task);
                                return firstAvailableTime;
                            }
//                          Listeyi tek tek dolaş,
//                          Statüsü Reject ise onun başlangıcından next node un başlangıcına
kadarki süreye bak
//                          yetiyorsa güncelle, yetmiyorsa next node null ise yine güncelle
//                          next node null değil ama Reject ise onun süresini de ekle, sayac tut
ve 2 yap
//                          Bu şekilde süre yetene kadar devam et,
//                          Bulursan güncelle, bulamazsan reject ya da boşluk aramaya devam
et
```

```
//                  Statüsü reject değilse, bitişi ile sonraki node arasındaki süreye bak
(statüsü rejectse sonrakine bak...)
//                  En kötü en sona ekle
                else {
                        int taskListSize = taskList.size();
                        boolean found = false;
                        for (int i = 0; i < taskListSize; i++) {
                Task task = taskList.get(i);
                States state = task.getState();
                int startTime = task.getStartTime();
                int endTime = task.getEndTime();
                String managerInfo = task.getManager();
                Task nextTask = null;
                int availableInterval = 0;
                int firstAvailableInterval = 0;
                int counter = 0;
                if (!(state.equals(States.preReject) ||
state.equals(States.definitiveReject))){
                        // listede tek eleman varsa, sonuna ekle.
                        if (i+1 == taskListSize) {
                                if (BESTPICEAVAILABLE && bestPrice > 0 &&
endTime >= bestPrice ) {
                                        return REFUSE;
                                }
                                Task task1 = new Task(manager, endTime,
endTime+offeredTask, offeredTask, States.preBidding);
                                taskList.add(task1);
                                return endTime; // önceki taskın bitişi.
                        }
                        else {
                                // Bir sonraki task ile arasında yeterli boşluk varsa
araya ekle
                                nextTask = taskList.get(i+1);
                                States nextState = nextTask.getState();
                                int nextStartTime = nextTask.getStartTime();
                                firstAvailableInterval = nextStartTime - endTime;
                                if (firstAvailableInterval >= offeredTask) {
                                        if (BESTPICEAVAILABLE && bestPrice > 0
&& endTime >= bestPrice) {
                                                return REFUSE;
                                        }
                                        Task task1 = new Task(manager, endTime,
endTime+offeredTask, offeredTask, States.preBidding);
                                        taskList.add(i+1, task1);
                                        return endTime;

                                }else { // Yoksa ve sonraki task Reject değilse döngüye
devam et
```

```
                                        if (!((nextState.equals(States.preReject) ||
nextState.equals(States.definitiveReject) &&
!managerInfo.equalsIgnoreCase(manager)))){
                                                firstAvailableInterval = 0;
                                                continue;
                                        } else { // sonraki task Reject ise availableTime
ı set edip gönder
                                                firstAvailableInterval = nextStartTime -
endTime;
                                                continue;
                                        }

                                }
                        }
                }
                if ((state.equals(States.preReject) ||
state.equals(States.definitiveReject) && !managerInfo.equalsIgnoreCase(manager))
){
                        int taskDuration = task.getDuration();
                        if (i+1 == taskListSize) {
                                availableInterval = taskDuration +
firstAvailableInterval;
                        } else {
                                nextTask = taskList.get(i+1);
                                availableInterval = firstAvailableInterval +
nextTask.getStartTime() - task.getStartTime();
                        }
                        if (availableInterval >= offeredTask) {
                                // mevcutun değerlerini güncelle
                                if (BESTPICEAVAILABLE && bestPrice > 0 &&
startTime-firstAvailableInterval >= bestPrice) {
                                        return REFUSE;
                                }
                                task.setStartTime(startTime-firstAvailableInterval);
                                task.setEndTime(startTime-
firstAvailableInterval+offeredTask);
                                task.setDuration(offeredTask);
                                task.setManager(manager);
                                task.setState(States.preBidding);
                                firstAvailableTime = startTime-firstAvailableInterval;
                                found = true;
                                break;
                        }
                        else {
                                // while ile nodelarda reject bulana kadar interval ara...
                                if (i+1 == taskListSize) {
                                        // next node is null, current node is the last one.
                                        // Thus update the current node (i)
```

