İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITILMIŞ BİLEŞİK NESNE TABANLI ORTAM ÜZERİNDE LOJİSTİK UYGULAMASI

YÜKSEK LİSANS TEZİ

Jamshid NAKHCHIVANSKI
(504071545)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı : Prof. Dr. Nadia ERDOĞAN

12 EKİM 2011

İTÜ, Fen Bilimleri Enstitüsü'nün 504071545 numaralı Yüksek Lisans Öğrencisi **Jamshid NAKHCHIVANSKI**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"DAĞITILMIŞ BİLEŞİK NESNE TABANLI ORTAM ÜZERİNDE LOJİSTİK UYGULAMASI"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Nadia ERDOĞAN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Nadia ERDOĞAN** İstanbul Teknik Üniversitesi	.....
	Doç. Dr. Şima UYAR ETANER İstanbul Teknik Üniversitesi	.....
	Yrd. Doç. Dr. Güray YILMAZ Hava Harp Okulu	.....

Teslim Tarihi : **15 Eylül 2011**
Savunma Tarihi : **12 Ekim 2011**

ÖNSÖZ

Ben hayatım boyunca beni son derece anlayan ve eğitimimi destekleyen babam Yusif ve annem İmaret'e minnettarım. Her zaman ben ve benim geleceğimle ilgili kaygılandıkları için kız kardeşim Ayten ve kardeşim Kamran'a da teşekkür borçluyum.

Ben danışmanım Prof. Dr. Nadia ERDOĞAN'a ve Yrd. Doç. Dr. Güray YILMAZ'a bana Yüksek Lisans eğitim dönemi sırasında yaptığı yardım, destek ve paha biçilmez rehberlik için teşekkür etmek istiyorum. Onların desteği olmadan bu çalışmanın hazırlanması mümkün olmazdı.

Eylül, 2011 Jamshid Nakhchivanski

İÇİNDEKİLER

ÖNSÖZ ..... **v**
İÇİNDEKİLER ..... **vii**
ÇİZELGE LİSTESİ ..... **ix**
ŞEKİL LİSTESİ ..... **xi**
KISALTMALAR ..... **xiii**
SEMBOL LİSTESİ ..... **xv**
ÖZET ..... **xvii**
SUMMARY ..... **xix**
1. GİRİŞ **1**
1.1. Tezin Amacı 2
1.2. Literatür Araştırması 2
1.2.1. Geleneksel IPC modellerinin değerlendirmesi 3
1.2.2. Vekil tabanli nesne modellerinin değerlendirmesi 4
1.2.3. Bölünmüş nesne modellerinin değerlendirmesi 5
1.2.4. Dağıtılmış bileşik nesne modeli ile bölünmüş nesne modelinin karşılaştırılması 7
1.3. Tez Planı 8
2. LOJİSTİK ORTAM **11**
2.1. Lojistik Ortamın Genel Görünümü 11
2.2. Lojistik Ortamının Çalışma Şekli 14
3. OPTİMAL GÖNDERİM PLANININ HESAPLANMASI **19**
3.1. GÇDTYP Probleminin Matematiksel İfadesinin Oluşturulması 20
3.2. GÇDTYP için Algoritmanın Oluşturulması 22
3.3. Değiştirilmiş Clack – Wright Savings Algoritması 25
4. DBN MODELI VE DBNTO ORTAMI **27**
4.1. DBN Modelinin Haberleşme Şekli ve Nesne Yapısı 27
4.2. DBN Modelinde Nesnelerin Alanlar Üzerinde Paylaşımı 28
4.3. DBN Modelinde Nesne Kontrolü ve Saydamlık 29
4.4. Bir Bileşik Nesnenin İç Yapısı 30
4.4.1. Bağlanti nesnesi 31
4.4.2. Kontrol nesnesi 32
4.5. DBN Modelinin Avantajları 32
4.6. Dağıtılmış Bileşik Nesne Tabanlı Ortam (DBNTO) 32
4.6.1. DBNTO sisteminin temel fonksiyonları 34
5. LOJİSTİK UYGULAMA ORTAMININ TASARIMI VE GERÇEKLENMESİ **37**
5.1. Lojistik Uygulama Ortamında DBNTO Sisteminin Kullanım Avantajları 37
5.2. Lojistik Uygulama Ortamının Genel Görünümü 39
5.3. Uygulamanın Gerçekleme Ayrıntıları 40
5.4. Kullanıcı Uygulamalarda Bileşik Nesne Yapısının Kullanımı 41

5.4.1. Alt nesnelerin gerçeklenmesi 42
5.4.2. Bileşik nesnelerin gerçeklenmesi 44
5.5. Kullanıcı Uygulamalara Sunulan Arayüzler 49
5.5.1. Satıcı düğümlerdeki kullanıcı uygulamalara sunulan arayüzler 49
5.5.2. Aracı düğümlerdeki kullanıcı uygulamalara sunulan arayüzler 51
5.5.3. Alıcı düğümlerdeki kullanıcı uygulamalara sunulan arayüzler 52
5.5.4. Kullanıcı uygulamalar arasındaki etkileşim 54
5.6. Lojistik Uygulamanın Çalışmaya Hazır Duruma Getirilmesi 56
6. SONUÇLAR VE DEĞERLENDİRMELER 59
6.1. GÇDTYP Algoritmasının Değerlendirilmesi 59
6.2. Lojistik Uygulamanın Değerlendirilmesi 61
7. SONUÇLAR 65
KAYNAKLAR 67
ÖZGEÇMİŞ 71

ÇİZELGE LİSTESİ

Sayfa

Çizelge 5.1: SatıcıDüğüm bileşik sınıfında yer alan metotların işlevleri 45
Çizelge 5.2: AracıDüğüm bileşik sınıfında yer alan metotların işlevleri 46
Çizelge 5.3: AlıcıDüğüm bileşik sınıfında yer alan metotların işlevleri 48
Çizelge 5.4: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotların işlevleri.....50
Çizelge 5.5: Aracı düğümlerdeki kullanıcı uygulamalarda yer alan metotların işlevleri.....52
Çizelge 5.6: Alıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotların işlevleri.....53
Çizelge 6.1: Test Çizelgesi 59
Çizelge 6.2: Giriş için Rastgele Değerler 60
Çizelge 6.3: Taşıt rotaları ve taşınan malzeme miktarları 61
Çizelge 6.4: DBNTO üzerinde yapılan testlerden elde edilen sonuçlar 63

ŞEKİL LİSTESİ

Sayfa

Şekil 2.1: Lojistik Ortamın genel görünümü 12
Şekil 2.3: Lojistik uygulama ortamının çalışma şekli 14
Şekil 3.1: GÇDTYP algoritmasının sözde-kodu 25
Şekil 4.1: Tekil bir adres alanında üç alt nesnesi ile birlikte yaratılmış bileşik nesne 28
Şekil 4.2: Dört farklı adres alanı üzerine yayılmış bir dağıtılmış bileşik nesne 29
Şekil 4.3: Alt nesne erişim düzeni 31
Şekil 4.4: DBNTO sisteminin genel yapısı 33
Şekil 5.2: Malzeme alt sınıfının sınıf kodu 42
Şekil 5.3: Taşıt alt sınıfının sınıf kodu 42
Şekil 5.4: Bilgi alt sınıfının sınıf kodu 43
Şekil 5.5: Mesaj alt sınıfının sınıf kodu 43
Şekil 5.6: SatıcıDüğüm bileşik sınıfının sınıf kodu 44
Şekil 5.7: AracıDüğüm bileşik sınıfının sınıf kodu 46
Şekil 5.8: AlıcıDüğüm bileşik sınıfının sınıf kodu 48
Şekil 5.9: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotlar 49
Şekil 5.10: Satıcı düğümlerdeki kullanıcı uygulama arayüzü 51
Şekil 5.11: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotlar 51
Şekil 5.12: Aracı düğümlerdeki kullanıcı uygulama arayüzü 52
Şekil 5.13: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotlar 53
Şekil 5.14: Alıcı düğümlerdeki kullanıcı uygulama arayüzü 54
Şekil 5.15: Alıcı, Aracı ve Satıcı düğümlerdeki kullanıcı uygulamalar arasındaki etkileşim 55
Şekil 5.16: Alıcı, Aracı ve Satıcı kullanıcı uygulamalar arasındaki dağıtılmış bileşik nesneler 56
Şekil 6.1: Simülasyon Uygulaması 60

KISALTMALAR

DBN : Dağıtılmış Bileşik Nesne
DBNTO : Dağıtılmış Bileşik Nesne Tabanlı Ortam
DSK : DBNTO Sistem Koordinatörü
DBNS : Dağıtılmış Bileşik Nesne Sunucusu
TYP : Taşıt Yönlendirme Problemi
TDTYP : Tek Depolu Taşıt Yönlendirme Problemi
ÇDTYP : Çok Depolu Taşıt Yönlendirme Problemi
PTÇDTYP : Parçalanmış Teslimli Çok Depolu Taşıt Yönlendirme Problemi
GÇDTYP : Geliştirilmiş Çok Depolu Taşıt Yönlendirme Problemi
GUI : Grafik Kullanıcı Arayüzü
RMI : Uzak Metot Çağırısı

