**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ**

# ÇOKLU ETMEN ORTAMINDA NESNE TABANLI DAĞITIK BELLEK PAYLAŞIMI

**YÜKSEK LİSANS TEZİ**

**Metehan PATACI**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Bilgisayar Mühendisliği Programı**

**MAYIS 2014**

**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ**

# ÇOKLU ETMEN ORTAMINDA NESNE TABANLI DAĞITIK BELLEK PAYLAŞIMI

**YÜKSEK LİSANS TEZİ**

**Metehan PATACI**
**(504081552)**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Bilgisayar Mühendisliği Programı**

**Tez Danışmanı: Prof. Dr. Nadia ERDOĞAN**

**MAYIS 2014**

İTÜ, Fen Bilimleri Enstitüsü'nün 504081552 numaralı Yüksek Lisans Öğrencisi **Metehan PATACI**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"ÇOKLU ETMEN ORTAMINDA NESNE TABANLI DAĞITIK BELLEK PAYLAŞIMI"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

| | | |
|---|---|---|
| **Tez Danışmanı :** | **Prof. Dr. Nadia ERDOĞAN**<br>İstanbul Teknik Üniversitesi | ..... |
| **Jüri Üyeleri :** | **Doç. Dr. Güray YILMAZ**<br>Hava Harp Okulu | ..... |
| | **Yrd. Doç. Dr. Tolga OVATMAN**<br>İstanbul Teknik Üniversitesi | ..... |

**Teslim Tarihi :** **5 Mayıs 2014**
**Savunma Tarihi :** **29 Mayıs 2014**

## ÖNSÖZ

Tez çalışmam süresince her zaman pozitif yaklaşımıyla yardımlarını, desteğini ve güler yüzünü esirgemeyen değerli hocam sayın Prof. Dr. Nadia Erdoğan'a teşekkür ederim.

Mayıs 2014

Metehan Patacı
(Bilgisayar Mühendisi)

**İÇİNDEKİLER**

**Sayfa**

**KISALTMALAR**

| | |
|---|---|
| **ACL** | **:** Agent CommunicationLanguage |
| **AMS** | **:** Agent Management System |
| **CDS** | **:** Cell Directory Server |
| **COM** | **:** Component Object Model |
| **CORBA** | **:** Common Object Request Broker Architecture |
| **DCOM** | **:** Distributed Component Object Model |
| **DF** | **:** Directory Facilitator |
| **DFS** | **:** Distributed File System |
| **DOS** | **:** Distributed Object Sharing |
| **DOSMAS** | **:** Distributed Object Sharing In Multi-Agent Systems |
| **DSM** | **:** Distributed Shared Memory |
| **FIPA** | **:** Framework For Intelligent Physical Agents |
| **IDL** | **:** Interface Definition Language |
| **IPC** | **:** Interprocess Communication |
| **JADE** | **:** Java Agent Development Framework |
| **MRMW** | **:** Multiple Reader Multiple Writer |
| **MRSW** | **:** Multiple Reader Single Writer |
| **M2M** | **:** Machine To Machine |
| **OMG** | **:** Object Management Group |
| **RMI** | **:** Remote Method Invocation |
| **RPC** | **:** Remote Procedure Call |
| **SL** | **:** Semantic Language |

## ÇİZELGE LİSTESİ

**Sayfa**

## ŞEKİL LİSTESİ

**Sayfa**

## ÇOKLU ETMEN ORTAMINDA NESNE TABANLI BELLEK PAYLAŞIMI

## ÖZET

Dağıtık sistem, farklı bilgisayarlardaki yazılım ve donanım bileşenleri arasında haberleşme ve koordinasyonun mesaj yoluyla yapılmasıyla ifade edilir. Dağıtık bir sistemin en önemli ve ayırt edici özelliği bir ağın varlığı ve sistem ile ilgili tüm ayrıntıların kullanıcıdan saydam olarak gerçeklenmesidir. Kullanıcı çok işlemcili dağıtık bir sistemde çalışıyor olmasına rağmen, sistem kendisine tek bir sanal işlemci olarak görünür. Ağ üzerinde dosya gönderme, alma gibi uygulama tabanlı işlemler kullanıcı tarafından yürütülür.

Ağda veri paylaşmak gibi dağıtık uygulamaların geliştirimi için çok zaman harcamak ve sistemin heterojen yapısından kaynaklanan çeşitli zorluklarla mücadele etmek gerekir. Dağıtık sistemleri diğer ağlardan yazılım katmanı ayırır. Bu sistemlere kaynakların paylaşma gerekliliğinden dolayı ihtiyaç duyulur. Paylaşılacak kaynaklar disk, yazıcı gibi donanımsal olabileceği gibi veri tabanları, nesneler, dosyalar gibi yazılım tabanlı da olabilir.

Dağıtılmış sistemler genellikle uzak nesne modelini kullanmaktadırlar. Uzak nesne modelinde, nesneler uzak bir alana yerleştirilir ve istemciler bir vekil aracılığıyla bu alana saydam olarak erişirler. Uzak nesne modelinde ölçeklenebilme eksikliği vardır ve sabit yapısından dolayı kullanıcıyı kısıtlar.

Uzak nesne erişim modellerinde nesne erişimi basit fonksiyonlar ile ve kullanıcıya saydam olarak yapılsa da dağıtık mimari üzerinde farklı erişim yöntemleri kullanmaya imkan tanımaz. Nesneler sabit bir konumda duracağından ve her erişim için sistemde nesne konumuna vekiller aracılığıyla çağrı yapmak gerekeceğinden erişim performansında sorunlar çıkması mümkündür.

Yapılan çalışmada çoklu etmen ortamında nesne paylaşımı gerçeklenmiştir. Etmen, tasarım hedefleri doğrultusunda bir ortam içerisinde bağımsız hareket edebilen bir bilgisayar sistemidir. Birden fazla etmenin aynı ortamda benzer ya da farklı görevleri üstlenerek çalıştıkları ve mesajlaşma yoluyla iletişim kurabildikleri mimariye çoklu etmen sistemi denir. Etmenler proaktiflik, mantıklılık, süreklilik, hareketlilik, bağımsız hareket etme ve reaktiflik gibi özelliklere sahiptir. Bu özelliklerin kullanılma yoğunluğuna göre etmenler farklı tiplere ayrılır. Tüm etmenler bağımsız hareket etme özelliğine sahiptir. Çeşitli dış etkilere karşı bir etmenin ne gibi tepkilerde bulunacağına yazılım karar verir. Geliştirilmiş çoklu etmen sistemleri, etmen haberleşme alt yapısının sunmuş olduğu avantajların yanında oluşturulmuş çoklu etmen ortamı için yararlı servisler sunarlar. Bu servisler genellikle sistem ilk çalıştığında oluşturulan ve tüm etmen platformunu izlemek ile görevli etmenler aracılığı ile sağlanır. Sisteme yeni bir etmen katılması, bir etmenin sistemden ayrılması ya da sistemde içinde sunulan servislerle ilgili mesajların platforma yayılması gibi görevleri üstlenen yardımcı etmenler sayesinde kullanıcı ve yazılımcı birçok hizmeti hazır olarak alabilir.

Geliştirilen çoklu etmen ortamında dağıtık nesne paylaşım sisteminde JADE etmen mimarisi kullanılmıştır. JADE, Java programlama dili ile geliştirilmiş ve FIPA'nın etmenler için hazırlamış olduğu standartları gerçekleyen çoklu etmen ortamıdır. Sistemin Java programlama dili ile yazılmış olması platform bağımsızlığını beraberinde getirir ve bu durum dağıtık nesne paylaşımının heterojen ortamlarda da yapılabilmesini sağlar.

Paylaşılan nesnelerin ağ üzerinde dağıtılması ve yerel erişime imkan tanıması yük dengeleme, performans, koruma ve kullanılabilirlik açısından büyük öneme sahiptir. Dağıtılmış sistemlerde yerellik çok önemli bir konudur. Ağ üzerinde kuyruklarda beklememek ve meydana gelebilecek olumsuz durumlardan uzak kalmak için yerel kopyaları taşımak önemlidir.

Dağıtık bir sistemde paylaşılan bir nesnenin en güncel kopyasını taşıyan birim nesnenin sahibidir. Nesne sahipliği verinin bulunduğu yerleşkeyle ifade edilir. Bir nesne bir yerleşkeye atanmış ve tüm erişim isteklerinde değişmez bir şekilde bu yerleşkede duruyorsa buna sabit sahiplik denir. Sabit sahiplik yönteminde istemcilerin veriye doğrudan yazma hakkı yoktur. Nesnelere yazma isteği nesne sahibine iletilir ve nesne sahibi ilgili nesneyi belleğinde günceller. Bu yöntemde yoğun istekler sorun teşkil eder. Dinamik sahiplik yönteminde; nesne sahipliği sistem içinde el değiştirir. Güncel veriye kimin sahip olduğunu belirleme problemi ortaya çıkar. Merkezi sahiplik bilgisini tutan sunucuyu belirlemek, sahiplik bilgisini tutan sunucuyu dağıtık olarak atamak, veri sahibini ağa yayın yaparak belirlemek gibi yöntemlerle nesnenin ağ üzerinde yerleşkesi belirlenir ve erişimler gerçekleştirilir.

Verinin sistem üzerinde dağıtılması ve veriye erişim süresinin azaltılması sistemde çözülmesi gereken en önemli problemlerdendir. Ayrıca erişilen verinin tutarlılığının ve güncelliğinin sağlanması da çok önemlidir. Erişilmek istenen nesnenin her zaman en son değerinin elde edilmesi ve sistem içinde farklı düğümlerin aynı nesne için farklı değerlere sahip olmaması gerekir.

Ortak belleğin sistem üzerinde yayılmasında kopyalama ve göç ettirmeye dayalı çeşitli algoritmalar bulunur ve bu algoritmaların birbirlerine göre çeşitli avantaj ve dezavantajları vardır. Tüm nesne sahiplik bilgisinin bir yöneticide bulunduğu merkezi ortak bellek algoritmasında, istemciler yöneticiden haberdardır ve yönetici ile haberleşecek alt yapıya sahiptirler. Veri erişim istekleri yöneticiye yapılır ve yönetici sistem içinde veriyi bularak istemciye iletmekle görevlidir. Okunabilir kopyalama algoritmasına göre paylaşılan veri sisteme yayılmış durumdadır. Sadece sahip düğüm değil kullanıcı düğümler de belleklerinde veriyi saklarlar. Bu algoritmaya göre veriye sadece sahip düğüm yazabilir. Göç ettirme algoritmasında bir yönetici yoktur. Veri erişim ve güncelleme işlemlerini her makine kendisi yapar. Veri aynı anda sadece bir düğümde bulunabilir ve sadece bir düğüm yazma ve okuma hakkına sahiptir. Tam kopyalamalı algoritmaya göre, veri kopyaları birden fazla düğümde bulanabilir. Farklı düğümler aynı zamanda veriyi değiştirme hakkına da sahiptir. Bu özelliğinden dolayı tam kopyalamalı algoritmada veri tutarlılığını sağlamak oldukça zordur.

Veri güncelleme işlemi için dağıtık sistemlerde farklı modeller vardır. Bu modeller verinin güncelleme gereksinimini, zamanını ve şeklini belirler. Güçlü veri güncelleme modellerinde ağ üzerinde her zaman en güncel veriye ulaşmak garanti edilir. Zayıf güncelleme modellerinde ise en güncel verinin alınmasında bir garanti yoktur. Sistem güncel ya da güncele en yakın veriyi sağlamak zorundadır.

Nesneler veri ile davranışların birlikte çalışmasını sağlayan ve veri yapısında oluşabilecek hataları engellemeye yönelik mekanizmalara sahip varlıklardır. Nesnelerin sahip olduğu veri gizleme özelliği ile dağıtık sistemlerde veri güvenliği sağlanır ve arayüz ile gerçekleme birbirinden ayrılır. Aynı arayüze sahip farklı nesnelerin ağ üzerindeki düğümlerde yorumlanması ve yönetimi daha kolay olur. Farklı düğümler nesnelerin veri kısımlarını yönetirken nesne içindeki metotları kullanacağından veri alanı yönetimi kolaylaşır ve yapılan işlemlerden kaynaklanan tutarsızlık problemi en aza indirgenir.

Çoklu etmen ortamı üzerinde gerçekleştirilen dağıtık nesne paylaşımı sistemi etmenler arasında mesajlaşmalarla nesne paylaşımını sağlamaktadır. Oluşturulan nesne paylaşım sistemi; sistem koordinatörü ve yerel sunucu yöneticisi olmak üzere iki temel sunucu sistemi üzerine kuruludur.

Yerel sunucu, istemciye yerleşke olarak yakın sunucular üzerinden nesnelere erişmeyi ve nesne erişim işlemlerini kullanıcı etmenden saydam olarak gerçekleştirmeyi sağlar. Yerel sunucu farklı görevlere sahip üç etmenden oluşur. Bunlardan birincisi, başlangıç yöneticisi etmeni; yerel sunucu ile ilgili ilklendirme işlemlerini gerçekleştirir. Sistem koordinatörü ile iletişime geçerek yaratılacak yerel sunucu etmenleri için bir tanımlayıcı isim uzantısı elde eder ve sistem koordinatörü erişim bilgileri gibi başlangıç bilgilerini sağlayarak diğer yerel sunucu etmenleri olan yerel sistem yöneticisini ve nesne yöneticisini yaratır. İkinci yerel sunucu etmeni olan yerel sistem yöneticisi; yerel sunucu için bir ağ geçidi görevi üstlenmektedir. Sistem içinde ve koordinatörle iletişime geçerken mesajlar öncelikle yerel sistem yöneticisine ulaştırılır. Geri dönüş zamanı açısından kritik mesajlar yerel sistem yöneticisi kullanılmadan doğrudan hedef etmene iletilebilir. Üçüncü yerel sunucu etmeni nesne yöneticisidir. Nesne yöneticisi, nesnelere olan erişim işlemlerini gerçekleştirir. Nesne kopyaları, erişim isteği listesi, nesne durumu gibi bilgiler nesne yöneticisinin kontrolündedir.

Yerel sunucular yaratılan her nesne erişimcisi etmen için ayrı olabileceği gibi belirli bir makine üzerinde birden fazla istemci etmene hizmet verebilir. Yerel sunucular sayesinde nesne sahipliği ağ üzerinde el değiştirir. Yani güncel nesne kopyası en son yazma isteği yapan yerel sunucunun nesne yöneticisinin belleğinde bulunur. Etmenler üzerinde çalışma, nesne sahipliğinin dinamik olarak el değiştirebilmesi, erişimlerin yerel olarak gerçekleştirilebilmesi özellikleri yapılan çalışmanın ayırt edici özelliklerindendir. Yerel bir sunucunun çökmesi durumunda ilgili sunucunun sahipliğini üstlendiği paylaşılan nesneler için sistem koordinatörü denetiminde olan nesne yöneticisi geçici sahiplik yapar.

Sistem koordinatörü, nesne paylaşım platformunda oluşabilecek beklenmedik durumları ve dağıtık yapıdan kaynaklanan problemleri çözen birimdir. İçinde farklı görevlere sahip beş birim vardır. Birincisi olan başlangıç yöneticisi, sistem için ilklendirme aşamalarını gerçekleştirir ve diğer sistem koordinatörü etmenlerini yaratır. İkinci birim sistem yöneticisi, ağ geçidi görevini üstlenir ve olayları izleyerek gerekli tepkileri verir. Üçüncü birim yerleşke yöneticisi, ağ üzerinde yaratılmış olan nesnelerin yerleşke bilgilerini saklar ve yapılan sorgulamalara hızlı bir şekilde cevap döner. Dördüncü birim isimlendirme yöneticisi, ağ üzerinde yaratılmış olan tüm nesnelerin isimlendirme bilgisini saklar. Yapılan isim uygunluğu sorgulamalarına yanıt verir ve nesnelerin ağ üzerinde tekilliğini sağlar. Beşinci birim nesne yöneticisidir. Sistem koordinatörü altında görevli nesne yöneticisi, yerel sunucu altında yer alan nesne yöneticisinin tüm özelliklerine sahiptir. Bu özelliklerin

yanında ağda gerçekleşen tüm erişim ve nesne yaratma işlemlerinin bilgilerini toplar. Yerel sunucunun sahipliğini üstlendiği bir nesnenin kopyası aynı zamanda sistem koordinatörü altında bulunan nesne yöneticisinde de vardır. Böylelikle paylaşılan nesneler ile ilgili ağda herhangi bir problem oluştuğunda sistem koordinatörü nesne yönetici ile bu sorunlara çözüm üretilir. Sistem koordinatörünün nesne yöneticisi normal koşullarda bir nesnenin sahipliğini almaz. Yerel sunucuları ve erişim işlemlerini denetlediğinden yerel sunucunun erişilemez olması gibi durumlarda sahip olduğu denetçi mekanizmalardan yararlanarak nesne sahipliğini üstlenir ve erişim isteklerine cevap verir. Yapılan bir yazma işlemi ile sahipliği geçici olarak üstlenilen nesne tekrar erişimci yerel sunucu nesne yöneticisine verilir.

Çok okuyuculu tek yazıcılı mimaride gerçekleştirilen sistem yerel nesne erişimlerine olanak tanıdığından aynı etmenin yapacağı sık yazma işlemleri yüksek performansla gerçekleştirilir. Sahiplik değişimi ile nesne kopyalarının ağ üzerinde yayılması, olumsuz koşullarda ağın herhangi bir yerinden bir nesne kopyasının bulunabilmesini sağlar.

Sistem kullanıcıya nesneler için okuma, yazma, gevşek okuma, silme, yeniden adlandırma, serbest bırakma özelliklerini sunmaktadır. Kullanıcılar erişim istekleri ile elde ettikleri nesnelerin yapısı konusunda daha önceden anlaşmış olmalıdır. Nesne üzerinde yapılan yazma istekleri sonucu nesne kilitlenir. Bu yüzden yazma erişimi hakkı kazanan bir etmen yazma işlemini çok hızlı bir şekilde tamamlayarak nesneyi serbest bırakmalıdır.

Nesne erişim işlemlerinde erişimlere zaman aşımı süresi tanımlanarak kullanıcı etmenin aynı yazılım noktasında takılması engellenmiş ve sistemin kendi kendine karar verebilme özelliğine uygun çözümler üretilmiştir. Nesne sahibi sunucuların etkinliğini (canlılığını) kaybetmesi gibi durumlarda sistem koordinatörü gerekli sorumlulukları alarak sistemin sorunsuz bir şekilde çalışmasını sağlayacak önlemleri alacak şekilde tasarım yapılmıştır.

Geliştirilen nesne paylaşım sistemi bir rezervasyon uygulaması ile test edilmiştir. Rezervasyon uygulamasında yaratılan salon ve içindeki koltuklar paylaşılan nesneyi temsil etmektedir. Kullanıcılar koltuk satın almak istediği salon ismi ile sistemde bir okuma işlemi gerçekleştirir ve bu işlem ile salonun güncel durumunu elde eder. Uygun koltuklar kullanıcı tarafından seçilir ve seçim onaylanır (yazma erişimi). Seçilen koltuklardan herhangi birisini daha önce onaylayan bir kullanıcı yoksa yazma işlemi başarı ile gerçekleşir. Başka bir kullanıcı ilgili koltuklardan herhangi birisini satın almışsa sistem kullanıcıya güncel salon durumunu döndürür.

Önerilen nesne paylaşım modelinin sağladığı yerel sahiplik avantajı ve olumsuz koşullarda sunduğu çözümlerin çoklu etmen mimarisinde uygun bir nesne paylaşım ortamı sunduğu görülmüştür.

## DISTRIBUTED OBJECT SHARING IN THE MULTI-AGENT ENVIRONMENT

## SUMMARY

Distributed system provides communication and coordination between several computer software and hardware components through messages. The existance of a network and transparency of system details are most distinctive properties of a distributed system. Users workona multi processor distributed network but perceivethe system as a single virtual machine. Application based operationssuch as sending and receiving files are executed by users.

Developing data sharing applications takes a long time and heterogenous structure of the system causes some challenges. Its software layer distinguishes distributed systems from other network systems. In general, distributed systems are used because of data sharing requirement. The shared resources can be hardware components such as discs, printers or software components such as databases, objects or files.

Distributed systems usually use remote object models. In a remote object model, object is located on a remote computer and users access this object by using a proxy. Remote object models scalibilty is very hard and it limits users because of non flexible structure.

In a remote object system, object access operations are simple metot calls and provide transparent access to users; howeverthey do not provide varied access operations. Access performance is low because of constant object location and use of proxy for each access.

Sharing objects on a distributed network and local object access operations are important for load balancing, performance, protection and reusability. Local object access is important forqueue performances and network failure situations.

The owner of a remote object is a unit in distributed system that owns the most up to date copy of the shared object. Object ownership is expressed by object location. If an object is located at a constant location and all object access operations are handled at this location, this kind of ownership is expressed as fixed ownership. Each piece of data is assigned a fixed owner. The owner can usually be calculated in an easy manner from the address of the data. Requesters can directly write access objects in fixed ownership. Requesters send their write requests to fixed owner, and then owner updates the object and returns a response to the requester. The disadvantages of this scheme are obvious; there is a large overhead on every write access, and the data owner becomes a bottleneck in the system. This approach is realized through centralized algorithms. An alternative approach is dynamic ownership where the ownership of a data item moves around the system. Locating object owner is a problem in such systems. As a solution to this problem, we introduce the concept of a manager for an item of data. The manager needs not own the data, but is responsible

for tracking the current owner of the data. Another way to determine object owner is broadcasting a message to the network and waiting for a reply.

Another problem is maintaining replicas of an object over the network and manage object access. Consistency of the object in system becomes a very important issue. When there are multiple copies of an object spread around the distributed system, it must be ensured that no user (agent, processor etc.) reads stale data once a write has been completed on a machine.

There are two main approaches for write synchronization. A processor can broadcast writes, or it can invalidate all other copies before doing an update. These two approaches are known as write-broadcast (write update) and write-invalidate. The write broadcast approach, broadcasts all writes to all nodes that have copies of the data, effectively replicating data. If a low level broadcast primitive is not available, then there will be a problem locating all the sites with copies of the data set. Write broadcast is usually considered too expensive to be used as a general solution. Ordering of broadcasts is a problem and replication algorithms are usually implemented using a single write sequencer site that broadcasts writes to all involved sites. Inthe invalidation approach, we synchronize write accesses and invalidate all the other copies of the data. This can easily be done with broadcasts, but the expense of broadcasting can grow rapidly in large distributed systems and hosts may spend a lot of time handling broadcast messages. If a distributed system uses multicasts for individual messages instead, then we again have a problem of location. We need to locate the processors that currently have a copy of data in the distributed system. We may have a manager that has a list of proccessors with a copy of the data. If write is allowed to proceed before the invalidation is complete, it is possible for non current read-only copies of an object to exists for a short period of time.

There are copy based and migration based algorithms for shared memory spread along distributed network. Those algorithms have some advantages and disadvantages. We usually use copy based algorithms for read only accesses. Read only copies of shared data can be at different processors on network and if one processor wants to make read only accessesto the data, a copy of the shared data is transferred to the processors memory. In this approach, write requests are problem for data consistency because too muchshared data copy is spread around the network. With migration based algorithms, only one processor has write access permisson and this processor is said to be the owner of the shared data. When a new write request is received, data ownership changes and the data migrates to requester processor’s memory.

There are several data update models for distributed systems. These models definethe need foran update, update time and the update procedure. Strong data update models always guarantee most up to date data access. Weak data update models do not guarantee up to date data access. The system provides up to date or near up to date data for accessors.

An object is a location in memory having a value and possibly referenced by an identifier. An object can be a variable, a function, or a data structure. An object has data hiding ability which provides data safety, separating abstraction and implementation. Different objects may have the same interface and this provides easy interpretation and management of data at distributed systems.

A software agent is a computer program that acts on behalf of a user or a program in a relationship of agency. If more than one agent exists and communicates for same or

different job in an environment we call this architecture multi agent system. In this work, a framework for distributed object sharing in a multi agent environment is implemented. Users of shared objects, actually agents, are provided an interface which allows them to access, to read or write remote objects, in a transparent manner, regardless of their current locations in the distributed system. The framework implements the protocols that enables object sharing and replica management with a high degree of fault tolerance.

We have use JADE as the implementation platform for the agent based architecture of the framework. JADE is a multi agent environment that using JAVA programming language and it is compatible with FIPA standards. The framework supports object sharing through agent interactions via messaging. The architecture relies on two main server components, namely the system coordinator and a local server manager for each remote site.The local server manager is the unit which intercepts agent users access requests to remote objects. It cooperates with the system coordinator to carry out the execution steps needed to answer those requests.

The local server manager includes three agents, each with different tasks. One of them is the initiation manager whose task is to run the initialization operations of the local server manager. It interacts with the system coordinator to get a unique descriptor name for the local system and system coordinator access information, which it uses to create the local server and the object manager. The second agent is the local server which acts as gateway for local server manager. Lastly, the third agent is the object manager which is responsible for all object access operations. It stores replicas of shared objects and provided object access management operations.

The local server manager servesall agents which are located on the same remote machine.Object ownership spreads around the distributed network by local server managers. Most up to date object copies (object ownership) live at local server managers object manager. When a write request is received from a local server manager, object ownership is transferred to requester agents local server manager. The most distinguishing property of the framework is its agent based architecture and dynamic sharing of object ownership.

The system coordinator controls the whole system and it solves unexpected situations and problems. It includes five agents with different tasks. The inititation manager makes initializations for the system and creates the other system coordinator agents. The system manager acts as gateway for the system coordinator. The location manager stores live object locations (object owner addresses) and gives quick responses for location queries. The naming manager stores shared object names in the distributed system. It gives quick responses to object name suitability queries and provides object name uniqueness. The object manager has all the capabilities of the local server manager’s object manager, and in addition to those it collects all object access and create informations in distributed system. Local server’s object replicasare maintained at the same time at system coordinators object manager (by information messages which comes from local server) to overcome problems. The system coordinator’s object manager doesn’t take object ownership in normal situations. It takes information messages from object owners and when an unexpected problem occurs on objects (such as an owner agent failure) it takes the object ownership of the objects owned by the failing agent.When a write access request arrives for those objects, the object ownership is transferred to the requester’s local server object manager.

The framework implements the MRSW (multiple reader/ single writer) access model and provides local object access. If an agent makes too many write accesses or it is already the object owner, object accesses occurlocally and therefore is very fast. When ownership changes in distributed network, local object copy spreads around and when an unwanted situation occurs in system we can find object copies for restore operations.

The framework provides users an interface with different kinds of access operations to shared objects. These are read, write, loose read (object value may not be up to date), delete, rename, release operations. Users must have agreed upon objects structure before. When a write request is received, the object is locked, preventing other agents read access requests, therefore an agent which has write permission should quickly finish its job and release the object.

The framework monitors the processing time of object access requests. There is a predefined and configurable maximum object access duration time limit in the system. With this approach, a request which can not be met within the defined period is cancelled with an error return code, thus avoiding gettings stuck in a block of code, and allowing users to make alternative decisions. The system coordinator takes solution oriented responsibilities for unwanted and unexpected situations such as local server failure.

The framework described enables distributed object sharing in a multi agent environment. It provides local object ownership and local access that minimizes network traffic, and solves problems that arise in distributed object sharing in a multi agent environment.

# 1. GİRİŞ

Bilgisayarlar arası haberleşmenin olduğu birçok ortamda ortak bellek kullanma ihtiyacı vardır. Dağıtılmış sistemler, ağ üzerinde oluşturulan haberleşmese alt yapısından faydalanarak bilgi ve sistem kaynaklarını paylaşabilirler.

Yüksek performans gerektiren işlemler için dönemin en üst teknolojisine sahip süper bilgisayarlar kullanmak bir çözüm olarak gözüksede, ihtiyaçların zaman içinde artması, teknolojinin hızla gelişmesi, yüksek donanım maliyetleri, yerel ya da geniş bir ağa hizmet verebilmek gibi unsurların ortaya çıkması sonucu birden çok bilgisayarın paralel olarak işlemleri yerine getirebilme ve uyumlu bir şekilde çalışabilmesi gereksinimini ortaya çıkartmıştır. Her biri kendi başına çalışabilen, işlemci, işletim sistemi ve bellek gibi kaynaklara sahip iş istasyonlarının bir bilgisayar ağı ile birbirlerine bağlanıp bir yazılım sayesinde etkileşim içinde, uyumlu ve gerektiğinde paralel çalışabilmesi fikri dağıtık sistemlerinin temelini oluşturmaktadır. Çok yüksek maliyetli donanımlara ihtiyaç duymayan bu yöntemin etkinliği ağ ve yazılımın performansına bağlıdır.

Dağıtık sistemlerde dağıtım ile ilgili ayrıntılar kullanıcılara saydam olarak yapılır. Birçok uygulamada, kullanıcı gerçekleştirdiği işlemler sırasında ağ üzerinde farklı bilgisayarları kullandığının farkında olmaz.

Ağ üzerinde veri paylaşımı gibi geniş alan uygulamalarının geliştirilmesi için özel yöntemler ve çok fazla efor gerekmektedir. Özellikle ağ üzerinde yer alan farklı işletim sistemi ve farklı mimarilere sahip bilgisayarların uyumlu bir şekilde çalışmasını sağlamak için fazladan efor harcamak gerekmektedir. Bu sorunları aşmak için platform bağımsız yazılım ve işletim sistemleri geliştirilmiş olsada ağ üzerinde herzaman heterojen bir yapının olması mümkündür.

Geçmişte dağıtık sistemler için genelde ağ üzerinde dosya işlemleri yapmayı hedefleyen ilkel haberleşme modelleri oluşturulmuştur. Bu modeller veri göç ettirilmesi ve kopyalama gibi problemlere çözüm sağlayamamaktadır ve sunulan çözümler uygulamaya özel nitelikte olmakla sınırlı kalmaktadır. Bu yüzden

dağıtılmış ortamda veri alışverişini ve paylaşımını genel bir arayüz altında gerçekleyecek bir modele ihtiyaç vardır. Model haberleşme ilkelerine göre daha soyut düzeyde ve uygulamadan bağımsız yapıda olmalıdır. Nesne tabanlı dağıtılmış sistem çözümleri, nesnenin doğal yapısından kaynaklanan işlevselliği gerçeklemeden ayırma özelliği sayesinde dağıtık yapıya uygundur. Aynı işlevselliğe sahip fakat farklı gerçeklemelerle ifade edilmiş özellikleri nesnelerle ifade etmek mümkündür. Nesneler, veriye erişimleri önceden tanımlanmış metotlar aracılığıyla sağladığından kullanım ve yönetimi kolaydır.

CORBA, DCOM gibi oldukça popüler dağıtılmış sistemler uzak nesne modelini kullanırlar [1,2]. Uzak nesne modeline göre nesneler uzak bir makinenin bellek alanına yerleştirilir ve istemci bilgisayarlar bir vekil aracılığıyla nesneye saydam bir şekilde erişirler. Kullanıcı açısından bakıldığında yapılan işlemler metot çağrılarından oluşmaktadır. Ağ üzerinde nesne erişimi ve yönetimi ile ilgili işlemlerden vekil yazılımı sorumludur. Uzak nesne modelinde ölçeklenebilme eksikliği vardır. Bu modelde nesne ile ilgili gerçekleştirilecek çağrı işlemlerini yerel olarak yapma imkanı yoktur.

Vekil farklı programlama dillerini ve işletim sistemlerini destekleyecek yazılım alt yapısına sahiptir ve bu alt yapıyı hazırlamak oldukça maliyetli bir işlemdir.

### 1.1 Tezin Amacı

Bu çalışmada amaç; çoklu etmen mimarisini kullanarak geniş alana yayılmış sistemlerde haberleşme problemine yönelik bir nesne paylaşım ortamı geliştirmek ve kullanıcıya sistem iç yapısından saydam bir çalışma ortamı sunmaktır. Geliştirilen sistem çok okuyuculu tek yazıcılı senkronizasyon modelinikullanmaktadır vebu mimariye göre nesneye yazma işlemi yapılırken, yazma işlemi tamamlanıp nesne serbest bırakılana kadar okuma işlemleri bekletilir. Bir ya da daha çok okuyucu nesneye aynı anda erişebilir ve tüm okuma istekleri serbest bırakıldığında varsa yazma isteğine izin verilir.

Uzak nesne modelinde veri belirli bir uzak bilgisayarda yer aldığından dağıtık yapının bazı avantajlarından yararlanılamaz. Paylaşılan bir nesne koruma, başarım, kullanılabilirlik gibi ilkelerin avantajlarından yararlanmak amacıyla çok sayıda alan üzerine yayılabilir. Etkin erişim, başarım ve ölçeklenebilirlik ilkeleri göz önünde

bulundurulduğunda nesnelerin yerelliği büyük önem taşır. Nesneleri yerelleştirmek amacıyla ağ üzerinde nesne kopyaları dağıtılmalıdır. Kopyalanmış nesne sistemleri yerel erişime imkan tanımasından dolayı uzak nesnelere oranla çok daha hızlı erişim sunarlar. Kullanılan güncelleme protokolüne ve erişim sıklığına bağlı olarak yerel nesneler sistem performansında büyük etkiye sahiptir.

Nesne kopyalama işlemi beraberinde tutarlılık problemini de getirir. Kopyalar arasında tutarlılığın sağlanması nesne paylaşım sisteminin görevidir. Tutarlılığın sağlanması ile ilgili geliştirilmiş birçok yöntem bulunmaktadır.

Bu çalışmada uzak nesne modellerine alternatif olarak çoklu etmen mimarisinde, yerel nesne kopyalarını kullanmaya imkan tanıyan, nesne sahipliğinin yazma istekleri sonucunda el değiştirdiği, kullanıcıya saydam bir arayüzle nesne erişimi imkanı sunan, etmenlerin sunduğu tüm özelliklere sahip, Java programlama dili ile gerçeklendiği için platform bağımsız, sistem içinde yük dengelemek amacıyla farklı görevlere sahip birimleri olan ve kopyalamaya dayalı bir dağıtık nesne paylaşım sisteminin tasarım ve gerçekleme ayrıntıları sunulmaktadır.

## 1.2 Çalışma Ortamı

Java nesnelerinin paylaşılmasını sağlayan sistem, paylaşılan nesnelerin ağ üzerindeki tüm bilgisayarlar tarafından anlamlandırılabilmesi ve kullanılabilmesi için Java sanal makinesine ihtiyaç duyar. Java platform bağımsız bir dil olduğundan farklı işletim sistemleri altında nesne erişimi yapmak mümkündür. Byte kodu adı verilen veri dizilerine çevrilen nesneler ağ üzerinde hedef bilgisayara ulaştığında tekrar Java nesnelerine dönüştürülür.

Çoklu etmen mimarisi kullanıldığından nesne erişim istemcisi bir etmen olmalıdır. Bu çalışmada JADE altyapısı kullanılarak dağıtık nesne paylaşımı sağlanmıştır[3]. JADE, Java programlama dili ile geliştirilmiş ve FIPA'nın etmenler için belirlediği tüm özellikleri içeren bir sistemdir. Kendi kendine karar verme, sosyallik (diğer etmenlerle iletişim halinde olmak), proaktiflik (çevresinden etkilenmenin yanında çevresini etkileyecek davranışlarda bulunma) gibi özelliklere sahiptir. Sistem etmenler arasında haberleşmeyi sağlayacak bir altyapıya sahiptir ve sunmuş olduğu metotlarla kullanıcının bu altyapıyı kullanmasına izin verir. Haberleşmeyi gerçekleştirmek için farklı protokol ve anlamsal ifadeleri destekler. Sistem olaylarını

izleyen ve gerektiğinde bu olayları diğer etmenlere yayınlama özelliğine sahip yardımcı birimleri vardır. Hareketli platformlar da dahil olmak üzere JADE sistemini bir çok platformda çalıştırmak mümkündür.

Dağıtık nesne paylaşım sistemi JADE kütüphanesinden gelen haberleşme metotlarını kullanarak nesne paylaşımını sağlar. Yapılan çalışmada JADE etmenlerine dağıtık nesne paylaşım özelliği eklenmiştir.

# 2. DAĞITIK BELLEK PAYLAŞIMI

## 2.1 Bellek Yönetimi

Ana belleğin süreçler arasında paylaşılmasına bellek yönetimi denir. İşletim sisteminin bu amaçla oluşturulan kesimine de bellek yöneticisi denir.

Bellek yöneticisinin görevi, belleğin hangi parçalarının kullanımda olduğunu, hangi parçalarının kullanılmadığını izlemek, süreçlere bellek tahsis etme, tahsis edilen belleği geri almak ve bellek ile disk arasındaki veri alışverişini gerçekleştirmektir [4,5].

Bellek yönetiminin temel amaçları şunlardır:

- Bellekteki herhangi bir işlemi başka bir yere aktarabilmek.
- Birden fazla işlem veya kullanıcı olduğunda bir kullanıcının veya işlemin diğerinin alanına müdehale etmesini engellemek.
- Kullanıcılar arasında kaynak paylaşımı sağlamak.
- Belleğin mantıksal alanlara bölünmesini sağlayarak bilgiye erişimi kolaylaştırmak.
- Belleğin yetersiz olduğu durumda, varsa başka bellek kaynaklarının alanlarını kullanabilmeyi sağlamak.