```
                                        // Kısaca : mevcutun değerlerini güncelle sadece
(üsttekiyle aynı) (i)
                                        if (BESTPICEAVAILABLE && bestPrice > 0
&& startTime-firstAvailableInterval >= bestPrice) {
                                                return REFUSE;
                                        }
                                        task.setStartTime(startTime-
firstAvailableInterval);
                                        task.setEndTime(startTime+offeredTask);
                                        task.setEndTime(startTime-
firstAvailableInterval+offeredTask);
                                        task.setDuration(offeredTask);
                                        task.setManager(manager);
                                        task.setState(States.preBidding);
                                        firstAvailableTime = startTime-
firstAvailableInterval;
                                        found = true;
                                        break;
                                } else {
                                        // i den sonraki node dan başlayıp ard arda
Reject statülü kayıtların sayısını bul
                                        // örn listede 3 tane reject kayıt olsun
                                        // i = 0, counter = 2 olur. Yani ilk node dan
sonra 2 tane daha available node var.
                                        for (int j = 0; j < taskListSize-i-1; j++) {
                                                nextTask = taskList.get(i+j+1);
                                                States nextState = nextTask.getState();
                                                if (nextState.equals(States.preReject) ||
nextState.equals(States.definitiveReject)) {
                                                        counter += 1;
                                                }
                                                else {
                                                        break;
                                                }
                                        }

                                        if (counter > 0) {
                                                if (counter+i+1 == taskListSize) {
                                                        // i den sonraki bütün elemanlar
reject, ilkini kullan(i)
                                                        // gerisini sil(i den sonra counter
kadar, 0 dan sonra 2 tane, 1 ve 2 indexlerini remove et)
                                                        if (BESTPICEAVAILABLE &&
bestPrice > 0 && startTime-firstAvailableInterval >= bestPrice) {
                                                return REFUSE;
                                        }
                                                        task.setStartTime(startTime-
firstAvailableInterval);
                                                        task.setEndTime(startTime-
firstAvailableInterval+offeredTask);
```

```
                                        task.setDuration(offeredTask);
                                        task.setManager(manager);
                                        task.setState(States.preBidding);

//                                      for (int j = 1; j <= counter; j++ ) {
//                                              taskList.remove(i+j);
//                                      }

                                        for (int j = counter+i; j > i; j-- ) {
                                                taskList.remove(j);
                                        }

                                        firstAvailableTime = startTime-
firstAvailableInterval;
                                                        found = true;
                                                        break;
                                                } else {
                                                        nextTask =
taskList.get(i+counter+1);
                                                        int nextTaskStartTime =
nextTask.getStartTime();
                                                        availableInterval =
firstAvailableInterval + nextTaskStartTime - startTime;

                                                        if (availableInterval >=
offeredTask) {
                                                                // uygun aralık bulundu,
yeteri kadarını kullan, fazlasını sil (counter a göre ayarla) // yukarıdaki ile aynı

                                                                if
(BESTPICEAVAILABLE && bestPrice > 0 && startTime-firstAvailableInterval >=
bestPrice) {
                                                                        return REFUSE;
                                                                }

        task.setStartTime(startTime-firstAvailableInterval);

        task.setEndTime(startTime-firstAvailableInterval+offeredTask);
                                                task.setDuration(offeredTask);
                                                task.setManager(manager);
                                                task.setState(States.preBidding);

    for (int j = counter+i; j > i; j-- ) {
                                                        taskList.remove(j);
                                                }
                                                firstAvailableTime = startTime-
firstAvailableInterval;
                                                        found = true;
                                                        break;
                                                }
```