SEMBOL LİSTESİ

m_{ij}	: satıcı ve j alıcı düğümü arasındakı en düşük maliyetli yolun maliyeti
dem_{ab}	: *a* alıcısının *b* malzeme çeşidine olan isteği
p_a	: *a* alıcısının istek sayısı (kaç tane farklı malzeme çeşidine talep var)
dis_{ij}	: düğümler arasındaki mesafe
v_k	: *k* deposunun dem_{ab} isteğine uygun malzeme taşıya bilecek taşıt sayısı
r_{kc}	: *k* deposunun *c* taşıtının rota sayısı
x_{ijkcl}	: eşittir *1*, eğer *k* deposunun *c* taşıtı *l.* rotasında *i* alıcı düğümünden sonra *j* alıcı düğümüne yolculuk yapıyorsa, aksi taktirde eşittir *0.*
vc_{kc}	: *k* deposunun *c* taşıtının birim uzunluğa göre taşıma maliyeti
amt_{kcl}	: *k* deposunun *c* taşıtının *l.* rotasında dem_{ab} isteğinin taşınan miktarı
mc_k	: *k* deposundaki dem_{ab} çeşitli malzemenin satış fiyatı
P_{abkl}	: *k* taşıtının *l.* rotasında *a* alıcısına teslim edilmiş *b* malzeme çeşidinin miktarı
u	: *a* alıcısına hizmet eden taşıtların sayısı.
r_k	: k taşıtının rota sayısı

DAĞITILMIŞ BİLEŞİK NESNE TABANLI ORTAM ÜZERİNDE LOJİSTİK UYGULAMASI

ÖZET

Günümüzde dinamik veri alış-verişi sağlayan paylaşımlı sistemlere ihtiyaç artmaktadır. Veri paylaşım noktalarının çokluğu, bu noktalar arasında iletilen veri yoğunluğu nedeniyle hızlı çalışan, gerektiğinde saydam veri gönderimine ve güncellenmesine ihtiyaç duyan paylaşımlı sistemlerin geliştirilmesine ilgiyi arttırmıştır.

Bu çalışmada dinamik ve paylaşımlı sistemlere örnek olarak güncel bir lojistik uygulaması incelenmiştir. Paylaşımlı sistemleri çekici kılan en önemli tasarım konularından biri, dağıtım ile ilgili tüm detayların kullanıcılara saydam olarak gerçekleniyor olmasıdır. Ancak, hali hazırdaki geniş alan dağıtılmış sistemlerinin bu konuyu tam olarak çözebildiklerini söylemek mümkün değildir. Bundan dolayı, bu lojistik uygulama Dağıtılmış Bileşik Nesne olarak adlandırılan yeni bir nesne modeli üzerinde, aynı zamanda, Java programlama dilini kullanarak dağıtılmış uygulamalar geliştiren yazılımcılar için tekdüze bir arayüz sağlayan, Dağıtılmış Bileşik Nesne Tabanlı Ortam ara katman üzerinde tasarlanmış ve gerçeklenmiştir. Bu ara katman dağıtık nesnelerin coğrafi olarak dağınık bulunan düğümler arasında paylaşımına olanak sağlamaktadır.

Çalışmada önce, bu ortam üzerinde dağıtılmış paylaşımlı sistemlere örnek olarak güncel bir lojistik uygulamasının genel yapısı anlatılmış, lojistik ortamı oluşturan alıcı, yönetici/aracı ve satıcı düğümlerin tanımları yapılmış, daha sonra onlar arasında etkileşim adımları anlatılmıştır. Ayrıca, lojistik sistemin yapısına uygun optimal gönderim planını hesaplamak için yeni Taşıt Yönlendirme Problemi (TYP) oluşturulmuş, problemi oluşturan matematiksel yöntemler ve lojistik maliyeti en aza indirgemek için gerekli olan TYP algoritmasının çalışma adımları açıklanmıştır. Daha sonra, lojistik ortamın Dağıtılmış Bileşik Nesne (DBN) modeline uygun tasarım ve gerçekleme aşamaları anlatılmış, DBNTO sisteminin bileşik yapısı ve kopyalamaya dayalı nesne tutarlılık yönetimi, yürütme zamanını azaltmak üzere istekçiye bileşik nesnenin yalnızca kullanılacak olan ilgili kısmının gönderilmesi sonucu çalışan uygulamanın başarımını yükseltmeğe yönelik konular incelenmiştir. Ayrıca, lojistik uygulamanın tasarım ve gerçekleme konuları anlatılmış, farklı düğümlerde çalışan kullanıcı uygulamaların tipleri ve özellikleri açıklanmıştır. Sistemde maliyeti optimize etmek için kullanılacak olan algoritmanın simulasyon ortamı hazırlanmış ve test değerlendirmeleri yapılmıştır.

DBN modeli uygulamanın temel çalışma ortamını oluşturmaktadır. Bu model, düğümler arasında arakatman vazifesini üstlenerek, dağıtılmış uygulamalar için gerekli olan haberleşme, verilerin kopyalanması, parçalanma, tutarlılık, uygulamaların dinamik olarak yüklenmesi v.b. gibi temel mekanizmaları sağlar.

LOGISTICS APPLICATION ON DISTRIBUTED COMPOSITE OBJECT BASED ENVIRONMENT

SUMMARY

Presently, demand for shared systems, providing dynamic data exchange, is constantly increasing. With large numbers of data sharing nodes, the intensity of communication traffic between the nodes increases interest in shared systems, which are fast and transparently transfer or update data when required.

In recent years, the data processing area of applications has moved from individual environments to the network and the internet environment. Thus, a new issue emerges; development of information-management applications distributed over a wide area. Consequently, the design and development of applications that are adaptable to distributed environments have become important. Distributed systems provide sharing of information on computer networks and system resources.

This paper presents the design of a logistic environment which is distributed over a wide area and the development of a logistic application that runs on this environment. The system is implemented on a platform which allows for distributed objects to be shared among nodes located at dispersed geographical locations.

A simple logistic environment consists of several *customer* and *provider* companies. Customers are served various types of goods by set of vehicles in different providers. In such an environment, both customer and provider sides not only undertake the required operations for buying and selling, but also partly undertake management operations. This aspect increases complexity of information processing and decreases scalability of the system when a customer or provider company works with many other companies. The environment that has been developed to support logistic applications addresses these problems and presents better solutions for these types of problems. In this environment, management functions between customer and provider companies are given to the *mediator* companies.

Customer, provider and mediator companies altogether constitute a logistic environment. Provider companies possess a variety of goods for sale and vehicles for transporting them. Customer companies buy goods from the provider companies and fulfill various goods requests. Mediator companies manage and supervise all buying and selling operations among provider and customer companies. Paths exist among the companies and goods are carried to companies via these paths. Typically, the costs of different paths vary. A vehicle that transports goods starts from a provider company visits some customer companies over its route, fulfilling customers' requests and finally returns to the owner provider company. Provider companies are retail companies. They have a variety of goods for sale and vehicles for transporting them. Every provider company is in agreement with at least one mediator company, and cooperates with customers over mediator companies. The amount of goods and the number of vehicles change dynamically in a course of time as new goods or vehicles maybe added or removed. Requests for goods by customer companies reach

provider companies only after being analyzed and finalized by mediator companies. Each provider company evaluates incoming requests and accepts only those that are suitable. Customer companies determine their requests, sent them to the mediator companies with agreement, and in thus way get answer over mediator companies. Every customer company must be an agreement with at least one mediator company, for sending its request to the provider companies by the mediator companies. In conditions when one customer company has agreement with several mediator companies, it can send its goods request to every mediator and can accept appropriate optimal sending plan offered by mediators. Mediator companies have bilateral agreement with different provider and customer companies, and undertake manager/mediator role between them. They also have some functions; to get goods requests from customer companies; to prepare an optimal transfer plan according to vehicle and material capacity of provider companies, path distance and transportation cost between customer and provider locations and then offers it to the customer side. Mediator companies know geographical locations of provider and customer companies. This information is used to find optimal transfer plan.

The structure of the logistic environment shows that, it is a dynamic and distributed environment. Many different types of companies (customer, mediator and provider) communicate with one another here. It is possible registration new companies on the environment and removal already registered companies from the environment. Emergence of new customer requests is possible always. Mediator companies must analyze new material requests come from the customer companies and calculate optimal transfer plan. In addition, all selling and buying operations must be controlled by them. That is why; mediator companies must always know the current condition of all companies having the agreement with it.

All of these require an implementation model of this type of environment, which support scalability, dynamic management of the shared system, fast and transparently transfer or update of information. Distributed Composite Object model gives us that support.

Distributed applications, implemented in collaboration, generally work on large and medium-grained objects. However, users of these types of applications are usually interested in the smaller parts of objects. Therefore, if a large object can be divided into smaller sub-objects and only useful part of the object can be sent to the requested area, then the amount of data to be accessed and transferred among various areas will be reduced, and productivity of applications will grow. The new DCO model, based on a replication, has been developed from this point of view. The main idea of this model is the physical distribution of a composite object, consisting of the union of many sub-objects, in various areas.

Furthermore, an actual logistics application includes several features such as dynamic control on a distributed environment, should be robust and scalable, and is expected to provide the optimum solutions to generate rapid responses to queries and to minimize costs. DCOBE system presents some important properties, which allow the development of applications that possess these features. DCOBE system supports composite object management. This feature allows us to model vehicles and materials which take place in logistics application, and some of their attributes as an object. In calculations associated with the requesting nodes, only the necessary parts of the composite objects are transmitted through the internet environment. This approach is advantageous as it reduces the amount of data to be transmitted, thus decreasing the network traffic load. DCOBE system supports two consistency

protocols, known as “write-update” and “write-invalidate”. If "write-update" protocol is used in the system, and any object modifications have occurred in the servers, then all copies of this object in the system are synchronously updated to reflect this modification. If the "write-invalidate" protocol is used and any object modifications have occurred in the servers, then all servers using a copy of this object are informed about these modifications. A server updates its local copy with the current copy of the object from the appropriate addresses only when necessary. These capabilities enable us to quickly achieve the state of the latest data from the nodes (servers), when the optimal values must be calculated in the application. Hence, DCOBE can provide significant help in increasing the speed and minimal use of system resources. Using object based approach in development of the logistic application is another advantage for us.