## 2.2 Ortak Bellek

Güncel veriye ulaşmak ya da iletişim kurmak için programların eş zamanlı olarak eriştiği paylaşılmış bellek alanına ortak bellek denir. Bu yapıyı kullanan sistemlerde ortak bir bellek ve bu belleği kullanan işlemciler vardır. Ortak bellek mimarisinde işlemciler birbirlerine doğrudan bağlı değildir. İşlemciler arasında her türlü haberleşme ortak bellek ile olur. Her işlemci sadece kendisinin erişebileceği bir belleğe sahip olabileceği gibi birden fazla işlemci ile ortak olarak kullanabileceği bir bellekte sistem mimarisinde bulanabilir.

Birden fazla işlemcinin bulunduğu bir mimaride ortak bellek kullanmanın avantajı, işlemciler arasında doğrudan bir bağlantı olmamasından dolayı programlama işleminin daha kolay olmasıdır. Aksi durumda tüm işlemcilerin haberleşmesi için ayrı veri yolları hazırlamak gerekecek ve veri tutarlılığını sağlamak için işlemciler arasında karmaşık mesajlaşmalar gerçekleştirilecektir.

Ortak bellekli mimaride tüm işlemciler aynı veri alanını kullandıkları için, aynı veri alanına birden çok işlemcinin eşzamanlı erişimi sırasında veride tutarsızlık ile karşılaşmak mümkündür. Bunun yanında, ortak bellek kullanımında işlemci sayısı arttıkça etkinlik düşmektedir. Ortak bellek erişiminde eş zamanlı erişimden dolayı veri talepleri bir kuyrukta saklanmakta ve kuyruk yönetimine bağlı olarak sırası gelen taleplere cevap verilmektedir. Bu sorunun çözümü için yerel bellek (cache) kullanılabilir ve bu sayede ortak belleğe erişim sıklığı azaltılabilir. Yerel bellek kullanımında ise veri tutarlılığı sorunu ortaya çıkmaktadır.

İşlemcilerin ortak belleğe erişim amaçları belirlenip performans iyileştirici önlemler almak mümkündür. Üç işlemcili bir mimaride, iki işlemci ortak belleğe sadece okuma amaçlı, bir işlemci ise hem okuma hemde yazma amaçlı erişiyorsa, hem okuma hemde yazma amaçlı erişen işlemci veriyi daha hızlı olan yerel belleğinde tutar ve sadece yazma yapacağı zaman yerel belleğinde ilgili kısmı geçersiz kılıp ortak belleği günceller. Ortak bellek erişimlerinde kullanılan karşılıklı dışlama (semafor) yapısı donanım tabanlı ve performanslı bir yapıda geliştirildiği taktirde performans artacaktır.

### 2.3 Dağıtılmış Ortak Bellek

Ortak bellekli mimaride her işlemci belirli bir adres alanına erişirken dağıtılmış ortak bellek mimarisinde kullanılan her makinenin kendine ait bir belleği vardır. Bu işlemciler kendi aralarında mesaj transferi ile haberleşirler. Mesajların belirlenmiş bir formatı vardır ve her işlemci bu formatı yorumlama yeteneğine sahiptir [6,7].

Mesaj formatı genellike bir başlık ve veri kısmından oluşmaktadır. İşlemciler başlık kısmından mesajın tipini yorumlar ve veri kısmını bu yorum doğrultusunda kullanırlar.

Dağıtılmış bellek mimarisinde işlemci sayısı konusunda sınırlama yoktur. Sistemdeki bilgisayarların homojen yapıda olması da zorunlu değildir. Ancak heterojen

sistemlerde olabilecek veri temsili farklılıklarına dikkat edilmelidir. Heterojen yapıda en sık karşılaşılan problem işlemcilerin verileri diziş sırası (little/big endian) farklılığıdır.

Dağıtılmış ortak bellek kullanan bir sistem tasarlamakta amaç ortak bellekli ve dağıtılmış bellekli sistemlerin avantajlarını birleştirerek yeni bir sistem oluşturmaktır. Programcı ortak bellek mimarisinde olduğu gibi güncel veri kontrolü, hata mesajları, veriye erişme gibi problemlerle uğraşmayacak hem de dağıtılmış bellek kullanan sistemlerde olduğu gibi çok sayıda işlemci kullanabilecektir. Veriye bu şekilde erişim bir arayüz sayesinde olur ve programcı dağıtılmış ortak belleğe erişirken kendi yerel belleğindeymiş gibi çağrılarda bulunur.

Tasarlanan arayüz veriyi bulup getirmek, hata kontrolü yapmak, tutarlılığı sağlamak gibi özelliklere sahiptir. Dağıtılmış ortak bellek sisteminde her işlemcinin kendine ait belleği vardır fakat bu belleğin içeriği işlemciler arası mesajlaşma yoluyla paylaşılabilmektedir. Tasarlanan arayüz sistemdeki mesajların transferi, veri tutarlığının sağlanması, kuyrukların ayarlanması gibi işlemlerden sorumludur ve bu işlemleri kullanıcıya yansıtmamalıdır.

Dağıtık ortak bellek paylaşımı arayüz programlamasında; veri erişiminde kullanılacak algoritmanın seçimi, erişimin nasıl ifade edildiği, veri güncelleme modelinin seçimi konuları önem taşımaktadır.

## 2.4 Veri Yerleşkesi

Verinin bulunduğu bellek alanını ifade etmektedir. Sadece veri sahibinin yazma hakkı vardır. Veri sahibi üzerinde yazma isteği bulunduğu sırada, diğer düğümlerin okunabilir kopyaya sahip olması durumunda, yazma senkronizasyonu problemi ortaya çıkmaktadır [8]. Veri sahipliği ile ilgili algoritmalar alt bölümlerde açıklanmıştır.

### 2.4.1 Sabit sahiplik

Veri değişmez bir sahibe atanmıştır. Sahip genellikle veri adresinden kolaylıkla anlaşılabilir. Diğer işlemcilerin veriye doğrudan yazma hakkı yoktur ve veriye yazmak istediklerinde sahip işlemci ile iletişime geçmelidirler [8]. Bu yöntemin en

büyük dezavantajı çok fazla yazma isteği olduğu durumda sahip işlemcinin darboğaza girecek olmasıdır.

### 2.4.2 Dinamik sahiplik

Sahiplik dağıtık sistemde el değiştirir. Güncel sahibin kim olduğunu belirlemek sorun oluşturur [8]. Bu problemi çözmek için paylaşılan verileri kontrol eden bir yönetici atamasından bahsedilebilir. Yönetici, veriye sahip olmak ve veriyi taşımak zorunda değildir. Verinin sahibinin kim olduğunu, nerede yer aldığını saklamak ve yapılan veri sahipliği sorgularına cevap vermek yöneticinin görevidir. Dinamik sahiplik; merkezi, dağıtık, yayınlama, değişken sahiplik olmak üzere dört farklı şekilde gerçekleştirilebilir.

Merkezi sahiplik; bir düğüm tüm paylaşılan veri için yönetici olarak seçilir ve bu düğüm her bir verinin anlık sahiplik bilgisini saklar. Diğer işlemciler veriye erişmeden önce verinin yerini belirlemek amacıyla yöneticiden sahiplik bilgisini alır. Yönetici düğüm ayrıca yazma okuma isteklerini bir kuyrukta tutar ve bu işlemleri takip eder.

Dağıtık sahiplik; merkezi yönetici modelinde oluşabilecek darboğazı engellemek amacıyla yöneticiyi dağıtık olarak ve el değiştirecek şekilde atamak mümkündür. Böyle bir atamada bu sefer yöneticinin o anda hangi düğüm olduğunu belirlemek problem yaratacaktır. Bu yöntemde yöneticiyi hızlıca bulabilecek dinamik yöntemler tanımlamak gerekecektir [5].

Yayınlama ile sahiplik; her bir işlemci veri ve bu veri ile ilgili yönetim bilgisini saklamak ile sorumludur. Tüm bu bilgiler gerektiğinde ağda farklı bir düğüme taşınır. Mesaj sahibini arayan işlemci tüm ağa bir sorgu mesajı yayınlar. Bu algoritmanın çok basit bir yapısı olmasına ragmen ağda çok fazla mesaj trafiğine sebep olduğundan oldukça düşük performansa sahiptir [5].

Değişken sahiplik; her işlemci muhtemel veri sahipliği bilgisi saklar[8]. Bu bilgi sadece bir tahmindir ve yanlış olma olasılığı vardır. Saklanan bilginin yanlış olması durumunda veri sahibi bulunana kadar ağdaki diğer işlemcilere sırayla sahiplik sorgusu göndermek gerekir. Muhtemel sahiplik bilgisi sahipliği gösteren mantıklı

bilgiler alındığında güncellenir. Veri güncelleme mesajı alındığında muhtemel sahiplik bilgisi de güncellenebilir.

### 2.5 Dağıtılmış Ortak Bellek Algoritmaları

Dağıtık ortak bellek erişiminde kullanılacak sistemin gereksinimlerine ve yapısına göre farklı tipte algoritmalar kullanılmalıdır. Aynı yerel sunucuya bağlı ve birbirlerine çok yakın olan sistemler ile geniş bir alana yayılmış sistem için kullanılacak ortak bellek paylaşım algoritmaları gereksinime göre farklılık gösterir [9,10,11].

Dağıtık ortak bellek paylaşımında karşılaşılan en önemli iki problem:

- Verinin mevcut sistem üzerinde dağıtılması ve en kısa sürede erişim sağlayacak yapıya getirilmesi.
- Dağıtık sistemde ortak erişilen verinin tutarlığının ve güncelliğinin sağlanması.

Ortak verinin dağıtık sistem üzerinde yayılması için kopyalama ve göç ettirme algoritmaları kullanılır. Kopyalama algoritmasında bir veri birden fazla yerel ya da cep belleğinde aynı anda bulunabilir. Bu durum veriye eş zamanlı olarak çok sayıda erişim yapılabilmesini sağlar. Göç ettirme algoritmasında bir veriyi kullanacak işlemci öncelikle ilgili veriyi kendi belleğine taşımalıdır ve taşıma işlemi tamamlandıktan sonra veri kullanılabilir.

Sistem ihtiyacına göre kullanılacak algoritma seçilir. Dağıtılmış ortak belleği gerçekleyen dört ana algoritma vardır.

#### 2.5.1 Merkezi ortak bellek

Yönetim bir merkezden gerçekleştirilir. Sistemi kullanan tüm düğümler yöneticiden haberdardır ve yönetici ile haberleşebilecek alt yapıya sahiptir. Bir düğüm sistemdeki bir veriye erişme isteğini yönetici düğüme iletir. Yönetici kendi içinde yapacağı bir arama işlemi ile verinin hangi düğümde olduğunu bulur ve ilgili düğümden veriyi alarak istemci düğüme iletir. Çalışma mantığı basit olan bu yapıda oluşabilecek sorunlar farklı yöntemlerle çözülebilir.

İstemci düğüm, isteğini yönetici bilgisayara gönderdikten sonra cevabı bekleme durumuna geçecektir ve bu talep yönetici bilgisayarda kaybolursa istemci sonsuza kadar beklemiş olur. Bu durumu engellemek için her bir istek için bir zaman aşımı süresi belirlenir. Gerekli sürede yöneticiden bir cevap gelmezse istek yenilenir. Veriye ulaşamama durumunda alınacak tutumlar ve maksimum istek yenileme sayısı istekçi düğümde belirlenir.

Merkezi ortak bellek algoritmasının en zayıf yanı bütün kritik işlemlerin tek bir sunucu üzerinde gerçekleşiyor olmasıdır. Merkezi sunucuda bir darboğaz oluşması herzaman mümkündür. Bu durumu engellemek için birden fazla yönetici bilgisayar kurulur ve veriler dengeli bir şekilde bu sunuculara dağıtılır.

### 2.5.2 Okunabilir kopyalama

Okunabilir kopyalama algoritmasında ortak veri sisteme yayılmış durumdadır. Veri sadece sahip'in belleğinde değil kullanıcı düğümlerin belleğinde de bulunmaktadır. Sadece veri sahibi olan düğümün yazma yetkisi vardır. Sistemde veriyi yerel belleğinde taşıyan diğer sunucular sadece okuma yapabilirler. Bu tipteki algoritmalara MRSW (çok okuyuculu tek yazıcılı) algoritmalar denir.

Ortak veri sistem üzerinde birden çok düğümde aynı anda bulunabileceğinden ve her düğüm veriye yerel belleğinden erişebileceğinden okuma işlemlerinin maliyeti çok düşüktür. Yazma işlemlerinde ise sahip düğüm veriye her yazma işlemi gerçekleştirdiğinde okunabilir kopyaya sahip diğer tüm düğümlere güncelleme mesajı göndermek zorundadır. Bunu sağlamak için sahip düğüm tüm okunabilir kopyaya sahip düğümleri izler ve kopyalama kümesi adreslerini saklar. Ayrıca sistemde kopyalama kümesi adreslerini saklayan ayrı bir sunucu bulundurmak da bir yöntemdir.Yazma işlemi yapacak bilgisayar, sahip bilgisayardan veriyi alırken kopyalama kümesini de alacaktır. Yazma yetkisi alan düğüm öncelikle veriyi kilitler ve kilitleme mesajını kopyalama kümesine iletir. Yazma işlemi tamamlandıktan sonra kopyalama kümesine güncel veriyi içeren kilitleme ortadan kaldırma mesajı gönderilir. Okuma yazma oranı yüksek olan sistemler için bu model uygundur.

### 2.5.3 Göç ettirme

Göç ettirme algoritmasında yönetici görevini üstlenen bir makine yoktur. Veri erişim ve güncelleme işlemlerini her makine kendisi gerçekleştirir. Bu algoritmada güncel verinin hangi düğümde olduğunu bulmak problemdir. Tüm sisteme yayın yapılarak, sadece verinin yerini tutan bir yönetici ekleyerek ya da her düğümde güncel veri adresleri saklayan bir tablo bulundurarak bu sorun çözülebilir.

Göç ettirme algoritmasına göre bir veri aynı anda sadece bir düğümde bulanabilir ve sadece bir düğüm tarafından erişilebilir. Tek bir düğümün okuma ve yazma hakkı olacağından bu algoritmaya tek okuyuculu tek yazılı (SRSW) algoritma da denir.

Veri aynı anda sadece bir düğümde bulanacağından ve aynı anda sadece bir düğüm tarafından veriye erişilebileceğinden veriye sahip bilgisayarın erişimi çok hızlı olacaktır; fakat farklı düğümlerden gelecek sık erişim isteği sahiplik değişikliği ve verinin taşınmasını gerektireceğinden daha az performanslı olacaktır.

### 2.5.4 Tam kopyalama

Paylaşılan verinin kopyaları aynı anda birden fazla düğümde bulunabilir ve kopyalama algoritmasından farklı olarak her düğüm aynı zamanda veriyi değiştirme hakkına da sahiptir. Aynı anda farklı düğümlerde yapılacak yazma erişimleri veride tutarsızlığa sebep olacaktır. Bu tutarsızlığı engellemek amacıyla sistem çapında tüm düğümlerde hassas ve senkron bir saat tanımlanıp her bir erişim zaman sırasına koyulabilir. Yapılacak bu işlemler sistem çapında ek mesajlaşmalara sebep olacaktır.

## 2.6 Dağıtık Ortak Bellek Gerçekleme Yöntemleri

Dağıtık sistemlerde ortak bellek yazılım tabanlı, donanım tabanlı ve karma (melez) olarak gerçeklenebilir. Bu yöntemleri ayıran en önemli faktörler maliyet, performans ve karmaşıklıktır. Sistemlerin birbirlerine göre avantaj ve dezavantajları vardır. Kullanılacak sistemin gereksinimlerine göre uygun bir yöntem seçilir.

### 2.6.1 Donanım tabanlı gerçekleme

Donanım tabanlı gerçeklemede verinin yerel belleğe ve önbelleğe kopyalanma işlemleri donanım tarafından gerçeklenir. Yazılım tabanlı sistemlere göre hata oranı

daha düşüktür. Sistem performansını donanım hızları belirler ve esnek değildir. Bu tip sistemler yazılım tabanlı sistemlere göre daha maliyetlidir.

### 2.6.2 Yazılım tabanlı gerçekleme

Sistemler arası mesajlaşma ile gerçeklenen bir yöntemdir. Programcının verinin güncelliğini ve geçerliliğini sağlaması programlama aşamasında zordur.

Bu sistemler kullanıcı seviyesi, programlama dili seviyesi ve işletim sistemi seviyesinde gerçeklenebilir.

### 2.6.3 Karma gerçekleme

Donanım ve yazılım tabanlı gerçeklemenin beraber kullanıldığı yöntemdir. Bu yöntemde sistem esnekliği üst düzeyde tutulurken maliyet azaltılmıştır. Mesajlaşmalar ve bellek kontrolünün bır kısmı yazılım tarafından kontrol edilir. Yazılım tabanlı yöntem ile karşılaştırıldığında sistem içinde kullanılan donanım desteği daha fazla olacağından performans artacaktır. Sadece donanım tabanlı sistemler ile karşılaştırıldığında ise esneklik artacak ve maliyet azalacaktır.

## 2.7 Veri Tutarlılık Modelleri

Verinin güncellenme gereksinimini, zamanını ve şeklini belirleyen modellerdir. Dağıtık bellek paylaşımı sağlayan sistemlerde performansın artması farklı düğümlerin aynı anda veriye erişebilmesinin sağlanması ile olur. Bu durumu sağlamak için farklı düğümlerde ortak belleğin kopyaları bulunur. Kopyaları ilgili düğümlere gönderme kararı vermek kullanılan modele göre değişiklik gösterir. Tam kopyalamalı ve okunabilir kopyalamalı algoritmalarda veri kopyası düğümler arasında iletilir. Kopyalama işlemi ile veriye erişim artacaktır; fakat verinin değiştirilmesi sırasında diğer kullanıcılara değişiklik bilgisi ya da güncel veriyi iletme gereksinimini ortaya çıkaracaktır. Bu problemin çözümü için gereksinime göre farklı tutarlılık modelleri kullanılır [10-13].

Burada güncel veriden kasıt bir okuma işlemi gerçekleştirildiğinde alınan değerin sistem içinde yazılan en son değer olmasıdır. Kullanılan güncelleme modeli verinin kullanım şeklini ve ağda hangi koşullarda taşınacağını belirlediğinden sistem performansına etkisi büyüktür. Güçlü yapıdaki güncelleme modelleri programlama

işlemini kolaylaştırmasına rağmen veri erişim hızını azaltmakta ve yüksek bant genişliği gerektirmektedir.

Zayıf güncelleme modellerisistem performansını arttırmasına rağmen bu modellerde senkronizasyon sağlamak zordur. Programcı senkronizasyonu sağlamak için daha fazla ve karmaşık kodlar yazmak zorunda kalır.

Ortak bellek erişiminin kullanım amacına göre uygun bir güncelleme modeli kullanmak performans açısından önemlidir. Bu aşamada sistemde kullanılan veri tipi, verinin kullanım sıklığı ve diğer faktörler uygun güncelleme protokolünü seçmeye yardımcı bilgilerdir.

### 2.7.1 Nedensel tutarlılık

Dağıtık sistemdeki bellek erişimleri, tüm düğümleri ilgilendirecek şekilde potansiyel olarak nedensel bir ilişki içindedir. Sistem içinde bir değişkenin değeri, potansiyel olarak diğer bir değişkene bağlı ise buna nedensel güncelleme denir [10].

Bir düğüm, bir yazma işleminden sonra okuma işlemi yapıyorsa buna nedensel ilişki denir. Yazma işleminden sonra saklanan değer okuma işleminde kullanılan değeri etkiliyor olabilir. Aynı durum yazma işleminin ardından gelen yazma, okuma işleminin ardından gelen okuma işlemi içinde geçerlidir.

Nedensel tutarlılık geçişlilik (transitive) özelliğine sahiptir. Bir A işlemi nedensel olarak B işleminden önceyse ve B işlemi de C işleminden nedensel olarak önce sıralanmışsa, A işlemi nedensel olarak C işleminden önce sıralanmıştır denebilir.

Nedensel olarak diğer işlemlerle ilişkisi olmayan işlemlere eş zamanlı işlemler denir. İşlemlerin farklı makineler tarafından farklı zamanda görülmeleri bir sorun teşkil etmez. Sistemde nedensel olarak ilişkili olan bir değişkene yazma işlemi uygulanması diğer işlemler tarafından aynı anda görülebilmelidir.

### 2.7.2 Delta tutarlılık

Güncelleme tüm sistem düğümleri boyunca yayılır ve sistemdeki tüm düğümler sabit bir $\Delta$ süresi sonunda tutarlı kopyalara sahip olacaktır. Bir başka deyişle yapılan bir okuma işleminden elde edilen değer, yazma işleminden kısa bir süre sonra tutarlı

olacaktır. Düğümde değiştirilmiş bir nesne kopyası varsa ve ilgili nesnenin değiştirilmesinden çok kısa bir süre sonra okuma yapılıyorsa, nesnenin tutarsız olması mümkündür. Yazma işleminden sabit kısa bir süre sonra değişiklik düğümler arasında yayılacak ve tutarlı veri elde edilebilecektir.

### 2.7.3 Giriş tutarlılık

Bu modelde nesne erişimi yapılırken kritik bölgeler belirlenir ve kritik bölgelere giriş işlemlerinde "elde et", çıkış işlemlerinde ise "serbest bırak" komutları kullanılır. Bu yöntemde her bir ortak bellek, senkronizasyon değişkenleri olarak ifade edilen kilit ve bariyer değişkenleri ile birlikte kullanılır.

### 2.7.4 Serbest tutarlılık

Paylaşılmış veriye erişim için "elde et" ve "serbest bırak" senkronizasyon değişkeni kullanılır. Bu iki değişken arasında veri ile ilgili işlemler gerçekleştirilir. Kullanılan "elde et" senkronizasyon değişkeni kritik bölgeye girilmek üzere olduğunu, "serbest bırak" değişkeni ise kritik bölgeden çıkılmak üzere olduğunu gösterir. Bu modelde senkronizasyon sağlamak için bariyer mantığını kullanmak da mümkündür. Bir programda senkronizasyon "n+1" nci mantıksal seviyede başlıyorsa, bütün işlemler n. seviyeyi bitirinceye kadar bu bariyere gelen işlem, diğerlerinin de aynı seviyeye gelmesini bekler.

### 2.7.5 Sıralı tutarlılık

Bü modelde tüm işlemciler arasında belleğe yapılan erişim için bir sıra söz konusudur ve herbir işlemci ilgili belleğe programın belirlediği sırayla erişir.

### 2.7.6 Zayıf tutarlılık

Her düğüm her yazma işleminin sonucunu görmek istemeyebilir. Bu modele göre önemli olan bir düğüm ya da işlemci ortak bir belleğe erişmek istediğinde her zaman güncel veriyi okuyabilmesidir.

### 2.7.7 Tam tutarlılık

Tüm erişimler, tüm düğüm ve işlemciler tarafından aynı sırada görülür. Böylelikle tüm düğümlerde her zaman tutarlı veri saklanır. Tek işlemcili mimarilerin tam güncelleme modeli kullandığı kabul edilir.

Tam tutarlılık modeli en güvenilir güncelleme modelidir. Tüm düğüm ve işlemcilerin herhangi bir anda ortak belleğe erişme isteği sonucunda her zaman en güncel veri elde edilir. Bu yaklaşım dağıtılmış sistemler için en ideal model olsada sistemde performans ve veri tutarlılığını sağlamak zordur.

## 2.8 Veri Güncelleme Protokolleri

Güncelleme modellerine ek olarak güncellemenin ilgili makine üzerinde nasıl yapılacağını ifade eden güncelleme protokolleri vardır. Ortak bellek erişiminde sistem gereksinimine göre iki temel güncelleme protokolü vardır. Bu protokoller yazma/iptal etme ve yazma/güncelleme protokolleridir.

### 2.8.1 Yazma/iptal etme

Bu protokol genellikle MRSW (çok okuyuculu tek yazıcılı) algoritmalarda kullanılmaktadır. Herhangi bir düğüm veriye erişmek isteğinde; veri sadece okunabilir moddadır ve bir veya birkaç işlem tarafından okunmuştur ya da okunmuştur ve devamında bir işlem tarafından yazılmıştır.

Okunabilir modda olan bir veriyi talep eden tüm düğümler veriyi kendi belleklerine kopyalayabilir ve bu veriyi kendi yerel belleklerinden istedikleri kadar kullanabilirler.

Okunabilir kopya almış bir düğüm paylaşılan belleğe yazmak istediğinde, diğer düğümlerin güncel veriyi kullanmasını sağlamak için tüm düğümlere yeni veriyi göndermek yerine, bu düğümlerin sahip oldukları verinin artık geçerli bir değer taşımadığı bilgisini gönderirler. Bu bilgi verinin geçerli olup olmadığını gösteren bir işaretçiyi günceller. Düğümler, paylaşılan veriye erişmek istediğinde öncelikle bu işaretin yönünü kontrol ederler ve verinin güncel olduğunu anlarlarsa doğrudan kullanırlar aksi halde ağ üzerinde veriye yazma işlemi yapan düğümü bulup nesnenin bir kopyasını belleklerine alırlar. Sistemde yazma işlemi gerçekleştirecek bir düğüm yapacağı işlemi tüm düğümlere duyurarak veriyi geçersiz kılar.

### 2.8.2 Yazma/güncelleme

Gerçeklemesi yazma/iptal etme protokolüne göre daha zor olan bir protokoldür. MRMW (çok okuyuculu ve çok yazıcılı) yapısından dolayı tam kopyalamalı algoritma ile birlikte kullanılır. Yazma işlemini tamamlayan bir düğüm, verinin kopyasını bulunduran tüm düğümlere güncel veri değerini gönderir.

# 3. DAĞITIK ORTAK BELLEK SİSTEMLERİ

Nesneler gerçek dünya varlıklarının bir temsilidir. Veriler ve bu verilerin yönetiminde kullanılan davranışları içeren mantıksal varlığa nesne denir. Nesneler bir amacı gerçekleştirmek için yaratılır. Durum ve davranışa sahiptirler.

Nesnenin durumu; nesne için tanımlanmış veriler ve bu verilerin o anki değerlerinden oluşur.

Davranış; veriler üzerinde etkili olan olaylardır. Programlama dillerinde kullanılan metotlar davranışların örneğidir.

## 3.1 Dağıtılmış Nesne

Birlikte çalışmak üzere birden çok bilgisayardan oluşan bir ağda ya da birden çok işlemciye sahip bir bilgisayarda kullanılan yazılım birimleridir [10].

Dağıtılmış nesne ile yerel nesne çeşitli farklılıklara sahiptir [10].

- Yaşam döngüsü; dağıtık bir nesnenin yaratılması, taşınması ve silinmesi yerel nesneye göre farklıdır. Dağıtık sistemde, sistem içindeki tüm bilgisayarlar bu işlemlerden haberdar olmalı ve gerekli davranışı sergilemelidir.
- Referans; dağıtık bir sistemde oluşturulan uzak referans yerel bir nesne referansına göre oldukça karmaşıktır.
- İstek gecikmesi; yerel bir sistemde yapılan nesne erişim çağrısına göre dağıtık bir sistemde yapılan nesne erişim çağrısı süre açısından daha uzundur.
- Nesne aktifliği; dağıtık bir sistemde yapılan bir nesne çağrısı sonucunda karşı sistemde ilgili nesne herzaman aktif olmayabilir.
- Paralellik; dağıtık bir sistem üst düzey paralel çalışma imkanları sunmaktadır.

- Haberleşme; dağıtık sistemler için çok çeşitli ve farklı haberleşme yapıları vardır.
- Başarısızlık; yerel bir sisteme göre dağıtık sistemde nesne erişiminde meydana gelebilecek başarısızlık oranı yüksektir.
- Güvenlik; nesne yerel bilgisayarın dışında bulunabileceği ve ağda taşınacağı için güvenlik oranı daha düşüktür ve güvenliği sağlamak daha zordur.

Dağıtılmış nesnelerin yönetimi, güvenliğinin sağlanması yerel nesnelere göre zordur ve ancak kullanılacak sistemin gereksinimlerini doğru bir şekilde belirleyip yapılan özel yaklaşımlar sayesinde bu zorluklar aşılabilir.

### 3.1.1 Dağıtılmış nesne yapısının sağlayacağı faydalar

Nesneler yapısı gereği veri ile davranışların birlikte çalışmasını sağlayan ve veri yapısında oluşabilecek hataları engellemeye yönelik mekanizmalara sahip varlıklardır. Programcı nesneleri kullanma yetkisi verirken, kullanıcı ile etkileşime geçmesini istemediği veri ve davranışları gizleme hakkına sahiptir.

Nesneler, sahip oldukları veri gizleme özelliği sayesinde veri güvenliğini sağlar ve arayüz ile gerçekleme arasında ayrıma izin verir. Bu sayede nesne yönelimli programla dillerinde aynı arayüze sahip farklı nesneler oluşturulabilir. Bu durum dağıtık bir sistemde veri yorumlamasını kolaylaştırır.

Nesneler programlama hatalarına karşı daha iyi koruma sağlarlar ve üst seviyede anlamsallıkları sayesinde paylaşım için uygundurlar.Modüler yapısı sayesinde yalnızca yeni kısımların eklenmesi ile geliştirilebilirler. Nesnelerin sahip olduğu kalıtım özelliği sayesinde nesne özellikleri genişletilebilir ve gerektiğinde farklı şartlara uyum sağlaması sağlanır.

Nesneler arasında haberleşme metot çağrıları şeklinde ve etkin teknikler kullanılarak üst seviyede gerçekleştirilebilir. Farklı sistemler nesnelerin veri kısımlarını yönetirken yine aynı nesne içindeki metotları kullanacağından tutarsızlık problemi en aza indirgenir.

### 3.1.2 Dağıtık sistemde nesne ve yönetimi

Dağıtık bir sistemde nesneler yönetilirken çeşitli problemlerle karşılaşılır. Paylaşma ve koruma, adlandırma, erişim, süreklilik, yerleşke bulma bunların en önemlileridir [10,14].

#### 3.1.2.1 Paylaşma ve koruma

Dağıtık bir sistemde bir nesnenin birden fazla makine ile ortak kullanılmasına nesne paylaşımı denir. Dağıtık bir ortamda nesne paylaşımı ağ üzerinde mesaj transferi ile gerçekleşir. Nesne paylaşımı standart mesaj iletimine göre daha üst seviyede bir soyutlama sağlar. Eş zamanlı olmayan nesne paylaşımları sürekli nesneler ile sağlanabilir. Eş zamanlı paylaşımlar ise karmaşık yapısından dolayı ve ağ üzerinde eş zamanlılığın sağlanması gerektiğinden bazı farklı mekanizmalar gerektirir. Nesne kopyalarını yerel olarak saklama, nesneleri göç ettirme, nesneleri geçersiz kılma gibi yöntemlerle sistemde nesne paylaşımı sağlanır.

Kullanılacak paylaşım yöntemleri veri tutarlılığını sağlamak amacıyla ek mekanizmalara ihtiyaç duyar ve tutarlılığı performanslı bir şekilde gerçeklemek zordur. Verinin ağ üzerinde kötü amaçlı kişilerin eline geçmesi ve bunun farkına varıp gerekli tedbirleri almak ise sistem tasarımı açısından ayrı bir zorluktur.

#### 3.1.2.2 Adlandırma

Nesnelerin adlandırılması iki aşama olarak düşünülebilir. Birincisi, kullanıcı seviyesinde adlandırma olan kullanıcıların nesnelerin ismini gördüğü ve kullandığı simgesel isimler, ikincisi ise fiziksel adreslere dönüşecek olan alt seviyedeki iç adlardır. İç adlar nesneler arası referanslarda kullanılır ve gizlilik ve koruma konularında da sisteme önemli katkılar sağlarlar. Dağıtık sistemlerde adlandırma çakışmalarını engellemek için genellikle adlandırmadan sorumlu bir birim bulunur. Yeni bir nesne yaratılırken nesne isminin sistemde tekilliği ve uygunluğunu bu sorumlu birim kontrol eder.

Evrensel adlar, tüm çalışma zamanında ve uzayında tekildirler fakat yerel adlar verilen bağlama özeldir.

#### 3.1.2.3 Erişim

Yerel bir sistemde bir nesneye erişim sadece bir adrese dallanmak kadar kolay iken dağıtık bir sistemde nesnelere yapılmak istenen erişim isteği için bir dizi işlem gerçekleştirilir. Bu işlemlerden genellikle kullanıcının haberi bile olmaz. Dağıtık bir sistemde nesneye erişimi; nesnenin sistem üzerinde aranıp bulunması, tip uyumluluğunun kontrol edilmesi, çağrının yerel ya da uzak olduğunun belirlenmesi, nesne henüz bağlanmamışsa nesnenin bağlanması (kod, veri), gerekli ise dinamik metot bağlaması yapılması ve çağrının yürütülmesi aşamaları olarak sıralanır.

Erişim sırasında oluşabilecek hatalar ve bu hatalara karşı alınacak tepkiler (yeniden erişmeye çalışmak, hata mesajı döndürmek gibi) sistem içinde tanımlanmalıdır [14].

#### 3.1.2.4 Süreklilik

Süreklilik; nesnenin bir kopyasının devamlı varolması ve nesnenin yaşam süresinin onu kullanan süreçlerden bağımsız olmasıdır. Dağıtık bir sistemde yaratılmış bir nesne sistem için özel bir çağrı gerçekleşmediği sürece (silme çağrısı) varlığını sürdürmelidir.

#### 3.1.2.5 Yerleşke bulma

Dağıtık bir sistemde nesnenin güncel bir kopyasının nerede olduğunu bulmak önemli bir problemdir. Özellikle hareketli sistemlerde nesneler yer değiştirdikleri için yerleşke takibini yapan ayrı bir birim belirlemek gerekir. Nesne yerleşkeleri bulunurken nesne isminden yola çıkılarak sistem üzerinde arama yapılır. Sistem içinde isim ve yerleşkeyi eşleştiren tablolar tutmak yerleşke bulma işlemini kolaylaştırır.

### 3.2 Dağıtılmış Nesne Modelli Sistemler

Dağıtılmış sistemlerde nesnenin sabit konumlu ya da hareketli olması, haberleşme şekli, nesnenin bütünlüğü gibi unsurlar gözönünde bulundurularak farklı dağıtık nesne modelli sistemler geliştirilmiştir. Geçmişte bu sistemler daha çok donanım ağırlıklı olmasına rağmen günümüzde yazılım ağırlıklı sistemler ön plandadır. Dağıtılmış nesne modelleri üç başlık altında sınıflandırılır.

Süreçler arası haberleşme (IPC); süreçler arası haberleşme protokolünü kullanan modellerdir. Bu modele göre dağıtılmış ortamdaki süreçler ağ üzerinde mesaj alış

verişi yoluyla ya da paylaşılan bir veri üzerinde yer alan işlemler aracılığıyla haberleşirler.

Vekil tabanlı (proxy based) nesne modelleri; ağ üzerinde kullanıcıya saydam olarak gerçekleşen bir sistemdir. Nesne tek bir makine üzerinde bulunmaktadır ve istekçinin nesne üzerindeki herhangi bir metoda çağrı göndermesi istekçi tarafında yaratılan bir vekil aracılığıyla olur. Java remote method call (RMI) mekanizması en çok kullanılan örneğidir [10,15].

Bölünmüş nesne modelleri; kullanıcılara saydam olarak nesneler bir çok makine üzerinde fiziksel olarak dağıtılır. Kullanıcı tarafında, nesne bir bütün olarak gözükür [16].

### 3.2.1 Süreçler arası haberleşme (IPC)

Bir ya da daha çok süreç arasında iplikler vasıtasıyla veri alışverişi sağlayan metotlardır. Süreçler ağ üzerinde bir ya da daha fazla bilgisayar üzerinde çalışabilir. Bant genişliği, ağ üzerindeki gecikme, ipliklerin performansı ve taşınan verinin tipi gibi değişkenler sistem performansını etkiler.

Bu sistem nesneler üzerine yoğunlaşmamıştır. Ağ üzerinde kaba veri kümeleri iletmeye odaklanılmıştır. Mesaj iletimi temelli bir protokol olduğu için geliştirimi zordur. Yerleşim, süreklilik, kopyalama ve tutarlılık yönetimi uygulamaya bırakılmıştır. Her uygulama bu özellikleri kendisi geliştirir.

#### 3.2.1.1 PVM (Paralel Virtual Machine)

Amaç ağ üzerinde heterojen yapıdaki makinelerin genel amaçlı hesaplama sistemi olarak görülmesini sağlamaktır. PVM Tennessee Üniversitesi, Oak Ridge National Laboratory ve Emory Üniversitesi tarafından geliştirilmiştir. PVM, dağıtık ve grid hesaplamalarını gerçekleyen güncel sistemlerin gelişmesi için önemli bir adım olmuştur. Yazılım taşınabilir özelliktedir ve büyük hesaplama yükü getiren problemler pek çok bilgisayarın bir araya gelmesiyle sağlanan hız ve bellek sayesinde çözülebilir. PVM, mesaj iletimi ve alımı, süreçlere çağrıda bulunma, paylaşılan bellek, yayın, karşılıklı dışlama, bariyer ile senkronizasyon gibi ilkeleri içerir. Senkron ve asenkron süreç başlatılmasına olanak sağlar [17].