```
                                                        else {
                                                                i = i + counter; // i = 0 + 2
= 2
                                                        }
                                                }
                                        }
                                }

                        }
                }
        }
                        if (!found) {

                                Task lastTask = taskList.get(taskListSize-1);
                                int newTaskStartTime = lastTask.getEndTime();
                                if (BESTPICEAVAILABLE && newTaskStartTime
>= bestPrice) {
                        return REFUSE;
                }
                                Task task = new Task(manager, newTaskStartTime,
newTaskStartTime+offeredTask, offeredTask, States.preBidding);
                                taskList.add(task);
                                firstAvailableTime = newTaskStartTime;
                                return firstAvailableTime;
                        }

                        // Clean the multiple instances

                        boolean canDelete = false;
                        for (int i = 0; i < taskListSize; i++) {
                Task task = taskList.get(i);
                String managerInfo = task.getManager();
                if (!canDelete && managerInfo.equalsIgnoreCase(manager)) {
                        canDelete = true;
                        continue;
                } else if (canDelete && managerInfo.equalsIgnoreCase(manager)) {
                        taskList.remove(i);
                        break;
                }
                        }

                        return firstAvailableTime;
                }
        }

        public void updateStatus(String manager, States state) {
                int taskListSize = taskList.size();
                for (int i = 0; i < taskListSize; i++) {
        Task task = taskList.get(i);
        String managerInfo = task.getManager();
```

```
            if (manager.equals(managerInfo)) {
                    task.setState(state);
                    break;
            }
        }
            }

            public int updateStatusForDefinitiveAcccept(String manager) {
                    int taskListSize = taskList.size();
                    boolean betterOfferFound = false;
                    int finalPrice = 0;
                    for (int i = 0; i < taskListSize; i++) {
            Task task = taskList.get(i);
            String managerInfo = task.getManager();
            States state = task.getState();
            if (!betterOfferFound && (state.equals(States.definitiveReject) ||
(state.equals(States.preReject)))) {
                    task.setState(States.definitiveAccept);
                    task.setManager(manager);
                    betterOfferFound = true;
                    finalPrice = task.getStartTime();
                    System.out.println("Better bid found for " + managerInfo);
                    continue;
            }
            if (!betterOfferFound && manager.equals(managerInfo)) {
                    task.setState(States.definitiveAccept);
                    finalPrice = task.getStartTime();
                    break;
            }
            if (betterOfferFound && manager.equals(managerInfo)) {
                    taskList.remove(i);
                    break;
            }
        }
                    return finalPrice;
            }

}
```

EK A3: Contractor.java

```
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;

public class Contractor extends Agent {
        private TaskList contractorTaskList = new TaskList();

        // Put agent initializations here
        protected void setup() {
try{
                // Register the transporting service in the yellow pages
                DFAgentDescription dfd = new DFAgentDescription();
                dfd.setName(getAID());
                ServiceDescription sd = new ServiceDescription();
                sd.setType("transporting");
                sd.setName("JADE-transporting");
                dfd.addServices(sd);
                try {
```

```
                DFService.register(this, dfd);
            }
            catch (FIPAException fe) {
                fe.printStackTrace();
            }

            // Add the behaviour serving queries from manager agents
            addBehaviour(new OfferRequestsServer());

            // Add the behaviour serving purchase orders from manager agents
            addBehaviour(new PurchaseOrdersServer());

            // behaviour to handle Reject messages
            addBehaviour(new HandleRejectMessages());

            // behaviour to handle Pre-Reject messages
            addBehaviour(new HandlePreRejectMessages());

            // behaviour to handle Accept messages
            addBehaviour(new HandlePreAcceptMessages());
} catch (Exception e ) {
        e.printStackTrace();
}
        }

        // Put agent clean-up operations here
```

```
	protected void takeDown() {
		// Deregister from the yellow pages
		try {
			DFService.deregister(this);
		}
		catch (FIPAException fe) {
			fe.printStackTrace();
		}
		// Printout a dismissal message
		System.out.println("Contractor-agent "+getAID().getName()+"
terminating.");
	}