The paper first defines the roles of customer, manager/mediator and provider nodes, which constitutes a logistic environment, and next describes the interactions among them. Furthermore, a new Vehicle Routing Problem (VRP) is constituted according to the structure of the logistic environment. Mathematical formulas of the problem and a new VRP algorithm, that minimizes logistical cost is also introduced and discussed in detail. Then, design and implementation stages of the logistic environment on Distributed Composite Object (DCO) model are discussed; features of user applications and interaction among them are examined.

DCO model forms the basis of the execution environment of the application. The middleware which implements the DCO model provides the basic mechanisms of communication and solves data sharing issues such as replication, fragmentation, consistency management, dynamic loading of applications, etc.

1. GİRİŞ

Son yıllarda uygulamaların bilgi işlem alanı bireysel ortamlardan ağ ve internet ortamına taşındığı için geniş alana yayılmış bilgiyi kontrol eden uygulamaların tasarımı konusu ortaya çıkmıştır. Burada ayrıca, yalnız uygulama tasarımı değil, hem de bu uygulamaların çalıştığı dağıtılmış ortamların geliştirilmesinin de önemi artmıştır. Dağıtılmış sistemler bilgisayar ağları üzerinden sistem kaynaklarının ve bilginin paylaşımını sağlarlar. Dağıtım ile ilgili tüm ayrıntıların kullanıcılara saydam olarak gerçekleniyor olması bu sistemleri çekici yapan önemli bir tasarım konusudur.

Geniş alan uygulamaları kullanıcılarına bilgi alış-verişi ve paylaşımı için bazı kolaylıklar sağlarlar. Bu bağlamda, dağıtılmış sistemler üzerine günümüze değin birçok proje geliştirilmiştir. Bu modeller genelde uzaktan yordam çağırma (Remote Procedure Call-RPC), noktadan-noktaya mesaj iletimi (point-to-point message passing), dağıtılmış paylaşılan bellek (distributed shared memory) kullanımı gibi düşük-düzeyli haberleşme ilkellerini kullanarak gerçeklenmişlerdir[1][2]. Ancak, bu nispeten düşük düzeyli haberleşme modelleri uygulamada bazı problemlerin de kaynağı olurlar. Öncelikle, bu modeller kopyalama (replication) ve verinin göç ettirilmesi (data migration) gibi, tüm dağıtılmış sistemler için geçerli olan haberleşme problemlerine çözüm sağlayamamaktadırlar. Bu noktada, geniş alan dağıtılmış sistemleri için bilgi alış-verişi ve paylaşımı konusunda karşılaşılan haberleşme problemlerini çözümleyecek bir modele ihtiyaç olduğu görülmektedir. Bizim burada inceleyeceğimiz ve Daha sonra, üzerinde lojistik uygulama tasarlayacağımız Dağıtılmış Bileşik Nesne modeli[3][4], basit haberleşme ilkellerine oranla daha soyut düzeyde ve aynı zamanda internet üzerinde ortaklaşa iş yürütmeyi destekleyen, farklı yapılardaki birçok geniş alan uygulamasının geliştirilebilmesine olanak sağlayan genel ve basit bir gerçekleme ortamı sunmaktadır.

Bu model ileride anlatacağımız dağıtılmış bileşik nesne yapısını desteklemektedir. Dağıtılmış sistemler için nesne tabanlı çözümlerin daha elverişli olduğu görülmüştür[5]. Öncelikle, nesne kavramı işlevselliği gerçeklemeden ayıran doğal bir yapı sunmaktadır. Bu ayırım gereklidir. Çünkü geniş alan uygulamaları bileşenlerin farklı özelliklerdeki alt sistemler üzerine dağıtılmalarını gerektirebilir. Bu durum,

aynı işlevselliğe sahip, farklı gerçeklemelerin bulunmasını zorunlu kılar. Diğer sebep ise, nesnelerin verilere yalnızca önceden tanımlanmış olan işlemler üzerinden erişilmesine izin veren doğal yapısıdır[6].

1.1. Tezin Amacı

Bu tezin amacı, geniş alana dağıtılmış, paylaşımlı ve dinamik yönetilebilen sistemlere örnek olarak "Dağıtılmış Bileşik Nesne(DBN) model"ini destekleyen "Dağıtılmış Bileşik Nesne Tabanlı Ortam(DBNTO)" üzerinde güncel bir lojistik uygulaması tasarlamak ve gerçeklemektir. DBN modelinin ve onun gerçekleme ortamı olan DBNTO'ın avantajları ilerideki bölümlerde anlatılmıştır. Amacımız internet ve ağ ortamında çalışabilen, ayrıca DBNTO'nın avantajları içeren bir uygulamaya tasarlamak. Böylelikle, farklı alanlarda bulunan ve bu uygulamayı kullanan çok sayıdaki kullanıcı arasında bilgisayar destekli birlikte iş yürütmeye, ortaklaşa çalışmaya imkân sağlayan, yeni bir dağıtılmış lojistik uygulama yapısı oluşturmaktır.

Lojistik uygulama üç farklı arayüzle üş çeşit (alıcı, aracı/yönetici, satıcı) düğümde çalışmaktadır. DBNTO ortamı farklı düğümlerdeki nesneleri (taşıtlar, malzemeler) ve onların özelliklerini DBN modelinin bileşik nesne yapısına uygunlaştırarak gerçeklemeyi sağlar.

1.2. Literatür Araştırması

Bir dağıtılmış sistem için en önemli tasarım konusu, sistem üzerinde geliştirilecek olan uygulamalar için kullanımı kolay, esnek yapılı ve çok amaçlı bir ara katman (middleware)'ın nasıl kurulacağıdır. Bu konuda, nesneler doğal yapılarında var olan özellikleri ve dağıtılmış sistemlere uyarlanabilmede göstermiş oldukları başarı ile, önemli bir tasarım alt yapısı sunmaktadırlar.

Nesne tabanlı dağıtılmış sistemlerin tasarım ve gerçeklemesi üzerine yapılmış çok sayıda akademik ve ticari çalışmalar bulunmaktadır. Var olan modellerin ve sistemlerin önemli özelliklerini tanımlamak amacıyla aşağıdaki gibi bir sınıflandırma yapmak konunun anlaşılması açısından faydalı olacaktır.

- *Geleneksel süreçler arası haberleşme (interprocess communication-IPC) yöntemlerini kullanan modeller:* Bu modeller dağıtılmış sistemlerin önemli bir sınıfını oluşturmaktadırlar. Modele göre, dağıtılmış ortamdaki süreçler mesaj

alış-verişi yoluyla ya da paylaşılan bir veri üzerindeki işlemler aracılığıyla haberleşirler.

- *Vekil-tabanlı (proxy-based) nesne modelleri:* Son yıllarda geliştirilen ve iyi bilinen bir çok sistem bu modeli benimsemiştir. Bu modelde, nesne tek bir makine üzerine yerleştirilir. Bu nesnenin herhangi bir metoduna çağrıda bulunma, istekçi tarafına bir vekil atamak suretiyle, ağ üzerinden saydam olarak yapılır.
- *Bölünmüş (partitioned) nesne modelleri:* Bu modelde, nesnenin birçok makine üzerinde fiziksel olarak dağıtılmış olduğu kabul edilir. Fakat kullanıcılar dağıtımın farkında değildirler. Nesne kullanıcılara bir bütün olarak görünür. Dağıtılmış olan parçaların başarım ve kullanılabilirliği arttırma amacıyla değişik alanlar üzerinde kopyalanması da mümkündür.

Konunun ilerleyen bölümlerinde yukarıda sözü edilen üç farklı model sınıfına giren ve literatürde kabul görmüş sistemler üzerinden her bir modelin değerlendirmesi yapılacaktır.

1.2.1. Geleneksel IPC modellerinin değerlendirmesi

Birçok dağıtılmış sistem mesajlaşma yoluyla ya da paylaşılan veri üzerindeki işlemle aracılığıyla süreçlerin haberleşmesi modelini destekler. Mesaj iletim modelleri düşük seviyede bir soyutlama sağlıyor olmalarına rağmen, iyi ölçeklenebilen sistemler olduklarından dolayı popülerliklerini hala devam ettirmektedirler. *Paralel Sanal Makine (Parallel Virtual Machine-PVM)* [7] ve *Mesaj İletim Arayüzü (Message Passing Interface-MPI)* [8] gibi haberleşme kütüphanelerine ve *Dağıtılmış Hesaplama Ortamı (Distributed Computing Environment-DCE)* [9] gibi RPC temeline dayanan dağıtılmış hesaplama uygulamaları, doğrudan TCP ve UDP gerçeklemelerine dayanmaktadırlar. Ancak, mesaj iletimi yoluyla dağıtılmış uygulamalar geliştirmek zordur. En önemli sınırlama verinin yerleşim, kopyalama, tutarlılık ve sürekliliğinin yönetimi gibi önemli gerçekleme konularının uygulamaya bırakılıyor olmasıdır. Yani, her uygulama bu mekanizmaları kendileri tasarlamalı ve gerçeklemelidir. Her bir uygulamanın kendi alt yapısını kurduğu bir ortamda yeni uygulamaların geliştirilip, diğerleri ile entegrasyonunun sağlanması da son derece güç olacak, hatta belki de mümkün olmayacaktır.