#### 3.2.1.2 MPI (Message Passing Interface)

Çeşitli paralel bilgisayarlar üzerinde çalışmak üzere ağ üzerinde mesaj iletimi tabanlı bir sistemdir [18]. Fortran ve C programlama dilleri için yazılmış kütüphanesi mevcuttur. Ölçeklenebilir ve taşınabilir paralel sistemler yaratmayı sağlar. Noktadan noktaya ve tümleşik iletişimi destekler. MPI standardı, mesaj iletim yeteneklerinin geniş bir arayüzü ve işlevselliği sağlaması amacıyla geliştirilmiştir.

MPI'ın performansı, ağ ve kullanılan makinenin performansına sıkı bir biçimde bağlı olduğundan sistemi daha verimli kullanabilmek için ilgili makine üzerinde çeşitli ayarlar yapmak gerekir.

#### 3.2.1.3 DCE (Distributed Computing Environment)

İstekçi ve sunucudan oluşan çalışma ortamı ve araçlar içerir. Çalışma ortamı RPC (uzak metot çağırımı), zaman servisi, yetkilendirme servisi ve DFS (dağıtık dosya sistemi) içerir [19].

Çok sayıda istemci ve sunuculu ortamların yayılması sonucu ortaya çıkmıştır ve dağıtık bir yapının oluşması için geliştirilen bir sistemdir. DCE sisteminde yer alan temel bileşenler:

- Güvenlik sunucusu; yetkilendirmeden sorumludur.
- CDS (hücre dizin sunucusu); kaynakları ve erişim kontrol dizinlerini saklamakla sorumludur.
- Dağıtık zaman sunucusu; yüksek doğruluklu zaman bilgisi saklar.

#### 3.2.1.4 NFS (Network File System)

Ağdaki bilgisayarların ortak dosya sistemine yerel disklerindeki kadar kolay ulaşmasını sağlayan RPC temelli dağıtık dosya sistemi yapısıdır. Farklı makineler, ağ mimarileri ve işletim sisteminden oluşan heterojen bir yapı üzerinde çalışma özelliğine sahiptir [9-11].

NFS sayesinde bir makinede yer alan ayrılmış bir disk bölümü, farklı makineler tarafından okunabilir ve yazılabilir. Sistem kullanıcıya saydamlık sağlar. Özellikle büyük organizasyonlarda disk alanından tasarruf sağlamak için kullanılır.

### 3.2.2 Vekil tabanlı nesne modeli ile gerçeklenmiş sistemler

Nesne tabanlı birçok sistem vekil tabanlı modeli kullanmaktadır. Bu modelde nesne tek bir düğüm üzerinde bulunur ve dağıtık sistemde yer alan diğer düğümler nesnenin bulunduğu düğüme erişir. Bu işlem, istemci düğümlerin nesneye erişmek için bir vekil atamasıyla ve bu vekil üzerinden yapılan çağrılarla gerçekleşir [20].

Nesne tabanlı bir model kullandığı için büyük ölçekli uygulamalar oluşturmayı kolaylaştırır ve iyi bir mimari model sunar.

#### 3.2.2.1 Spring

Sayfalama, adlandırma, dosya sistemleri, ağ işlemleri gibi hizmetler kullanıcı seviyesi sunucular olarak gerçeklenmiştir ve sistem nesneye yönelik arayüzler sağlar [21]. İstemci uzak bir nesneye erişmek istediğinde nesne üzerinde bulunan bir metoda çağrıda bulunur.

Nesneleri sunucular yönettiği için dağıtık sistemde bulunan tüm düğümler sunucu üzerindeki hizmetleri kullanma hakkına sahiptir. Spring ayrıca sunucu tabanlı olmayan nesneleri de desteklemektedir.

#### 3.2.2.2 DCOM (Distributed Component Object Model)

Dağıtık sistemler üzerinde çalışan ve nesneye yönelik RPC için geliştirilmiş bir modeldir. COM'un (nesne bileşini modeli) geliştirilmesi ile dağıtık bir yapıya sahip olmuştur. COM'dan farklı olarak ağa verilerin diziliş şekli, ağda oluşabilecek nesne referansları ile ilgili problemler ve bant genişliği kullanımı ile ilgili geliştirmeler içerir [22].

Sistemin dağıtık yapısından kaynaklanan problemleri çözmede RPC mekanizmasının mevcut alt yapısından faydalanılmıştır.

#### 3.2.2.3 CORBA (Common Object Request Broker Architecture)

Obje Yönetim Grubu (OMG) tarafından tasarlanmış dağıtık sistemlerde yazılım tabanlı nesne paylaşımını amaçlayan bir mimaridir. Farklı işletim sistemleri, programlama dilleri ve donanımlar ile uyumlu bir şekilde çalışma özelliğine sahiptir. Nesne yönelimli programlama ile çok benzer bir tasarım mimarisi vardır [10,23]. Nesne yönelimli bir mimari kullanmasına rağmen, CORBA kullanan sistemlerin nesneye yönelik olma zorunluluğu yoktur.

Yakın ve uzak nesne çağrıları için standard bir yapısı vardır. Objeleri genelleştirmek amacıyla bir IDL (Arayüz Tanımlama Dili) kullanılır ve bu sayede farklı programlama dillerince oluşturulan nesneler ortak bir dilde ifade edilir. Dağıtık nesnelere erişme aşamaları:

- İstemci düğümde nesne erişim arayüzü ile ilklendirmeler yapılır.
- Nesne istem aracısı (object adapter) ile bağlantı kurulur.
- Kullanılan ortam ve programlama diline bağlı olarak yaratılmış olan nesne sınıfları arayüz tanımlama dili yapısına uygun olarak dönüştürülür ve gerekli kayıt işlemleri yapılarak sistem kullanıma hazır hale getirilir.

Bazı programlama dillerinin IDL dönüşümü CORBA mimari yapısına yakınlığından dolayı kolay olurken (Java), bazılarını dönüştürmek için (C++, C) karmaşık kodlama ve zaman zaman kullanıcı müdehalesi gerekmektedir.

CORBA kullanıcının nesnelere saydam olarak erişmesini sağlar. CORBA mimarisinde yer alan ORB (nesne istek aracısı) istemci/sunucu ilişkilerini kuran bir arayüzdür. İstemci ORB ile yakın ilişkilidir. Nesne erişimi yerel bir çağrı yapar gibi ORB arayüzünün sağladığı metotlar aracılığıyla yapılır.

### 3.2.3 Bölünmüş nesne modelleri

Nesneler bir bütünü teşkil ettiğinden ince taneli olarak düşünülür. Ağ üzerinde nesneler bir bütün olarak taşınabilmesine rağmen yapısı gereği büyük veri alanı kaplayan nesneler iri taneli olarak düşünülür. Bölünmüş nesne modelinde iri taneli nesneler kullanıcıdan saydam olarak küçük alt parçalara bölünüp ağ üzerinde yayılırlar [24,25]. Dağıtık sistemlerde bant genişliği ve nesne işlem süresi düşünüldüğünde iri taneli nesneler sorun teşkil etmektedir ve bu yüzden iri taneli nesneleri ince taneli olacak şekilde alt parçalara bölmek sistem performansını arttıracaktır.

Nesneler parçalanırken veri bütünlüğünün korunması konusuna dikkat edilir. Bir nesnenin parçalanması koruma, yük dengeleme, başarım ve kullanılabilirlik konularında dağıtık sistemlerde avantaj sağlar. Parçalanmış nesneler kullanıcı bakış açısına göre bir bütündür ve kullanıcı bir nesneye erişirken bir arayüz üzerinden nesnenin tamamını elde ediyor hissi ile erişim yapar. Sistem içinde ise bir nesne birbirleriyle iş birliği yapan alt nesnelerin birleşmesiyle oluşur. Nesnelerin dağıtımı,

aralarında haberleşmesi, hata durumunda yapılacak işlemler gibi konular tasarlanan sistem tarafından çözülür.

Bölünmüş nesne modelini kullanan sistemler alt nesneler için ayrı bir mekanizma tasarlarlar. Bir nesneye erişim yapılmak istendiğinde, aslında ağ üzerinde ilgili nesne alt parçalar halinde bulunmaktadır ve sistem kullanıcının hangi alt nesneye erişmek istediğini, alt nesnenin hangi yerleşkede bulunduğu gibi problemleri kendi içindeçözer. Ayrıca tek parçalı bir nesnenin tutarlılığının sağlanmasına göre parçalanmış bir nesnenin tutarlılığının sağlanması için ek mekanizmalar gerekmektedir.

# 4. ETMENLER

Etmenler için farklı tanımlar bulunmaktadır. Etmenleri aşağıdaki özellikleri içeren varlıklar olarak tanımlayabiliriz [26-28]:

- Başkaları adına özerk bir biçimde hareket edebilme.
- Proaktif (çevresini etkileyen işlemler yapabilme) ve reaktif (çevresindeki değişikliklere tepki verebilme) düzeyde kendi eylemlerini gerçekleştirebilme.
- Öğrenme, işbirliği ve hareketlilik özelliklerini belirli bir düzeyde sergileyebilme.

Bu tanımlama yazılım tabanlı etmenler için geçerlidir. Yazılım tabanlı etmenler yukarıdaki özellikleri içerir. Bilgisayar ve ağı yöneterek, bilgisayar tabanlı görevlerle kullanıcıya yardımcı olurlar. Çeşitli dış etkilere karşı bir etmenin neler yapıp yapamayacağına kullanıcı yazılımı karar verir. Bir etmenin davranışı gerektiğinde daha üst düzeyde tanımlanmış bir etmen tarafından yardım amacıyla değiştirilebilir.

Birden fazla etmenin belirli bir amacı gerçekleştirmek için uyumlu bir şekilde haberleştikleri ve gerek kendilerinden, gerekse çevresinden kaynaklanan olaylara tepki verdikleri ortama çoklu etmen sistemi denir.

## 4.1 Etmenlerin Özellikleri

Etmenler standart yazılım objelerinin özelliklerine ek olarak özel amaçları için ek özellikler içerirler. Yapılacak uygulamanın gereksinimine göre etmenler bu özelliklerin gerekli olanlarını içerirler [26,27].

Proaktiflik; çevresini etkileyebilecek işlemler yapabilme yeteneğidir. Proaktif etmenler sadece çevresinde meydana gelen olaylara tepki vermezler, aynı zamanda çevresini etkileyecek değişiklikler ve etkiler yaparak ortamdaki diğer etmenlerin tepki vermesine neden olurlar. Bu işlemler gerçekleşirken bir kullanıcı müdehalesi olmaz. Amaç ve durum değişiklikleri de otomatik olarak etmen içindeki karar

mekanizmaları ile meydana gelir. Bu özellik etmenleri makineler arası iletişim (M2M) için uygun kılar. Trafik kontrol sistemleri, iletişim ağları buna örnek olarak gösterilebilir.

Haberleşme; daha önce anlaşılmış bir iletişim dili kullanarak diğer etmen ve kullanıcılar ile haberleşme yeteneğidir. Bu yetenek sayesinde etmenler çevresindeki ve içsel olayları birbirlerine duyurur ve gerekli tepkileri verirler.

Adaptiflik; çevresinde oluşan olay ve durumları öğrenerek gerekli tepkileri alıp sistem içinde uyumlu bir şekilde çalışma yeteneğidir.

Reaktiflik; çevrede meydana gelen değişikliklere doğru ve uyumlu bir şekilde tepki verebilme yeteneğidir.

Açıklık; yeni özellikler eklenebilme ve mevcut özelliklerin geliştirilebilir olma özelliğidir.

Mantıklılık; farklı koşullara bağlı olarak hedef ve davranışlarını, amacını tam anlamıyla yerine getirebilmek için değiştirebilme yeteneğidir.

Süreklilik; sorumluluğunu yerine getirene kadar ve uzun bir süre sonlanmadan çalışabilme yeteneğidir.

Hareketlilik; etmenin gerektiğinde bulunduğu ortamdan başka bir ortama taşınabilmesidir. Ortamlar farklı bir fiziksel konumda olabileceği gibi aynı konumda da olabilir ve etmenler taşınma sırasında tüm durum ve özelliklerini korurlar.

Bağımsız hareket etme; etmenlerin kendi başlarına karar verebilme yeteneğidir. Bağımsız hareket etme özelliği tüm etmenlerde bulunması gereken en temel özelliklerden birisidir.

Çok yönlülük; farklı ortamlarda kullanılabilme özelliğidir.

Etmenler içerdikleri bu özellikler aracılığıyla kullanıcıya esnek bir çalışma ortamı sunmakta ve bazı temel özellikleri kendi içerisinde yerine getirerek daha basit ve hatasız çalışmaya yardımcı olmaktadır.

### 4.2 Etmen Tipleri

Etmenler sahip oldukları özelliklere göre farklı sınıflara ayrılır [26,27]. Tüm etmenler bağımsız hareket etme özelliğine sahiptir ve bu özelliğin yanında sahip

olunan diğer özellikler etmenin tipini belirtmektedir. Ayrıca etmenlerin görevleri, kontrol mimarileri, sahip oldukları iç durumlar, duyarlılık düzeyleri, eylemlerinin etki alanı gibi özellikler de etmenin tipinde önemli rol oynar [26].

### 4.2.1 Bilgi etmenleri

Bilgi etmenleri, çeşitli dağıtık kaynaklardan gelen bilgilerin toplanması, düzenlenmesi ve yönetiminden sorumludur. Bilgi etmenleri yalnız ya da birlikte çalışabilir, durgun ya da hareketli olabilir ve öğrenme yeteneğine sahip olabilirler.

Bilgi etmenleri yerel ya da ağ üzerinde çeşitli sorgulamalar yaparak bilgi toplama özelliğine sahiptir.

### 4.2.2 Arabirim etmenleri

Arabirim etmenleri; kontrolü altında bulunduğu etmenler için bazı görevleri yerine getirme ve öğrenme yeteneğine sahip kullanıcı yardımcısı etmenlerdir. Amaçları kullanıcı hareketlerini takip ederek önerilerde bulunmak ve kararlar vermektir. Hem kullanıcılar ile hemde etmenlerle iletişim halinde olduklarından diğer etmenlerden ayrılırlar. Kullanıcılardan çeşitli görevler alabilir ve kullancılara pozitif ya da negatif geri dönüşlerde bulunabilirler.

### 4.2.3 İşbirliği etmenleri

Birlikte çalışma özelliğine sahip etmenlerdir. Karar verme yeteneği, proaktiflik ve reaktiflik özelliğini güçlü bir şekilde gerçeklerler. Bilgi etmenleri kadar öğrenme yeteneklerinin geniş olmasına gerek yoktur. İşbirliği etmenleri birlikte çalışmaya başlamadan önce bazı anlaşmalar yapabilirler. Hedeflerine ulaşmak için diğer etmenlerle iletişim halindedirler.

### 4.2.4 Hareketli etmenler

Hareketli etmenler ağ üzerinde gezme özelliğine sahiptirler. Ağda farklı konumdaki etmenler ile iletişime geçme, bilgi toplama, ağda amaca yönelik olarak çeşitli bölgelerde konakladıktan sonra görevlerini tamamladıklarında tekrar ilk konumuna geri dönebilme gibi özelliklere sahiptirler.

Hareketli etmenleri diğer etmenlerden ayıran kilit nokta, sabit bir konumda yaşamak zorunda olmamalarıdır. Ağ üzerinde iletişim maliyetini azaltma, yerel kaynakları

düşük olan makineler için ağır maliyeti olan işlemleri yerine getirebilme, kolay yönetilme, asenkron çalışma, esneklik gibi üstün özelliklere sahiptirler.

### 4.2.5 Duyarlı (reaktif) etmenler

Etki tepki mantığında çalışan etmenlerdir. İç olaylardan ziyade çalışmakta olduğu sistemde meydana gelen olaylara (etkilere) tepki vermeyi amaçlarlar. Acil durum fonksiyonelliği olması gereken uygulamalarda yoğun olarak kullanılırlar. Etmenler arası haberleşmede basit yapılı mesajlar kullanılır.

### 4.2.6 Melez etmenler

Diğer etmen tiplerinden farklı olarak tek bir özellik üzerine yoğunlaşmaktan ziyade birçok özelliği kabul edilebilir düzeyde içererek farklı uygulamalarda kullanalabilen etmenlerdir. Melez etmenler reaktif ve hareketli olabilir ve arayüz etmenleri gibi diğer etmenlerin tüm özelliklerini ve güçlü yönlerini mümkün olan en iyi seviyede içermeyi amaçlar.

## 4.3 Java Etmen Geliştirme Ortamı (JADE)

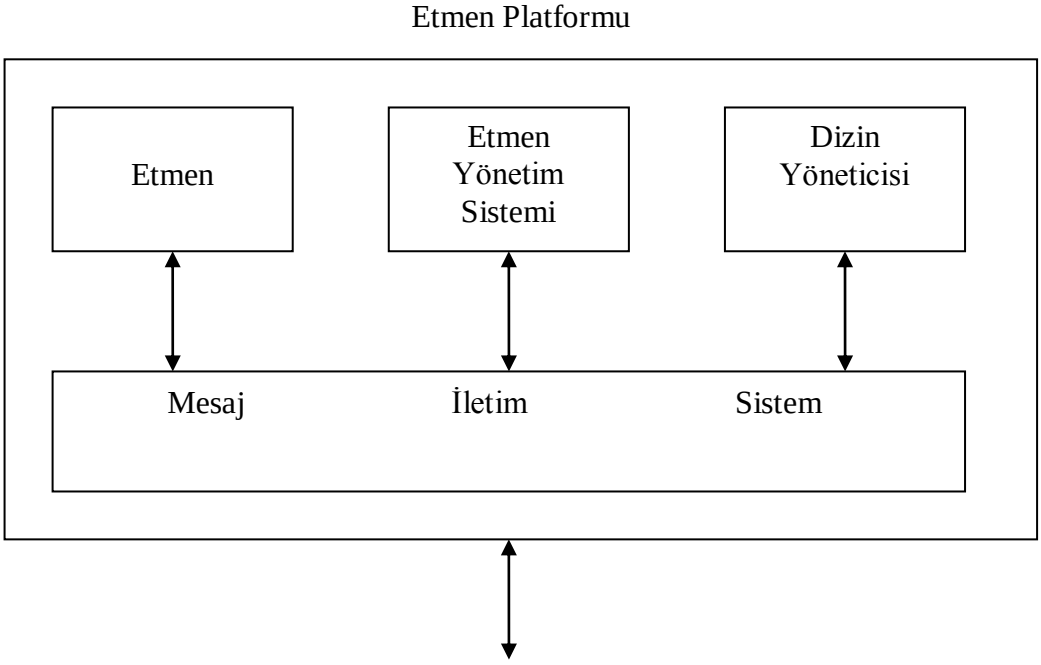
Java programlama dili ile geliştirilmiş bir yazılım sistemidir [28-31]. Çoklu etmen mimarisini, FIPA standartlarını sağlayan bir orta katman yazılımı kullanarak çeşitli ayıklama araçlarının da yardımıyla basit bir şekilde gerçeklenmesini ve dağıtılmasını sağlayan bir sistemdir [28,32]. Etmen platformu makineler arasında dağıtılabilir (makineler aynı işletim sistemine sahip olmak zorunda değildir) ve uzak bir arayüz yardımıyla tüm ayarlar kontrol edilebilir. Sistem yapılandırması gerektiğinde ve istendiğinde yeni etmenler yaratarak ve mevcut etmenleri bir makineden diğer bir makineye taşıyarak değiştirilebilir.

İletişim katmanı mimarisi, her bir etmen için ayrı bir kuyruğa ve ACL mesajlarını esnek ve verimli bir şekilde almayı sağlayan yapıya sahiptir [33]. Etmenler mesaj kuyruklarına; bloklama, sorgulama, zaman aşımı ve örnek eşleştirme ile erişebilirler. JADE sisteminde tüm FIPA iletişim modelleri gerçeklenmiştir [34]. Etkileşim protokolleri, zarf yapısı, ACL, içerik dilleri, kodlama şemaları, ontolojiler ve iletim protokolleri JADE sisteminde gerçeklenmiştir. İletim katmanı kullanıcının istediği geçerli protokolleri seçebileceği şekilde tasarlanmıştır. SL ve etmen yönetim ontolojileri; kullanıcı tanımlı içerik dilleri ve ontolojileri destekleyecek şekilde

etmenlere kayıtlanmış ve otomatik bir şekilde çalışma ortamında kullanılabilecek duruma getirilmiştir.

### 4.3.1 Etmen platformu ve özellikleri

Etmen platformu için FIPA tarafından tanımlanmış standart model Şekil 4.1'de gösterilmiştir [28].




**Şekil 4.1 :** Etmen platformu.

Etmen platformu; etmen, etmen yönetim sistemi (AMS) ve dizin yöneticisinden (DF) oluşmaktadır.

Etmen yönetim sistemi (AMS) platform kullanım ve erişimlerini yönetir. Bir platformda sadece bir tane etmen yönetim sistemi bulunur. Yaşam döngüsü, beyaz sayfa servisleri, etmen tanımlayıcıları ve durumları bilgilerini tutar. Her bir etmen, etmen yönetim sistemine kayıt olarak geçerli bir AID (etmen tanımı) almalıdır.

DF (Dizin Yöneticisi) platformda sarı sayfalar servisini sağlar. Mesaj iletim servisi bir başka adıyla ACC (Etmen İletişim Kanalı) uzak platformlarda dahil olmak üzere platform içinde mesaj alışverişini sağlayan yazılım bileşenidir.

JADE platformu çalıştırıldığında otomatik olarak etmen yönetim sistemi ve dizin yöneticisi yaratılır. AMS ve DF ana taşıyıcıda (main container) yaşarlar. Sistem

içindeki diğer taşıyıcılar ana taşıyıcıya bağlanarak etmen tabanlı çalışma ortamını oluştururlar.

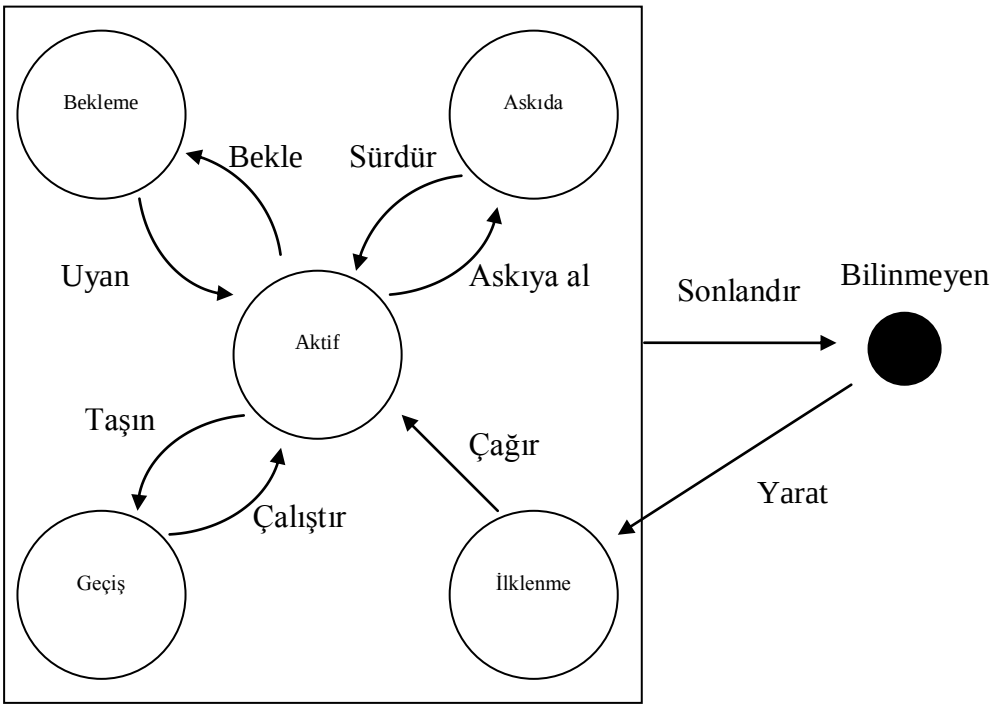
### 4.3.2 Etmenler

Etmen sınıfı, kullanıcı tanımlı etmenler için bir taban sınıf görevi yapar. Programcı gözüyle bir JADE etmeni Agent sınıfından türemiş basit bir Java sınıfı örneğidir [31]. Agent sınıfından miras kalan özellikler vasıtasıyla platform ile etkileşim (kayıtlanma, yapılandırma, uzak yönetim gibi) sağlanır ve etmene özel davranışlar gerçeklenir.

Etmenler çoklu iplik modeline sahiptir ve iplikler eş zamanlı olarak çalıştırılır. Etmenler için herbir servis ve fonksiyonellik bir ya da daha fazla davranış (behaviour) ile sağlanır. Zamanlayıcı (schedular), etmen temel sınıfı içinde programcıdan gizlenmiş bir şekilde hazır olarak gelir ve davranışların zamanlamasını otomatik olarak ayarlar.

#### 4.3.2.1 Etmen yaşam döngüsü

Etmenler yaşam döngüsü boyunca FIPA bildirimine göre farklı durumlarda bulunabilirler [28]. Etmen yaşam döngüsü Şekil 4.2'de gösterilmiştir.




**Şekil 4.2 :** FIPA tanımına göre etmen yaşam döngüsü.

İlklenme; etmen yaratılmıştır fakat henüz kendisini AMS'ye kayıt ettirmemiştir. Henüz ismi, adresi yoktur ve diğer etmenler ile haberleşemez.

Aktif (etkin); etmen AMS'ye kayıt olmuştur. İsmi ve adresi vardır ve birçok JADE özelliklerine erişebilir durumdadır.

Askıda; etmen çalışması durdurulmuştur. Etmenin ana ipliği askıya alınmış durumdadır ve hiçbir davranış çalışamaz.

Beklemede; etmen bloke olmuş durumdadır ve devam etmek için bir koşulun oluşmasını beklemektedir (genellikle bir mesajın etmene ulaşması).

Silinmiş; etmen silinmiştir, ipliği sonlanmıştır ve AMS'den kaydı silinmiştir.

Geçiş; hareketli bir etmen yeni bir konuma göç ederken (taşınırken) bu duruma geçer. Sistem etmene gelen mesajları tamponlamaya devam eder ve etmen yeni konuma geçişini tamamlayınca mesajlar iletilir.

Sistem, etmen yaşam döngüsünü programatik olarak kontrol edebilmek için çeşitli fonksiyonlar sağlamaktadır. Etmenler, davranışları sadece aktif durumdayken çalıştırabilir. Herhangi bir davranış doWait() metodu ile beklemeye alınmışsa tüm etmen ve bu etmenin tüm faaliyetleri bloklanmış durumdadır. Bunun yerine block() isimli metot ile sadece ilgili davranış beklemeye alınabilir.

#### 4.3.2.2 Etmen yaratma

Etmen yaratma işlemi sırasında izlenen yol; etmen kurucu metodu çalıştırılır, etmene jade.core.AID sınıfı aracılığıyla benzersiz bir isim verilir, yeni etmen AMS'ye kayıt edilir, etmenin durumu etkin yapılır ve son olarak etmenin kurulum (setup) metodu çağrılır [28,31]. FIPA standartlarına göre bir etmenin evrensel benzersiz bir isime ve başka etmenlerin yeni yaratılan etmenle haberleşmesi için bir adres kümesine ihtiyacı vardır [28,32].

Programlama aşamasında JADE platformu öncelikle taban sınıftan gelen ilklendirmeleri gerçekleştirir ve programcı etmene farklı özellikler katmak için setup isimli metodun üzerine yazar. Etmenin JADE platformu tarafından tanımlanmış şekilde yürütülmesi bu metodun çağrılması ile başlar. Etmenin setup metodu çağrıldığında, etmen AMS'ye kayıt olmuş ve etkin durumdadır. Programcı setup metodunu şu işleri yapmak için kullanabilir:

- Gerekli görülen durumlarda AMS'ye kayıt edilmiş verileri değiştirmek.
- Gerekli görülen durumlarda, etmenin açıklamasını ve ortama sunduğu servisleri ayarlamak. İhtiyaç duyulursa etmeni birden çok DF'ye kayıt ettirmek.
- Görev kuyruğuna addBehaviour metodunu kullanarak yeni davranışlar eklemek. Davranışlar, setup metodu sonlandıktan sonra çalıştırılırlar ve etmen sistemine yeni özellikler katmak için en sık kullanılan yöntem yeni davranışlar eklemektir.

Sistemde setup metodunun üzerine yazılırken en az bir adet görev (davranış) tanımlanmalıdır. Etmen setup metodunun sonlanması ile otomatik olarak hazır davranış kuyruğundan alınan ilk davranış çalıştırılır ve davranışlar sonlandıkça kuyruktan alınan sıradaki davranış işlenir. Bir davranış çalışırken başka bir davranış onun çalışmasını kesemez. Davranışların zamanlanması round-robin algoritmasına göre yapılır.

#### 4.3.2.3 Etmen sonlandırma

Herhangi bir davranış Agent.doDelete() metodu ile etmen çalışmasını sonlandırabilir. Etmenin kullandığı kaynakları temizlemesi ve kayıt olduğu servislerden kaydını silme işlemlerinin yapılabilmesi için otomatik olarak takeDown() metodu çağrılır. takeDown() metodunun çağrılması ile etmen silinmiş durumuna geçer; yani yok edilecektir. Bu metot çağrıldığında etmenin AMS kaydı silinmemiş haldedir. Başka etmenlere mesaj gönderebilir ve sonlandırılmakta olduğunu haber verebilir. takeDown metodu sonlandıktan sonra etmen için yaratılan iplik sonlandırılır [28,31].

#### 4.3.2.4 FIPA iletişim sınıfı

FIPA standartlarına göre, etmenlerin uygulama tarafından talep edilen işlemleri yapabilmeleri için anlamlı ve bir düzene sahip haberleşme yapıları bulunur [32]. Etmenler arasında mesaj iletim yöntemi, temsil edilme yöntemi (XML gibi) ve diğer özellikleri tanımlayan ACL (Etmen Mesajlaşma Dili) FIPA standardına göre belirlenmiştir. FIPA standartlarına göre bir ACL mesajında bulunan alanlar aşağıdaki gibidir [33]:

- sender: Mesajı gönderen etmenin tanımıdır.

- receiver: Mesajın ulaşması gereken hedef etmen ya da etmenleri içerir.
- reply-with: Alıcı etmenin gönderilen mesaja cevap verirken kullanması gereken alandır. Gönderici etmen bu alan vasıtasıyla hangi mesaja cevap geldiğini ayırt edebilir.
- in-reply-to: Mesajın hangi tipten bir mesaja cevap olarak gönderildiğini belirten alandır.
- reply-by: Mesaja son cevap verme tarihini içerir.
- reply-to: Gönderilen mesajın cevabının hangi etmene gitmesi gerektiğini belirten alandır.
- performative: Mesajın hangi eylemi/amacı gerçekleştirdiğini belirten alandır. FIPA standartlarında belirtilmiştir.
- content: Gönderen etmen tarafından mesajın veri kısmı olarak doldurulur. Veri kısmının anlamlı olabilmesi için gönderici ve alıcı içerikle ilgili önceden anlaşmış olmalı ya da içeriği karşılıklı olarak yorumlayabilmelidir.
- language: İçerik (content)alanının biçimsel dilini (SQL, FIPA-SL, Prolog, vb.) belirtir. Gönderici ve alıcı etmen bu biçimsel dilleri yorumlama yeteneğine sahip olmalıdır.
- encoding: İçerik (content) alanının karakter kodlamasını belirtir. Gönderici ve alıcı etmen kodlamayı yorumlayabilme yeteneğine sahip olmalıdır.
- conversation-id: Etmenlerin karşılıklı olarak kullandıkları mesajlaşmaları özelleştiren alandır. Yapılan haberleşmenin konusunu belirtir ve bu alan kullanılarak mesajlar uygulamaya göre ayırt edilir.
- ontology: Mesajın içerik (content) kısmında yer alan sembollere anlam verebilmek için kullanılan kelime kümesini belirtir. Gönderici ve alıcı etmen kelime kümesini yorumlayabilme yeteneğine sahip olmalıdır.
- protocol: Mesajın FIPA tarafından tanımlanmış bir protokole göre gönderilmesi halinde kullanılan protokolü belirten alandır.

Etmenler arasında gönderilen mesajların performative (bildirim, eylem) alanı FIPA standartlarında açıkça belirtilmiştir. Bu eylemler tüm kullanım senaryoları düşünülerek oluşturulduğundan, kullanıcı tarafından farklı bildirim tipleri kullanılmaması genel anlaşılırlık açısından iyi olacaktır. FIPA standartlarına göre oluşturulan bildirim tipleri aşağıdaki gibidir [34].

- CFP: Mesajı alan etmenden bir öneri/teklif beklendiğini belirten alandır.
- confirm: Gönderici etmenin alıcı etmene bir önermenin doğru olduğunu ya da önermenin kabul edildiğini belirttiği alandır.
- disconfirm: Gönderici etmenin alıcı etmene bir önermenin yanlış olduğunu ya da önermenin kabul edilmediğini belirttiği alandır.
- accept-proposal: Alıcı etmene daha önce iletilen propose mesajına karşılık verilen kabul mesajıdır.
- query-if: Mesajı gönderen etmenin, alıcı etmene bir önermenin doğru olup olmadığını sorduğu mesajdır.
- query-ref:Mesajı gönderen etmen, alıcı etmenden bir nesneyi ister ve bu isteği yaparken ilintili nesnenin kaynağını belirten bir ifade yollar.
- propagate: Gönderici etmenin, mesaj içinde gömülü bir mesajın, alıcı etmen tarafından yine mesaj içinde belirtilen etmenlere göndermesini istediği mesaj tipidir.
- propose: Gönderici tarafından, alıcıya gönderilen bir öneri mesajıdır. Alıcı etmen uygun koşullarda bu mesaja karşılık gerekli eylemleri gerçekleştirir.
- proxy: Gönderici etmen alıcı etmeni vekil olarak kullanır. Mesaj içinde yer alan bir açıklamaya göre alıcı etmen hedef etmenleri seçerek ilgili mesajı iletir.
- agree: Alıcı etmene daha önce gönderilen request mesajına karşılık olarak gönderilir. Yapılması istenen eylemin kabul edildiğini belirten mesajdır.
- cancel: Bir etmen daha önce mesaj göndererek bir eylem gerçekleştirmesini istediği etmene artık daha önce yapmasını istediği eylemin geçerli olmadığını belirttiği mesajdır.

- request: Gönderici etmenin alıcı etmene bir eylem gerçekleştirmesini belirttiği mesajdır.
- request-when: Alıcı etmenden belirli bir koşul sağlanması halinde bir eylemi gerçekleştirmesi istenen mesajdır.
- request-whenever: Gönderici etmenin alıcı etmene belirli bir koşulun her oluştuğunda periyodik olarak gerçekleştirmesini istediği eylemi belirttiği mesaj tipidir.
- failure: Daha önce yapılması istenmiş bir eylemin başarısız olduğunu belirten mesajdır.
- inform: Alıcı etmene bir önermenin doğru olduğunun bildirildiği mesaj tipidir.
- inform-if: Alıcı etmene bir önermenin doğru ya da yanlış olduğunun bildirildiği mesaj tipidir.
- not-understood: Gönderici etmenin alıcı etmenin yapmış olduğu bir eylemi anlamadığını belirttiği mesaj tipidir.
- refuse: Daha önce yapılması istenmiş bir eylemin ilgili etmen tarafından reddedildiğini belirten mesajdır.
- reject-proposal: Daha önce ilgili etmene gelen propose mesajı ile yapılan önerinin kabul edilmediğini belirten mesajdır.
- subscribe: Bu mesajla gönderici, alıcı tarafında gerçekleşen seçilmiş bir eyleme kayıt olur ve bu eylem her gerçekleştiğinde haberdar olmak ister.

Etmenler, aralarında gerçekleşen mesajlaşmalarda mesajı ve içerisindeki bilgileri almak ve anlamak için, mesajın içeriğinde yer alan biçimsel dili ayrıştırabilmelidir. İlgili biçimsel dilde yer alan kavram ve semboller de karşılıklı olarak anlaşılır olmalıdır. Kavram ve semboller kümesi kullanılarak oluşturulan biçimsel dillere ontoloji denir [28].

#### 4.3.2.5 Etmenler arası haberleşme

Etmen sınıfıetmenler arası haberleşme için çeşitli metotlar içermektedir. FIPA bildirimine göre etmenler asenkron mesaj transferi ile haberleşir ve haberleşmede veri olarak ACLMessage sınıfının nesneleri taşınır.