	/**
	  Inner class OfferRequestsServer.
	  This is the behaviour used by transporting agents to serve incoming
requests for offer from manager agents.
	  If the requested task can be carried out, the contractor agent replies  with a
PROPOSE message specifying
	  the price. Otherwise a REFUSE message is sent back.
	 */
	private class OfferRequestsServer extends CyclicBehaviour {
		public void action() {
			MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.CFP);
			ACLMessage msg = myAgent.receive(mt);
			if (msg != null) {
```

```
                    // CFP Message received. Process it
                    String messageContent = msg.getContent();
                    ACLMessage reply = msg.createReply();
                    int bestPrice =
Integer.parseInt(messageContent.substring(0, messageContent.indexOf(";")));
                    int offeredTask =
Integer.parseInt(messageContent.substring(messageContent.indexOf(";") + 1));
                    int firstAvailableTime =
contractorTaskList.searchAndAdd(bestPrice, msg.getSender().getName(),
offeredTask);

                    if (firstAvailableTime == -1) {
                        // The requested job can NOT be handled for a
better offer.
                        reply.setPerformative(ACLMessage.REFUSE);
                        reply.setContent("not-available");
                    } else {
                        // A better offer may be available. Reply with
the firstAvailableTime

        reply.setPerformative(ACLMessage.PROPOSE);

        reply.setContent(String.valueOf(firstAvailableTime));
                    }
                    myAgent.send(reply);
                }
                else {
                    block();
```

```
                }
            }

        }  // End of inner class OfferRequestsServer

        /**
          Inner class PurchaseOrdersServer.
          This is the behaviour used by Transporting agents to serve incoming
          offer acceptances (i.e. purchase orders) from manager agents.
          The contractor agent adds the new task to its task list
          and replies with an INFORM message to notify the manager that the
          the task can be carried out.
         */
        private class PurchaseOrdersServer extends CyclicBehaviour {
            public void action() {
                MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
                ACLMessage msg = myAgent.receive(mt);
                if (msg != null) {
                    // ACCEPT_PROPOSAL Message received. Process it
                    String messageContent = msg.getContent();
                    ACLMessage reply = msg.createReply();

                    String offeredTask =
messageContent.substring(messageContent.indexOf(";") + 1);
                    String manager = msg.getSender().getName();
```

```
                        int finalPrice =
contractorTaskList.updateStatusForDefinitiveAcccept(manager);

                        reply.setContent(String.valueOf(finalPrice));
                        reply.setPerformative(ACLMessage.INFORM);

                        myAgent.send(reply);
                    }
                    else {
                        block();
                    }
                }
            }  // End of inner class OfferRequestsServer

            private class HandleRejectMessages extends CyclicBehaviour {
                public void action() {
                    MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.REJECT_PROPOSAL);
                    ACLMessage msg = myAgent.receive(mt);
                    if (msg != null) {
                        String manager = msg.getSender().getName();
                        contractorTaskList.updateStatus(manager,
States.definitiveReject);
                    }
                    else {
                        block();
                    }
```

```
            }

        }  // End of inner class HandleRejectMessages


        private class HandlePreRejectMessages extends CyclicBehaviour {
            public void action() {
                MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.DISCONFIRM);
                ACLMessage msg = myAgent.receive(mt);
                if (msg != null) {
                    // Pre Reject Message received. Process it
                    String manager = msg.getSender().getName();
                    contractorTaskList.updateStatus(manager,
States.preReject);
                }
                else {
                    block();
                }
            }
        }

        private class HandlePreAcceptMessages extends CyclicBehaviour {
            public void action() {
                MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.CONFIRM);
                ACLMessage msg = myAgent.receive(mt);
```