Mesaj iletimine dayanan sistemlerin ardından, paylaşılan veri üzerinde işlem yürütme esasına dayanan yeni bir prosesler arası haberleşme modeli ortaya çıkmıştır. Bu modelin tipik bir örneği *Ağ Dosya Sistemi (Network File System-NFS) ve dağıtılmış paylaşılan bellek (Distributed Shared Memory-DSM)* gerçeklemeleridir [10]. Paylaşılan veri modeli, sekizlilerin okunup yazılması için küçük bir ilkeller kümesi sunmaktadır. Buradaki ana problem, verinin tutarlılığını korurken, aynı zamanda başarım ve ölçeklenebilirliği de sağlayabilmektir. Örneğin, paylaşılan dosyaların başarımı düşürmeden özel kilit yöntemleri kullanılarak korunması ihtiyacı olabilir. Aynı şekilde, DSM sistemleri gevşek bellek tutarlılığı gerektirebilir. Sonuç, anlaşılması çok güç bir haberleşme modelidir. Ayrıca, sayfa-tabanlı DSM sistemleri bilgi gizleme konusunda bir desteğe sahip değillerdir. Yine, ciddi başarım problemleri vardır ve asla çok geniş alana ölçeklenemezler.

1.2.2. Vekil tabanlı nesne modellerinin değerlendirmesi

Yukarıda bahsedilen IPC-tabanlı modellere alternatif olarak, nesne tabanlı yaklaşımlar daha çok benimsenmiştir. Nesneler yalnızca büyük ölçekli uygulamalar oluşturmayı kolaylaştırmakla kalmaz, aynı zamanda dağıtılmış uygulamalar için iyi bir mimari modelini de beraberinde getirirler. Nesneler özellikle *ince-taneli (fine-grained)* hizmet sağlayıcıları olarak görülebilirler. Vekil-tabanlı sistemlere *Uzaktan Metot Çağırma (Remote Method Invocation-RMI)* [1], RPC'deki teknik kullanılarak gerçeklenebilir. Ancak, bu yaklaşım vekil-tabanlı modellerin dünya çapına ölçeklenebilmelerinin de ana engellerinden biridir. Bu yüzden, nesne kopyalama ve asenkron metot çağırma gibi ek mekanizmalara ihtiyaç vardır.

En basit şekliyle RMI, geleneksel RPC yapısı gibi gerçeklenebilir. Örneğin, *Dağıtılmış Bileşen Nesne Modeli (Distributed Component Object Model-DCOM)*'da [11] bu yöntem kullanılmıştır. DCOM'da istekçiler için nesnelerden çok, uzak sunucular tarafından gerçeklenen arayüzler sunulur.

Nesneleri açıkça destekleyen yaklaşımlar daha ilginçtir. Örneğin Legion[12] sisteminde, nesneler farklı adres alanlarına yerleştirilmişlerdir ve metot çağrısı doğrudan mesaj iletimi ile gerçeklenir. Bu durum dağıtım saydamlığını bozmasına rağmen, Legion yaklaşımı geniş alana ölçeklenebilmeyi sağlayan ender yaklaşımlardan biridir. Bu model nesnelerin nasıl gerçekleneceği konusunda bir kural koymaz. Sadece, onların nasıl etkileşimde bulunacağını belirtir. Globus[13]

projesi esnek gerçeklemeleri desteklemek amacıyla global işaretçiler geliştirmiştir. Global işaretçi uzaktaki bir nesneye bir referanstır. İşaretçi, istekçi tarafından seçilen bir çok nesne ile haberleşme protokollerini tanımlar. Dağıtım saydamlığı henüz sağlanamamış olmasına rağmen, global işaretçiler Legion yaklaşımından daha üst derecede esneklik sağlar.

Dağıtım saydamlığı en açık şekliyle *(Object Request Broker-ORB)*'larda görülmektedir. ORB, nesneler ile onların istekçileri arasında bir ara bulucudur. Nesne bir sunucunun adres alanı içinde yer alır ve istekçiler metotlara kendi adres alanlarında bulunan bir vekil nesnesi aracılığıyla çağrıda bulunurlar. Temelde ORB'lar, ILU'daki gibi sadece dilden bağımsız ve yerleşkeye saydam metot çağrılarını destekler. *Genel Nesne İstek Aracı Mimarisi (Common Object Request Broker Architecture-CORBA)* [14]'ya ait olan ORB'lar adlandırma, süreklilik ve *hareket (transaction)* gibi ilave dağıtım hizmetleri de sunarlar. Fakat ne yazık ki, CORBA henüz nesnelerin saydam olarak kopyalanması ve kopyalar arasındaki tutarlılığın sağlanması için gerekli hizmetler tanımlamamıştır. Oysa, CORBA'daki nesneler kendi kopyalama politikalarını oluşturabilirler. Ayrıca, hali hazırdaki hizmetlerin ne derecede ölçeklenebilir olarak gerçekleneceği de çok açık değildir.

ORB dağıtım hizmetlerinden sorumlu olduğunda, ek olarak çekirdek nesne modelinin bağımsız mekanizmalarına gereksinim duyulur. Bu şekil bir mekanizma Spring sisteminde kullanılan *alt-sözleşme (subcontract)* [15] mekanizmasıdır. Alt-sözleşme bir çağrı protokolünü gerçekler; istekçi tarafındaki bir metot çağrısının etkisini nesne tarafındaki metot çağrısı ya da çağrıları şeklinde tanımlar. Örneğin kopyalanma durumunda, bir istekçi tarafından yapılan metot çağrısı, her bir kopyanın metodunun çağrılması şeklinde gerçeklenir. Çağrının kopyalanması alt-sözleşme yapısının içinde gerçeklenir ve istekçiden gizlidir. Ancak, genel bir mekanizma olarak, alt-sözleşme yapısı oldukça sınırlıdır. Örneğin, çok sayıdaki istekçi tarafından paylaşılan nesneler grubunun tutarlılığın alt-sözleşmeler ile sağlanması zordur.

1.2.3. Bölünmüş nesne modellerinin değerlendirmesi

Genel dağıtım hizmetleri geliştirmeye bir alternatif olarak sunulan bölünmüş nesne modelinde, nesnenin bölünme şekli, kopyalanması, göç ettirilmesi gibi nesnenin dağıtımı ile ilgili tüm ayrıntılar onun gerçeklemesinin bir parçasıdır ve nesnenin sunduğu arayüzün arkasına gizlenmiştir. Bölünmüş nesneler ilk olarak SOS işletim

sisteminde [16] *Parçalı Nesneler (Fragmented Objects-FO)* modeli [17] ile ortaya atılmıştır. Globe [18] projesindeki dağıtılmış paylaşılan nesneler bölünmüş nesnelerin bir diğer gerçeklemesini oluşturur. DUPLEX'te [19] ise, müşterek bir grup çalışmasının sağlanması amacıyla parçalanan bir doküman dağıtılmış bir nesne olarak ele alınır. Fakat, DUPLEX'te geniş alana bir dağıtım sağlanmış olmasına rağmen, yöntemin ilgilendiği nesneler sayıca azdır ve yalnızca bir dokümanın paylaşımı söz konusudur. Bu amaçla, yalnızca uygulamaya özel bir yapı geliştirilmiştir. Ayrıca, ilgilenilen nesne sayısı az olduğu için adlandırma yapısı basittir.

Bölünmüş nesne modelleri arasında FO ve Globe'un dağıtılmış paylaşılan nesneleri iddialı iki modeldir. Bu iki model arasındaki en önemli farklılık, FO modelinin geniş alana yayılmış ağlar için tasarlanmamış olmasıdır. FO için temel olarak kullanılan haberleşme nesneleri yalnızca yerel alan ağları için tasarlanmış ve gerçeklenmiştir. Ayrıca, FO modelinde nesneye özel kopyalama stratejileri için de bir yöntem sunulmamıştır. Bu model farklı tutarlılık politikalarının gerçeklenmesi yönünde bir yaklaşım sunmamakta ve platform heterojenliği konusunu da ele almamaktadır. Fakat, bölünmüş nesne modelini ilk olarak ortaya atan bir yaklaşım olduğu için, diğer yaklaşımlara ilham kaynağı olmuştur.

Globe, programlama dilinden bağımsız olan nesnelerinin alt parçalara ayrılması, alt parçaların kopyalanması, her biri için farklı kopyalama stratejilerinin belirlenebilmesi, nesne durumunun göç ettirilebilmesi gibi her türlü dağıtım politikalarının nesne bazında düzenlenebildiği yapısı ile mevcut nesneye-yönelik dağıtılmış sistem perspektifine radikal bir değişim getirmiştir. Alt tabakada bulunan ve dünya çapına dağılmış milyonlarca kullanıcının binlerce nesnesine hizmet verebilecek kapasitede olduğunu iddia ettikleri adlandırma ve yerleşke bulma hizmetiyle de diğer yaklaşımlardan ayrılmaktadır.

AspectIX [20], CORBA uyumlu ve bir dağıtılmış nesnenin farklı adres alanlarına yüklenmiş yerel parçalardan oluştuğu yapıyı destekleyen bir ORB sunar. AspectIX'te, nesneler istekçilerin ihtiyaçları doğrultusunda ve değişen sistem koşullarına göre uyarlanabilir olma özelliğine sahiptir. Globe nesne modeli içinde bulunan yerel nesnenin farklı dağıtım, kopyalama, tutarlılık politikaları sunan gerçeklemelere izin veren yapısı itibariyla iki model birbirine oldukça benzemekle

beraber, AspectIX'in CORBA uyumlu olması ve hatta CORBA uygulamalarına sunuculuk yapabilmesi gibi özellikleri de vardır.