Etmen sınıfında(Agent.class) yer alan send() metodu, ACL mesajlarını göndermeyi sağlar. ACLMessage sınıfı içinde yer alan receiver alanı mesajı alacak etmenlerin tanımlarını saklar. Metot çağrısı etmenlerin nerede yaşadığı (yerel ya da uzak) bilgisinden bağımsızdır [28].

### 4.3.3 Etmen haberleşme mesajları (ACL)

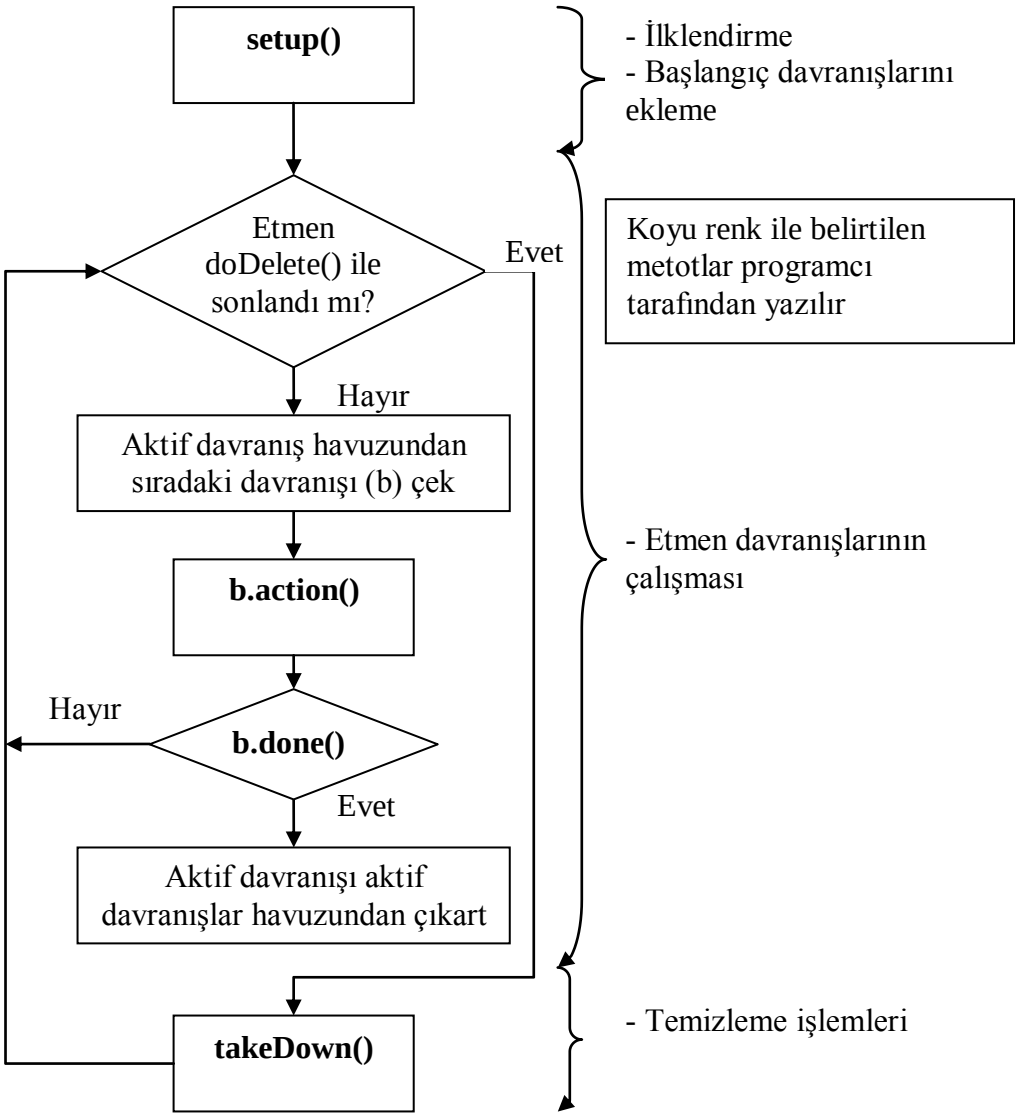
Etmenler arası bilgi alışverişinde kullanılan mesajları ACLMessage sınıfı temsil eder. Bu sınıf FIPA bildiriminde yer alan özellikleri içerir.

Mesaj göndermek isteyen bir etmen; yeni bir ACLMessage nesnesi yaratmaları, nitelikleri uygun değerler ile doldurmalı ve son olarak Agent.send() metot çağrısını yapmalıdır. Benzer şekilde mesaj almak isteyen bir etmen Agent sınıfı içinde yer alan receiver() ya da blockingReceive() metotlarını çağırmalıdır.

Mesaj gönderim ve alım işlemleri alıcı ve gönderici davranışları eklenerek zamanlandırılabilir.

### 4.3.4 Etmen iş parçacıkları

Bir etmen dış olaylara tepki verebilecek şekilde eş zamanlı görevcikleri yerine getirebilmelidir. Etmen yönetimini verimli bir şekilde greçekleştirmek amacıyla her bir etmen tek bir iplik üzerinde çalışır ve tüm görevcikleri davranış olarak modellenir. Çoklu iplik içeren etmenler JADE platformunda gerçeklenebilmesine rağmen ACL mesaj kuyruğu senkronizasyonu dışında özel bir platform desteği yoktur.




**Şekil 4.3 :** Etmen ipliğinin çalışması.

Şekil 4.3'te etmen ipliğinin akış diyagramı gösterilmiştir. Aktif davranış havuzunda bir davranış olduğu sürece etmen yaşamını sürdürecektir.

#### 4.3.4.1 Davranışlar ve çeşitleri

Bir etmenin görevleri yerine getirmesi, davranışlar (behaviour) aracılığı ile olur [28,29,31]. Bir davranış, kalıtım aracılığı ile jade.core.Behaviours.Behaviour sınıfının genişletilmesi ile tanımlanır. Bu davranışın etmen tarafından yürütülebilmesi için etmen sınıfı içerisinde addBehaviour metoduna parametre olarak eklenerek, addBehaviour(b) şeklinde çağrılması gerekir. Davranışlar, etmenin açılışında çalıştırılan setup metodunda veya daha sonra başka bir davranış içerisinden eklenebilir.

Behaviour sınıfını genişleten her sınıf, action ve done özet metotlarını gerçeklemek zorundadır. Kullanılan action metodunda, davranışın çalışması sırasında yapılacak işler tanımlanır. Kullanılan done metodu ile, davranışın çalışmasının sonlanıp sonlanmadığı bilgisi true veya false dönüş değerleri ile JADE platformuna bildirilir. Eğer etmen sonlanmadıysa, action metodunun tekrar çalıştırılması için davranış görev kuyruğuna atılır. Bir davranışın action metodu sonlanmadan başka bir davranışın action metodu çalıştırılmaz. Şekil 4.3'te gösterilen bu yöntemin faydaları aşağıdaki gibi sıralanabilir [20]:

- Her JADE etmeninin bir iplik olarak çalıştırılabilmesini sağlar, böylece performansı az olan sistemlerin de desteklenmesi sağlanır.
- Davranışlar arasındaki geçişler, Java iplikleri arasındaki geçişlerden daha hızlı olduğu için performans artışı sağlanır.
- Aynı anda birden fazla davranış aynı veri üzerinde işlem yapamayacağı için eş zamanlı çalışma sorunlarına yönelik önlem almaya gerek yoktur.
- Davranışlar arasında bir geçiş olduğunda herhangi bir yığın bilgisi kayıt edilmediği için, etmenin anlık durum bilgisi alınabilir. Böylece etmenin daha sonra çalışmasına devam etmesi için sabit bir ortama kaydedilmesi ve hareketli etmenlerin durumlarını ortamlar arasında taşıması kolaylaşır.

JADE kütüphanesi, 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 özet metotları gerçeklenerek genişletilebilir.

#### 4.3.4.2 Davranış sınıfı (Behaviour)

Behaviour sınıfı soyut (abstact) olarak tanımlanmıştır ve diğer davranışlar bu soyut sınıfı genişleterek gerçeklenir. Davranış sınıfı durum geçişlerine izin vererek (başlama, bloklama, yeniden başlama) zamanlayıcı için temel özellikleri gerçekleştirir.

Davranış sınıfı içinde yer alan block() metodu bir olay gerçekleşene kadar (genellikle bir mesaj gelene kadar)ilgili davranışı bloklar. Bu metodun kullanılması diğer davranışları etkilemez ve çok iplikli yapı için ince taneli (fine grained) kontrol sağlar. Bu metot, davranışı bloklanmış davranış kuyruğuna koyar ve action() metodundan

dönüş sağlanır sağlanmaz bu durum etkili olur. Tüm bloklanmış davranışlar etmene bir mesaj gelmesiyle tekrar zamanlanır. Bunun yanında, bir davranış block() metoduna geçtiği bir zaman parametresi ile kendisini milisaniye düzeyinde belirli bir süre bloklayabilir. Bir davranış restart() metodu ile tekrar çalıştırılabilir.Bir bloklanmış davranış şu olaylar sonucunda tekrar çalışır:

- Davranışla ilişkili bir mesajın etmene gelmesi.
- Daha önce block() metoduna bloklama süresi geçilerek yapılmış çağrı sonucu zaman aşımı oluşması.
- restart() metodunun açık bir şekilde çağrılması.

Davranış sınıfı ayrıca onStart() ve onEnd() isimli iki yer tutucu metot içerir. Bu metotlar, kullanıcı tarafından genişletilmiş özellikler ile üzerine yazılarak davranış çalışmasında önce ve sonra gerçeklenmesi istenen işlemler yerine getirilir.

Davranış sınıfı içerisinde yer alan onEnd() metodu, davranış tüm işlerini tamamladıktan ve etmen davranış havuzundan çıkarıldıktan sonra çağrılır. Bu yüzden onEnd() metodu içerisinde reset() metodunu çağırmak, davranışı tekrarlamak için yeterli değildir. Davranış tekrar çalıştırılmak isteniyorsa restart() çağrısından sonra ilgili davranış, etmen davranış havuzuna tekrar eklenmelidir.

Davranış sınıfı çeşitli bilgileri saklamak amacıyla DataStore isimli bir veri alanı içerir. DataStore davranışlar arası veri paylaşımı için kullanışlıdır.

#### 4.3.4.3 Basit davranış (SimpleBehaviour)

Bu davranış Behaviour sınıfını genişleterek basit atomik bir davranış oluşturur. İçerisinde bulunan reset() metodunun kullanıcı tarafından üzerine yazılmadıkça bir işlevi yoktur.

#### 4.3.4.4 Tek kullanımlık davranış (OneShotBehaviour)

OneShotBehaviour sınıfının genişletilmesiyle, bir defa çalışan ve hemen sonlanan davranışlar oluşturulur [28,31]. Bu davranışlar, tüm işlemlerini action metodunun bir defa çağrılacağını göz önünde bulundurarak yapmalıdır. Davranışın bir defa çağrılması, done metodunun varsayılan değer olarak true döndürmesi ile sağlanmıştır.

#### 4.3.4.5 Tekrarlı davranış (CyclicBehaviour)

CyclicBehaviour sınıfının genişletilmesiyle oluşturulan davranışlar, hiç bir zaman sonlanmazlar. Davranışın her çağrılmasında action metodundaki aynı işlemler tekrarlanır [28,31]. Davranışın sonlanmaması, done metodunun sürekli false değerini döndürmesi ile sağlanmıştır. Davranış, davranışlar havuzundan removeBehaviour metodu ile çıkarılabilir.

#### 4.3.4.6 Karma (birleşik) davranış (CompositeBehaviour)

CompositeBehaviour sınıfının genişletilmesiyle oluşturulan davranışlar, farklı sayıda davranışın birleştirilmesi ile oluşur [29,31]. Asıl yapılması gereken işlemler, davranışın kendi içinde gerçekleştirilmez. Birleşik davranış içerisinde yer alan alt davranışların belirtilen zamanlama politikasına göre çalıştırılmasından sorumludur.

CompositeBehaviour sınıfı, zamanlama için sadece bir arayüz tanımlar ve zamanlama politikasını geliştirmek alt davranışların sorumluluğundadır.

#### 4.3.4.7 Sıralı davranış (SequentialBehaviour)

Birden fazla davranışın sırayla çalıştırılmasını sağlayan davranıştır. SequentialBehaviour sınıfı, kalıtımla genişletilmeden kullanılır. Bu sınıfın addSubBehaviour metodu kullanılarak, çalıştırılması istenen davranışlar sırayla eklenir. Davranış çalışmaya başladığında tüm alt davranışları sırayla çağırır. Bir alt davranışın done metodu true döndürdüğünde, alt davranış sonlanmış kabul edilir ve sıradaki alt davranışa geçilir [28,31].

#### 4.3.4.8 Eş zamanlı davranış (ParallelBehaviour)

Birden fazla davranışın paralel olarak çalıştırılmasını sağlayan davranıştır. Kalıtımla genişletilmeden kullanılır. Bu sınıfın addSubBehaviour metodu ile çağrılmasıyla istenen davranışlar listeye eklenir. Paralel davranışlar, geliştiricinin tercihine bağlı olarak bir alt davranış sonlandığında veya tüm alt davranışlar sonlandığında sonlanabilirler. Bu çalışma koşulu, kurucu fonksiyona verilen bir parametre ile belirlenir [28,31].

#### 4.3.4.9 Sonlu durum makine bazlı davranış (FSMBehaviour)

Birden fazla davranışın, sonlu durum makine modeline göre çalıştırılmasını sağlayan davranıştır. FSMBehaviour sınıfı kalıtımla genişletilmeden kullanılır. Sonlu durum

makine davranışını sağlamak için bir durum geçiş tablosu tutulur. Bu durum geçiş tablosundaki her düğüm bir alt davranışı temsil eder. Alt davranış sonlandığında, onEnd metodu çağrılarak dönüş değeri alınır ve çalıştırılacak yeni davranışı belirlemek için durum tablosundaki değerlerle karşılaştırılır. Karşılaştırma sonucunda çalıştırılacak yeni alt davranış belirlenir veya eğer sonlanma durumuna ulaşıldıysa ana davranış sonlandırılır [28,31].

#### 4.3.4.10 Uyanan davranış (WakerBehaviour)

WakerBehaviour sınıfının genişletilmesiyle oluşturulan davranışlar, belirli bir süre geçtikten sonra veya belirli bir tarih ve saatte gerçekleştirilmesi istenen görevler için kullanılır [28,31]. Davranışın çalışması istenen tarih ve saat geldiğinde, davranışın onWake metodu çalıştırılır ve bu metodun çalışması bittikten sonra etmen sonlandırılır.

#### 4.3.4.11 Zamanlanmış davranış (TickerBahaviour)

TickerBehaviour sınıfının genişletilmesiyle oluşturulan davranışlar, belirli zaman aralıklarıyla gerçekleştirilmesi gereken görevler için kullanılır [28,31]. Davranışın kurucu fonksiyonuna parametre olarak verilen zaman aralığı her dolduğunda, onTick metodu çağrılır. Bu metot sonlandığında ilgili davranış zaman aralığı tekrar dolduğunda çalışacak şekilde görev kuyruğuna eklenir.

# 5. ÇOKLU ETMEN ORTAMINDA NESNE TABANLI DAĞITILMIŞ BELLEK PAYLAŞIMI

## 5.1 Giriş

Bu çalışmada çoklu etmen ortamında dağıtık nesne paylaşımı gerçeklenmiştir. Etmenlerin sahip oldukları proaktiflik, hareketlilik, öğrenme, kendi kendine karar verebilme, iş birliği yapabilme gibi özellikler kullanılarak dağıtık nesne paylaşım ortamı sunulmuştur.

Dağıtık sistemin oluşturulmasında JADE (Java Agent Development Framework) sisteminin özelliklerinden faydalanılmıştır [28-31]. Sistemi oluşturan etmenler aynı makinede yaşayabileceği gibi JADE sisteminde aynı platforma bağlı olarak farklı makinelerde de yaşayabilirler. JADE'nin sağlamış olduğu haberleşme altyapısı kullanılarak etmenler arası etkileşim sağlanmıştır.

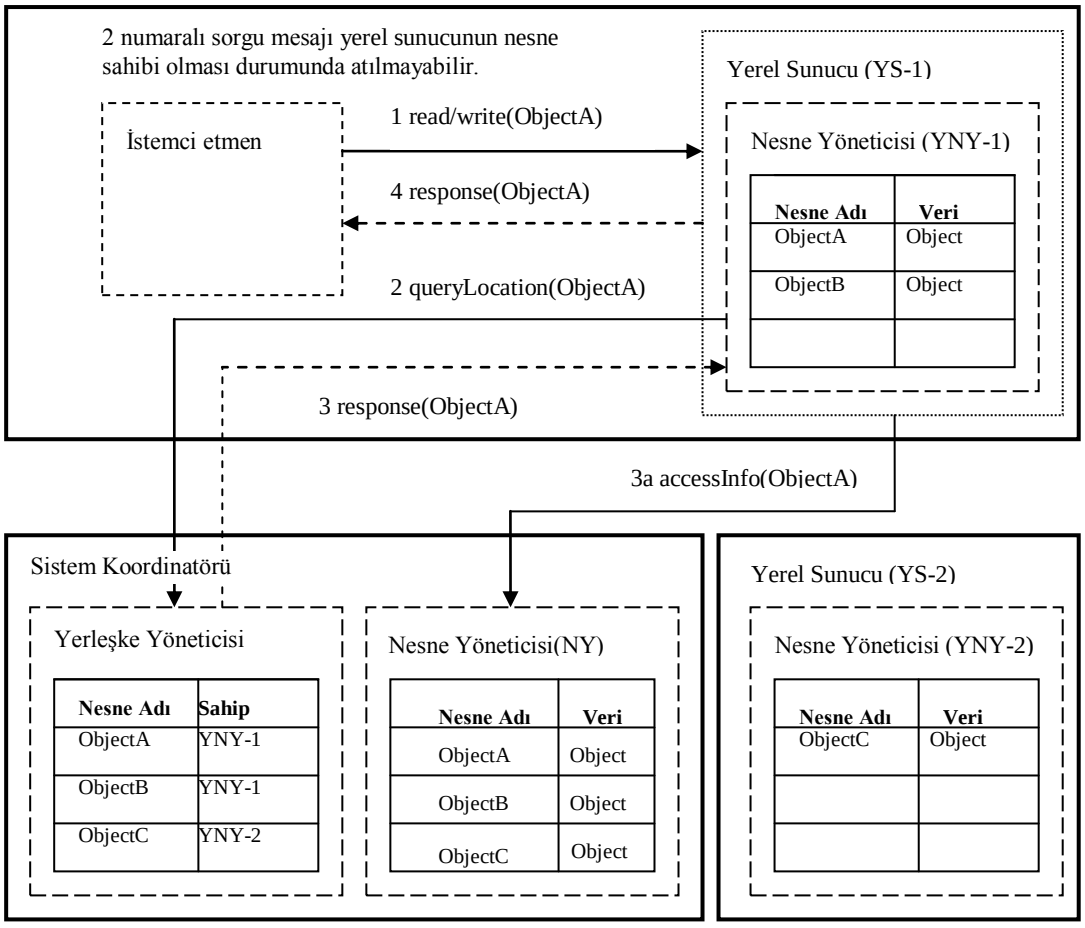
Dağıtık nesne paylaşım sistemi etmenler üzerinde çalışır ve nesne sahipliği sistem üzerinde yazma istekleri neticesinde el değiştirir. Burada el değiştirmek ile ifade edilen işlem nesnenin en güncel kopyasının yazma isteği yapan istemcinin bağlı olduğu yerel yöneticide bulunmasıdır. Nesnenin yazma istekleri ile el değiştirmesi bir etmenin sık yazma isteği yapması durumunda paylaşılmış belleğe erişimini yerel sunucu aracılığıyla yapmasını sağlayacağından erişim hızında artmayı beraberinde getirecektir. Ayrıca bu yaklaşım etmen mimarisinde yer alan hareketlilik kavramı ile uyumludur. Farklı nesnelerin farklı yerel sunucularda bulunabilmesi sağlandığından işlem ve erişim maliyeti azaltılmış ve merkezi yönetim algoritmalarında olduğu gibi bir sunucu üzerine bütün işlemlerin yüklenmesi engellenmiştir.

Yerel ve yönetici sistem alt birimlere ayrılarak görev paylaşımı sağlanmış ve böylelikle gelen istekler için oluşacak kuyruklar sunucu bazında azaltılmıştır. Nesnelerin kopyası erişim istekleri neticesinde ağ üzerinde farklı sunucular üzerinde el değiştireceğinden olumsuz koşullarda (mesaj kaybı oluşması, kullanıcı uygulamadan kaynaklanan hatalı yazma işlemi) nesnelerin ağ üzerinde bulunup kurtarılabilceği bir ortam oluşturulmuştur.

## 5.2 Dağıtılmış Paylaşılan Nesne

Birlikte çalışmak üzere birden çok bilgisayardan oluşan bir ağda ya da birden çok işlemciye sahip bir bilgisayarda kullanılan ve ortak erişime imkan tanıyan nesnelerdir. Dağıtılmış nesneler ağ üzerinde herhangi bir yerde bulunabilir ve bu durum nesnenin yerleşkesinin bulunması, isim nesne ilişkisinin kurulması ve erişim hızı gibi problemleri yanında getirir [10]. JADE etmenleri ile oluşturulan dağıtılmış nesne paylaşım sistemi bu problemler için uygun çözümler bularak dağıtık nesne paylaşımına imkan verir.

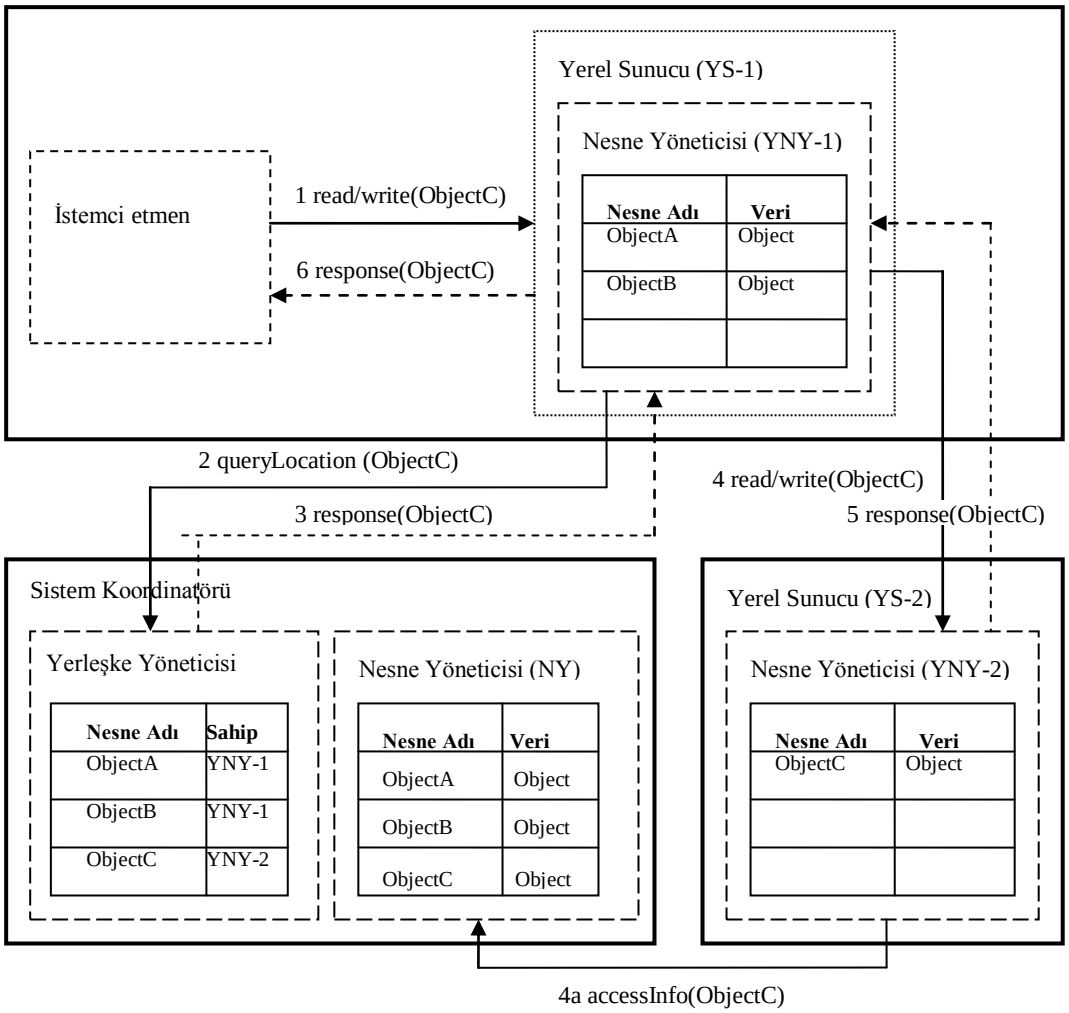
Nesneler yerel ya da uzak düğümlerde saklanabilir. Nesnelerin uzak düğümlerde bulunması nesneyi ağda bulup getirme, taşıma gibi problemleri yanında getirir. Nesneyi yaratıp sistem üzerindeki düğümlerin ortak kullanımına açan düğüme nesne yaratıcısı denir. Nesne yaratıcısına özel yetkiler vermek mümkündür. Gerçeklenen sistemde yaratılmış bir nesneyi silme yetkisi sadece yaratıcı etmene verilmiştir.




**Şekil 5.1 :** Okuma/Yazma isteği sonucu yerel nesne erişimi.

Şekil 5.1‘de okuma/yazma isteği sonucunda bir nesneye yerel sunucu aracılığıyla erişim gösterilmiştir.İstemci tarafından ObjectA isimli nesneye erişim isteği yapılır. Yerel sunucu ilgili isteği yerel nesne yöneticisine aktarır ve yerel nesne yöneticisi, nesne sahibinin kendisi olduğunu sistem koordinatörünün yerleşke yöneticisine yaptığı sorgu ile teyit eder. Yerel nesne yöneticisi nesnenin bir kopyasını istemciye ulaştırır.

Dağıtılmış bir ortamda nesnenin en güncel kopyasını saklayan düğüme nesne sahibi denir. Uzak nesne modelinde, nesneler merkezi bir düğümde saklandığından ve tüm erişimler bu düğümün kontrolünde gerçekleştiğinden nesne sahibi sabittir. Nesne sahipliğinin sabit olması merkezi yapısından dolayı erişim kontrolünü kolaylaştırsa da uzak bir makineye erişmek ve ağ üzerinde nesne kopyasının taşınması genellikle zordur. Sabit nesne sahipliğine alternatif olarak nesne sahipliğinin dinamik olarak el değiştirmesi sağlanabilir. Gerçeklenen sistem çok okuyuculu ve tek yazıcılı mimaridedir ve nesne sahipliği gelen yazma istekleriyle el değiştirmektedir. Yazma isteği yapan bir düğüm aynı zamanda nesne sahipliğini de üstlenerek bundan sonra gelecek okuma isteklerini yanıtlar. Böylelikle ağ üzerinde farklı nesnelerin sahipliği farklı düğümler tarafından üstlenileceğinden erişimlerde oluşabilecek darboğaz engellenmiş, yerel nesne erişimi sağlanmış ve etmenlerin hareketli yapısına uygun bir dağıtık ortam sunulmuştur.




**Şekil 5.2 :** Okuma/Yazma isteği sonucu uzak nesne erişimi.

Şekil 5.2'de uzak okuma yazma işlemi gösterilmiştir. İstemci tarafından ObjectA isimli nesneye erişim isteği yapılır. Yerel sunucu ilgili isteği yerel nesne yöneticisine aktarır ve yerel nesne yöneticisi, nesne sahibinin kendisi olmadığını sistem koordinatörü yerleşke yöneticisine yaptığı sorgu ile anlar. Yerel nesne yöneticisi nesnenin sahibi ise yerleşke yöneticisine sormadan erişime izin verme yetkisine sahiptir. Yerel nesne yöneticisi erişim isteğini yerleşke yöneticisinden edindiği bilgiden yararlanarak nesnenin sahibi olan nesne yöneticisine iletir. Nesne sahipliğini üstlenmiş olan nesne yöneticisi erişimin mümkün olduğu durumda (nesne kilitli olmadığında) nesne kopyasını istemci nesne yöneticisine oradan da istemci etmene iletir.

### 5.2.1 Dağıtılmış paylaşılan nesnenin içyapısı

Oluşturulan sistemde Java nesnelerinin ağ üzerinde dağıtık olarak paylaşılması sağlanmıştır. Sistemde Java yazılım kütüphanesinden gelen Serializable arayüzünü

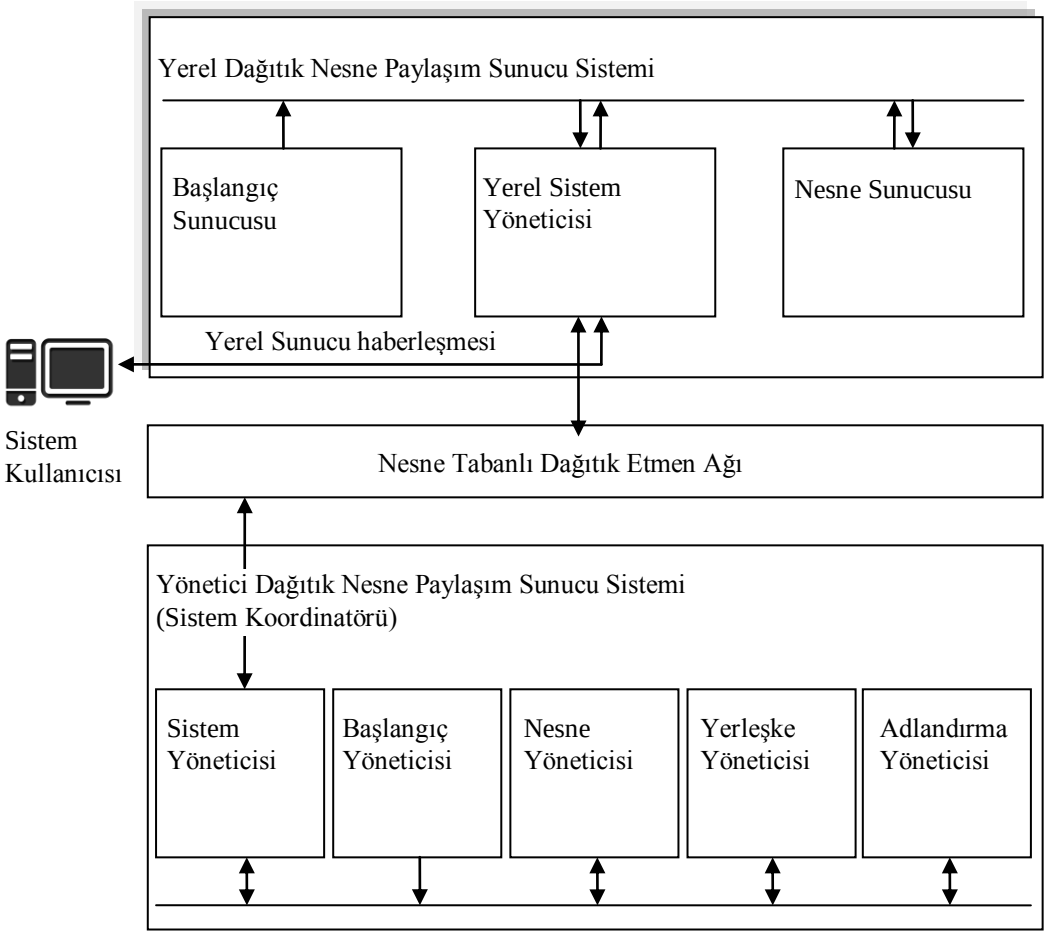
gerçekleyen nesneler paylaşılabilir. Nesneler paylaşılırken ağ üzerinde mesajlar içinde iletildiğinden paylaşılacak nesnenin byte dizisi olarak ağa verilebilir olması gerekmektedir. Java programlama dilinde Serializable (serileştirilebilir) arayüzünü gerçekleyen nesneler byte dizisi haline çevrilebilir ve disk üzerinde saklanabilir. Byte dizisi haline getirilmiş nesneler diskten okunarak tekrar Java nesnesine dönüştürülebilir. Java platform bağımsız bir programlama dili olduğundan serileştirilmiş nesneler farklı işletim sistemlerinden de okunup tekrar Java nesnelerine dönüştürülebilir.

Sistem içinde paylaşılacak nesneler için Serializable arayüzünü gerçekleme gereksinimi dışında bir kısıt yoktur.

### 5.3 Sistem Yapısı

JADE etmenlerinden oluşan dağıtık sistem, etmenlerin birbirleriyle yaptığı mesaj alışverişleri ile nesne paylaşımını sağlar. Mesaj alışverişi ile yapılan nesne paylaşımı tutarlılık problemini ortaya çıkarır. Sistem içinde etmenlere verilen farklı görevler ile tutarlılık probleminin üstesinden gelinir. JADE etmen sisteminde çalışan DOS (Dağıtık Nesne Paylaşım) sistemi, sistem koordinatörü ve yerel sunucu yöneticisi olmak üzere iki temel sunucu sistemi üzerine kuruludur. Bu sunucu sistemleri kendi içinde alt birimlere ayrılarak farklı görevleri üstlenmiştir. Her birim farklı bir etmen tarafından gerçeklenir. Birimler kendi aralarında haberleşme yeteneğine sahiptir. Şekil 5.3'te çoklu etmen ortamında dağıtık nesne paylaşım sistemi mimarisi gösterilmiştir.

Paylaşılan nesnelere yazma, okuma gibi erişimler yapan düğümlere sistem kullanıcısı denir. Sistem kullanıcısı, erişim isteklerinin yapıldığı yazılım noktası ya da doğrudan sistemi kullanan kullanıcı olabilir. Sistem kullanıcısı, paylaşılan nesnelere doğrudan ya da dolaylı olarak yazma/okuma isteği yapan varlıktır. Sistem kullanıcısı, yerel sistem yöneticisi ile iletişim halindedir. Yerel sistem yöneticisi kullanıcıdan aldığı erişim isteklerini mesaj tipine göre ilgili etmene iletir ve gerekli ise yanıtı kullanıcıya ulaştırır.




**Şekil 5.3 :** Dağıtık nesne paylaşım sistemi mimarisi.

Yerel sunucu sistemi ve sistem koordinatörü kendi içlerinde sistem yöneticisine sahiptir ve bu sistemlerin içinde görevli etmenler sistem dışına mesaj göndermek istediklerinde sistem yöneticilerini ağ geçidi olarak kullanırlar.

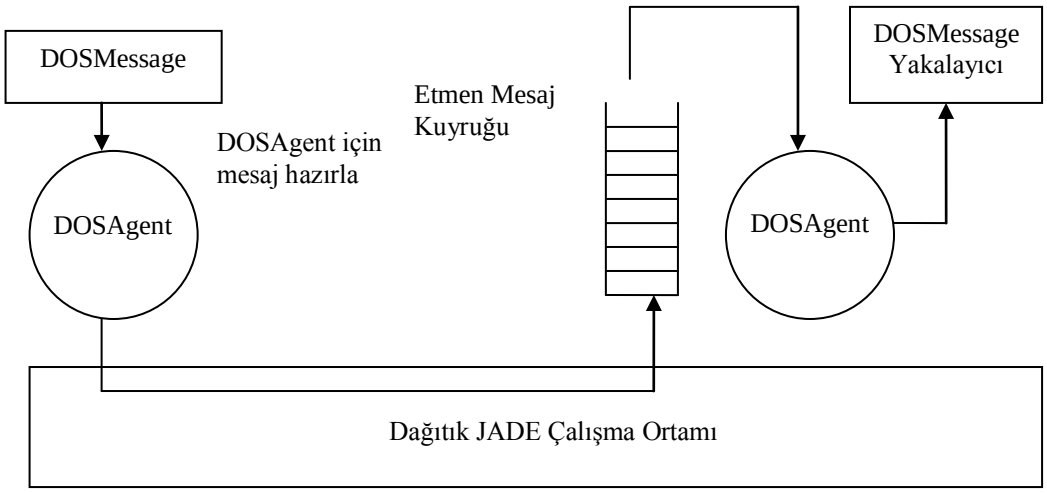
### 5.3.1 Sistem içi haberleşme

JADE platformu hazır bir haberleşme alt yapısını sunduğundan dağıtık nesne paylaşım sistemi ACLMessage yapısının alanlarını kullanarak haberleşme sağlar [28,33]. Platformda bulunan tüm etmenler birbirleriyle haberleşme yeteneğine sahiptir. Her etmen kendisini ilgilendiren mesajları alır ve işler. Mesaj tipi tanımlanmamış ya da etmeni ilgilendirmeyen mesajlar göz ardı edilir.

#### 5.3.1.1 Mesaj iletimi

Etmenler arası haberleşmede JADE'in sağlamış olduğu Agent.send() ve Agent.receive() metotları kullanılır [28,31]. JADE platformunda tanımlı olan ve

etmenlerin adresini ifade eden AID (etmen tanımı) sınıfı alıcı ve gönderici etmenlerin tanımlayıcısıdır. Agent.send() metodu ile gönderilen mesajlar periyodik davranış (periyodik olarak çalışan bir sınıf) sınıfının Agent.receive() metodunun kullanılması ile alınır. Şekil 5.4'te dağıtk JADE çalışma ortamı mesajlaşması gösterilmiştir.