```
                        if (msg != null) {
                                // Accept Message received. Process it
                                String manager = msg.getSender().getName();
                                contractorTaskList.updateStatus(manager,
States.preAccept);
                        }
                        else {
                                block();
                        }
                }
        }

}
```

EK A3: Manager.java

```
import java.util.ArrayList;
import java.util.List;

import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;

public class Manager extends Agent {
	private List<AID> contractorAgents;
	private List<AID> contractorAgentsTemp;
	private int offeredTask = 30;
	private int hopCount = 0;

	long negotiationStartTime = System.nanoTime();
	// Put agent initializations here
	protected void setup() {
		try {
		try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		// Printout a welcome message
		System.out.println("Hallo! Manager-agent "+getAID().getName()+" is
ready.");
		contractorAgents = new ArrayList<AID>();

		DFAgentDescription template = new DFAgentDescription();
		ServiceDescription sd = new ServiceDescription();
		sd.setType("transporting");
		template.addServices(sd);
		try {
			DFAgentDescription[] result = DFService.search(this,
template);
			for (int i = 0; i < result.length; ++i) {
				contractorAgents.add(result[i].getName());
			}
		} catch (FIPAException fe) {
			fe.printStackTrace();
		}
		contractorAgentsTemp = (List<AID>) ((ArrayList<AID>)
contractorAgents).clone();
```

```
                addBehaviour(new RequestPerformer());
                        } catch (Exception e) {
                    e.printStackTrace();
                }
        }

        // Put agent clean-up operations here
        protected void takeDown() {
                // Printout a dismissal message
                System.out.println("Manager-agent "+getAID().getName()+"
terminating.");
        }

        /**
          Inner class RequestPerformer.
          This is the behaviour used by Manager agents to request contractor
          agents the given task.
         */
        private class RequestPerformer extends Behaviour {
                private AID bestContractor; // The agent who provides the best offer
                private int bestPrice = 0;  // The best offered price
                private int previousBestPrice = -1;
                private int repliesCnt = 0; // The counter of replies from contractor
agents
                private MessageTemplate mt; // The template to receive replies
                private int step = 0;

                public void action() {
                        try {
                        switch (step) {
                        case 0:
                                // Send the cfp to all contractors
                                ACLMessage cfp = new
ACLMessage(ACLMessage.CFP);
                                for (int i = 0; i < contractorAgentsTemp.size(); ++i) {
                                        cfp.addReceiver(contractorAgentsTemp.get(i));
                                }
                                cfp.setContent(bestPrice + ";" + offeredTask);
                                cfp.setConversationId("transporting");
                                cfp.setReplyWith("cfp"+System.currentTimeMillis());
// Unique value
                                myAgent.send(cfp);
                                // Prepare the template to get proposals
                                mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("transporting"),

        MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
                                hopCount += 1;
                                step = 1;
```

```
                        break;
                case 1:
                        try {
                                Thread.sleep(2000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        long step1StartTime = System.nanoTime();
                        // Receive all proposals/refusals from contractor agents
                        ACLMessage reply = myAgent.receive(mt);
                        if (reply != null) {
                                // Reply received
                                if (reply.getPerformative() ==
ACLMessage.PROPOSE) {
                                        // This is an offer
                                        int price =
Integer.parseInt(reply.getContent());
                                        if (bestContractor == null || (bestPrice !=
0 && price < bestPrice)) {
                                                // This is the best offer at present
                                                bestPrice = price;
                                                bestContractor =
reply.getSender();
                                        }
                                }
                                repliesCnt++;
                                if (repliesCnt >= contractorAgentsTemp.size())
{
                                        // We received all replies
                                        hopCount += 1;
                                        step = 2;
                                }
                        }
                        else {
                                long estimatedTime = System.nanoTime() -
step1StartTime;
                                if (estimatedTime > 2000000000) {
                                        step = 2;
                                        hopCount += 1;
                                        System.out.println("Manager-agent
"+getAID().getName() + " continue to STEP 2 with repliesCnt : " + repliesCnt);
                                }
                                block();
                        }
                        break;
                case 2:
                        // Teklifleri kıyasla, best price, previous price
dan(initial = 0) iyiyse en iyi teklif veren hariç sellerlist hazırla, Step = 0 yap.
                        // Değilse step = 3 yap
```