1.2.4. Dağıtılmış bileşik nesne modeli ile bölünmüş nesne modelinin karşılaştırılması

Dağıtılmış bileşik nesne modeli ile bölünmüş nesne modeli karşılaştırıldığında, bir çok açıdan benzerlikler içermekle birlikte, bazı yönlerden birbirlerinden ayrılmaktadırlar. İki model arasındaki benzerlik ve farklılıklar şu şekilde sıralanabilir.

- *Geniş alan uygulamaları için destek sağlama:* Her iki model de geniş alana yayılmış ve müşterek yürütülen uygulamaları destekleyecek bir alt yapı sunmaktadır.
- *Kopyalama:* Her iki model de belli ölçülerde nesne kopyalama yöntemini desteklemektedirler. Tüm nesne erişimleri yerel kopyalar üzerinden yapılır. Ancak, DBN modelinde tüm yerel kopyalar birbirleri ile aynı iken, örneğin Globe nesne modelinde, bir yerel kopya alt nesneye ait bir durum bilgisi içermeyip, tüm çağrıları bir uzak-kopyaya ileten bir vekil gibi davranabilmektedir.
- *Adlandırma ve yerleşke bulma hizmetleri:* Bölünmüş nesne modeli sınıfına giren FO ve DUPLEX'in sunduğu adlandırma ve yerleşke bulma hizmetinin yapısı, DBN modeli tarafından sunulan yapı ile benzerlikler göstermektedir. Üçünde de yönetilen nesne sayınının çok fazla olmadığı (binler mertebesinde) kabul edilir. Diğer yandan, Globe projesinin sunduğu yapı, dünya çapına dağılmış milyonlarca kullanıcının her birinin on binlerce nesnesini destekleyecek şekilde tasarlanmıştır.
- *Nesne erişim şekli:* DBN modelinde, yeni yaratılan bir bileşik nesne kullanıcı tanımlı bir ad ile *kaydedildikten (registration)* sonra, bir başka alanda çalışan uzak kullanıcıya ait bir uygulama tarafından bir *arama (lookup)* komutu yürütüldüğünde, o alana alt nesneleri olmaksızın yalnızca kabuk nesne kopyalanır. Daha sonra yürütülecek metot çağrıları ile gerekli alt nesneler de ilgili alan üzerinde kopyalanırlar. Yani, alt nesnelerin varlığı kabuk nesne içine gizlenmiştir. Bölünmüş nesne modeli sınıfına giren projelerde ise, erişim doğrudan alt nesneleredir ve kullanıcı erişeceği alt nesneye ait bir arayüz tablosu

aracılığıyla metot çağrılarını gerçekleştirir. Bundan dolayı, DBN modeli daha üst seviyede bir soyutlama sağlamaktadır. Öyle ki, bu yapıda kullanıcı dağıtılmış ve parçalı bir nesneye erişiyor olduğunu dahi bilmiyor olabilir.

- *Dinamik yayılma:* DBN modelinde bir bileşik nesne tek bir alan üzerinde tüm alt nesneleri ile birlikte yaratılır ve zaman içerisinde diğer alanlardan gelen istekler ile dinamik olarak yayılarak dağıtılmış bileşik nesne görüntüsünü kazanır. Bölünmüş nesne modelinde ise, bir nesne farklı alanlar üzerinde yaratılabilir.
- *Dinamik uyarlama***:** Büyük ölçekli geniş alana yayılmış olan bir uygulama dinamik olarak yeni bileşenlerin eklenmesi ve var olanların çıkarılabilmesine olanak tanımalıdır. DBNTO, mevcut durumda nesneye bağlantı sağlamış olan istekçileri etkilemeksizin, DBN modeli bir bileşik nesneye yürütme zamanında yeni bir alt nesne eklenmesine, var olanların çıkarılmasına ya da değiştirilmesine imkan tanır. Fakat, bölünmüş nesne modeli sınıfına giren çalışmalar üzerinde yapılan incelemelerde böyle bir özelliğe rastlanılmamıştır.
- *Saydamlık:* DBN modelinde, dağıtılmış ortamda bulunan çok sayıdaki kullanıcı tarafından ortaklaşa yürütülen bir uygulama merkezi bir yapıda çalışılıyormuş gibi geliştirilebilmektedir. Alt nesnelerin dağıtımı, senkronizasyon ve tutarlılık ile ilgili konular uygulama kodundan bağımsız olarak çözümlenmektedir. Aynı yapı, Globe ve FO projelerinde de görülmektedir. Bölünmüş bir nesnenin parçaları için ayrı ayrı düzenlenebilen dağıtım, senkronizasyon ve tutarlılık yöntemlerini gerçekleyecek alt seviye nesneler bir kütüphaneden elde edilebilmektedir.

1.3. Tez Planı

Tezin ikinci bölümünde Lojistik ortamın tasarım aşamaları anlatılmıştır. Burada önce ortamı oluşturan üç çeşit alıcı, aracı ve satıcı düğümlerin özellikleri açıklanmıştır. Her düğüm sadece kendi çeşidine uygun işlevleri yapmakta ve bu yolla lojistik ortamın karmaşıklığı azalmakta, ölçeklenebilirliği ise artmaktadır. Daha sonra, lojistik ortamın protokole benzeyen çalışma yapısı incelenmiştir. Bu yapı tüm etkileşim ve çalışma adımlarını, ayrıca optimal gönderim planının hesaplanması adımlarını da içermektedir.

Üçüncü bölümde lojistik ortama uygun optimal gönderim planını hesaplamak için geliştirilmiş Taşıt Yönlendirme Problemi (TYP) [21] oluşturulmuş, Daha sonra, matematiksel yöntemlerin analizi ve TYP'ni gerçekleyen algoritmanın çalışma adımları açıklanmıştır.

Dördüncü bölümde Dağıtılmış Bileşik Nesne (DBN) modeli tanıtılmış, onun nesne yapısı ve haberleşme şekli anlatılmıştır, Daha sonra, nesnelerin farklı alanlar üzerinde paylaşımı ve dağıtılmış bir bileşik nesnenin oluşumu açıklanmıştır. Nesnelerin kontrolü ve dağıtılmış bir ortam için gerekli olan saydamlık konusu incelenmiştir. Ayrıca DBN modelinin başka modellere göre avantajları anlatılmıştır. Daha sonra, bu modelin gerçekleme ortamı "Dağıtılmış Bileşik Nesne Tabanlı Ortam"ın genel yapısı açıklanmış, sistemin temel fonksiyonları hakkında bilgi verilmiştir.

Beşinci bölümde uygulamanın gerçekleme aşamaları incelenmiş, farklı tipte kullanıcı arayüzlerinin tasarımı, düğümlerde sistemin ve uygulamaların başlatılması, uygulamalar arası etkileşim konuları anlatılmıştır. Lojistik ortamın gerçeklenmesinde DBN modelinin kullanımı ve avantajları anlatılmıştır. Bu model lojistik uygulamanın ortam modeli olarak kullanılmıştır ve geniş alan uygulamalarının geliştirilmesine olanak sağlayan genel ve basit gerçekleme ortamı sunmaktadır. DBN modeli ayrıca internet üzerinden ortaklaşa iş yürütme imkânı da tanıyor.

Altıncı bölümde benzetim ortamı, deneysel sonuçlar ve değerlendirmeler yapılmıştır. Burada lojistik uygulamadan elde edilen test sonuçları ve geliştirilmiş TYP algoritmasından elde edilen test sonuçları değerlendirilmiştir.

Yedinci ve son bölüm olan sonuçlar bölümünde, geliştirilen lojistik uygulama ve bu modelin çalışma ortamı olan DBNTO sisteminin sağladığı katkılar anlatılmış ve öneri olarak, gelecekte sistemin daha da geliştirilmesine yönelik yapılabilecek çalışmaların neler olabileceği üzerinde durulmuştur.

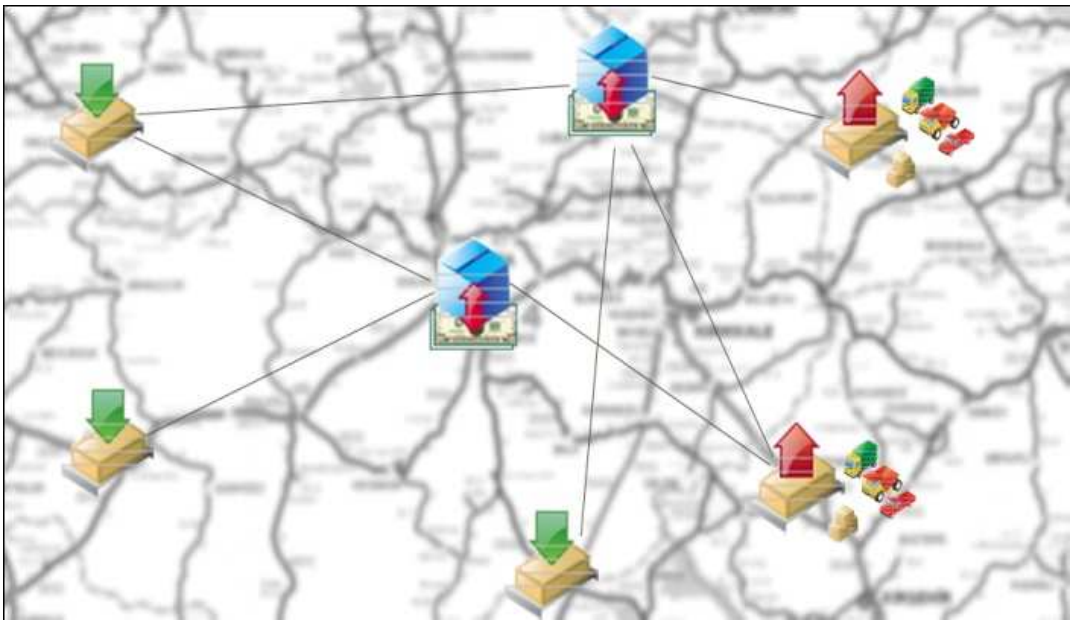
2. LOJİSTİK ORTAM

Basit haliyle bir lojistik ortam incelendiğinde ortamda *satıcı* ve *alıcı* taraf adlandırılan firma çeşitleri arasında istekler doğrultusunda malzeme alış-verişi gerçeklendiği görülmektedir[22]. Bu tip ortamlarda hem satıcı, hem de alıcı taraf sadece gerekli satım veya alım işlemlerini değil, kısmen de olsa yöneticilik vazifesini üstlenmektedir. Bu durum bir alıcı veya satıcı firmanın daha çok firmalarla işbirliği yaptığı durumlarda bilgi işlem karmaşıklığını artırmakta, sistemin ölçeklenebilirliğini azaltmaktadır[23][24].