**Şekil 5.4 :** Dağıtık JADE çalışma ortamı mesajlaşması.

Etmenler AMS'den gelen mesajlar için doğrudan ACLMessage içeriğini kullanırlar, fakat DOS sistemi tüm mesajlaşmalarında içerik bölümünü DOSMessage ile doldurmaktadır.

#### 5.3.1.2 Mesaj yapısı

Mesajlar FIPA standardına uygun olarak JADE sisteminde yer alan ACLMessage yapısında oluşturulur [33]. ACLMessage sınıfının sahip olduğu gönderici, alıcı, eylem gibi alanlar kullanılarak mesaj parametreleri ayarlanır ve mesaj gönderilir. ACLMessage sınıfı içerisinde yer alan içerik (content) bölümüne DOS sistemi mesaj yapısı (DOSMessage) koyularak etmenler arasında haberleşme sağlanır.

DOSMessage düzeni, içinde mesaj tipi olarak ayrılmış bir alan içerir ve sistem içinde gönderilen mesajların tipi bu alan aracılığıyla anlaşılır. Her etmen kuyruğundan çektiği mesajları tip kontrolü yaparak görevi doğrultusunda yorumlar, kendisini ve çevresini etkileyecek tepkiler verir.

Dağıtık nesne paylaşım platformundaki mesaj tipleri:

- Is agent exists; ismi verilen etmenin DOS sisteminde kayıtlı olup olmadığını sorgulayan mesajdır.
- Is agent exists response; ismi verilen etmenin DOS sisteminde kayıtlı olup olmadığının yanıt mesajıdır.
- Query object name; ismi verilen nesnenin DOS sisteminde kayıtlı olup olmadığını sorgulayan mesajdır.
- Query object name response; ismi sorgulanan nesnenin varlığının yanıt olarak gönderildiği mesajdır. Nesne platformda yoksa mesajın içerik kısmında null gönderilir.
- Query location; ismi verilen nesnenin sahibinin adresini sorgulama mesajıdır.
- Query location response; ismi verilen nesne platformda kayıtlı ise nesne sahipliğini üstlenen etmenin adresi, sorgulanan nesne sistemde yoksa null mesaj içeriği olarak gönderilir.
- Register object; yeni yaratılan bir nesne, adı ve içeriği ile birlikte bu mesajla kayıt edilir.
- Register object response; kayıt istemi yapılan nesnenin kayıt işleminin sonucunu döndüren mesajdır. Mesaj içeri null döndürüldüğünde kayıt başarısız olmuş demektir.
- Access object; nesne erişim işlemleri bu mesaj tipi ile yapılır. Erişilmek istenen nesnenin adı ve erişim tipi mesaj içeriğinde bildirilir.
- Access object response; erişim isteğinin cevabı olarak gönderilir. Içerik null ise erişim başarısızdır.
- Invalidate object; nesneleri geçersiz kılmak için gönderilen mesajdır.
- Invalidate object response; geçersiz kılma işleminin başarılı olup olmadığını bildiren mesajdır.

- Access info; nesneye erişim yapıldığında nesne sahibi etmenin sistem yöneticisine gönderdiği bilgi mesajıdır. Sistem yöneticisi bu mesaj ile erişimleri denetler.
- Access info response; erişim bilgisinin alındığına dair bilgilendirme mesajıdır.
- Is agent alive; etmenlerin yaşamını kontrol etmek için kullanılan mesajdır. İçerik olarak boş mesaj gönderilir.
- Is agent alive response; etmen yaşam kontrol mesajına cevap olarak gönderilir. İçerik olarak boş mesaj gönderilir.
- Delete object; sistemde kayıtlı bir nesnenin silinmesi için kullanılan mesajdır. Nesneleri silme hakkına sadece yaratıcı etmenler sahiptir.
- Delete object response; etmenin silme isteğinin başarım durumunu bildiren mesajdır.
- Rename object; sistemde kayıtlı bir nesne için yeniden adlandırma isteğidir. Sadece nesne yaratıcısı etmen nesnenin ismini değiştirme hakkına sahiptir.
- Rename object response; isim değiştirme isteğinin başarım durumunu bildiren mesajdır.

Sistemde görevli etmenler ihtiyaç durumuna göre bu mesajları işleyen metotları genişletir.

### 5.3.2 Yerel sunucu

Nesne paylaşım sisteminde nesne erişim isteklerini karşılayan ve sistem koordinatörü ile uyumlu bir şekilde çalışan birimdir. Yerel sunucu, dağıtık nesne paylaşım sistemi kullanıcısının etkileşim halinde olduğu, nesnelerin yerel kopyalarının tutulduğu ve sadece yerel olarak değil sahip olduğu nesneler için tüm platforma hizmet edebilen bir birimdir. Yerel sunucu başlatma yöneticisi, nesne yöneticisi ve yerel sistem yöneticisinden oluşur. Yerel sunucu, sahibi olduğu nesneler için yapılan yerel erişim isteklerine hızlı erişim hizmeti sunulmasını sağlar. Nesne sahipliği sistemde yaşayan yerel sunucular arasında yazma istekleriyle beraber değişir. Herbir yerel sunucu sahibi olduğu nesneler için tüm platforma hizmet eder.

DOS sistemini diğer nesne paylaşım sistemlerinden ayıran en önemli özellik yerel sunucu kullanımına ve nesne sahipliğinin ağ üzerindeki yerel sunucular bazında el değiştirmesine olanak sağlamasıdır. Bu özellik sayesinde sık yazma isteği yapan etmenler nesne sahipliğini elinde bulunduracağından nesne erişimlerini yerel olarak yapabilir. Yerel erişimler ağ üzerindeki mesaj yoğunluğunun azalması ve erişim hızının artmasını sağlar.

Yerel sunucu şu üç birimden oluşur:

- Başlatma yöneticisi,
- Yerel sistem yöneticisi,
- Nesne yöneticisi.

Bu üç birim DOS sisteminde ayrı etmenler olarak gerçeklenirler ve bağımsız bir şekilde çalışma özelliğine sahiptir. Yerel sunucu birimleri gerektiğinde birbirleriyle haberleşerek çevrelerinde ve kendi içlerinde olan olaylar ile ilgili birbirlerini bilgilendirirler. Oluşan olaylara göre etmenler gerekli davranışları gösterirler.

#### 5.3.2.1 Başlatma yöneticisi

Sistemin ilklendirme işlemlerini gerçekleştirmek ile görevlidir. Yerel sunucu sisteminin oluşturulma aşamasında ilk yaratılan etmendir. Daha sonra yaratılacak etmenler (yerel sistem yöneticisi, nesne yöneticisi) için ön hazırlık aşamalarını yerine getirir ve etmenlerin hizmet verecek duruma gelmelerini sağlar.

Başlatma yönetici, sistemde önceden belirlenmiş bir iskele üzerinde çalıştırılır. Başlangıç aşamasında sistem yöneticisine erişim yetkisine sahiptir. Etmen öncelikle DOS ortamında mesajlaşma yeteneği edinmek için mesajlaşma davranışını yaratır. İlklendinme işlemleri tamamlandıktan sonra sistem koordinatörü üzerinden SCNamingManager'e (Adlandırma Yöneticisi) yaratacağı yerel sistem için bir isim uzantısı isteği yapar. Elde edilen isim uzantısı standart yerel sunucu isimlerine eklenir ve yerel sunucular yaratılır.

Yerel sunucuların yaratılma işlemi tamamlandıktan sonra başlangıç yöneticisi görevini tamamlamış olur ve sistemden kendisini hatasız bir şekilde silmek için bir sonlandırma davranışı yaratır. Etmen sistemi içinde yer alan zamanlayıcı tarafından sonlandırma davranışı çalıştırılır ve başlatma yöneticisi platformdan silinir.

#### 5.3.2.2 Yerel sistem yöneticisi

Dağıtılmış nesne paylaşımı ortamında sistem için ağ geçidi görevini yerel sistem yöneticisi yapmaktadır. Kullanıcı uygulamanın yapmak istediği tüm erişim istekleri öncelikle yerel sistem yöneticisine iletilir.

Kendisine iletilen mesajları değerlendirmeye alarak ilgili etmenlere iletme, dışarıdan gelen mesajları ilgili yerel sunucu birimlerine teslim etme ve hata durumlarını algılayarak sistem içi gerekli önlemleri alma gibi sorumlulukları vardır. Başlangıç yöneticisi, yerel sistem yöneticisini yaratırken yerel sistemde görevli tüm etmenlerin isim ve görev bilgilerini içeren bir listeyi yerel sistem yöneticisine gönderir. Etmenler ve görevlerinden oluşan bu liste yerel sunucuda saklanır ve erişim isteklerinde bu bilgiler kullanılır.

Yerel sistem yöneticisi sunucuların birbirleriyle iletişim kurabilmeleri için gerekli alt yapıyı hazırlar.

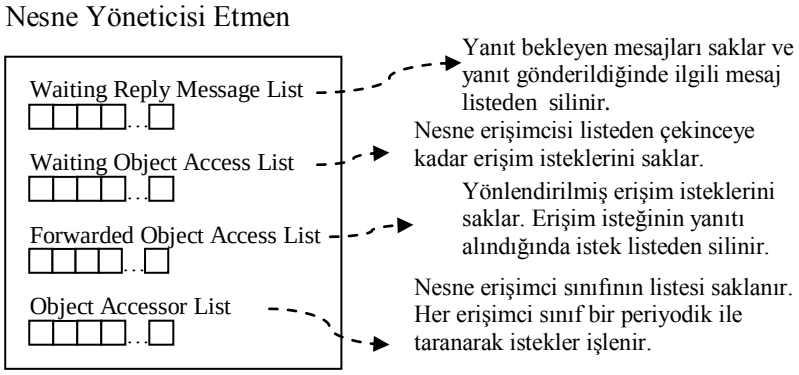
#### 5.3.2.3 Nesne yöneticisi

Dağıtık nesne paylaşım sisteminde kilit öneme sahip birimdir. Gelen nesne erişim isteklerini değerlendirme, yönetme ve gerektiğinde nesnenin durumu ile ilgili karar verme yetkilerine sahiptir.

Nesne yöneticisi temel olarak iki önemli görevi yerine getirir. Bunlardan birincisi, nesne sahipliği yapmaktır. Dağıtık nesne paylaşım sisteminde nesne sahipliği yazma isteği ile el değiştirir. Bu platform içinde yer alan tüm nesne yöneticilerinin nesne sahibi olabileceği anlamına gelmektedir. İkinci görevi ise, gelen erişim isteği üzerine sahibi olmadığı nesnelerin sistem içinde varlığını tespit ederek erişim isteğini nesne sahibine ulaştırmak ve erişim ile ilgili gerekli bilgileri toplayıp sonucu istemciye iletmektir.

Sahip olduğu nesneleri saklamanın yanında, erişim istekleri sonucu edinilen nesnelerin kopyaları da nesne yöneticisinde saklanır. Bu özelliği sayesinde nesne yöneticisine yapılan ve veri güncelliği konusunda esnek davranılabilecek istekler yerel olarak hızlı bir şekilde yanıtlandırılır. Veri güncelliği kritik olmayan bir okuma isteği yapıldığında nesne sahibini bulmadan doğrudan yerel nesne istemciye gönderilir.

Yönetiminden sorumlu olduğu listeler; cevap bekleyen istemci, erişim isteği bekleyen istemci, yönledirilmiş mesaj, nesne erişim listesi olmak üzere dört tanedir. Listeler Şekil 5.5'te gösterilmiştir.



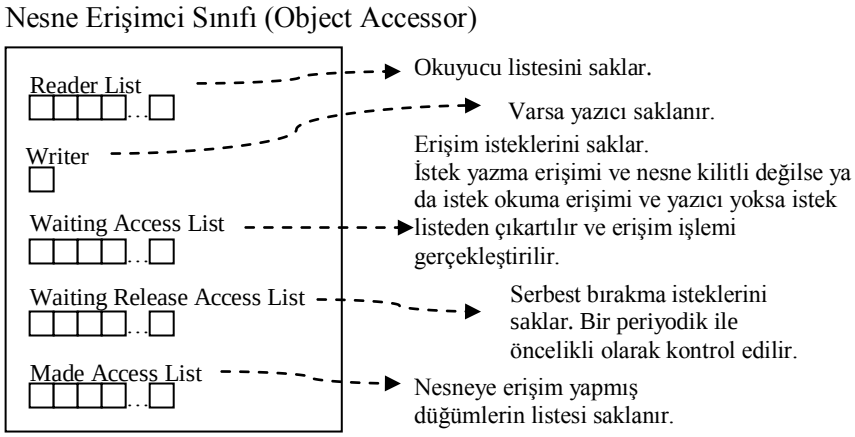

**Şekil 5.5 :** Nesne yöneticisi etmen içindeki listeler.

Cevap bekleyen istemci listesi; nesne yöneticisine gelen mesajları ve bu mesajların kimden geldiğini saklayan listedir. Gelen mesajın işleniş şekline göre cevap bekleyen istemci listesi, mesaj yanıtının gönderileceği etmeni belirlemek için ya da mesajı gönderen istemci bilgisinin gerekli olduğu yerlerde bu liste üzerinde bi arama yapılarak bulunmasını sağlar. Mesaj ile ilgili işlemler tamamlandıktan sonra istek listeden çıkartılır.

Erişim isteği bekleyen istemci listesi; nesne yöneticisine gelen tüm erişim isteklerinin saklandığı kuyruktur. Etmene ulaşan erişim mesajları doğrudan bu listeye koyulur ve bu listeyi tarayan periyodik davranış tarafından istekler işlenir ve listeden çıkartılır.

Yönlendirilmiş mesaj listesi; nesne yöneticisine gelen fakat sistem içinde yer alan bir karar mekanizması ile yönlendirilen mesajları içerir. Bu liste erişim isteği yapılan nesnenin sahibinin başka bir nesne yöneticisi olması durumunda kullanılır. Nesne yöneticisi erişim isteğini nesne sahipliğini yapan yöneticiye iletir iletmez bu listeyi günceller ve iletilen mesajın yanıtının gelmesini bekler. İletilen mesajın yanıtının gelmesiyle ilgili istek listeden çıkartılır. Bu yapı ile nesne yöneticisinin sahip olmadığı fakat gelen istek üzerine sahibini öğrendiği ve yönlendirdiği erişim isteklerinin sonuçları ve dolayısıyla nesne kopyaları sistemde saklanır. Bu durum nesne yöneticisinin daha çok nesneden haberdar olmasını ve yerel belleğinde daha fazla nesne bulundurmasını sağlar.

Nesne yöneticisi erişimleri kontrol etmek için nesne erişim listesine sahiptir ve bu listenin her bir elemanı bir dağıtılmış paylaşımlı nesneyi ve bu nesnenin erişim yönetim özelliklerini içerir. Sisteme bir erişim isteği geldiğinde nesne erişim listesinde ilgili nesne yoksa, sunucu bu nesnenin bir kopyasına sahip değildir. Nesne erişim listesinde her bir eleman dağıtılmış nesneyi ve bu nesneye olan erişimleri yönetecek ek bilgileri içerir. Bu bilgiler her bir nesne için ayrı olup yazma/okuma erişim listesi, serbest bırakma listesi, kilitlilik durumu, nesne ismi, nesne versiyon bilgisi, nesne yaratıcısı, nesne sahibi, nesneye en son erişme tarihi, erişim yapan etmen listesi, yazıcı listesi ve okuyucu listesinden oluşur. Nesne erişimci sınıfı içindeki listeler Şekil 5.6'da gösterilmiştir.




**Şekil 5.6 :** Nesne erişimci sınıfı içindeki listeler.

Diğer etmenlerden gelen okuma ve yazma istekleri ile nesneler kilitli durumda kalır. Kilidi kaldırmak için erişim isteği gönderen etmenden ilgili istek için serbest bırak (release) komutu gelmelidir. Nesne yöneticisi, sistem içinde nesneleri saklayıp paylaşmanın yanı sıra erişimleri de denetler. Sistem içinde yapılan bir isteğin serbest bırakılması için ayarlanabilir bir zaman aşımı süresi vardır. Bu süre içinde istemci nesneyi serbest bırakmazsa nesne sahibi yönetici ilgili nesneyi serbest bırakıp istemciye bu işlemi bildirir. İstemci zaman aşamı süresi sonunda elde etmiş olduğu nesneyi serbest bırakmak isterse bir hata mesajı alır ve bu mesajla yapmış olduğu erişimin başarısız ve geçersiz olduğunu anlar.

Nesne yöneticisi sakladığı her bir nesne için yaşama süresi tanımlar. Bir nesneye kullanıcının nesneyi yaratırken verdiği ya da sistem için tanımlanmış bir süre boyunca erişim yapılmazsa yerel bellekten ve arama işlemlerinden kazanç sağlamak

amacıyla zaman aşımı sonunda silme işlemi uygulanır. İlgili nesne yöneticisi nesne yaratıcısı ya da nesne sahibi ise bu mekanizma uygulanmaz.

### 5.3.3 Sistem koordinatörü

Dağıtık nesne paylaşım ortamının yönetiminden ve bakımından sorumlu olan birimdir. Yerel sunucular üzerinde meydana gelen olayları izlemek ve gerekli tepkiyi vermekle görevlidir. Sistem koordinatörü daha sonradan yaratılacak yerel sunucuların bileceği bir adreste bulunur. Yerel sunucular yaratılırken sistem koordinatörüne erişebilmeleri için gerekli bilgiler sağlanır.

Sistem koordinatörünün görevleri:

- Sistem içinde sunucuların birbirinden ayırt edilmesini sağlayacak bir tanımlayıcı bilgisine ihtiyaç vardır. JADE platformunda her bir etmen farklı bir tanımlayıcıya sahip olmalıdır [28]. Sistem koordinatörü yerel sunuculara yaratılma aşamasında benzersiz bir tanımlayıcı bilgisi sağlar.
- Sunucular (dağıtık nesne paylaşım sistemi etmenleri) arasında haberleşmeyi sağlamak için gerekli alt yapıyı sağlar. Yerel bir sunucu bir nesneye erişmek istediğinde, nesne sahibi etmeni bulmak için sistem koordinatörüne bir sorgu yapar. Sistem koordinatörü sahip olduğu yönetici sunuculardan faydalanarak cevabı istemci etmene iletir.
- Yaratılmak istenen nesne isminin sistem çapında tekilliğinin kontrolünü yapar ve uygunluk durumunun cevabını döndürür.
- Sistemde kayıtlı tüm nesnelerin bilgilerini saklar.
- Sunucular ve sahip oldukları nesnelere ait yerleşke bilgilerini saklar.
- Dağıtık nesne paylaşımı sisteminde oluşacak hata durumları için gerekli çözümleri üretir.

Sistemde yerel sunucular yer alsada bu sunucuların birbirlerinden haberdar olmaları ve etkileşimi için bir sistem koordinatörüne ihtiyaç vardır. Koordinatör içerisinde farklı görevlere sahip, sistemin çalışmasını düzenleyen ve birbirleriyle iletişim halinde olan şu sunucular yer almaktadır:

- Başlatma yöneticisi,
- Sistem yöneticisi,

- Yerleşke yöneticisi,
- İsimlendirme yöneticisi,
- Nesne yöneticisi.

Her birim bir etmen tarafından gerçeklenir ve farklı görevler üstlenerek birim zamanda verilecek cevap sayısı ve hızı arttırılır.

#### 5.3.3.1 Başlatma yöneticisi

Sistem koordinatörü sunucularının yaratılma aşamasında ilk çalışan etmendir ve bir defaya mahsus olmak üzere çalışır. Görevlerini yerine getirdikten sonra kendisini sonladırır. Sistemin yerel sunuculara ve istemcilere hizmet verecek duruma gelmesi için gerekli işlemleri yapar. Bu amaçla daha önce belirtilmiş adres ve iskele üzerinde sistem yöneticisi, yerleşke yöneticisi, isimlendirme yöneticisi, nesne yöneticisi etmenlerini yaratır ve bu etmenlerin birbirlerinden haberdar olmasını sağlayacak bilgileri yaratma aşamasında gönderir.

#### 5.3.3.2 Sistem yöneticisi

Dağıtılmış nesne paylaşımı ortamında sistem koordinatörü için ağ geçidi görevini sistem yöneticisi yapmaktadır. Yerel sunucuların mesajları öncelikle sistem yöneticisine iletilir.

Kendisine iletilen mesajları değerlendirmeye alarak ilgili etmenlere iletme ve dışarıdan gelen mesajları ilgili yerel ve sistem sunucu birimlerine teslim etme, hata durumlarını algılayarak sistem içi gerekli önlemleri alma sorumlulukları vardır. Başlangıç yöneticisi tarafından yaratılan etmenlerin listesi yaratılma aşamasında sistem yöneticisine aktarılır ve bu etmenler gerektiğinde kullanılmak üzere bir listede saklanır. Liste etmenler ve görevlerinden oluşmaktadır.

Sistem yöneticisi tüm sunucuların birbirleriyle iletişim kurabilmeleri için gerekli alt yapıyı hazırlar.

#### 5.3.3.3 Yerleşke yöneticisi

Dağıtık nesne paylaşım sisteminde paylaşılan nesnelerin yerleşke yönetiminden sorumludur. Nesnelerin yerleşkelerini saklamak amacıyla nesne tanımı ve yerleşke

bilgisini içeren bir hash tablosuna sahiptir. Dağıtık nesne paylaşım sisteminin temel görevleri:

- Herhangi bir dağıtık sistem etmeninden gelen nesne yerleşke sorgusuna yanıt vermek.
- Sisteme bir paylaşılmış nesne katılması ya da mevcut paylaşılmış nesnenin sahipliğinin değişmesi durumunda yerleşke tablosunu güncellemek.

Yerleşke tablosunda yer alan nesnelere ait sahiplik bilgisinin yazma işlemleri sonunda güncellenmesi gerekir. Sahiplik bilgisi nesne üzerinde en son güncellemeyi yapmış etmenin kimlik bilgisidir. Nesne yöneticisi yazma isteğini işleme alır almaz yerleşke yöneticisine nesnenin yeni sahibini bildirir. Yerleşke tablosu üzerinde bir arama yapılarak ilgili nesnenin sahiplik bilgisi güncellenir.

Sistemde meydana gelebilecek tutarsızlıkları engellemek amacıyla nesne erişimlerinde yerleşke tablosu üzerinden nesne sahibi sorgulanır ve bu sorgulama işleminin performansı yerel sunucularda yapılan erişim hızını önemli ölçüde etkiler. Yerleşke yöneticisi nesne isimlerinin tekilliğinden faydalanarak sahiplik bilgisini hash tablosunda tutar ve böylelikle gelen sorgulara çok hızlı yanıt verir.

#### 5.3.3.4 İsimlendirme yöneticisi

Dağıtık nesne tanımlayıcı bilgisi içeren bir komut tablosuna (hash table) ve sunucu tanımlayıcı bilgilerini saklayan bir listeye sahiptir. İsimlendirme yöneticisi yöneticisinin görevleri aşağıda belirtilmiştir:

- Dağıtık nesne paylaşım platformunda yaratılmak istenen nesnelerin isimlerinin tekilliğini kontrol etmek ve istemciye istekte bulunduğu nesne isminin uygunluk durumu bilgisini vermek.
- Sistemde kayıtlı tüm nesnelerin isimlendirme bilgisini saklamak.
- Yeni yaratılan yerel sunuculara eşsiz bir tanımlayıcı isim vermek.

Nesne kayıt işlemlerinde isimlendirme yöneticisine bir isim tekillik kontrolü yapılır ve bu çağrıya karşılık nesne isminin kullanılıp kullanılamayacağı bilgisi istemci etmene gönderilir. Bir nesnenin sistemden tamamen silinmesi durumunda

isimlendirme tablosundan ilgili nesne ismi silinir ve böylelikle silinen nesne ismi tekrar kullanılabilir duruma gelir.

#### 5.3.3.5 Nesne yöneticisi

Sistem yöneticisi olarak çalışan sunuculardan birisidir. Nesnelerin sunucular arasında iletilmesi, nesne sahipliği, nesne sahiplik yönetimi ve bakımı işlemlerini gerçekleştirir. Yerel yönetici altında bulunan nesne yöneticisinin tüm özelliklerine sahiptir.

Yerel nesne yöneticisinin sahip olduğu özelliklere ek olarak aşağıdaki hizmetleri verir:

- Yerel sunuculardan gelen nesne erişim bilgilerini saklar ve bu bilgiler aracılığıyla nesnelerin durumunu takip eder.
- Nesne sahiplerinin erişilebilir olup olmadıklarını periyodik olarak kontrol eder ve kayıtlı nesne sahibinin devre dışı olması durumunda ilgili nesnenin sahipliğini üstlenir. Sahipliğini üstlendiği nesne için ağda geçersiz kılma mesajı yayınlar. Sistem içinde sahiplik yazma isteği ile değiştiğinden, bir etmenin yazma isteği yapmasıyla sahiplik değişimini gerçekleştirir.

Zorunlu bir durum olmadıkça (sahip yerel nesne yöneticisinin ulaşılamaz olması) nesne sahipliği görevini üstlenmez. Asıl görevi sistemde nesneler ile ilgili oluşabilecek tutarsızlıkların farkına varmak ve gerekli çözümleri üretmektir.

### 5.4 Dağıtık Nesne Erişim İşlemleri

Nesne erişim işlemlerinde DOS sisteminde kullanıcılar paylaşılan nesneler üzerinde şu işlemleri gerçekleştirebilirler:

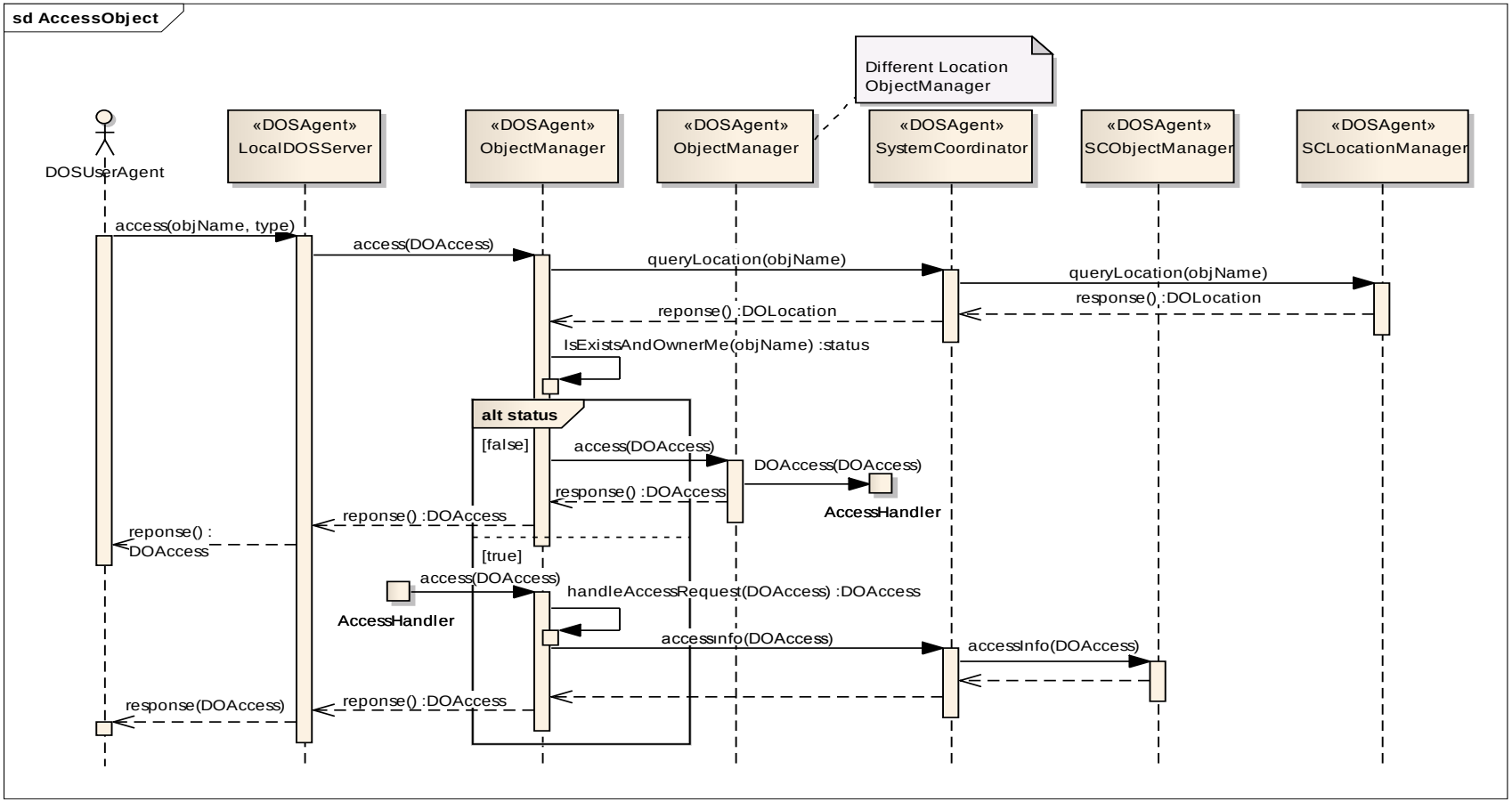
- Nesne yaratma,
- Nesneyi okuma,
- Nesneye yazma,
- Nesneyi serbest bırakma,
- Nesneyi gevşek okuma,

- Nesneyi silme,
- Nesneyi yeniden adlandırma.

Kullanıcı erişim türüne göre gerekli parametreleri sağlar ve kalan tüm işlemler kullanıcıya yansıtılmadan (saydam olarak) gerçekleşir. Nesnenin yerel ya da uzak bir sunucuda yer aldığından kullanıcının haberi yoktur. Kullanıcı çağrısını gerçekleştirir ve sistem erişim ile ilgili gerekli işlemleri yaparak kullanıcıya tepki verir. Erişim işlemi sonunda kullanıcıya olumlu ya da olumsuz tepki verilebilir. Olumsuz tepki sonucunda kullanıcı tekrar istek yapma hakkına sahiptir.

Kullanıcının yapmak istediği erişimler için zaman aşamı süresi tanımlanmıştır. Böylelikle kullanıcı etmenin aynı yazılım noktasında takılı kalması engellenmiştir. Zaman aşımına uğramış bir erişim isteği için kullanıcıya erişim başarısızlığı sonucu döndürülür ve bu durum karşısında kullanıcı uygulamasının gereksinimine göre gerekli eylemleri yapması beklenir.

Şekil 5.7'de okuma, yazma, serbest bırakma işlemleri için ardışık erişim diyagramı gösterilmiştir.




**Şekil 5.7 :** Nesne erişim işlemi ardışık diyagramı.

### 5.4.1 Çağrı yönetimi

DOSAgent sınıfını genişleten tüm etmenler nesne paylaşım çağrısı yapma hakkına sahiptir. Sistemde çağrı işlemleri DOAccess sınıfı metotları ile gerçekleşir. Nesne paylaşımı ile ilgili bir işlem yapacak olan etmen öncelikle getAccessObject() metodu ile yapılacak çağrıya ilişkin metotlara sahip ve çağrı ile ilgili erişim bilgilerini saklayan bir nesne elde etmelidir. Yapılacak her çağrı için yeni bir DOAccess nesnesi edinilmelidir fakat serbest bırakılması gereken bir erişim yapılıyorsa, nesne edinilip gerekli işlemler tamamladıktan sonra yine aynı DOAccess nesnesi üzerinden serbest bırakma işlemi gerçekleştirilmelidir. Serbest bırakma işlemi daha önce yapılmış bir erişimin tipini ve erişim bilgilerini saklamış nesne üzerinden yürütüldüğünden, gelen serbest bırakma isteğinin daha önceki hangi erişim tipi ile ilişkili olduğu sistem tarafından anlaşılır. DOS sisteminde kullanıcıya bırakılmış tek serbest bırakma işlemi yazma istediğidir. Nesne erişim işlemleri Çizelge 5.1'de gösterilmiştir.

**Çizelge 5.1 :** Nesne erişim işlemleri.

| Erişim Tipi | Çağrı | Açıklama |
|---|---|---|
| Kayıt | IObjectAccess register(String objName,Object obj) | Serializable olarak gerçeklenmiş bir nesneyi kayıtlamak için kullanılır. Nesne ismi ve nesne parametre olarak verilir. |
| Kayıt | IObjectAccess register(String objName,Object obj,long timeOut) | Serializable olarak gerçeklenmiş bir nesneyi kayıtlamak için kullanılır. Nesne ismi, nesne ve en geç işlem tamamlanma süresi milisaniye cinsinden parametre olarak verilir. |
| Okuma | IObjectAccess read(String objName) | Nesne okumak için kullanılır. Nesne adı parametre olarak verilir. |
| Okuma | IObjectAccess read(String objName,long timeOut) | Nesne okumak için kullanılır. Nesne adı ve en geç işlem tamamlanma süresi parametre olarak verilir. |
| Gevşek Okuma | IObjectAccess readLoose(String objName) | Nesne okumak için kullanılır. Nesne adı parametre olarak verilir. |
| Gevşek Okuma | IObjectAccess readLoose(String objName, long timeOut) | Nesne okumak için kullanılır. Nesne adı ve en geç işlem tamamlanma süresi parametre olarak verilir. |
| Yazma | IObjectAccess write(String objName) | Nesne yazma işlemi için kullanılır. Nesne adı parametre olarak verilir. |
| Yazma | IObjectAccess write(String objName, long timeOut) | Nesne yazma işlemi için kullanılır. Nesne adı ve en geç işlem tamamlanma süresi parametre olarak verilir. |
| Silme | IObjectAccess delete(String objName) | Nesne silme işlemi için kullanılır. Nesne adı parametre olarak verilir. |
| Silme | IObjectAccess delete(String objName, long timeOut) | Nesne silme işlemi için kullanılır. Nesne adı ve en geç işlem tamamlanma süresi parametre olarak verilir. |
| Yeniden Adlandırma | IObjectAccess rename(String objName, String newName) | Nesne yeniden adlandırma işlemi için kullanılır. Nesne adı parametre olarak verilir. |
| Yeniden Adlandırma | IObjectAccess rename(String objName, String newName, long timeOut) | Nesne yeniden adlandırma işlemi için kullanılır. Nesne adı ve en geç işlem tamamlanma süresi parametre olarak verilir. |
| Serbest bırakma | boolean release() | Erişim nesnesi ile yapılan son erişim işlemini serbest bırakmak için kullanılır. |
| Serbest bırakma | boolean release(long timeOut) | Erişim nesnesi ile yapılan son erişim işlemini serbest bırakmak için kullanılır. En geç işlem tamamlanma süresi milisaniye cinsinden parametre olarak verilir. |

Kullanıcı yazma isteği yapıp nesneyi elde ettikten sonra hızlı bir şekilde nesneyi güncellemeli ve erişim için edindiği DOAccess sınıfı ile nesneyi serbest bırakmalıdır.