```
                        if (bestPrice != 0 && (previousBestPrice == -1 ||
bestPrice < previousBestPrice)) {
                                //contractorAgentsTemp = contractorAgents;
                                contractorAgentsTemp = (List<AID>)
((ArrayList<AID>) contractorAgents).clone();
                                contractorAgentsTemp.remove(bestContractor);
                                previousBestPrice = bestPrice;
                                hopCount += 1;
                                step = 0;

                                ACLMessage preRejectInform = new
ACLMessage(ACLMessage.DISCONFIRM);
                                for (int i = 0; i < contractorAgentsTemp.size();
++i) {

        preRejectInform.addReceiver(contractorAgentsTemp.get(i));
                                }
                                preRejectInform.setContent("Pre-Rejected");

        preRejectInform.setConversationId("transporting");

        preRejectInform.setReplyWith("order"+System.currentTimeMillis());
                                myAgent.send(preRejectInform);

                                ACLMessage preAcceptInform = new
ACLMessage(ACLMessage.CONFIRM);
                                preAcceptInform.addReceiver(bestContractor);
                                preAcceptInform.setContent("Pre-Accepted");

        preAcceptInform.setConversationId("transporting");

        preAcceptInform.setReplyWith("order"+System.currentTimeMillis());
                                myAgent.send(preAcceptInform);

                                try {
                                        Thread.sleep(5000);
                                } catch (InterruptedException e) {
                                        e.printStackTrace();
                                }

                        }
                        else {
                                step = 3;
                                hopCount += 1;
                        }
                        break;
                case 3:
                        // Send the purchase order to the seller that provided the
best offer
                        if (bestContractor != null) {
```

```
                                        ACLMessage order = new
ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                                        order.addReceiver(bestContractor);
                                        order.setContent(bestPrice + ";" + offeredTask);
                                        order.setConversationId("transporting");

        order.setReplyWith("order"+System.currentTimeMillis());

                                        myAgent.send(order);
                                        // Prepare the template to get the purchase order
reply
                                        mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("transporting"),

        MessageTemplate.MatchInReplyTo(order.getReplyWith()));
                                        // Send the reject message to other seller agents
                                        ACLMessage rejectInform = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
                                        contractorAgentsTemp = (List<AID>)
((ArrayList<AID>) contractorAgents).clone();
                                        contractorAgentsTemp.remove(bestContractor);
                                        for (int i = 0; i < contractorAgentsTemp.size();
++i) {

        rejectInform.addReceiver(contractorAgentsTemp.get(i));
                                        }
                                        rejectInform.setContent("Rejected");
                                        rejectInform.setConversationId("transporting");

        rejectInform.setReplyWith("order"+System.currentTimeMillis());
                                        myAgent.send(rejectInform);

                                        step = 4;
                                        hopCount += 1;

                                        try {
                                                Thread.sleep(5000);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }

                                } else  {
                                        System.out.println("RESTARTING
NEGOTIATION SINCE ALL CONTRACTORS REFUSED");
                                        step =  0;
                                        hopCount += 1;
                                }
                                break;
                        case 4:
                                // Receive the purchase order reply
```