Çalışmada önerilen lojistik ortam bu problemlere daha iyi bir çözüm sunmaktadır. Bu ortamda alıcı ve satıcı firmalar arasında yöneticilik görevi *aracı* firmalar adlandırılan firmalara verilmektedir.

2.1. Lojistik Ortamın Genel Görünümü

A*lıcı*, *satıcı* ve *aracı* firmalar birlikte lojistik ortamı oluştururlar. Satıcı firmalarda satışa sunulan farklı malzemeler ve onları taşıyacak taşıtlar vardır. Alıcı firmalar satıcı firmalardan farklı malzeme çeşitleri almakla kendilerinin bu malzemelere olan taleplerini karşılıyorlar. Aracı firmalar satıcı ve alıcı firmalar arasındaki tüm alım-satım işlemlerini yönetir ve denetlerler. Düğümler arasında mesafe ölçüsünden yollar vardır ve malzemeler bu yollarla firmalara taşınırlar. Genelde, bu yolların maliyeti iki düğüm arasında farklı yönlerde değişik değerlerde olabilir. Malzeme taşıyan taşıtlar alıcı firmalardan başlayıp, rotaları üzerindeki farklı alıcı firmaların malzeme isteklerini karşıladıktan sonra tekrar kendi satıcı firmalarına dönerler. Şekil 2.1’de lojistik ortamın genel görünümü ifade ediliştir. Yeşil renkli ok bulunan kutucuklar alıcı firmaları, kırmızı renkliler ise taşıtları ve malzemeleri ile satıcı firmaları gösterir. Mavi renkli kutucuklar aracı firmalardır. Onlar hatlarla gösterilmiş birkaç alıcı ve satıcı firmaları yönetirler.



Şekil 2.1: Lojistik Ortamın genel görünümü

Bu noktada ayırt edilmesi gereken diğer bir konu düğümler arasında haberleşme yolu (veri iletim yolu) ile mesafe ölçüsünden yol anlayışı arasındaki farktır. Haberleşme yolları farklı düğümlerde çalışan kullanıcı uygulamalar arasında bilgi alış-verişini sağlamak için gereklidir. Bu işlemler genelde yerel ağlar ve internet ağları üzerinden gerçeklenir.

Firma çeşitleri aşağıdaki gibi sıralanabilirler:

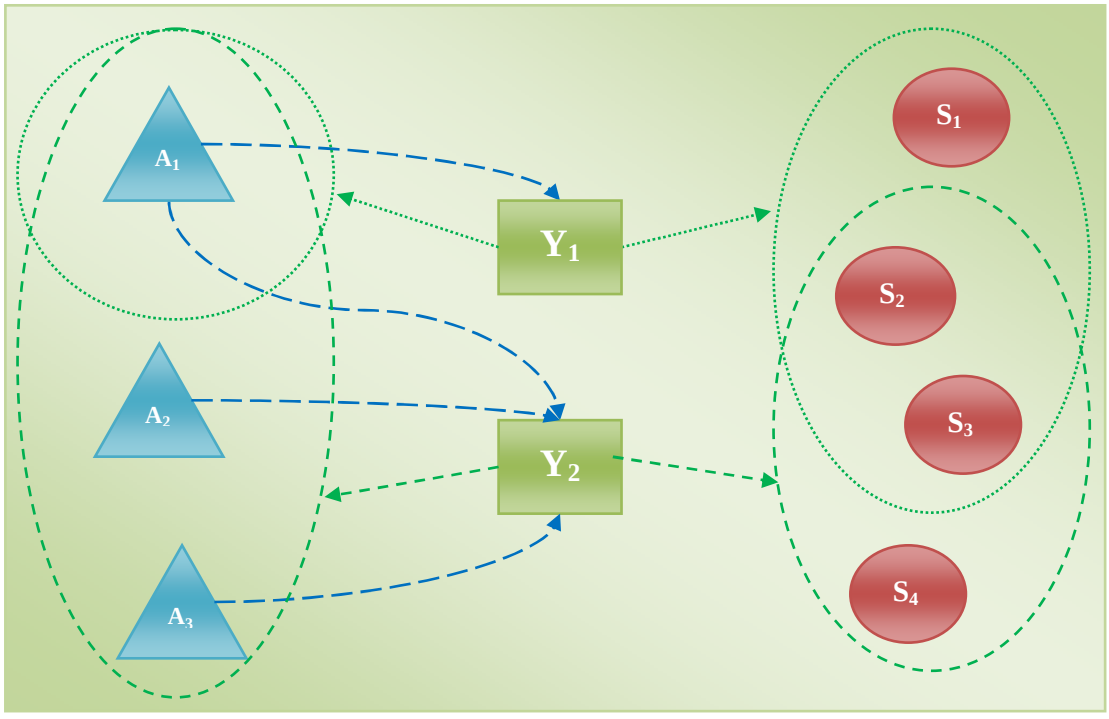
- ***Satıcı firmalar***. Bu firmalarda satışa sunulan farklı malzemeler ve onları taşıyacak farklı taşıt araçları yer alır. Her satıcı firmanın en az bir aracı firma ile anlaşması vardır ve alıcı firmalarla ile aracı firma üzerinden işbirliği yapmaktadır. Bu firmalarda olan malzeme ve taşıt miktarı zaman içinde dinamik olarak değişken değerlerdir(yeni malzemelerin veya taşıtların eklenmesi veya çıkarılması). Alıcı firmalar tarafından gönderilen malzeme istekleri aracı firmalar tarafından değerlendirildikten sonra yalnız gerekli satıcı firmalara ulaşır. Her satıcı firmanın bu istekleri incelemek ve sadece uygun gördüklerini kabul etmek özelliği vardır.
- ***Aracı firmalar***. Bu firmaların farklı satıcı ve alıcı firmalar ile ikili anlaşmaları vardır ve onlar arasında aracı/yönetici rolünü üstlenirler. Anlaşmada oldukları alıcı firmalardan gelen istekleri karşılamak için anlaşmada oldukları satıcı firmaların taşıt ve malzeme kapasitesine, alıcı ile satıcı firmalar arasındaki yol mesafesine, taşıma maliyetine göre optimal

gönderim planı hazırlamak ve bunu alıcı tarafa sunmakla görevlidirler. Aracı/yönetici firmalar ayrıca anlaşmada oldukları tüm alıcı ve satıcı firmaların coğrafi yerlerini de kendi veritabanında tutarlar. Bu değerler optimal gönderim planını hesaplamada kullanılır.

- ***Alıcı firmalar***. Bu firmalar isteklerini belirler ve anlaşmada oldukları aracı firmalara göndererek onlar üzerinden isteklerini karşılaya bilirler. Alıcı firmanın isteklerinin satıcı firmalara iletmesi için onun en az bir aracı firma ile anlaşmada olması zorunludur. Bu firmalar birkaç aracı firma ile anlaşmada oldukları durumda, tüm bu aracı firmalara aynı isteği göndererek onlara sunulan optimal gönderim planlarından en uygun olanını kabul ede bilirler.

Lojistik ortamındakı firmaların istedikleri zaman uygun yeni firmalarla anlaşmak ve anlaşmada oldukları firmalarla anlaşmayı bozmak özellikleri vardır.

Şekil 2.2'de, lojistik ortamın haberleşme yapısı gösterilmiştir.