#### 5.4.1.1 Kayıt

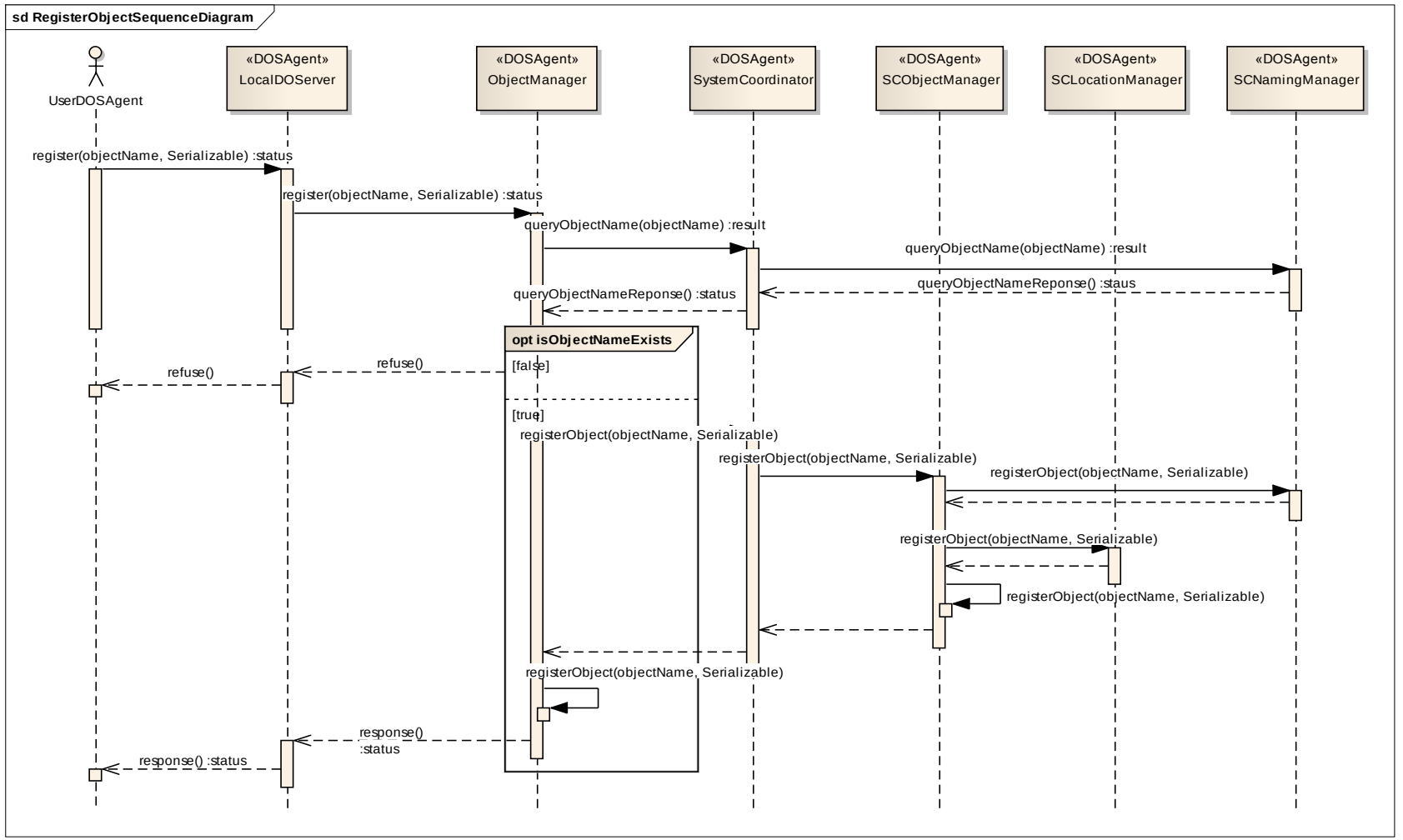
Sistemde yeni bir nesne yaratma işlemidir. Kullanıcılar yaratılan nesnenin yapısı konusunda önceden anlaşmış olmalıdır. Yaratılacak nesnenin yapısı ile ilgili tek kısıt nesnenin Serializable arayüzünü gerçeklemesidir.

Kayıt işlemi aşamaları:

- DOSAgent sınıfını genişleterek oluşturulmuş kullanıcı etmen, nesne kaydı için getAccessObject() metodu ile bir erişim nesnesi elde eder.
- Elde edinilen erişim nesnesi kullanılarak doAccess.register(String objectName, Serializable data) metot çağrısı, yaratılmak istenen nesne adı ve içeriği verilerek yapılır.
- Sistem erişim isteğine benzersiz bir tanımlayıcı atar ve isteği yerel sistem yöneticisine (LocalDOServer) iletir. Yerel sistem yöneticisi ağ geçidi görevi yaparak isteği yerel nesne yöneticisine iletir.
- Yerel nesne yöneticisi daha önce bu isimde bir nesne yaratılıp yaratılmadığını anlamak için sistem yöneticisine bir isim sorgulama çağrısı gönderir.
- Sistem yöneticisi sorguyu isimlendirme yöneticisine iletir ve isimlendirme yöneticisi daha önce aynı isimde bir nesnenin sisteme kayıtlı olup olmadığını sahip olduğu isimlendirme tablosunda sorgular ve sonucu sistem yöneticisine iletir.
- Sistem yöneticisi sorgu sonucunu yerel nesne yöneticisine iletir. Yerel nesne yöneticisi sistemde sorgulanan isimde bir nesne olduğunu anlarsa yerel sistem yöneticisine bir reddetme çağrısı gönderir ve yerel sistem yöneticisi de bu reddi kullanıcı etmene yollayarak erişim işlemi tamamlanır. Verilen isim ile nesne kaydı uygunsa nesne yöneticisi kayıt isteğini nesne yaratıcısı ve sahibi kendisi olduğunu belirterek sistem yöneticisi üzerinden koordinatörün nesne yöneticisine (SCObjectManager) gönderir. SCObjectManager nesneyi kayıt etmek üzere isimlendirme (SCNamingManager) ve yerleşke (SCLocationManager) yöneticilerine sırasıyla çağrı gönderir.

- Adlandırma ya da yerleşke yöneticisinden gelen herhangi bir olumsuz cevap sonucu kayıt işlemi başarısız sayılır ve red cevabı kullanıcı etmene kadar döndürülür.
- Adlandırma ve yerleşke yöneticisine başarılı bir şekilde kayıt edilen nesnenin sahiplik ve yaratıcı bilgisi ile sistem koordinatörü içinde yer alan nesne yöneticisine kaydedilir ve başarılı kayıt sonucu istemci yerel nesne yöneticisine ulaştırılır.
- Yerel nesne yöneticisi nesneyi ve erişim bilgilerini erişim tablosuna ekler ve başarılı kayıt sonucunu yerel sistem yöneticisi üzerinden kullanıcı etmene ulaştırır.

Kayıt işlemi tamalanan nesnelerin okuyucu listesi ve yazıcı durumu boş olarak ilklendirilir ve ilgili nesne platformundan gelecek okuma ve yazma erişimlerine hazırdır. Şekil 5.8'de nesne kaydetme ardışık diyagramı gösterilmiştir.




**Şekil 5.8 :** Nesne kaydetme ardışık diyagramı.

#### 5.4.1.2 Silme

Kayıtlı bir nesneyi silme yetkisi, sadece nesnenin yaratıcısı olan etmene verilmiştir. Nesne kayıt işlemi sırasında yaratıcı yerel nesne sunucunun erişim bilgisi de saklanır. Zaman içinde nesne sahibi etmen değişse de nesnelerin saklandığı ve yönetildiği DOAccess sınıfı içinde yer alan yaratıcı bilgisi değişmeyeceğinden silme isteği geldiğinde isteğin yaratıcıdan gelip gelmediği anlaşılır.

- Nesne adı ve silme isteği bilgisi ile erişim isteği yapan kullanıcı etmenin isteği yerel sistem yöneticisine iletilir.
- Yerel sistem yöneticisi isteği yerel nesne yöneticisine iletir ve yerel nesne yöneticisi yerleşke yöneticisine yapacağı bir çağrı ile nesne sahiplik bilgisini alır. Yerel nesne sunucusu nesne sahibi ise isteği kendi içinde değerlendirir, sahip değilse yerleşke yöneticisinden gelen sahiplik bilgisi doğrultusunda sahip yerel sunucuya silme isteğini iletir.
- Silme isteğini alan sahip nesne sunucusu isteğin yaratıcı sunucudan gelip gelmediğini kontrol eder. İstek yaratıcı sunucudan gelmiyorsa silme isteğini reddeder ve red mesajı istekçiye kadar ulaştırılır.
- Silme isteğinin yaratıcı etmenden gelmesi durumunda nesnenin kilit durumu kontrol edilir. Nesne kilitliyse, yani yazıcı ya da okuyucu varsa istek bekletilir ve kilit ortadan kalkınca istek işlenir.
- Kilit ortadan kalktığında yerleşke ve isimlendirme yöneticilerine tablolarından ilgili nesneyi kaldırmaları için silme isteği gönderilir. Nesneyi daha önce kullanan nesne sunucularına geçersiz kılma mesajı gönderilir ve ilgili nesnenin erişim isteği kuyruğundaki tüm etmenlere red mesajı gönderildikten sonra nesne sunucu tablosundan silinerek istemci etmene silmenin başarılı olduğunu gösteren bir cevap mesajı gönderilir.

Silme işlemi gerçekleştikten sonra herhangi bir kullanıcı etmen ilgili nesneye erişmek istediğinde yerleşke tablosunda bu nesne yer almayacağından başarısız erişim mesajı alacaktır.

#### 5.4.1.3 Yeniden adlandırma

Kayıtlı bir nesnenin yeniden adlandırma yetkisi sadece yaratıcı etmene verilmiştir. Nesneler kaydedilirken yaratıcı yerel nesne sunucu bilgisi de saklanır. Zaman içinde

nesne sahibi etmen değişsede nesnelerin saklandığı ve yönetildiği DOAccess sınıfı içinde yer alan yaratıcı bilgisi değişmeyeceğinden yeniden adlandırma isteğinin yaratıcıdan gelip gelmediği anlaşılır.

- Nesne adı, nesne yeni adı ve yeniden adlandırma isteği ile erişim isteğinde bulunan kullanıcı etmenin isteği yerel sistem yöneticisine iletilir.
- Yerel sistem yöneticisi isteği yerel nesne yöneticisine iletir ve yerel nesne yöneticisi koordinatör sunucu sisteminde yer alan yerleşke yöneticisine yapacağı bir çağrı ile sahiplik bilgisini sorgular. Yerel sunucu nesne sahibi ise isteği kendi içinde değerlendirir değilse nesne sahibi yerel sunucuya yeniden adlandırma isteğini iletir.
- Yeniden adlandırma isteğini alan sahip nesne sunucusu, isteğin yaratıcı sunucudan gelip gelmediğini kontrol eder. İstek yaratıcı sunucudan gelmiyorsa yeniden adlandırma isteğini reddeder ve red mesajı istekçiye kadar ulaştırılır.
- Yeniden adlandırma isteği yaratıcı etmenden gelmesi durumunda nesnenin kilit durumu kontrol edilir. Nesne kilitliyse yani yazıcı ya da okuyucu varsa istek bekletilir ve kilit ortadan kalkınca istek işlenir.
- Kilit ortadan kalktığında yeni ismin sistemde benzersizliğini sorgulamak için isimlendirme yöneticisine yeni nesne isminin kayıtlı olup olmadığı sorgusu gönderilir. Yeni isim ile daha önce kaydedilmiş bir nesne varsa istemci etmene kadar red cevabı ulaştırılır. Yeni nesne ismi benzersiz ise yerleşke ve isimlendirme yöneticilerine tablolarındaki ilgili nesne ismi ve yerleşkeyi güncellemeleri için yeniden adlandırma isteği gönderilir. Nesneyi daha önce kullanan nesne sunucularına geçersiz kılma mesajı gönderilir ve ilgili nesnenin erişim isteği kuyruğundaki tüm etmenlere red mesajı gönderildikten sonra nesne yeniden adlandırılır, istemci etmene yeniden adlandırmanın başarılı olduğunu gösteren bir cevap mesajı gönderilir.

Yeniden adlandırma işlemi gerçekleştikten sonra herhangi bir kullanıcı etmen nesnenin eski ismiyle nesneye erişmek istediğinde yerleşke tablosunda bu nesne yer almayacağından başarısız erişim mesajı alacaktır. Sisteme eski nesne ismiyle yeni bir nesne kaydetmek mümkündür.

#### 5.4.1.4 Okuma

Kayıtlı bir nesnenin, dağıtık nesne paylaşım sisteminde bulunup istemci etmenin yerel belleğine kopyasının getirilme işlemidir. Sahip nesne sunucusu kendisine gelen bir okuma isteğini nesne üzerinde bir yazıcı yoksa kabul eder. Aksi taktirde okuma isteği bekletilir ve yazıcı erişimini serbest bıraktıktan sonra okuma işlemi gerçekleştirilir.

Okuma erişimlerinde, nesne kopyası istemci etmene ulaştırılıp serbest bırakma mesajı alınana kadar nesne kilitlenir ve sadece kuyrukta bulunan okuma isteklerine cevap verilir. Nesne erişim kuyruğunda bir yazma isteği ile karşılaşılırsa, tüm okuma isteklerinden serbest bırakma mesajı gelene kadar kuyruğun işlenmesi bekletilir. Okuma isteği sonucu nesnenin kopyası istemci etmene ulaşır ulaşmaz sistem otomatik olarak serbest bırakma mesajını sahip etmene gönderir. Bu yüzden okuma işlemlerinde kullanıcının nesneyi serbest bırakmasına gerek yoktur. Okuma işlemi için gerçekleşen adımlar:

- İstemci etmen bir erişim nesnesi elde eder ve erişim nesnesini kullanarak read(nesneAdı) metodunu çağırır.
- İstek yerel sistem yöneticisine ve yerel yöneticiden de yerel nesne yöneticisine ulaştırılır.
- Yerel nesne yöneticisi yerleşke yöneticisinden nesne sahiplik bilgisini alır ve ilgili yerel nesne yönetecisi erişilmek istenen nesnenin sahibinin kendisi olup olmadığını kontrol eder. Sahip kendisi ise istek ile ilgili işlemleri gerçekleştirir. Nesne yöneticisi erişilmek istenen nesnenin sahibi ise yerleşke yöneticisine sahiplik sorgusu yapmadan işlemlere devam edebilir. Yerel nesne yöneticisi nesnenin sistemde bulunmadığını içeren bir sorgu cevabı alırsa yerel sistem yöneticisine ve oradan da erişim istekçisi etmene erişim reddi yanıtı gönderir. Aksi durumda erişim isteğini sahip etmene iletir.
- Nesne sahibi etmen gelen okuma isteğini erişim bekleyen istekçiler listesine koyar.

- Erişim istekleri periyodik olarak erişim isteği kuyruğundan çekilir ve yazıcı yoksa okuma isteği okuyucular listesine eklenerek nesne kilitlenir. Nesnenin bir kopyası istemci etmene iletilir.
- Okuma isteği sonucu nesne kopyasını kullanıcı etmene döndürmeden önce sistem otomatik olarak serbest bırakma mesajını sahip nesne sunucusuna ulaştırır. Sahip nesne sunucusu okuyucular listesinden etmeni çıkarır ve başka okuyucu kalmadıysa nesne kilidini kaldırır.

Okuma isteklerinde zaman aşımı süresi vardır. Okuma isteği yapan etmen belirli bir sürede erişimi için cevap alamazsa, etmene erişimin başarız olduğunu gösteren sonuç döndürülür.

#### 5.4.1.5 Gevşek okuma

Dağıtık nesne paylaşım platformunda istekte bulunulan nesnenin en yakın nesne yöneticisinden getirilmesi ve sistemde herhangi bir kilitleme işlemi yapılmadan okunması işlemidir. Gevşek okuma isteği sonucunda erişilen nesne kopyasının güncelliği garanti edilmez. Nesne karşılaşılan ilk yerdeki yerel kopya olabileceği gibi nesne sahibinden alınmış güncel bir kopya da olabilir.

- İstemci etmen bir erişim nesnesi elde eder ve erişim nesnesini kullanarak readLoose(nesneAdı) metodunu çağırır.
- İstek yerel sistem yöneticisine ve yerel yöneticiden de yerel nesne yöneticisine ulaştırılır.
- Yerel nesne yöneticisi erişilmek istenen nesnenin nesne erişim listesinde olup olmadığını kontrol eder. Nesne erişim listesinde varsa, daha önceden eklenmiş bu nesnenin yerel bir kopyası sistem içinde vardır ve sistem yöneticisi üzerinden istemciye doğrudan nesne kopyası ulaştırılır. Yerel nesne yöneticisinin nesne erişim listesinde sorgulanan nesne yoksa, sistem yöneticisine nesne yerleşkesi sorgulama mesajı gönderir. Sistem yöneticisi sorguyu yerleşke yöneticisine iletir ve sorgu cevabı yerel nesne yöneticisine kadar ulaştırılır. Yerel nesne yöneticisi nesnenin sistemde bulunmadığını içeren bir sorgu cevabı alırsa yerel sistem yöneticisine ve oradan da erişim istekçisi etmene erişim reddi yanıtı gönderir. Aksi durumda erişim isteğini sahip etmene iletir.

- Nesne sahibi etmen, gelen gevşek okuma isteğini erişim bekleyen istekçiler listesine koyar.
- Erişim istekleri periyodik olarak erişim isteği kuyruğundan çekilir ve sistemde kayıtlı en son kopya istemci etmene iletilir.

#### 5.4.1.6 Yazma

Dağıtık nesne paylaşımı sisteminde kayıtlı bir nesnenin değerini değiştirme isteğidir. Sistemde yazma isteği yapan ve nesneyi elde eden etmen, nesnenin sahibi olarak atanır ve daha sonra gelecek tüm erişim isteklerini nesne sahibi etmen olarak karşılar. Nesne sahibi etmen nesnenin herzaman en güncel kopyasını taşır. Sahipliğin etmenler arasında el değiştirmesi, aynı etmenin yapmak isteyeceği sık yazma erişimleri için yüksek performans getirir ve bu durum etmenlerin hareketliliği ile uyum sağlar.

Nesne sahipliğinin tek bir etmene verilmesi sistem içinde tutarlılık problemini ortadan kaldırır. Nesne değerini değiştirmek isteyen bir etmen öncelikle nesne sahibi etmenden yazma izni almalı, izin işlemi tamamlandığında ilgili nesnenin güncel kopyasını ve sahipliğini yerel nesne yöneticisine taşımalıdır. Yazma işlemi sırasında gerçekleşen adımlar:

- İstemci etmen bir erişim nesnesi elde eder ve bu erişim nesnesini kullanarak write(nesneAdı) metodunu çağırır.
- İstek yerel sistem yöneticisine ve yerel yöneticiden de yerel nesne yöneticisine ulaştırılır.
- Yerel nesne yöneticisi erişilmek istenen nesnenin sahibi ise yerleşke yöneticisine sahiplik sorgusu yapmadan işlemlere devam edebilir. Yerel nesne yöneticisi nesne sahibi değilse ya da sahiplik bilgisini teyit etmek amacıyla yerleşke yöneticisine nesne sahipliği sorgusu yapar. Nesne yöneticisi erişmek istenen nesnenin sahibinin kendisi olup olmadığını kontrol eder. Yerel nesne yöneticisi nesnenin sistemde bulunmadığını içeren bir sorgu cevabı alırsa yerel sistem yöneticisine ve oradan da erişim istekçisi etmene erişim reddi yanıtı gönderir. Aksi durumda erişim isteğini sahip etmene iletir.

- Nesne sahibi etmen gelen yazma isteğini erişim bekleyen istekçiler listesine koyar.
- Erişim istekleri periyodik olarak erişim isteği kuyruğundan çekilir ve okuyucu yoksa ve istemci yerel nesne sunucusu ile sahip nesne sunucusu aynı ise yani nesne yerleşkesi değişmeyecekse yazıcının varlığı işaretlenerek nesne kilitlenir. Nesnenin bir kopyası istemci etmene iletilir. Nesne sahibi ile yazma isteği yapan nesne yöneticisi aynı değilse sahiplik değişimi gerçekleştirilir. Yerleşke yöneticisine nesnenin yeni sahibini bildiren bir mesaj atılır ve yerleşke tablosunun güncellenmesi sağlanır. Nesne erişimi yapmış tüm istemcilere sahip oldukları kopyaların artık geçersiz olduğunu bildiren bir nesne geçersiz kılma mesajı gönderilir ve nesnenin güncel kopyası yazma isteği yapan nesne yöneticisine gönderilir. Nesnenin yeni sahibi nesneyi kilitler ve güncel nesne kopyasını istemci etmene yollar.
- Yazma isteği yapıp güncel nesne kopyasını alan etmen çok hızlı bir şekilde (yazma işleminin hızı erişimci etmenin nesne kullanım yöntemi ve performansına bağlı olarak değişebilir) yazma işlemini tamamlayıp serbest bırakma mesajını yayınlar. Serbest bırakma mesajını alan sahip nesne yöneticisi nesnenin kilitini kaldırır ve varsa sıradaki erişim isteğine cevap verir.

Yazma erişimi yapan etmen nesneyi kilitlemiş olacağından nesneyi elde eder etmez yazma işlemini gerçekleştirip serbest bırakma mesajını yayınlamalıdır.

#### 5.4.1.7 Serbest bırakma

Okuma ya da yazma isteği yapılarak kilitlenmiş bir nesne ile ilgili erişim işlemlerinin tamamlandığını belirten ve kilitinin kaldırılması için yapılan erişim çağrısıdır. Yazma işlemini aynı anda bir etmen yapacağından serbest bırakma mesajı ile nesne kiliti kaldırılır. Okuma işleminde bir etmen okuma işlemi yaparken gelen diğer okuma işlemlerine de izin verileceğinden tüm okuma işlemleri için gelen serbest bırakma mesajı sonunda nesne kiliti kaldırılır.

Okuma işleminde nesne istemci etmene ulaşır ulaşmaz sistem otomatik olarak serbest bırakma mesajını iletir. Yazma işlemlerinde nesne istemciye ulaşıp yazma

işlemi tamamlandıktan sonra kullanıcı etmen açıkça daha önce edinmiş olduğu nesne erişim sınıfı ile serbest bırakma çağrısını gerçekleştirmelidir.

Serbest bırakma istekleri sahip nesne yönetici sisteminde nesne erişim istekleri listesinden daha öncelikli olan nesne serbest bırakma istekleri listesine girer. Nesne yöneticisinde görevli nesne erişim yöneticisi birimi periyodik olarak öncelikle serbest bırakma listesini kontrol eder ve bu listede istek olduğu sürece diğer erişim işlemlerine (okuma, yazma, silme, gevşek okuma, yeniden adlandırma) izin verilmez.

## 5.5 Nesne Erişim Yönetimi

### 5.5.1 Erişim isteği yönetimi

Erişim istekleri DOAccess isimli sınıfın sağladığı metot ve değişkenler ile yönetilir. Erişim isteği yapacak etmen DOAccess sınıfı tipinde olan bir erişim nesnesi elde eder ve tüm erişimlerini bu nesne üzerinden gerçekleştirir. Nesne, her erişimi farklı kılacak nesne erişim tanımlayıcısını kendisi otomatik olarak yaratır ve kullanıcıya sadece yetkisi olan metotlara erişim hakkı sunar.

### 5.5.2 Dağıtık nesne bakımı

Dağıtılmış nesneler için yapılan erişimleri ve ilgili erişimler için zaman aşımını denetleyen kontrol mekanizmaları vardır. Nesne sahibi sunucunun erişilemez olması gibi sistem çapında etkili durumlar için sistem yöneticisi sahipliği geçici olarak üstlenme gibi gerekli önlemleri alır.

### 5.5.3 Sunucu canlılık kontrolü

Tüm dağıtık nesne paylaşım etmenleri etmen canlılık kontrolü özelliğine sahiptir. DOSMessage'ın sahip olduğu "IS_AGENT_ALIVE" mesaj tipi ile nesne canlılık kontrolü yapılır. Bu mesajın içeriği boştur ve mesajı alan etmen "IS_AGENT_ALIVE_RESPONSE" mesajı ile yanıt gönderir.

### 5.5.4 Erişim sırasında oluşabilecek durumlar

Dağıtık mimariden kaynaklanan nedenlerden dolayı nesne erişimi sırasında bazı problemler ortaya çıkabilir. Oluşan problemler sistem içinde kullanıcıya yansıtılmadan çözülür ya da kullanıcıya başarısız işlem mesajı gönderilir.

#### 5.5.4.1 Erişim sonrası nesnenin serbest bırakılmaması durumu

Okuma ya da yazma erişimi yapmış bir etmenin erişim işlemi tamamlandıktan sonra nesneyi serbest bırakmaması durumunda, sistemde tanımlanmış olan bir maksimum erişim süresi devreye girer. Çeşitli sebeplerle (uygulamanın nesneyi serbest bırakmaması, serbest bırakma mesajı yayınlanacağı sırada kullanıcı etmenin canlılığını kaybetmesi gibi) erişim isteği cevaplanmış fakat serbest bırakma mesajı alınmamış erişimler için zaman aşımı süresi dolduğunda sistem nesneyi otomatik olarak serbest bırakır ve istemci etmene geçersiz kılma mesajı gönderir. Yapılan erişim serbest bırakılmamış bir yazma erişimi ise nesne geçerli son kopyası ile güncellenir. Zaman aşımı sonunda gelen serbest bırakma erişimlerine red cevabı döndürülür ve erişim yapan etmen yaptığı işlemin başarısız olduğunu anlar. Etmen tekrar erişim yapma hakkına sahiptir.

#### 5.5.4.2 Yerel nesne sunucusuna erişilememesi durumu

Nesne sahipliğini elinde bulunduran sunucunun erişilemez olması durumunda; nesneye erişmek isteyen sunucular isteklerine cevap gelmeyeceği için zaman aşımına uğrar ve başarısız erişim yanıtını alır. Sistem koordinatörünün sahip olduğu nesne yöneticisi, periyodik olarak etmenlerin canlılık kontrolünü yaptığından, canlı olmadığını belirlediği sunucunun sahip olduğu nesnelerin sahipliğini yeni bir yazma isteği gelinceye kadar otomatik olarak üstlenir. Böylelikle nesnelerin erişilebilirliği sağlanır. Normal koşullarda sistem koordinatöründe yer alan nesne sunucusu nesne sahipliğini üstlenmez, fakat yerel bir nesne sunucusunun devre dışı kalması durumunda ilgili yerel sunucunun sahipliğini üstlendiği nesnelerin sorumluluğunu alarak bir yerel sunucu gibi davranabilir.

#### 5.5.4.3 Nesne erişim isteklerine yanıt alınamaması durumu

Yapılmak istenen erişime cevap gelmemesi durumunda; nesne erişimi yapmak isteyen etmen için belirli bir cevap alma süresi vardır. Yapılmak istenen erişime belirli bir süre cevap gelmezse sistem otomatik olarak erişim yapmak isteyen etmene erişim reddi cevabı döner. Erişim metotlarında yer alan zaman aşımı parametresi ile kullanıcı yapmak istediği erişimler için maksimum cevap alma süresini tanımlayabilir.

#### 5.5.4.4 Hatalı serbest bırakma istemi durumu

Aynı etmenin birden fazla paylaşılan nesne erişimi yapmak istemesi durumunda, ilgili etmen yapacağı her erişim için yeni bir erişim nesnesi almalıdır. Alınan erişim nesneleri yapılan istekler için tekil bir erişim tanımlayıcısı yaratacağından birden fazla erişim yapılabilir.

Etmenin yanlış erişim nesnesi ile serbest bırakma isteği yapması durumunda; sistem başarısız erişim mesajı döndürür. Yazma işlemlerinde kullanıcıya bırakılmış olan serbest bırakma işleminde, kullanıcı yazma erişim isteği için kullandığı erişim nesnesi ile serbest bırakma işlemini yapmalıdır. Her erişim tekil bir tanımlayıcı ile ifade edildiğinden yapılan serbest bırakma işleminin hangi tekil tanımlayıcı ile gerçekleşen erişim işlemi ile ilişkili olduğunu anlaması gerekmektedir. İlgili bağıntı kurulamayan erişim istekleri sonunda başarısız erişim mesajı döndürülür.

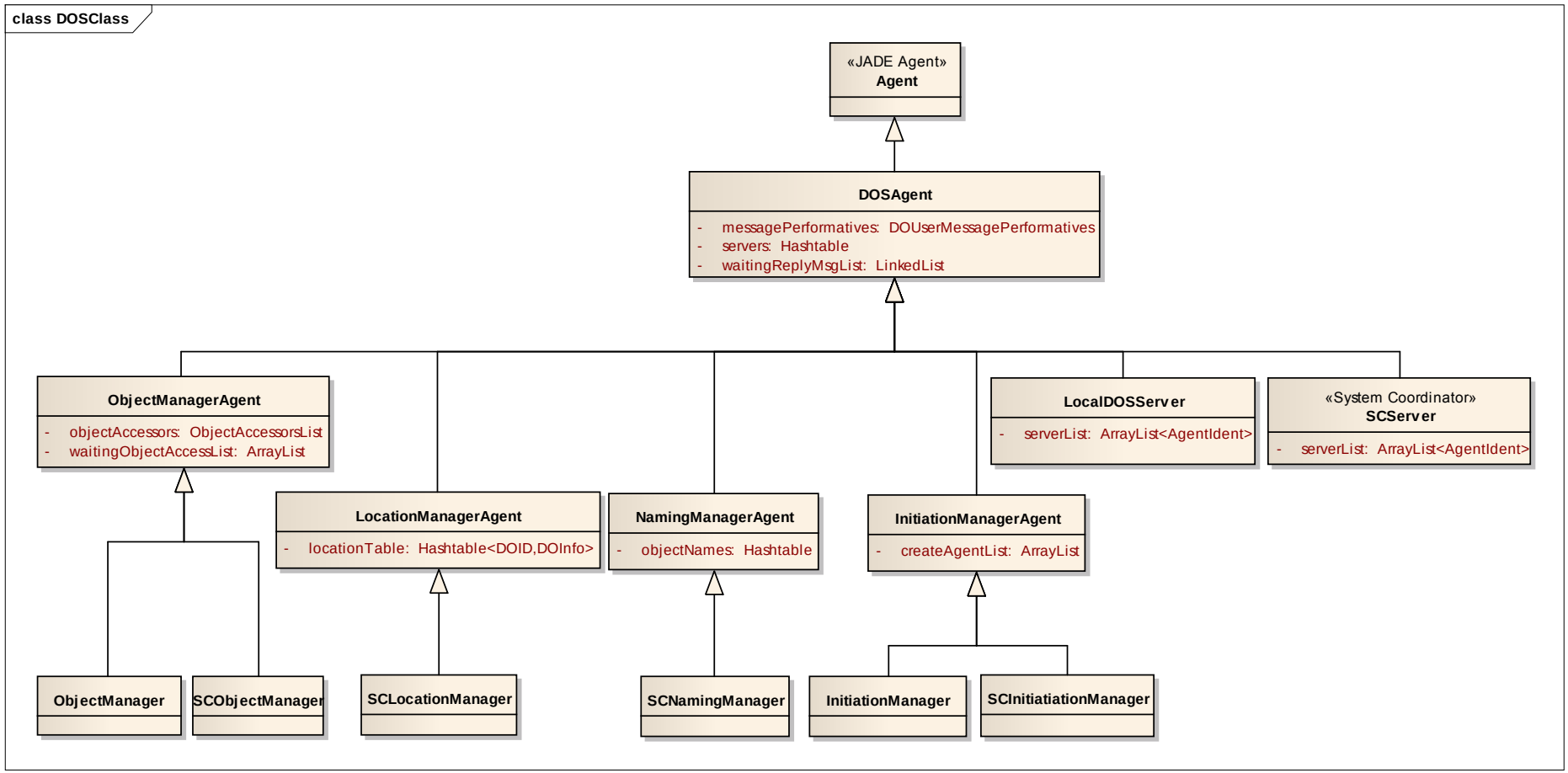
### 5.6 Yazılım Mimarisi

Bu bölümde dağıtık bellek paylaşımı sağlayan yazılım sınıfları ve bu sınıflara ait görevler açıklanmıştır.

#### 5.6.1 Dağıtık nesne paylaşım etmeni

Dağıtık nesne paylaşım sistemi etmenleri JADE kütüphanesinden gelen Agent sınıfını genişleterek oluşturulur. DOSAgent sınıfı Agent sınıfının sahip olduğu tüm özelliklerin yanında dağıtık nesne paylaşımı için gerekli metotlar ve nitelikler içerir. Nesne paylaşım sistemini kullanacak bir etmen DOSAgent sınıfını genişletmelidir.

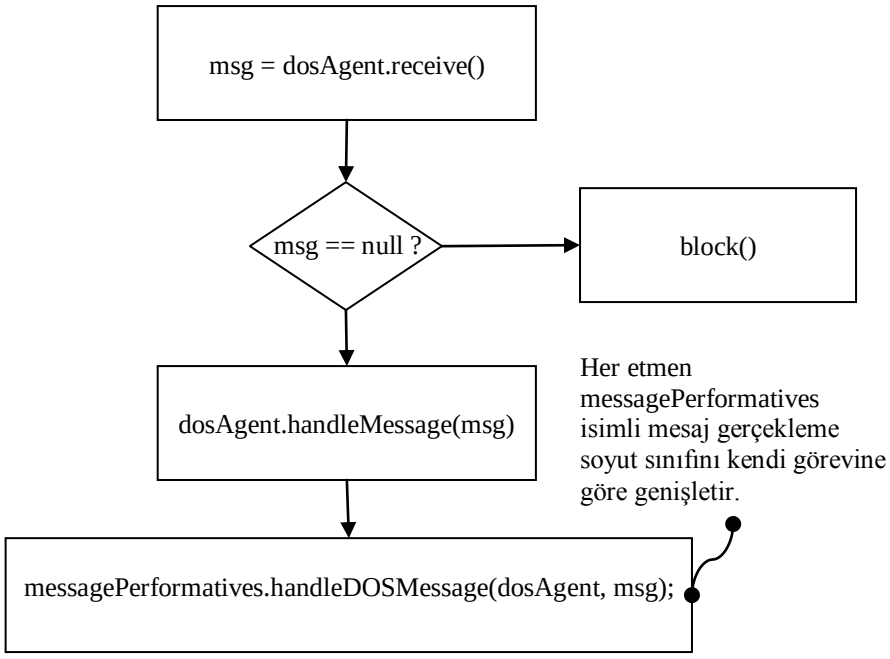
DOSAgent soyut bir sınıf olarak tanımlanmıştır ve içerisinde cevap bekleyen mesajlar listesi, haberleşme sağlanacak sunucu listesi, mesaj işleme sınıfı (DOMessagePerformatives) ve mesaj gönderip almak için yararlı metotlar içerir. Şekil 5.9'da gösterilen sınıf diyagramında yerel sunucu sistemi ve koordinatör sistemine ait etmenlerin yazılım sınıfları gösterilmektedir.




**Şekil 5.9 :** DOSAgent sınıf diyagramı.

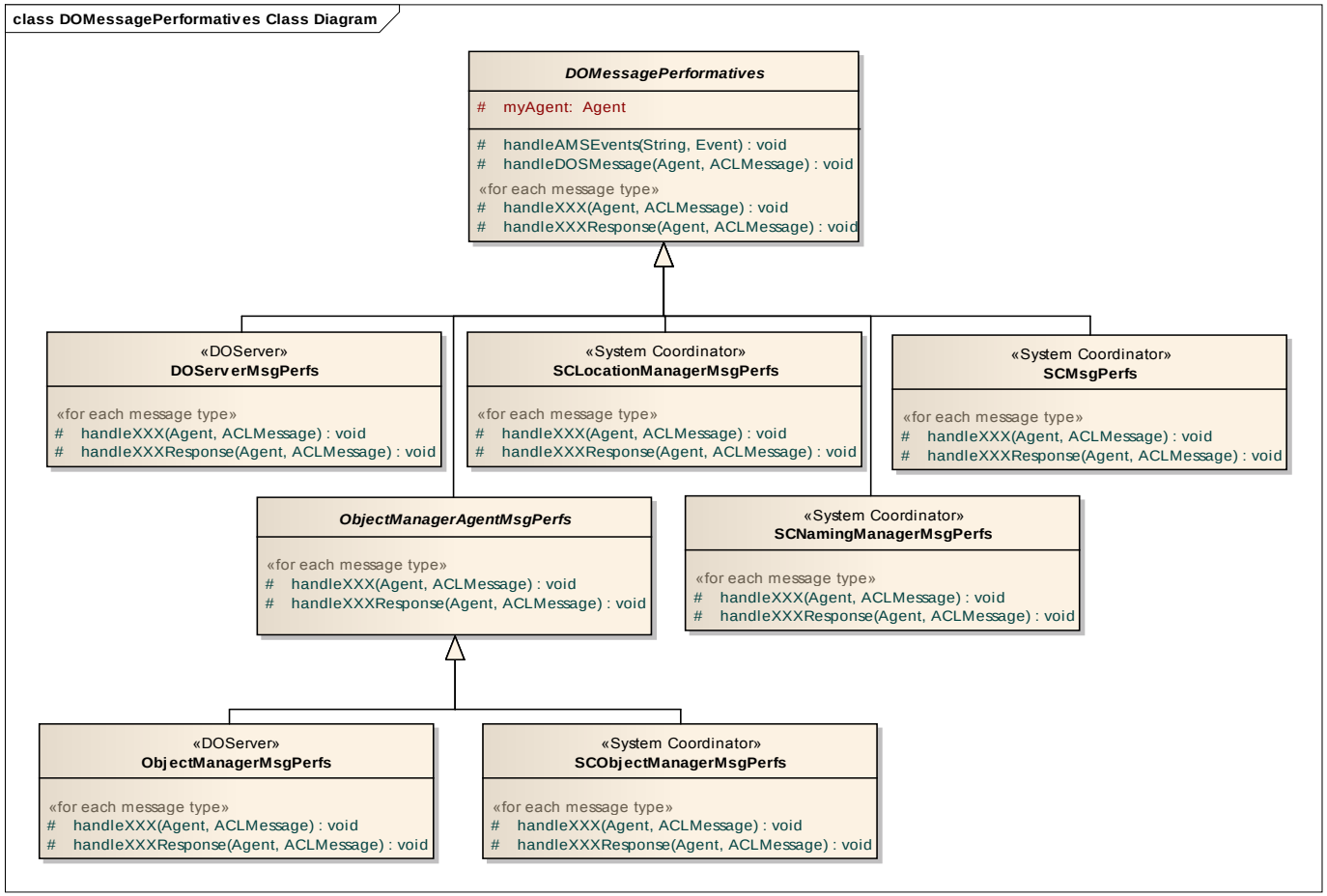
### 5.6.2 Mesajlaşma yönetimi

Dağıtılmış nesne paylaşım yazılımını gerçekleyen etmenler, ortak bir arayüz altında mesajları alma özelliğine sahiptir. Her etmen sistemde sahip olduğu sorumluluğa göre mesajları yorumlar. Mesaj alıcı sınıfı tüm etmenler için ortak bir arayüze sahiptir ve soyut olarak tasarlanmıştır. Etmenler ilgilendikleri (sorumlu oldukları) mesajların metotlarının üstüne yazarak gerekli eylemleri gerçekleştirirler. Gerçeklenen dağıtık nesne paylaşım sistemi için mesaj işlenişi Şekil 5.10'da gösterilmiştir.