```
                                long step4StartTime = System.nanoTime();
                                reply = myAgent.receive(mt);
                                if (reply != null) {
                                        // Purchase order reply received
                                        if (reply.getPerformative() ==
ACLMessage.INFORM) {
                                                // Purchase successful. We can terminate
                                                String finalPrice = reply.getContent();
                                                System.out.println("Manager-agent
"+getAID().getName() + " successfully purchased from agent
"+reply.getSender().getName() + " --> PRICE = " + finalPrice); // bestPrice);
                                                System.out.println(getAID().getName()
+ "Manager-agent "+getAID().getName() + " için HOP COUNT : " + hopCount);
                                                myAgent.doDelete();
                                                step = 5;
                                                hopCount += 1;
                                                long negotiationDuration =
(System.nanoTime()-negotiationStartTime)/1000000000;
                                                System.out.println("Manager-agent
"+getAID().getName() + " için ihale süresi : " + negotiationDuration);
                                        }
                                        else {
                                                System.out.println("Attempt failed:
requested time interval already commited to another manager.");
                                                System.out.println(myAgent.getName()
+ " RESTARTING THE NEGOTIATION.");
                                                hopCount += 1;
                                                step = 0;
                                                bestPrice = 0;
                                                previousBestPrice = -1;
                                                repliesCnt = 0;
                                                bestContractor = null;
                                                contractorAgentsTemp = (List<AID>)
((ArrayList<AID>) contractorAgents).clone();
                                        }

                                }
                                else {
                                        block();
                                        long estimatedTime = System.nanoTime() -
step4StartTime;
                                        if (estimatedTime > 2000000000) {   // 2
seconds
                                                step = 0;
                                                hopCount += 1;
                                        }
                                }
                                break;
                        }
                        } catch (Exception e) {
```

```
                        e.printStackTrace();
                    }
                }

                public boolean done() {
                    if (step == 3 && bestContractor == null) {
                        System.out.println("Attempt failed: "+offeredTask+"
can not be carried out for now");
                    }
                    return ((step == 3 && bestContractor == null) || step == 5);
                }
            }  // End of inner class RequestPerformer
}
```

ÖZGEÇMİŞ

Ad Soyad **:** Çiğdem Albuz

Doğum Yeri ve Tarihi **:** Bandırma / 22.04.1985

E-Posta **:** cigdemalbuz@gmail.com

ÖĞRENİM DURUMU:

- **Lisans** **:** 2007, İstanbul Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği
- **Yükseklisans** **:** 2015, İstanbul Teknik Üniversitesi, Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

Nisan 2010 - Finansbank Bilgi İşlem (IBTECH)

- Nisan 2010 - Eylül 2012 Core Database Administrator
- Eylül 2012 - Şubat 2013 Mevduat Ekibi Yazılım Geliştirme
- Şubat 2013 - … Muhasebe Ekibi Yazılım Geliştirme

Temmuz 2007 – Nisan 2013 Nortel Netaş

- OAM Yazılım Geliştirme

Ağustos 2006 - HAVELSAN

- Yazılım stajı

Eğitimler :

2014- Advanced Java Programming (Bilge Adam)

2014- WPF Application Development (Bilginç IT Academy – İstanbul)

2012- Java EE 5 Design Patterns (Bilginç IT Academy – İstanbul)

2011- Oracle Database 11g: SQL Tuning Workhop

2011- Reporting Workshop (Boğaziçi Eğitim – İstanbul)

2011- Writing Optimal SQL & Troubleshooting & Tuning with Jonathan Lewis (ORACLE – İstanbul)

2011- Oracle Database 11g: Exadata Administration Workshop (ORACLE)

2010- Oracle Database 11g: Data Guard Administration Ed 2 (ORACLE)

2010- Oracle Grid Infrastructure 11g : Manage Clusterware and ASM (ORACLE)

2010- Oracle Database 10g: Administration Workshop II (Bilginç IT Academy)

2010- Oracle GoldenGate workshop (Partner HUB & Linkplus)

2010- Perl Programming workshop (Bilginç IT Academy – İstanbul)

2008- Java Programming Language

2008- Oracle Advanced Replication

2007- Oracle Database 10g: Administration Workshop I (Bilginç IT Academy

2006- MSSQL Server 2005

2006- C# Object Oriented Programming

2005- C++ and STL

2005- System Programming and Advance C Applications

2004- C Programming Language