Y_i – aracı firmalar, S_i – satıcı firmalar, A_i– alıcı firmalar

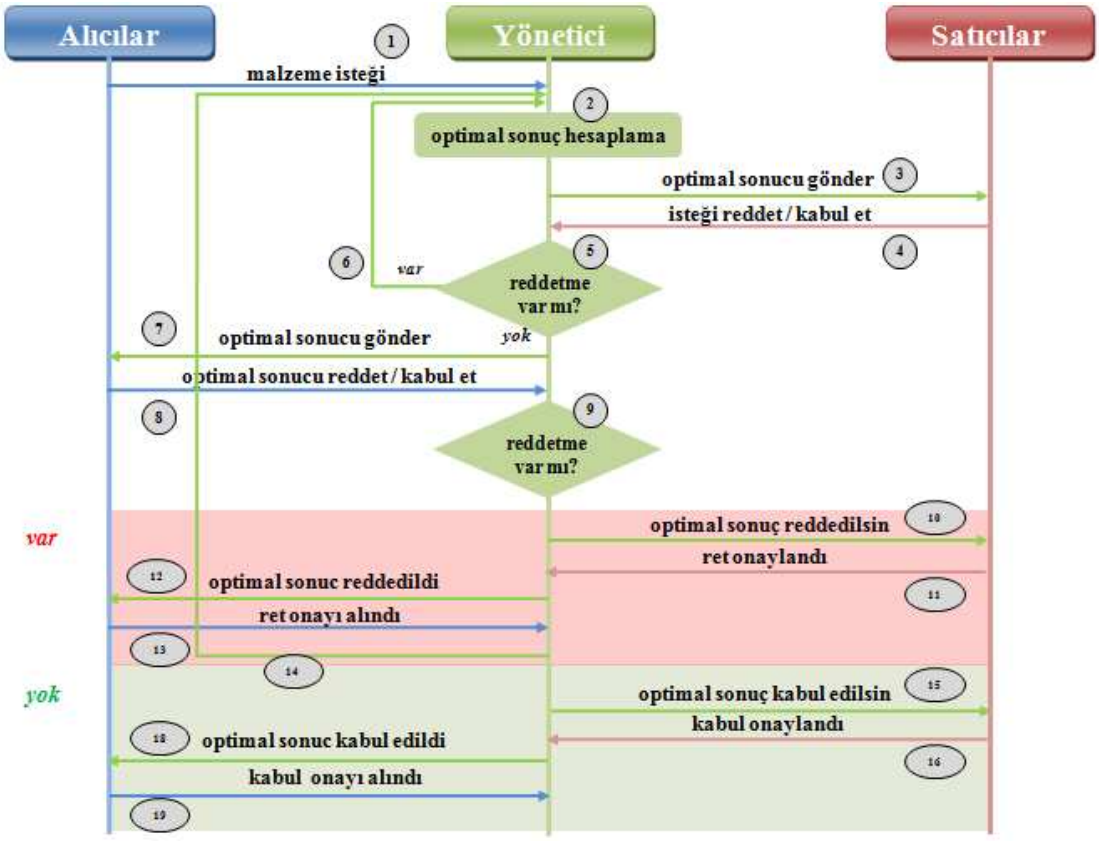
Şekil 2.2: Lojistik Ortamın haberleşme yapısı

Burada Y_1, Y_2 alıcı ile satıcı arasında aracı firmaları, S_1, S_2, S_3, S_4 satıcı firmaları, A_1, A_2, A_3 alıcı firmaları göstermektedir. Y_1 aracı firmasının S_1, S_3, S_4 satıcı firmaları ve A_1 alıcı firması ile, Y_2 aracı firması ise S_2, S_3 ,S_4 satıcı firmaları ve A_1, A_2, A_3 alıcı firmaları ile anlaşması vardır. A_1 alıcı firması hem Y_1, hem de Y_2 aracı

firmalarına istek gönderir. Değerlendirme ve hesaplamalara uygun olarak Y_1 ve Y_2 aracı firmaları sonucları alıcı A_1 firmasına gönderir. A_1 alıcı firması gönderim maliyetini, malzemelerin satış fiyatlarını ve başka değerleri kontrol ettikten sonra daha uygun sonucu kabul eder.

2.2. Lojistik Ortamının Çalışma Şekli

Burada, lojistik uygulama ortamında firmalar arası etkileşim adımlarını gösteren, ortam için protokol özelliği taşıyan bir yapı incelenecektir[25]. Bu yapı alıcılar tarafından gönderilen isteklerin aracı firmalar üzerinden değerlendirilmesini, optimal sonuçların bulunmasını, satıcı firmalar tarafından karşılanan isteklerin aracı firmalar aracılığıyla alıcılara gönderilmesini ve genel olarak sistemin çalışma şeklini oluşturur. (İlerleyen bölümlerde aracı ve yönetici kelimeleri karşılıklı kullanılacaktır).



Şekil 2.3: Lojistik uygulama ortamının çalışma şekli

Şekil 2.3'te birkaç alıcı firma, bir aracı firma ve birkaç satıcı firma arasında lojistik uygulama ortamının çalışma şekli anlatılmıştır. Birden fazla aracı firması olan bir

lojistik uygulama ortamı da aynı şekilde incelenebilir, çünkü aracı firmaların kabul ettiği kararlar birbirinden bağımsızdır.

Protokolün çalışma adımlarını Sekil 3 üzerinde inceleyelim:

(i) İlk önce aracı firma ile anlaşmada olan alıcı firmalardan aracı firmaya "malzeme isteği" mesajı (1) gönderilir. Bu mesaj farklı çeşitte olan malzeme isteklerini içerir (örnek: 5 ton kum, 7 ton çimento).

(ii) Alıcı firma tarafından kabul edilmiş istekler protokol için hazırlanmış politikalara uygun olarak belirli bir zaman periyodundan sonra veya anında değerlendirile bilir. Bu zaman aracı firma kendisi ile anlaşmada olan tüm etkin satıcı firmalardan malzeme türü, malzeme miktarı, malzemenin satış fiyatı, taşıt çeşidi, taşıt sayısı, taşıt kapasitesi, yol mesafesi, taşıma maliyeti ve başka gerekli bilgilerini öğrenir ve "optimal gönderim planı"nı hesaplar (2). Bir sonraki alt bölümde "optimal gönderim planı"nın nasıl hesaplandığı anlatılmıştır.

(iii) Daha sonra, optimal gönderim planına uygun olarak anlaşmada olan satıcı firmalara "optimal sonucu gönder" mesajı (3) gönderilir. Bu mesaj uygun satıcı firmadan taşınması gereken malzemenin çeşidi ve miktarını, hangi taşıtların kullanılacağını, her taşıtın ne kadar malzeme taşıyacağını ve hangi rotayı izlemesi gerektiğini, hangi alıcı firmaya ne kadar malzeme taşıması gerektiğini içerir. Aracı firma bu mesajla her satıcı firmaya yalnız ona gerekli bilgileri gönderir.

(iv) Satıcı firmalar onlara gönderilen mesaj içeriğini analiz ettikten sonra ya bu isteği kabul ederler ya da reddederler. Bu yanıt "isteği kabul et" ya da "isteği reddet" yanıtı (4) şeklinde aracı firmaya bildirilir.

(v) Aracı firma satıcı firmalardan gelen yanıtları kabul eder ve herhangi bir satıcı firmadan "isteği reddet" yanıtı geldiği durumda (5) reddetme yanıtı gönderen satıcı firmaların dışındaki, anlaşmada olduğu firmalarla optimal gönderim planını tekrar hesaplar (6). Bu durum tüm satıcı firmalardan " isteği kabul et" yanıtı gelene kadar devam eder. Protokolde böyle bir geri dönüşümlü ve tekrar-tekrar optimal gönderim planının hesaplanması durumu sistemin hızını negatif yönde etkileyebilir, ama satıcı firmaların onlara gönderilen isteklerde

maddi çıkarları olduğuna göre bir isteği reddetme olasılığı düşük olacaktır. Bu da sistemin kötü yönde etkilenmesini azaltacaktır.

(vi) Tüm satıcı firmalardan "isteği kabul et" yanıtı geldikten sonra, aracı firma tarafından istekleri değerlendirilmiş tüm alıcı firmalara "optimal sonucu gönder" mesajı gönderilir (7). Bu mesaj alıcı firmalara hizmet verecek satıcı firmalar, onların taşıtları, her taşıtın malzeme miktarı, her satıcı firmadaki malzeme satış fiyatı, toplam maliyet hakkında ve başka bilgileri içerir. Aracı firma "optimal sonucu gönder" mesajında her alıcı firmaya yalnız ona gerekli bilgileri gönderir.

(vii) Alıcı firmalar bu mesajın içeriğini analiz ettikten sonra ya sonucu kabul eder ya da reddederler. Bu yanıt "optimal sonucu kabul et" ya da "optimal sonucu reddet" yanıtı (8) şeklinde aracı firmaya geri döner.

(viii) Aracı firma alıcı firmalardan gelen yanıtları denetler ve herhangi bir "optimal sonucu reddet" yanıtı (9) geldiği durumda anlaşmada olan uygun satıcı firmalara "optimal sonuç reddedildi" bilgisi (10) gönderir. Bu bilgi satıcı firmaları optimal sonucun reddedildiği ile bilgilendirmek içindir.

(ix) Bilgiyi alan satıcı firmalar "ret onaylandı" yanıtı (11) ile bilginin onlara ulaştığını aracı firmaya bildirirler.

(x) Aynı şekilde "optimal sonuç reddedildi" bilgisi (12) ret yanıtı gönderen alıcı firmalar dışındaki istek gönderen alici firmalara da gönderilir.

(xi) Alıcı firmalar tarafından gönderilen "ret onayı alındı" (13) yanıtı ile bilginin karşı tarafa ulaştırıldığı anlaşılır.

(xii) Daha sonra, optimal gönderim planının tekrar hesaplanması (14) işlemi yürütülür. Bu durum sık-sık oluşmasa da sistemi negatif yönde etkileyen bir durumdur. Durumun oluşma nedeni daha çok alıcı firmanın isteği tamamen reddetmesi veya isteği değişmesi ile alakalı olacaktır, çünkü her reddetme sonrası hesaplanan yeni optimal gönderim planı bir öncekinden daha optimal olmayacak, bu da alıcı firmanı aynı malzeme isteğinden dolayı daha çok para ödemek zorunda bırakacaktır. Burada bazı politikalar belirleyerek (alıcılara sadece isteğin tamamını reddetmeğe izin vermek, isteğin değiştirilmesine izin vermemek gibi) olayların daha hızlı gerçeklenmesi sağlana bilir. Ama aracı firma her iki çeşit(alıcı ve satıcı) firmalar arasında

yalnız yöneticilik değil, aynı zamanda onların isteklerinin maksimuma yakın karşılanması vazifesini üstlendiğinden dolayı bu durumun sistemin hızlı yürütülmesini sağlamak için tamamen iptali doğru olmayacaktır.