**Şekil 5.10 :** Dağıtık nesne paylaşım sistemi mesaj işlenişi.

Mesajlar Agent.receive() metodu ile ACLMessage tipinde alınır ve mesajın içerik kısmında yer alan verinin DOSMessage tipinde olup olmadığı kontrol edilir [28,33]. Tüm etmenler için ortak olarak çalışan ACLMessagingBehaviour sınıfı JADE sisteminde bulunan CyclicBehaviour (periyodik çalışan bir davranış) sınıfını genişletir. ACLMessagingBehaviour sınıfı etmen kuyruğunda mesaj varsa bu mesajı kuyruktan çeker ve alınan mesaj DOSMessage tipinde ise etmenin mesaj işleyici sınıfı olan DOMessagePerformatives sınıfının handleMessage(msg) isimli metoduna gönderir. Geliştirilen nesne paylaşım sistemindeki mesaj eylemleri Şekil 5.11'de gösterilmiştir.

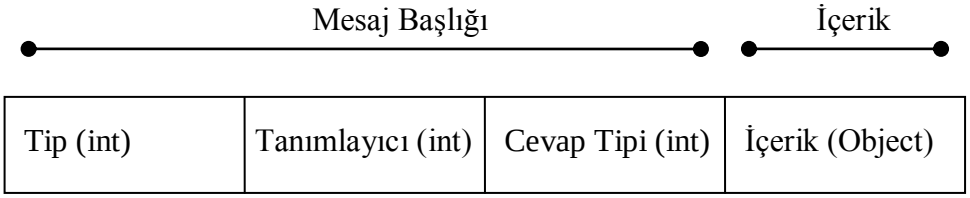



**Şekil 5.11 :** DOS ortamında mesaj eylemleri (performatives) diyagramı.

Her etmenin yapacağı işe göre genişlettiği DOMessagePerformatives soyut sınıfı için gösterilen handeleXXX(Agent,ACLMessage) ve handleXXXResponse(Agent, ACLMessage) metotlarında yer alan XXX gösteriminin yerini mesaj tipinin adı alır. Sistemde erişme isteği ve erişme isteğine yanıt mesaj tipleri için handleAccessObject(Agent, ACLMessage), handleAccessObjectResponse(Agent, ACLMessage) metotları gerçeklenmiştir.

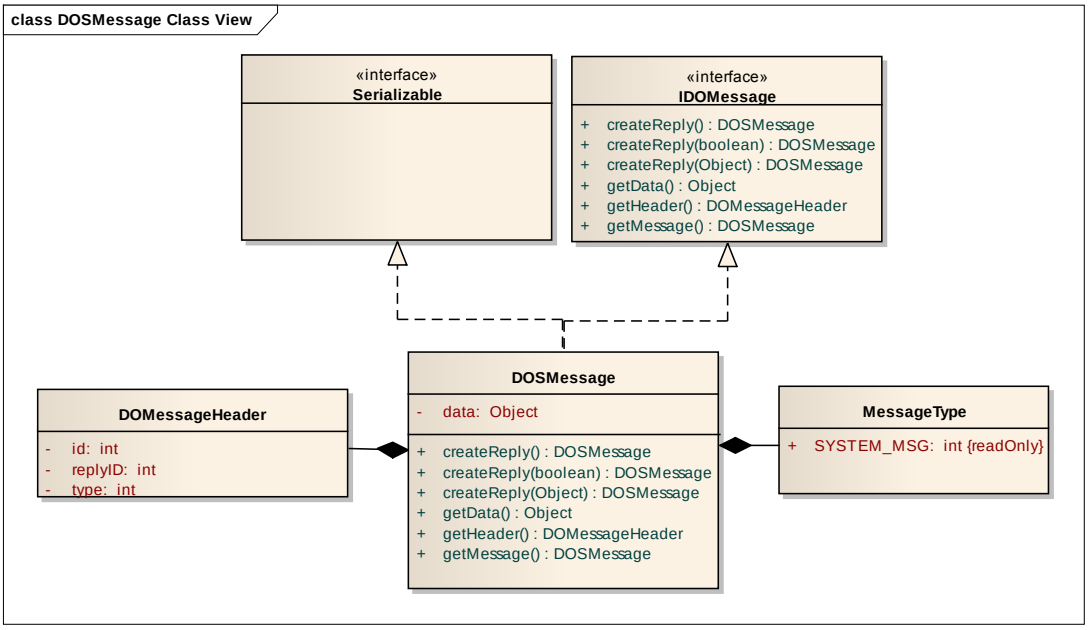
#### 5.6.2.1 Mesaj yapısı

Etmenlerin mesajlaşmalarda ACLMessage sınıfını kullandıklarından ve bu yapının içeriğinden bahsedilmişti [28,33]. Dağıtık nesne paylaşım sistemi DOSMessage isimli mesajlaşma sınıfını ACLMessage sınıfının içerik bölümüne koyarak mesaj alışverişi gerçekleştirir. Şekil 5.12'de DOSMessage isimli sınıfın veri alanları gösterilmiştir.




**Şekil 5.12 :** DOSMessage sınıfının veri alanları.

DOSMessage mesaj başlığı (message header) ver içerik (content) kısımlarından oluşur. Mesaj başlığı tip, tanımlayıcı ve cevap tipi olmak üzere üç veri alanı içerir. Tip alanı özel kullanımlar için ayrılmıştır. Sistem içinde tip alanı mesajın sistem mesajı olduğunu ifade eden eden "0" değerini alır. Tip mesajına yeni bir değer eklenerek tanımlayıcı ve cevap tipinin farklı yorumlanabilmesi sağlanmıştır. Tanımlayıcı alanı mesajın tanımlayıcı bilgisidir ve bölüm 5.3.1.2'de anlatılan mesaj tiplerine karşılık gelen değerleri alır. Cevap tipi alanı gönderilen mesaj için beklenen cevabın tipini belirtir. İçerik alanı mesaj tipine göre yorumlanacak bir nesne içerir. Şekil 5.13'te DOSMessage sınıf diyagramı gösterilmiştir.

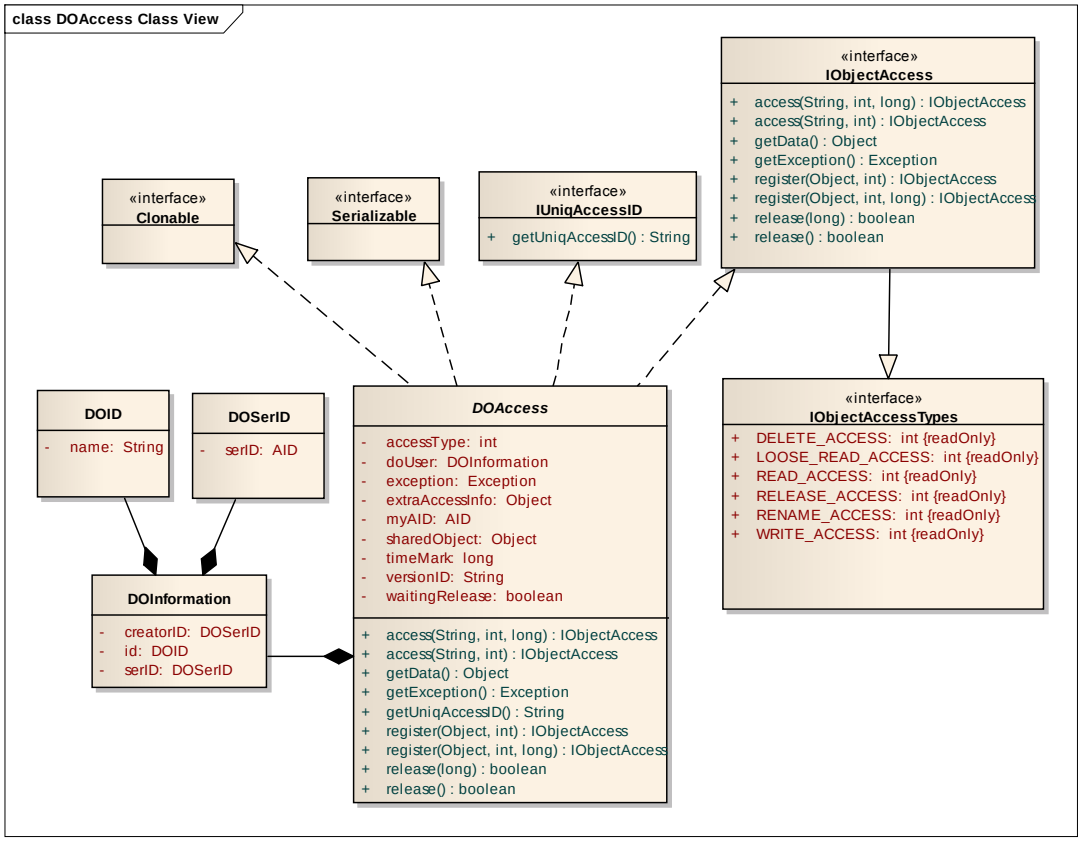



**Şekil 5.13 :** DOSMessage sınıf diyagramı.

MessageType olarak isimlendirilmiş sınıf DOSMessage içinde statik olarak tanımlanmıştır.

### 5.6.3 Erişim sınıfları

Nesne erişimi yapılırken DOMessage'ın içerik bölümüne DOAccess isimli yazılım sınıfı koyulur. DOAccess sınıfı nesne ismi, yaratıcı sunucunun tanımlayıcı bilgisi, erişim yapan sunucunun tanımlayıcısı, erişim tipi, tekil bir erişim belirteci, yapılan erişimin serbest bırakma gerektirip gerektirmediği, erişim zamanı, nesne versiyon bilgisi ve erişim ile ilgili farklı bilgiler taşımak için Object tipinde alanlardan oluşmaktadır. Nesne erişimi için kullanılan DOAccess isimli sınıf için hazırlanmış olan sınıf diyagramı Şekil 5.14'te gösterilmiştir.




**Şekil 5.14 :** DOAccess sınıf diyagramı.

Kullanıcı etmenin DOAccess sınıfını açıkça yaratma (Java programlama dilinde new anahtar kelimesi ile yaratma) yetkisi yoktur. Sistem erişim tanımlayıcısını otomatik olarak kullanıcıya bırakmadan yaratacağından erişim işlemlerinde bu sınıf bir metot çağrısı ile elde edilir.

## 5.7 Nesne Erişim Süreleri

Çoklu etmen ortamında gerçeklenen nesne paylaşım sistemi yerel ve uzak sunucular aracılığıyla nesne paylaşımını sağlamaktadır. Merkezi sunucu yönteminde nesneler bir sunucu üzerinde yer alır ve erişim istekleri bu sunucu tarafından karşılanır. Bu yüzden merkezi sunucunun çalıştığı bilgisayarın performansı doğrudan sistem performansını etkiler. Bunun yanında merkezi sunucunun yerleşke olarak istemciye olan uzaklığı da paylaşılan nesnenin istemciye ulaşma süresinde önemli bir etkiye sahiptir.

Geliştirilen sistemde nesneler sadece olumsuz koşullarda geçici olarak koordinatör görevini üstlenen sunucuda yer almaktadır. Normal çalışma şartlarında, yazma isteği sonucu sahipliği ele geçirmiş yerel sunucular aracılığıyla nesne erişimi

gerçekleşmektedir. Yapılan nesne erişim isteklerinin sonuçlanma süresi yerel sunucuların işlem gücü ile yakından alakalıdır, fakat nesnelerin ağ üzerinde farklı sunucularda yayılmış olduğu düşünüldüğünde her bir sunucuya düşen erişim isteği azaltılmış olacağından dağıtık sistem yapısı yüksek seviyede kullanılmıştır.

Çizelge 5.2'de DOSMAS (çoklu etmen ortamında nesne paylaşım sistemi) kontrolünde gerçekleşen yerel ve uzak sunucular kullanılarak yapılan nesne erişim işlem süreleri gösterilmiştir.

**Çizelge 5.2 :** DOSMAS'da nesne erişim süreleri.

| Erişim Tipi | Yerel Erişim (ms) | Uzak Erişim (ms) |
|---|---|---|
| Okuma | 24.684 | 35.736 |
| Gevşek Okuma | 10.157 | 18.631 |
| Yazma | 25.736 | 39.714 |
| Yeniden Adlandırma | 14.600 | 29.500 |
| Serbest Bırakma | 11.157 | 14.423 |
| Silme | 13.285 | 27.341 |
| Kayıt | 18.700 | 21.368 |

Nesne erişim süreleri ölçülürken 3.2 GHz hıza sahip Intel I5 işlemcili bilgisayarlar kullanılmıştır. Yapılan ölçümler ortalama nesne erişim sürelerini göstermektedir ve işlemcinin yoğunluğu, kullanılan işletim sistemi, ağ durumu, kullanılan çoklu etmen sisteminin mimarisi gibi etkenler erişim süresini etkilemektedir. Süre olçümünde kullanılan paylaşılan nesne içerisinde String tipinde veri içermektedir ve yerel nesne yöneticileri sahipliğini üstlendikleri nesneler içinde teyit amaçlı olarak yerleşke yöneticisine sorgu yapmıştır.

Çizelge 5.3 ve Çizelge 5.4'te sırasıyla Java RMI mekanizması kullanarak ve çoklu etmen ortamında nesne tabanlı dağıtık bellek paylaşım sistemi kullanarak içerisinde 1 Kbyte, 10 Kbyte, 100 Kbyte ve 1000 Kbyte veri dizisi bulunan paylaşılan nesnelere yapılan okuma/yazma erişimlerinin ortalama süreleri verilmiştir. Geliştirilen nesne paylaşım sistemi JADE mesajlaşma altyapısını kullandığından, nesne erişim yönetim sistemi iplikleri JADE davranış sınıfları olarak gerçeklendiğinden ve nesne paylaşım periyodikleri JADE zamanlayıcısının kontrolünde olduğundan bu süreler kullanılan çoklu etmen mimarisi performansı ile yakından ilişkilidir. Yerel olarak yapılan erişimler kullanıcı ile aynı makinede yaşayan sunucular aracılığıyla, uzak olarak yapılan erişimler ise kullanıcının yaşadığı makineden farklı bir makinede yaşayan sunucular aracılığıyla yapılan erişimleri göstermektedir.

**Çizelge 5.3 :** Java RMI ile nesne erişim süreleri.

| Nesne Boyutu | Yerel Okuma (ms) | Uzak Okuma (ms) | Yerel Yazma (ms) | Uzak Yazma (ms) |
|---|---|---|---|---|
| 1 K | 3.444 | 13.598 | 4.888 | 14.543 |
| 10 K | 4.777 | 14.865 | 5.888 | 15.435 |
| 100 K | 13.678 | 15.128 | 14.467 | 16.348 |
| 1000 K | 44.111 | 94.534 | 96.134 | 101.534 |

**Çizelge 5.4 :** DOSMAS ile farklı boyutlardaki nesnelere erişim süreleri.

| Nesne Boyutu | Yerel Okuma (ms) | Uzak Okuma (ms) | Yerel Yazma (ms) | Uzak Yazma (ms) |
|---|---|---|---|---|
| 1 K | 15.185 | 36.394 | 15.841 | 31.275 |
| 10 K | 15.444 | 39.721 | 15.765 | 29.488 |
| 100 K | 17.543 | 50.367 | 17.681 | 47.601 |
| 1000 K | 92.183 | 204.001 | 112.672 | 227.105 |

# 6. ÇOKLU ETMEN ORTAMINDA NESNE TABANLI DAĞITILMIŞ ORTAK BELLEK UYGULAMASI

Geliştirilen sistem etmenler arasında iletilen nesne erişim mesajlarını izlemek için bir arayüze sahiptir. Arayüz, kullanıcının yerel ve sistem koordinatörü birimlerini yaratırken ayarladığı bir parametrenin değerine göre gösterilir ya da gösterilmez. Geliştirilen grafik arayüz ile nesnelere yapılan erişimleri izlemek mümkündür.

Sistemin sahip olduğu arayüzlerin haricinde bir rezervasyon uygulaması geliştirilmiştir. Bu uygulama ile kullanıcıya koltuk kapasitesi ve benzersiz bir isimle bir etkinlik salonu (sinema, konferans salonu gibi) yaratma ve rezervasyon yapma olanağı sunulmuştur. Geliştirilen uygulamada yaratılan salon nesnesi bir bütün olarak ele alınır. Salon nesnesi yaratılma aşamasında verilen kapasite kadar koltuk içermektedir ve nesne okuma yazma işlemleri yapılırken nesne bir bütün olarak ele alınır. Salon durumunu görüntülemek isteyen bir kullanıcı görüntülemek istediği salonun ismi ile bir okuma işlemi gerçekleştirir ve bu işlem sonunda ilgili salonun durumunu (satılmış ve uygun koltuklar) görüntüler. Kullanıcı için hazırlanmış arayüzden faydalanarak koltuk seçimi yapılır. Satışı gerçekleşmiş koltuklar için sistem seçim yapmaya izin vermez ve kullanıcı koltuk seçimini yaparken işlemler yerel nesne kopyası üzerinde gerçekleşir (paylaşılan nesneye bir çağrı yapılmaz). Kullanıcı seçimlerini tamamlayıp onay düğmesine bastığında (yazma isteği) sistem ilgili salon nesnesinin en güncel kopyasını istemciye iletir. Kullanıcının seçtiği koltuklar uygunluğunu koruyorsa koltuk seçim işlemi (yazma işlemi) başarılı mesajı döndürülür. Kullanıcı seçim işlemleri ile uğraşırken bir başka kullanıcı seçilen koltuklardan herhangi birini satın almışsa sistem başarısız koltuk seçimi mesajı döndürerek güncel salon durumunu kullanıcıya gösterir.

## 6.1 Sistem Arayüzleri

Çoklu etmen ortamında dağıtılmış nesne paylaşım uygulamasını kullanırken kullanıcı çeşitli grafik arayüzler yardımıyla kurulum parametrelerini ayarlayabilir. Kullanıcı etmen, yerel sunucu ve koordinatör, yazılımının sunduğu yapıcılar

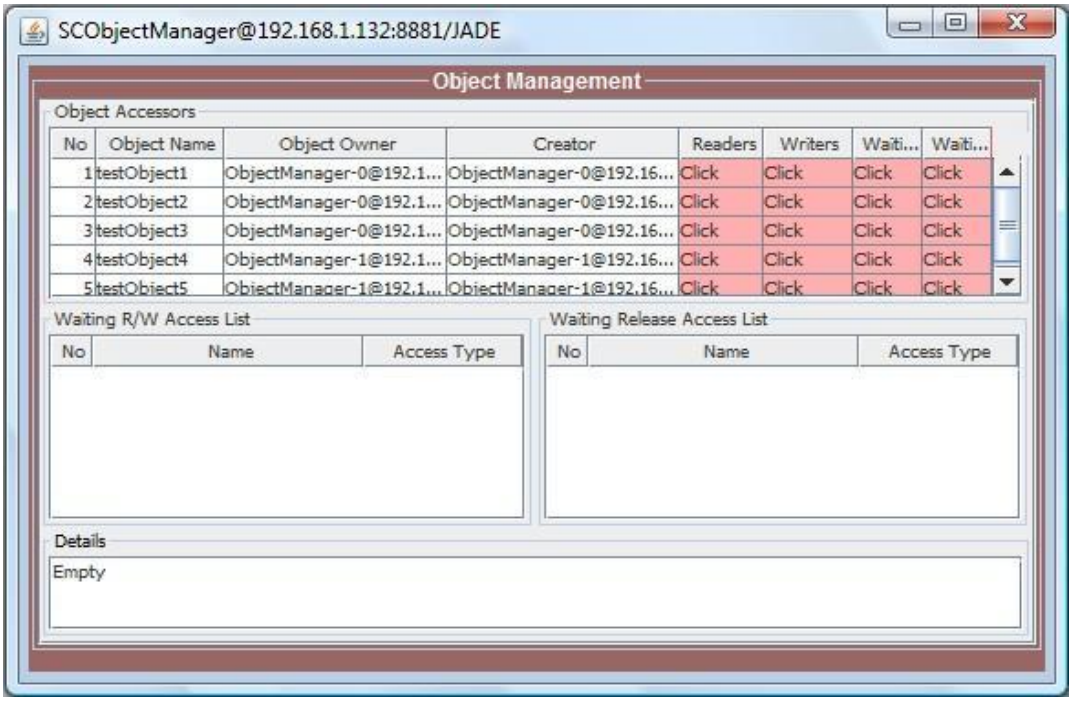
aracılığıyla herhangi bir arayüze gereksinim duyulmadan çalıştırılabilir.Grafik arayüzlerin kullanılması sistemde meydana gelen olayları izlemeyi kolaylaştırır, fakat grafik arayüzü işlemleri kullanılan bilgisayar ve uygulamanın performansını düşürür.

### 6.1.1 Sistem koordinatörü arayüzü

Sistem koordinatörü yaratılırken "SystemCoordinator" varsayılan ismi ile yaratılır ve yaratılan makinenin IP adresi ve "8881" numaralı portu kullanılarak platform isimlendirilir. Yapıcı sınıfa koordinatör ismi, IP ve port numarası değerlerinin verilmesiyle bu parametreleri değiştirmek mümkündür.

Sistem koordinatörü yaratılırken yapıcı sınıf aracılığıyla grafik arayüzün gösterilmesi seçilmişse sistem düzeyinde yaşayan tüm nesnelerin yaratıcı, sahiplik, okuyucu, yazıcı, erişme isteği bekleyen, serbest bırakma isteği bekleyen listeleri görüntülenebilir. Sistem koordinatöründe yer alan bu bilgiler uygulamayı izlemek amacıyla kullanılır. Sistem düzeyinde paylaşılan bir nesnenin sahipliğini üstlenen bir sunucuda problem oluşması durumunda sistem koordinatörü sahipliği üstleneceğinden bu arayüz aracılığıyla sahiplik bilgisi görüntülenebilir.

Şekil 6.1'de sistem koordinatörünün sistem çapında kontrol ettiği nesneler gösterilmektedir. Test amaçlı olarak kaydedilen(String türünde bilgi içeren Java nesneleri) testObject1, testObject2, testObject3 isimli nesneler ObjectManager-0 isimli yerel sunucunun, testObject4, testObject5, testObject6 isimli nesneler ise ObjectManager-1 isimli yerel sunucunun sahipliğindedir.

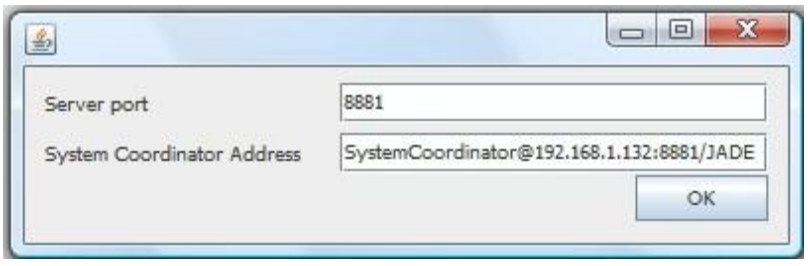



**Şekil 6.1 :** Sistem koordinatörü nesne yönetim arayüzü.

Paylaşılan nesnenin sahiplik bilgisi ve ilgili nesneye erişen etmenler arayüz üzerinde ilgili alanlar üzerine tıklanarak izlenir.

### 6.1.2 Yerel sunucu arayüzü

Çoklu etmen ortamında dağıtık bellek paylaşım sisteminde yerel sunucu yaratmak için basit bir arayüz bulunmaktadır. Şekil 6.2'de gösterildiği gibi, sistem koordinatörünün tam adı (platform adını da içerecek şekilde) ve bağlanılmak istenen port numarası verilerek yerel sunucu varsayılan bir isimle yaratılır.

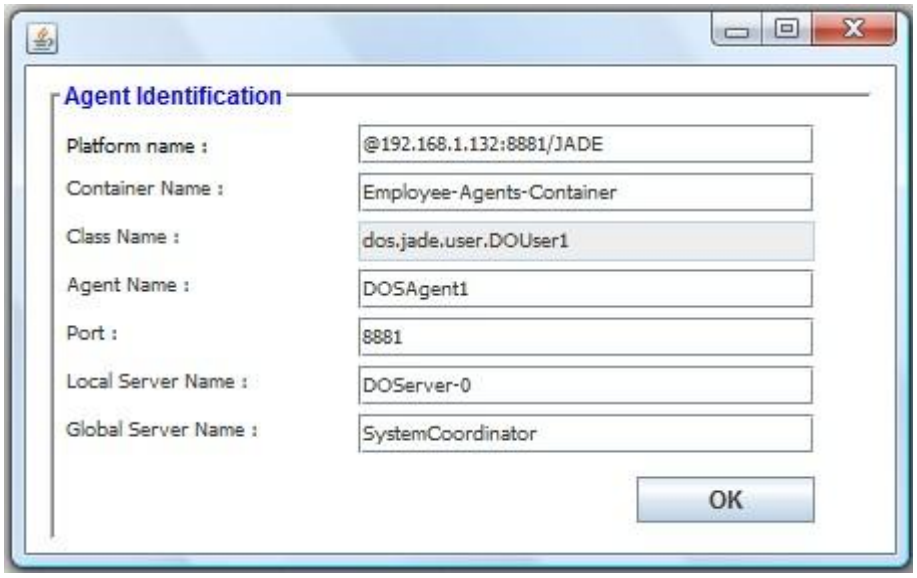



**Şekil 6.2 :** Yerel sunucu yaratma arayüzü.

Sistem içinde yerel sunucu yapıcı sınıfının sahip olduğu parametreler aracılığıyla farklı arayüzler geliştirilerek ya da yazılımsal olarak farklı parametreler ile yerel sunucu yaratılabilir. Sunucuya seçilen bir ismi vermek gibi esnek yapı kullanıcı uygulamanın yapıcı sınıfa geçtiği parametreler ile ayarlanabilir.

### 6.1.3 Kullanıcı etmen arayüzü

Yaşayan bir yerel sunucu ve sistem koordinatörüne bağlanmak amacıyla kullanıcı etmen sistem içinde yer alan yapıcı sınıf parametrelerini kullanabileceği gibi Şekil 6.3'te gösterilen arayüzden de faydalanabilir.

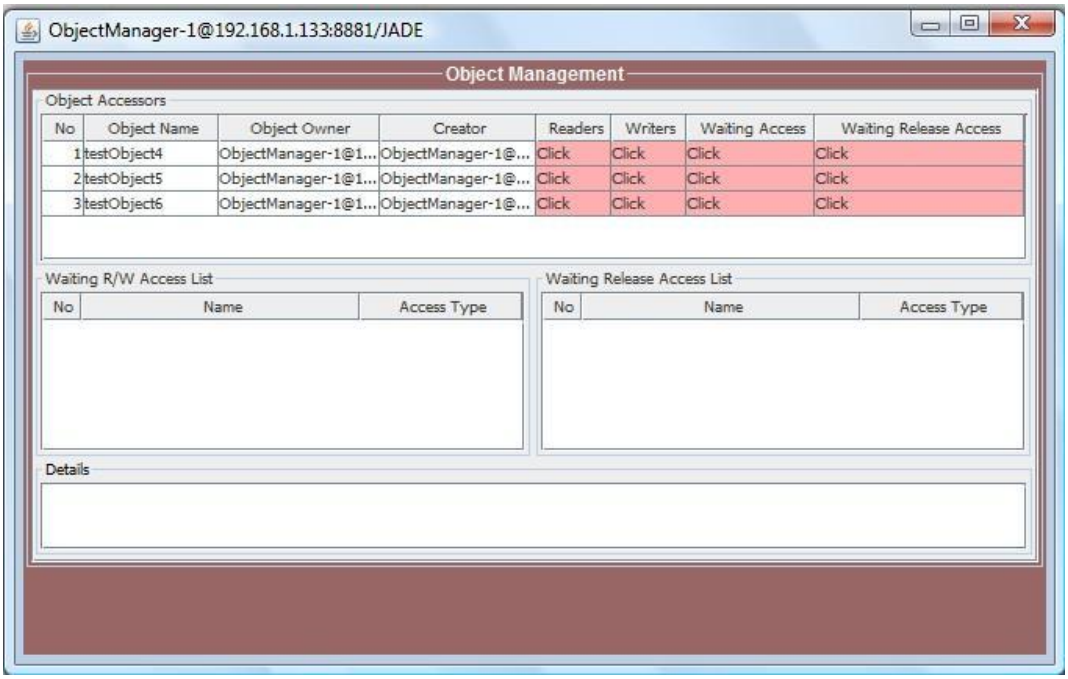



**Şekil 6.3 :** Kullanıcı etmen yaratma arayüzü.

Kullanıcı etmen bağlanacağı platform ismi, yaşayacağı container adı, yaratılacağı sınıfın tam ismi, bağlanacağı port numarası, etmen ismi, yerel sunucu ismi parametrelerini ayarlayarak nesne paylaşım platformuna bağlanır.

### 6.1.4 Paylaşılan nesne arayüzü

Paylaşılan nesne arayüzü sistem koordinatörünün sahip olduğu nesne arayüzü ile aynı işleve sahiptir. Arayüz üzerinde sadece yerel sunucunun daha önce eriştiği ya da yarattığı nesneler görüntülenir. Şekil 6.4'te ObjectManager-1 isimli yerel sunucunun nesne yöneticisinde bulunan nesneler gösterilmiştir.

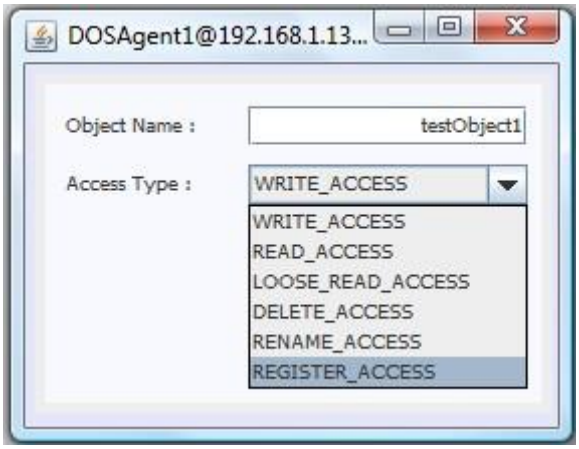



**Şekil 6.4 :** Yerel sunucu nesne yönetim arayüzü.

Nesne erişimi sırasında okuyucu, yazıcı, erişim bekleyen istekler, serbest bırakmayı bekleyen istekler görüntülenebilir.

### 6.1.5 Kullanıcı etmen nesne erişim arayüzü

Nesne erişimleri, sisteminin sunmuş olduğu metotlar aracılığıyla yapılır. Kullanıcı etmenin yaratılması aşamasında arayüz gösterme özelliği yapıcı sınıf parametresi ile etkin duruma getirilirse test amaçlı olarak Şekil 6.5'te gösterilen nesne erişim arayüzü yaratılır.




**Şekil 6.5 :** Yerel sunucu nesne erişim arayüzü.

Arayüz aracılığıyla test amaçlı olarak nesne yaratma, okuma, gevşek okuma, yazma, silme, yeniden adlandırma istekleri üretilebilir. Yaratılan nesneler String alanı içeren Java nesneleridir ve bir yazma isteği sonucunda sistem tarafından yaratılan rastgele bir String tipinde değer mevcut değerin sonuna eklenir.

### 6.2 Çalışmanın Uygulama Alanı

Dağıtık ortamda nesne paylaşmayı amaçlayan tüm uygulamalarda nesne paylaşım sistemi kullanılabilir. Rezervasyon, bilet satın alma, müzayede sistemleri gibi uygulamalar örnek olarak gösterilebilir.

### 6.3 Rezervasyon Uygulaması

Rezervasyon uygulaması, yeni bir etkinlik yaratmak, yaratılmış bir etkinlikten yer ayırtmak, etkinlik silme, etkinliği yeniden adlandırma, etkinlik yer durumunu görüntüleme özelliklerini içerir.

Yaratılan her bir etkinlik paylaşılan bir nesnedir ve tekil tanımlayıcıya sahip olmalıdır. Nesne paylaşım sistemi için nesneler bir bütün olarak ele alındığından, nesnenin iç yapısıyla ilgili meydana gelecek değişikliklerden uygulama programcısı sorumludur.

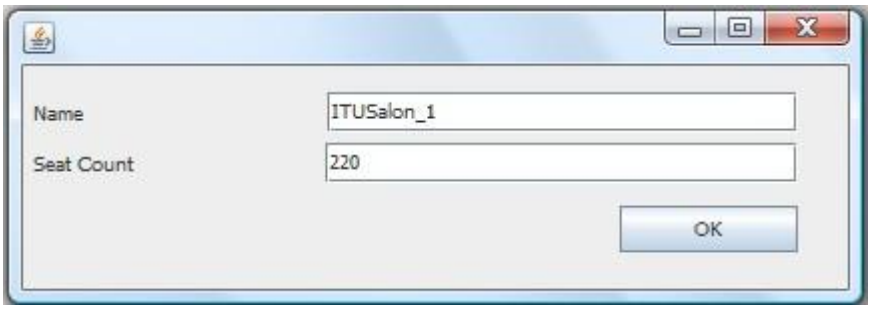
Nesne paylaşım sistemi için programlanmış rezervasyon uygulaması içerisinde yaratılma aşamasında tanımlanmış sayıda koltuk içeren bir salon nesnesinin paylaşımını amaçlamaktadır. İlgili nesne paylaşılırken salon bir bütün olarak ele alınır ve kullanıcı bir koltuk seçimi yapmak istiyorsa yazma isteği için tüm salon nesnesi elde edilir. Sistem gerekli uygunluk kontrollerini yaparak kullanıcıya yapılan seçimin başarılı olup olmadığı konusunda bilgi verir.

#### 6.3.1 Etkinlik yaratma

Rezervasyon uygulaması ile nesne paylaşım sistemine katılan bir kullanıcı etmenin, benzersiz bir etkinlik ismi ve satılabilecek bilet sayısı bilgilerini girerek yeni bir etkinlik yaratması işlemidir. Bu işlem nesne paylaşım sistemi açısından bir kayıt işlemidir.

Nesne paylaşım sistemi için her bir etkinlik paylaşılan bir nesneyi temsil eder. Rezervasyon uygulamasını kullanan tüm etmenler yeni bir etkinlik yaratma hakkına sahiptir. Etkinlik yaratma işlemi sırasında, etkinlik için kayıt edilmek istenen tanımlayıcı ile daha önce nesne yaratılmış ise ilgili etkinlik nesnesinin sisteme kaydına izin verilmez.

ITUSalon_1 isimli ve koltuk kapasitesi 220 olan bir salon'un yaratılması Şekil 6.6'da gösterilen arayüz aracılığıyla yapılır.




**Şekil 6.6 :** Rezervasyon uygulaması etkinlik yaratma arayüzü.

Yapılan salon yaratma isteği başarısız olursa, ekranda bir uyarı mesajı gösterilir. İsteğin başarılı olması durumunda yaratılan salon arayüz aracılığıyla gösterilir.

### 6.3.2 Rezervasyon durumu

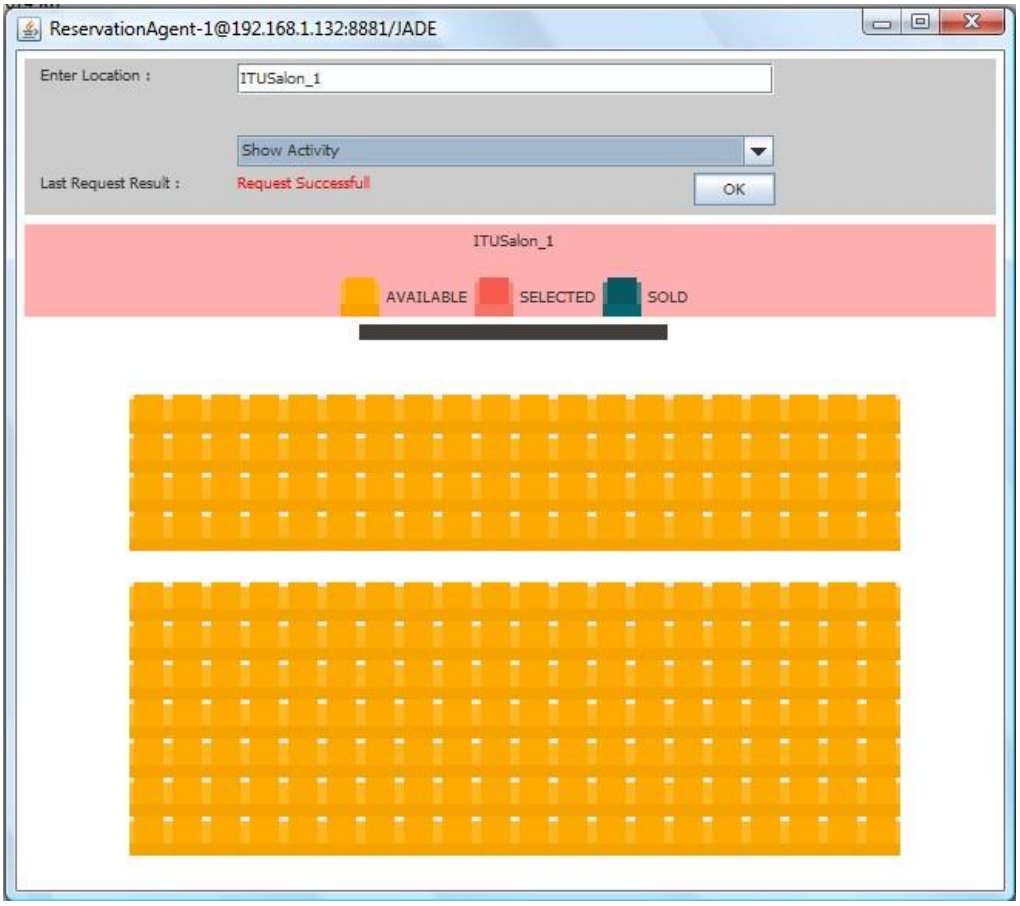
Etkinliğin (salonun) benzersiz ismi okuma parametresi olarak girilerek gerçekleştirilen dağıtık nesne paylaşım sisteminde rezervasyon nesnesinin okunma işlemidir. Okunan nesne etkinlik için satışa sunulmuş koltukların durumunu gösterir. Kullanıcı, etkinlik ismi ile yaptığı okuma isteği sonucu olarak bir arayüz aracılığıyla koltukların durumunu gözlemler. Okunmak istenen nesne (etkinlik) sistemde kayıtlı değilse ya da erişim ile ilgili bir sorun yaşanıyorsa uyarı mesajı döndürülür.

Rezervasyon durumu incelenirken bir başka kullanıcı satın alma işlemi gerçekleştirmiş olabilir. Kullanıcı okuma isteği yaptıktan sonra, farklı bir istemci tarafından yapılan yazma sonucu olarak ekranda gösterilen uygun koltukların durumu değişmiş olabilir.

Rezervasyon durum bilgisi alınan bir etkinlik için koltuk satın alma işlemi arayüz aracılığıyla yapılır. Arayüzde gösterilen koltuklardan istenilenler seçilir ve yazma isteği yapılarak rezervasyon gerçekleştirilir.

Bir etkinlik için rezervasyon durumunu görüntülemek nesne paylaşım sistemi için basit bir okuma işlemidir. Yerel sunucu sistemi ve sistem koordinatörü birimleri yardımıyla nesne okunur ve arayüz ile güncel durum kullanıcıya gösterilir.

Şekil 6.7'de salon ismi "ITUSalon_1" olarak girilip "Show Activity" seçimi yapıldıktan sonra "OK" isimli düğmeye basılarak "ITUSalon_1" isimli nesnenin güncel durumu uygulama tarafından gösterilmiştir.




**Şekil 6.7 :** Rezervasyon durumu görüntüleme.

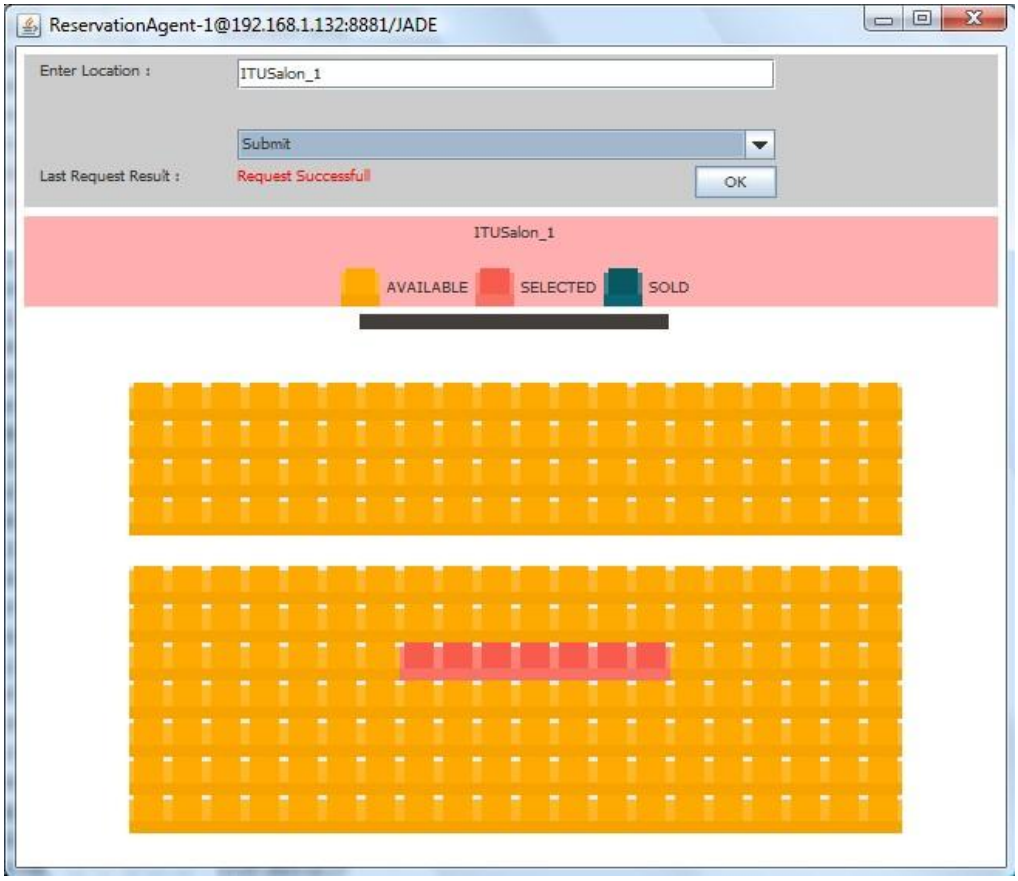
Sorgulanan nesnenin sistemde kayıtlı olmaması ya da nesneye ulaşılamaması durumunda işlem başarısız mesajı alınır.

### 6.3.3 Bilet satın alma isteği

Okuma isteği ile görüntülenen etkinlik için koltuk seçimi yapıp onaylama işlemidir. Bilet satın alma dağıtık nesne paylaşım sisteminde yazma işlemidir. Seçim yapabilmek için öncelikle rezervasyon yapılmak istenen salonun kayıtlı paylaşılan

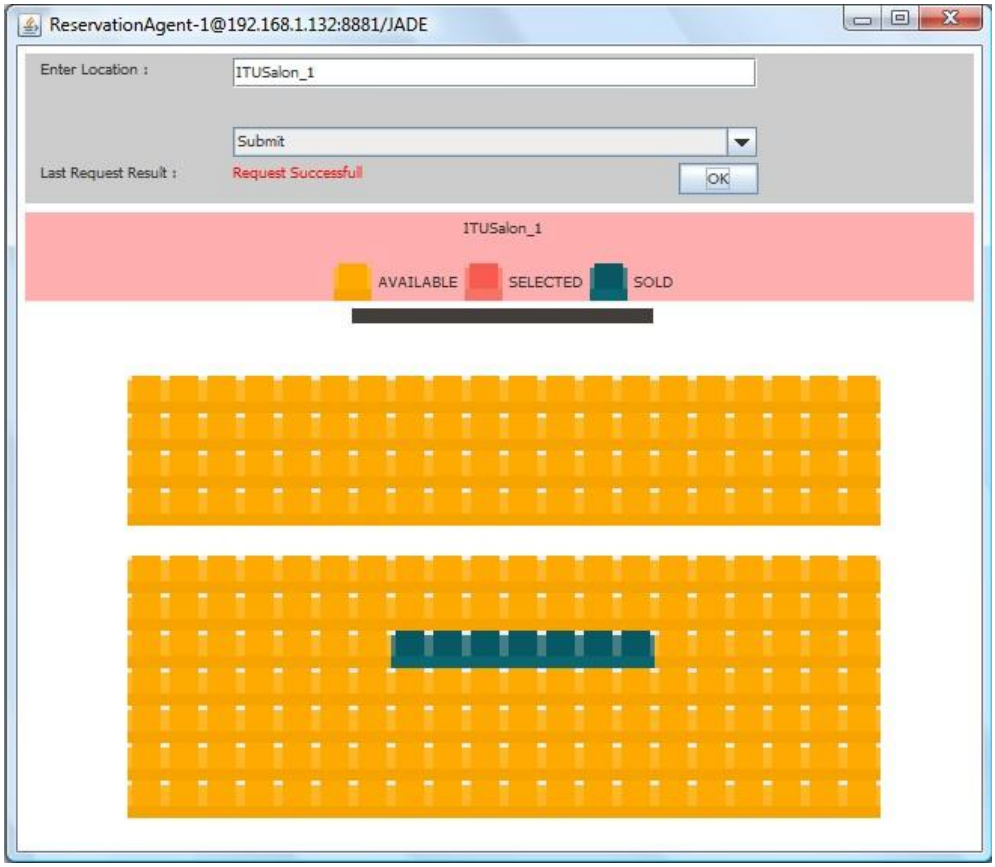
nesne ismi ile okuma işlemi gerçekleştirilir ve ekranda güncel koltuk durumu görüntülenir.

Şekil 6.8'de "ITUSalon_1" isimli salon için koltuk seçimi gösterilmiştir. Burada yapılan koltuk seçimi sistem çapında etkili değildir. Değişiklik sadece nesneyi kullanan etmenin yerel kopyasında (yerel sunucu ve sistem koordinatörünün değişiklik ile ilgili bilgisi yoktur) yapılmıştır. Değişikliğin sistem düzeyinde yani paylaşılan nesne üzerinde etkili olması için "Submit" seçeneği seçildikten sonra "OK" düğmesine basılarak yazma isteği yapılmalıdır.




**Şekil 6.8 :** Rezervasyon koltuk seçimi.

Şekil 6.9'da "ITUSalon_1" isimli nesne için koltuk seçimi yapılmış ve başarılı bir rezervasyon (yazma isteği) gerçekleşmiştir.




**Şekil 6.9 :** Rezervasyon koltuk onayı.

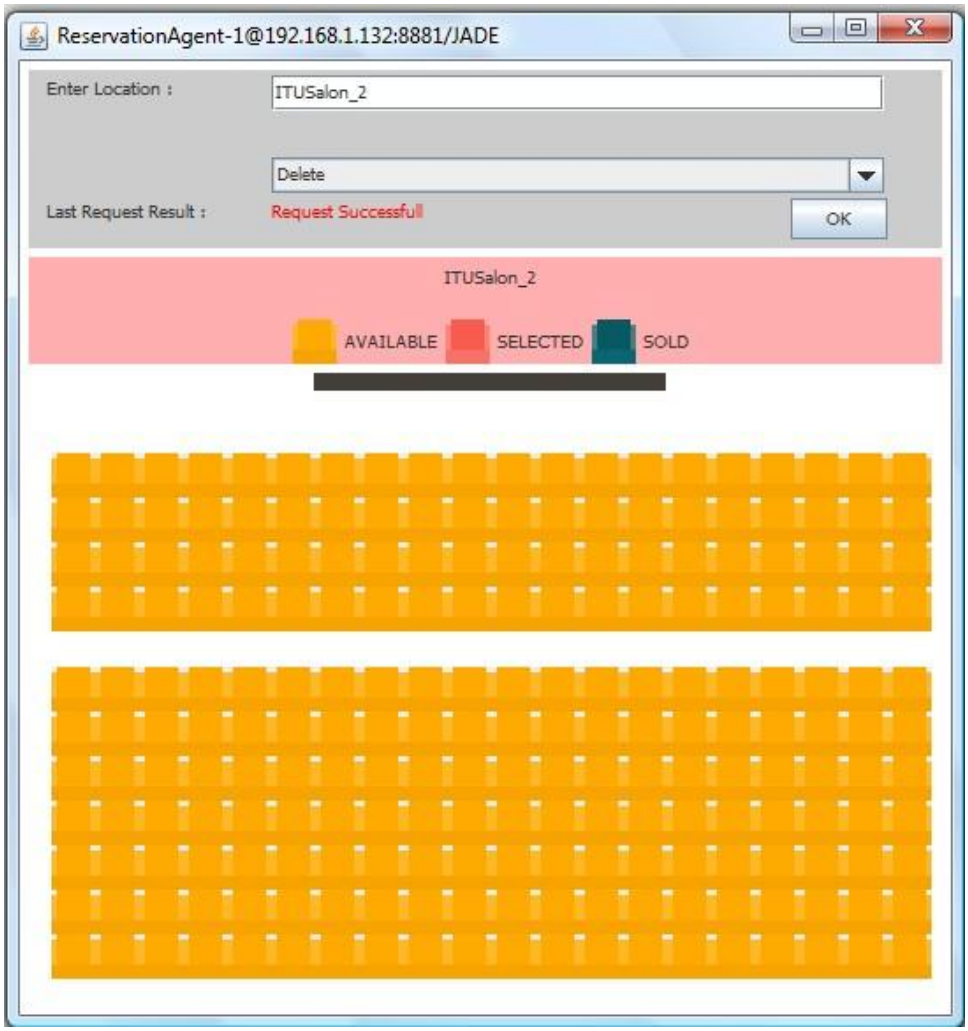
Okuma isteği ile görüntülenen bir rezervasyon, koltuk seçimi yapılıp onaya gönderildikten (yazma isteği) sonra başarısız olma olasılığı vardır. İşlemler sırasında farklı bir kullanıcı aynı koltukları ya da en az bir tane çakışan koltuk için erişim isteğinde bulunmuş ise yazma isteğini ilk yapan kullanıcının seçimleri geçerli olur ve diğer kullanıcıya güncel salon durumu gönderilir. Yazma isteği başarısız olan kullanıcı yeniden seçim yaparak yazma işlemini gerçekleştirebilir.

İki ya da daha fazla kullanıcı aynı anda aynı nesneye erişip çakışmayan (tamamen farklı) koltukları seçmesi durumunda sorunsuz bir şekilde rezervasyon işlemi gerçekleşir.

### 6.3.4 Etkinlik silme

Daha önce yaratılmış bir etkinliği, sistemden tamamen silme işlemidir. Etkinliğin silinmesi sonunda aynı isimle tekrar bir etkinlik yaratılabilir. Salonu silme yetkisi salonu yaratmakta kullanılan yerel sunucuya verilmiştir. Salonu yaratan sunucu üzerinden yapılan bir istek ile etkinlik sistemden tamamen silinebilir. Şekil 6.10'da

gösterilen etkinlik silme işleminde, silinmek istenen salon adı (paylaşılan nesne adı) ve silme seçeneği seçilip "OK" düğmesine basılarak daha önce yaratılmış olan "ITUSalon_2" isimli nesne koşulsuz bir şekilde silinir.



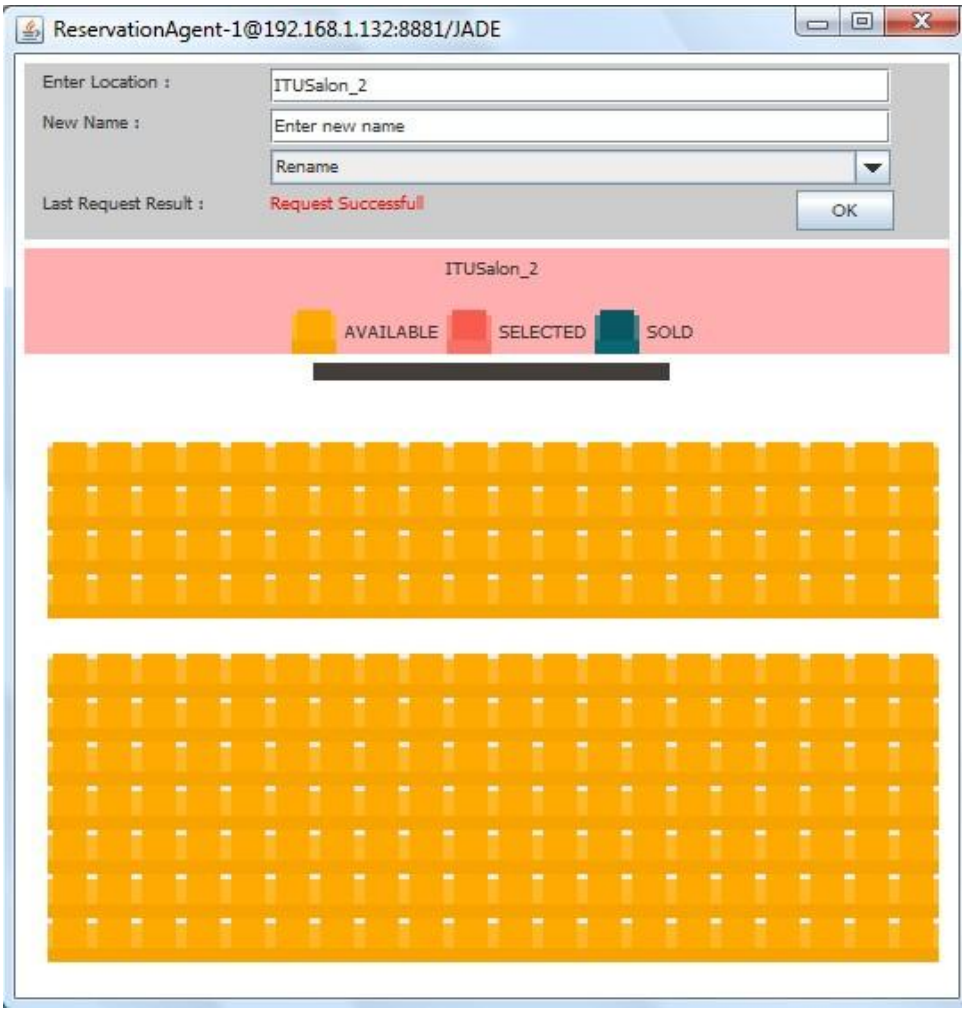

**Şekil 6.10 :** Rezervasyon silme işlemi.

Paylaşılan nesnenin silme işleminde gerekli kontrollerin yapılması uygulamanın sorumluluğundadır. Örneğin, daha önce koltuk seçimi yapılmış bir rezervasyon için silme isteğine uygulama tarafından onay verilmeyebilir ve kullanıcıya gerekli uyarı gösterilerek önlemler alması sağlanabilir.

### 6.3.5 Etkinlik yeniden adlandırma

Daha önce benzersiz bir isimle yaratılmış bir etkinliğin isminin benzersiz olan yeni bir isimle değiştirme isteğidir. Nesne isminin değiştirilmesi sırasında nesne içeriği değiştirilmez. Sadece nesneye erişmek için kullanılan benzersiz isim değiştirilir.

Şekil 6.11'de ismi "ITUSalon_2" olan bir salonun (paylaşılan nesnenin) isminin yeni bir isimle değiştirilmesine olanak tanıyan arayüz ekranı gösterilmiştir. Kullanıcı isim değiştirme seçeneğini seçtikten sonra, ismini değiştirmek istediği salonun (nesnenin) kayıtlı ismini, benzersiz yeni salon (nesne) ismini girip "OK" düğmesine basarak salonu (nesneyi) yeniden adlandırır.




**Şekil 6.11 :** Rezervasyon yeniden adlandırma işlemi.

Seçilen yeni isim sistemde daha önce kayıtlı ise başarısız yeniden adlandırma işlemi mesajı gösterilir. Yeniden adlandırılan bir nesnenin eski ismi sistem çapında yeni bir nesne yaratmak için kullanılabilir.

Yeniden adlandırma sonucunda nesnenin içeriği değişmediğinden gerekli kontrolleri yapıp uygun bir şekilde nesne içerisinde gerekli isim alanlarını değiştirmek uygulamanın sorumluluğundadır.

### 6.3.6 Etkinlik rezervasyon durumunu gevşek okuma

Nesne okumak için yapılan ve güncelliği garanti edilmeyen okuma işlemidir. Program içinde "readLoose" metoduna verilen nesne ismi ileağ üzerinde karşılaşılan ilk nesne kopyası istemci etmene ulaştırılır. Rezervasyon uygulamasında gevşek okuma gerçekleştiğinde kullanıcı güncel olmayan salon durumu görüntüleyebileceğinden satılmış olan koltuklar uygun durumda gözükebilir ve bu koltuklar için yapılan satın alma isteği başarısız olarak sonuçlanır.

# 7. SONUÇ VE ÖNERİLER

Bu tez çalışması, çoklu etmen ortamında nesne tabanlı dağıtık nesne paylaşımının ayrıntılarını içermektedir. Dağıtık nesne paylaşım ortamında çıkabilecek sorunlar ve çözümleri uygun gerçekleme yöntemleri ile sağlanmış ve yazılım diyagramları ile ifade edilmiştir.

Önerilen sistem, farklı alanlarda bulunan çok sayıda kullanıcı arasında nesne paylaşımına ve ortak iş yürütmeye imkan sağlayan uygulamalar için çoklu etmen ortamında nesne tabanlı bir paylaşım ortamı sunmuştur.

Kullanılan bir çok dağıtık nesne paylaşım sistemi, merkezi sunucu kontrolünde ve nesne yerleşkesi yine merkezi bir sunucu olacak şekilde nesne paylaşımı sağlamaktadır. Bu sistemler genellikle uzak nesne modelini benimsemişlerdir. Uzak nesne modelinin ölçekleme konusunda bazı dezavantajları vardır. Uzak nesne modeli nesne kullanım ve sistem içinde gerçeklenmesini basitleştirse de merkezi yapısından dolayı dağıtık sistemin getirdiği avantajlardan faydalanılmasını engeller. Dağıtık nesne paylaşım mimarisinde başarım, koruma, kullanılabilirlik ve yük dengeleme amacıyla nesnelerin farklı alanlara yayılması gereksinimi ortaya çıkabilir. Farklı alanlara nesnelerin yayılması beraberinde yerelliği de getirir. Nesnelerin yerelleştirilmesi ve farklı alanlara kopyalanması etkin erişim ve kullanılabilirlik açısından büyük önem taşır. Uzak nesne modellerinde bu avantajlardan yararlanılamaz.

Sistem gerçeklenirken JADE ortamının özelliklerinden yararlanılmış ve FIPA standardlarına uygun bir çoklu etmen ortamı meydana getirilmiştir. Etmenler sahip oldukları görevlere göre kendi kendilerine karar verecek şekilde akıllı bir yapıda tasarlanmıştır. Nesne erişim olayları gerekli etmenlere bildirilerek etmenlerin çevresindeki olayları algılaması ve tepki verebilmesi sağlanmıştır. JADE sisteminin sağlamış olduğu etmen yaratılması, etmen sonlanması gibi olay bildirimlerini algılayacak yazılım geliştirilmiştir.

Dağıtılmış nesne paylaşım sistemi kullanıcıdan saydam olarak oluşturulmuştur. Dağıtık nesne paylaşım sistemi üzerinde çeşitli kontrolleri yapan ve sistem çapında bilgileri saklayan bir sistem koordinatörü ve nesne sahipliği görevini üstlenen kullanıcı makineye yakın yerel sunucular ile dağıtık nesne paylaşımı gerçeklenmiştir.

Yerel sunucu her kullanıcı etmen için ayrı yaratılabileceği gibi, yerel bir ağ üzerinde tek bir sunucu yaratılarak bu ağ üzerindeki tüm kullanıcılar erişimlerini bu sunucu vekilliğinde gerçekleştirebilir. Yerel sunucu farklı görevlere sahip alt birimlerden oluşur ve bu durum yük dengeleme açısından önemlidir. Yazma istekleriyle nesne sahipliği istemci etmenin yerel sunucusuna aktarıldığından sık ve ardışık yazma isteklerinde performanslı bir erişim sağlanır ve dağıtık ortamının en önemli avantajından yararlanılmış olur.

Sistem koordinatörü de yük dengelemek için alt birimlere ayrılmıştır. Koordinatör içinde yerleşke yöneticisi, başlangıç yöneticisi, nesne yöneticisi, isimlendirme yöneticisi, sistem yöneticisi birimleri vardır. Yerleşke yöneticisi sistemde kayıtlı nesnelerin sahiplik bilgisini saklar. Erişim yapmak isteyen etmenin yerel sunucusu koordinatör üzerindeki yerleşke sunucusuna sahiplik sorgusu yapar. Yerleşke yöneticisi, nesnelerin tekil nesne tanımlayıcı bilgisinden faydalanarak oluşturduğu hash tablosunu kullanarak çok hızlı bir şekilde yerleşke sorgu sonucunu verir. Başlangıç yöneticisi ilklendirme işlemlemlerinden sorumludur. İsimlendirme yöneticisi sistem çapında tekil tanımlayıcıya sahip nesneler yaratılmasını sağlar. Koordinatör altında yer alan sistem yöneticisi bir ağ geçidi görevini üstlenir. Koordinatör nesne yöneticisi normal şartlar altında nesne sahipliği üstlenmeyerek sistemi merkezi yaklaşımdan uzak tutar. Koordinatör nesne yöneticisi nesneler ile ilgili oluşabilecek tutarsızlıkları denetler ve çözüm oluşturur. Nesne sahibi bir yerel sunucunun etkinliğini kaybetmesi durumunda koordinatör nesne sunucusu bu durumu algılar ve nesnenin sahipliğini üstlenir. Nesne sahipliği gelen yazma isteği ile yine bir yerel sunucuya aktarılır.

Kullanıcıya sadece kendisini ilgilendiren metotlara erişim izni verilmiş ve böylelikle kullanıcının hatalı davranışlarda bulunmaması ve sistem içinde kendini ilgilendirmeyen olaylara dahil olmaması sağlanmıştır.

Sistem içinde kullanıcıya nesne yaratma, okuma, yazma, gevşek okuma, silme ve yeniden adlandırma yetkileri verilmiştir. Kullanıcıya bu işlemler esnek bir yapıda

sunulmuştur. Yapılan herhangi bir erişim işleminde kullanıcının herhangi bir yazılım noktasında takılı kalmasını engellemek amacıyla zaman aşımı süresi tanımlanmışıtır. Kullanıcıya yapacağı erişimlerde, erişim metotlarına zaman aşımı süresini ihtiyacına göre girebilme imkanı sunularak uygulamaya özel bir alt yapı sağlanmıştır. Zaman aşımı süresi sonunda başarısız erişim mesajı bilgisi ile kullanıcının durumdan haberdar olması ve istediği tepkiyi verebilmesi sağlanmıştır. Burada kullanıcı, zaman aşımına uğramış erişim istekleri için tekrar istek yapabilir ya da sistem içinde uygulama bazlı önlemler alabilir. Erişim isteği için kullanılacak nesnelerin sistem tarafından otomatik yaratılması ile kullanıcının erişim isteklerinde hata yapması engellenmiştir.

Çok okuyuculu tek yazıcılı mimari kullanılarak eş zamanlı erişim kontrolü yapılmıştır. Nesne sahipliğini elinde bulunduran sunucu, sahip olduğu nesne erişim kuyrukları ile erişimleri denetler. Sistemde okuma isteği olduğu durumda nesne kilitlenir ve sadece kuyruktaki okuma isteklerine cevap verilir. Tüm okuma istekleri serbest bırakıldığında kilit kaldırılır. Yazma isteği ile karşılaşıldığında izin verilmiş okuma isteklerinin serbest bırakma mesajı ulaşır ulaşmaz nesne kilidi kaldırılır. Yazma isteğine izin verilerek ilgili yazma isteği serbest bırakılana kadar nesne okuma ve yazma isteklerine karşı kilitlenir.

Nesne erişimlerinde nesne güncelliği konusunda çok sıkı şartları olmayan uygulamalar için gevşek okuma adı altında bir okuma isteği tanımlanmıştır. Bu istek ile nesne güncelliği garanti edilmeden ağda karşılaşılan ilk nesne kopyası kullanıcı etmene ulaştırılır.

Nesne kullanıcısı etmen erişimlerini sistemde tanımlı bir metot çağrısı ile elde ettiği nesne erişimcisi üzerinden gerçekleştirir. Nesne erişimcisi, yapılan her erişim isteği için yeniden edinilir. Erişim nesnesi içinde yer alan ve kullanıcıdan saydam olarak oluşturulan erişim tanımlayıcı bilgisi her erişim için farklı olacağından, erişim isteği üreten etmen aynı anda farklı ipliklerinden (thread) çağrı gerçekleştirebilir. Dağıtık nesne paylaşım sistemi, erişim nesnesinde yer alan tanımlayıcı bilgi sayesinde işleme alınan erişim isteklerini birbirinden ayırır.

Gerçekleştirilen nesne erişimleri sonunda, yerel nesne sunucusunun nesne tablosunda bulunan nesneler belirli bir zaman aşımı (belirli bir süre erişim yapılmaması durumunda) sonunda silinerek tablolarda yapılacak arama gecikmesi ve bellek yükü

azaltılır. Sunucu nesnenin sahibi ya da yaratıcısı ise nesneler silinmez ve zaman aşımı bakımı uygulanmaz.

Sistem içinde gerçekleşen olayları gözlemleyebilmek amacıyla grafik arayüz hazırlanmıştır. Bu arayüz sayesinde yerel sunucu ve sistem koordinatörü birimlerinde yer alan erişim tabloları izlenebilir ve sistemde meydana gelen olaylar kolaylıkla analiz edilebilir.

Sistem, dağıtık mimaride nesne paylaşımı ve erişim senkronizasyonu gerektiren tüm alanlarda kullanılabilir. Bilet satış, rezervasyon uygulamaları, bankacılık sistemleri, paralel işlem gerektiren uygulamalar örnek olarak gösterilebilir.

Geliştirilen çoklu etmen ortamında nesne paylaşım sistemi, gerçek zamanlı çalışmaya uygundur. Geliştirilen sistemde oluşturulan tablolar, veri tabanına kayıt edilerek bilgilerin kalıcı olarak saklanması ve sistem açılışında tekrar yüklenebilmesi sağlanabilir.

Geliştirilen sistem kullanıcı açısından tek bir bilgisayar üzerinde çalışıyormuş görüntüsü altında ve erişimler için basit bir arayüz sunan yapıya sahiptir. Nesnelere yerel sahiplik atanılarak dağıtık yapının en önemli özelliklerinden biri uygulanmış ve erişimler esnek ve hızlı duruma getirilmiştir. Java programlama dilinin platform bağımsızlığı özelliğinden dolayı kullanım alanı geniş bir ortam sunulmuştur. Java dilinde kod yazabilen bir kullanıcının sistemde tanımlı metotlar aracılığıyla kolaylıkla dağıtık nesne paylaşımı yapabileceği bir ortam hazırlanmıştır.

## KAYNAKLAR


[1] **Orfali, R., Harkey, D.** (1998). Client/Server Programming with Java and CORBA. ISBN-10: 047124578X, ISBN-13: 978-0471245780.

[2] **Thai, T. L.** (1999). Learning DCOM. O'Reilly Media, 1st edition. ISBN-10: 1565925815, ISBN-13: 978-1565925816.

[3] **Bellifemine, F., Poggi, A. ve Rimassa, G.** (2000). Developing Multi-Agent Systems With JADE. Seventh International Workshop on Agent Theories, Boston, MA.

[4] **Khetan, G.** (2002). Comparison of Memory Management Systems of BSD. Windows and Linux. Department of Computer Science, University of Southern California, 16 Aralık 2002.

[5] **Fu, W., Liu, L. ve Chen, T.** (2013). Direct Distributed Memory Access for CMPs. Elsevier, J. Parallel Distrib. Comput. 74 (2014) 2109–2122. 14 Kasım 2013.

[6] **Liu, X., Jiang, H., ve Kiat, L.** (2004). A Distributed Shared Object Model Based on a Hierarchical Consistency Protocol for Heterogeneous Clusters. In IEEE International Symposium on Cluster Computing and the Grid 2004.

[7] **Weijian, F.** (2004). Distributed Object Sharing for Cluster-based Java Virtual Machine (doktora tezi). University of Hong Kong, 2004.

[8] **Judge, A., Nixon, P. A., Cahill, V. J. ve diğ.** (1998). Overview of Distributed Shared Memory. Distributed Systems Group, Dept. of Computer Science, Trinity College, Dublin, Ireland 29 Ekim 1998.

[9] **Tanenbaum, A. S.** (2007). Modern Operating Systems 3rd Edition. Pearson Prentice Hall.

[10] **Tanenbaum, A. S., Steen, M. V.** (2007). Distributed Systems Principles and Paradigms 2nd Edition. Pearson Prentice Hall.

[11] **Tanenbaum, A. S.** (1994). Distributed Operating Systems. Pearson Prentice Hall. 4 Eylül 1994.

[12] **Hill, M. D.** (1998). Multiprocessors Should Support Simple Memory-Consistency Models. IEEE Ağustos 1998.

[13] **Steinke, R. C., Nutt, G. J.** (2004). A Unified Theory of Shared Memory Consistency. Journal of the ACM, Volume 51 Issue 5, doi:10.1145/1017460.1017464.

[14] **Wong, A. K. L., Zhu, W.** (2002). A Multi-locking Mechanism on Shared Object DSM. Proceedings of the Ninth International Conference on Parallel and Distributed Systems (ICPADS'02). IEEE 2002.

[15] **Hagimont, D., Boyer, F.** (2001). A Configurable RMI Mechanism for Sharing Distributed Java Objects. Sirac Laboratory (INPG-INRIA-UJF). IEEE 2001.

[16] **Tudor, D., Macariu, G., Cretu, V., Schreiner, W.** (2010). Shared Data Grid Programming Improvements Using Specialized Objects. 2010 International Conference on Complex, Intelligent and Software Intensive Systems. IEEE 2010.

[17] **Geist, A., Beguelin, A., Dongarra, J. ve diğ.** (1995). PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing (Scientific and Engineering Computation).

[18] **Gropp, W., Lusk, E., Doss, N., Skjellum, A.** (1996). A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Elsevier Science Parallel Computing, 22 (1996) 789-828.

[19] **Rosenberry, W., Kenney, D., Fisher, G.** (1992). Understanding DCE. O'Reilly Media, 1. Sürüm, 8 Ekim 1992. ISBN-10: 1565920058, ISBN-13: 978-1565920057.

[20] **Fatoohi, R. ve Jensen, D.** (2003). Migration of DCE applications into CORBA and SOAP environments. John Wiley & Sons Ltd. 24 Ağustos 2002.

[21] **Mitchell, J. G., Gibbons, J. J., Hamilton, G. ve diğ.** (1994). An Overview of the Spring System. IEEE 1994.

[22] **Thompson, D., Exton, C., Garret, L., Sajeev, A. S. M., Watkins, D.** (1997). Distributed Component Object Model (DCOM). Department of Software Development Faculty of Computing and Information Technology Monash University Melbourne Australia. 5 Şubat 1997.

[23] **Object Management Group** (2012). Common Object Request Broker Architecture (CORBA) Specification, Version 3.3. Kasım 2012.

[24] **Makpangou, M., Gourhant, Y., Pierre, J. ve diğ.** (1994). Fragmented Objects for Distributed Abstractions. Readings in Distributed Computing Systems, pp. 170-186, IEEE Computer Society Press.

[25] **Yilmaz, G., Erdogan, N.** (2005). DCOBE: Distributed Composite Object Based Environment. The Computer Journal, 48 (3):273-291, Oxford Press, SCI, Mayıs 2005.

[26] **Nwana, H. S.** (1996). Software Agents: An Overview. Neural. Knowledge Engineering Review, Vol. 11, No 3, pp. 205-244, Ekim/Kasım 1996.

[27] **Wooldridge, M.** (2002). An Introduction to MultiAgent Systems. John Wiley & Sons (Chichester, England), Şubat 2002.

[28] **Bellifemine, F. L., Caire, G., Greenwood, D.** (2007). Developing Multi-Agent Systems with JADE. WILEY ISBN: 978-0-470-05747 - 6 Şubat 2007.

[29] **Caire, G.** (2009). JADE Tutorial, JADE Programming For Beginners.

[30] **Bellifemine, F., Caire, G., Trucco, T. ve diğ.** (2010). JADE Administrator's Guide.

[31] **Bellifemine, F., Caire, G., Trucco, T. ve diğ.** (2010). JADE Programmer's Guide.

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

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

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

## ÖZGEÇMİŞ


**Ad Soyad** **: Metehan PATACI**

**Doğum Yeri ve Tarihi** **: Şişli 09.05.1986**

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

**ÖĞRENİM DURUMU:**

- **Lisans** **:** 2009, Gebze Yüksek Teknoloji Enstitüsü, Bilgisayar Mühendisliği Fakültesi, Bilgisayar Mühendisliği Programı
- **Yükseklisans** **:** 2014, İstanbul Teknik Üniversitesi, Bilgisayar Mühendisliği Anabilim Dalı , Bilgisayar Mühendisliği Programı

**MESLEKİ DENEYİM VE ÖDÜLLER:**

**TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:**

- Patacı, M., Erdoğan, N., 2014. Object Based Distributed Data Sharing In Multi-Agent Environment, 5th *International Conference on Software Engineering and Service Science (ICSESS 2014).*

**DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:**