(xiii) Aracı firmaya tüm alıcı firmalardan “optimal sonucu kabul et” yanıtı (8) geldikten sonra aracı firma anlaşmada olan uygun satıcı firmalara “optimal sonuç kabul edildi” bilgisi (15) gönderir. Bu bilgiyi alan satıcı firmalar optimal gönderim planına uygun malzemeleri gerekli alıcı firmalara göndermeğe başlarlar.

(xiv) Aynı zamanda satıcı firmalar “kabul onaylandı” yanıtını (16) bilginin onlara ulaştığını bildirmek için aracı firmaya gönderirler.

(xv) Optimal sonucu yerine yetirecek olan tüm satıcı firmalardan “kabul onaylandı” yanıtını alındıktan sonra aracı firma “optimal sonuç kabul edildi” bilgisini (17) malzeme isteğinde bulunmuş tüm alıcı firmalara gönderir.

(xvi) Bilgiyi alan alıcı firmalar tarafından gönderilen “kabul onayı alındı” (18) yanıtı ile bilginin karşı tarafa ulaştırıldığı ve alıcı firmaların isteklerine uygun malzemelerin satıcılar tarafından gönderilmeğe başladığı anlaşılır.

Bu işlemler sonucunda aracı düğüm alıcı ve satıcı düğümler arasında birim-iş (transaction) olarak nitelendire bileceğimiz yöneticilik vazifesinin bir kısmını tamamlamış olur. Daha sonraki durumlarda aracı firma alıcılar tarafından gönderilen yeni isteklerin karşılanmasını devam ettirir, aynı zamanda önceki isteklere uygun sistemin genel işleyişini kontrol eder.

3. OPTİMAL GÖNDERİM PLANININ HESAPLANMASI

Çalışmanın bu bölümünden sonra, satıcı düğüm veya depo, alıcı düğüm veya istekçi düğüm kelimeleri eşanlamlı olarak kullanılacaktır.

Bir önceki bölümde anlattıklarımızdan anlaşılıyor ki, aracı firmalar optimal gönderim planını hesaplamakla yükümlüdürler. Bu plan mümkün olan tüm alıcı firma isteklerini karşılamak ve her bir alıcı firma için toplam maliyeti en aza indirgemek zorundadır[26]. Lojistik ortamın özelliklerine göre farklı satıcı firmaların çeşitli malzemeleri satması mümkündür ve satışa sunulan bu malzemeler arasında aynı çeşitte olan malzemelerin farklı satıcı firmalardaki birim fiyatları farklı değerler olabilir[27]. Ayrıca aynı çeşitli malzeme taşıyan taşıtların özelliklerinin de (taşıt kapasitesi, taşıma maliyeti) farklı olması mümkündür. Böylece, her bir alıcı firma için toplam maliyetin hesaplanması yalnızca güzergâhın uzunluğuna bağlı değil, aynı zamanda malzeme maliyetine(satış fiyatı), malzeme miktarı ve taşıt maliyetine(taşıma maliyeti) de bağlı olacak.

Bu parametrelerle optimal gönderim planının hesaplanması ortaya çözülmesi gereken yeni bir problem çıkarır. Bu problem güzergâh uzunluğuna, malzemelerin satış fiyatlarına, miktarına ve malzemeleri taşıyan taşıt özelliklerine bağlı olan geliştirilmiş Çok Depolu Taşıt Yönlendirme Problemidir (Multi-Depot Vehicle Routing Problem)[28].

Çok Depolu Taşıt Yönlendirme Problemi (ÇDTYP) Tek Depolu Taşıt Yönlendirme Probleminin (TDTYP)[28] genelleştirmesidir. Bu problemde taşıtlar birçok depodan kalkar, rotaları üzerindeki düğümlere uğradıktan sonra tekrar kendi depolarına dönerler. Problemin geleneksel amacı tüm rotaların uzunluğunu en aza indirgemektir.

TDTYP'de deponun taşıtların bir kısmının yardımıyla taşınması gereken birkaç istek vardır. Her istek gerekli miktarda malzemenin depodan alınması ve başka istekçi düğüme ulaştırılmasından oluşur. Bu problemin amacı taşıtlar için tüm istekleri karşılaya bilen rotalar bulmaktır ve toplam maliyeti ve ya seyahat yolunu en aza indirgemektir. Her taşıtın mümkün rotası verilmiş depodan başlamalı, taşıtın

kapasitesini aşmayacak birkaç istekçi düğüme hizmet vermeli ve sonda aynı depoya dönmelidir.

Anlaşılır olması için bizim problemin ismini Geliştirilmiş Çok Depolu Taşıt Yönlendirme Problemi (GÇDTYP) adlandıralım. İleride bu problemi analiz ettiğimizde onun Heterojen Malzemelerin Taşınması için Taşıtların Değişken Dağılımı ve Parçalanmış Teslimi özellikli Çok Depolu Taşıt Yönlendirme Problemi (Multi-Depot Vehicle Routing Problem with Split Delivery and Variable Distribution of Vehicles for Transporting Heterogeneous Materials) olduğunu anlayacağız.

3.1. GÇDTYP Probleminin Matematiksel İfadesinin Oluşturulması

Aşağıda GÇDTYP problemi tanımlanmıştır.

1) Ortamda M tane muhtemel depo (D kümesi olarak belirtilmiştir) vardır. ∀i(i ϵ D, |D|=M) deposunun, v_{i_k} tane taşıtı ve m_{i_l} tane farklı malzeme çeşidi vardır.

2) ∀i(i ϵ D) deposunun v_p,(p ϵ{1,2,...,i_k}) taşıtının kapasitesi ($vcap_p$), birim uzunluğa göre maliyeti (vc_p) ve yük çeşidi (vl_p) farklı değerler olabilir. ∀v_p taşıtının kendi deposundan başlayan, birkaç tane alıcıya hizmet veren ve aynı depoya dönen R_p rotası vardır. Aynı taşıt yeterli kadar taşıt olmadığı durumlarda rotasını bitirip deposuna döndükten sonar tekrar aynı depodan yeni rotayla birkaç defa hizmet verebilir.

3) Depolarda ∃ m_q , (q ϵ{1,2,...,i_l}) malzeme çeşidi vardır ki, satış fiyatı(malzeme maliyeti) (mc_q) farklı depolarda farklı olabilir.

4) Ortamda N tane muhtemel alıcı (C kümesi olarak belirtilmiştir) vardır. ∀j(j ϵ C, |C|=N) alıcısı istekleri doğrultusunda hizmet edilmek zorundadır ve onların her biri için birden fazla taşıtın hizmet etmesi mümkündür.

5) ∀j(j ϵ C) alıcısının isteği birkaç malzeme isteğinden Dem_j ={dem_{j1}, dem_{j2},…, dem_{jr}} oluşmuş olabilir ve ∀dem_s(s=j_1,j_2,…,j_r), farklı depoların taşıtları tarafından karşılanabilir.

6) Alıcı isteğinin ona hizmet eden taşıtın kapasitesini aşması mümkündür, bu durumda istek daha fazla taşıt tarafından karşılanılır. Bu şekildeki karşılanma Parçalanmış Teslim(Split Delivery) ismiyle tanınır[29].

GÇDTYP problemi aşağıdaki program modeli şekilde ifade edilmiştir.

$$\text{Toplam Maliyet} = \sum_{a \in C} \sum_{b=1}^{p_a} Z(dem_{ab}) \tag{3.1}$$

$$\text{En Aza İndirge } Z(dem_{ab}) =$$

$$\sum_{i \in C \cup D} \sum_{j \in C \cup D} dis_{ij} \sum_{k=D} \sum_{c=1}^{v_k} \sum_{l=1}^{r_{kc}} \left(x_{ijkcl} \cdot vc_{kc} + amt_{kcl} \cdot mc_k \right) , \tag{3.2}$$

$$a \in C, b = \{1, 2, \dots, p_a\}$$

$$\sum_{k=1}^{u} \sum_{l=1}^{r_k} P_{abkl} = dem_{ab}, \qquad a \in C, b = \{1, 2, \dots, p_a\} \tag{3.3}$$

dem_{ab} – a alıcısının b malzeme çeşidine olan isteği

p_a – a alıcısının istek sayısı (kaç tane farklı malzeme çeşidine talep var)

dis_{ij} – düğümler arasındaki mesafe

v_k – k deposunun dem_{ab} isteğine uygun malzeme taşıya bilecek taşıt sayısı

r_{kc} – k deposunun c taşıtının rota sayısı

x_{ijkcl} – eşittir *1*, eğer k deposunun c taşıtı l. rotasında i alıcı düğümünden sonra j alıcı düğümüne yolculuk yapıyorsa, aksi taktirde eşittir *0*.

vc_{kc} – k deposunun c taşıtının birim uzunluğa göre taşıma maliyeti

amt_{kcl} – k deposunun c taşıtının l. rotasında dem_{ab} isteğinin taşınan miktarı

mc_k – k deposundaki dem_{ab} çeşitli malzemenin satış fiyatı

P_{abkl} – k taşıtının l. rotasında a alıcısına teslim edilmiş b malzeme çeşidinin miktarı

u – a alıcısına hizmet eden taşıtların sayısı.

r_k – k taşıtının rota sayısı

Denklem (3.1) tüm malzeme isteklerinin toplam maliyet fonksiyonunu ifade ediyor. Toplam maliyet - tüm alıcıların farklı malzeme çeşidi için olan isteklerinin maliyet fonksiyonlarının toplamına eşittir. Denklem (3.2) seçilmiş herhangi bir a alıcı

düğümünün herhangi bir *b* malzemesine olan isteğinin maliyet fonksiyonunu gösterir. Denklem (3.3) *a* alıcısının dem_{ab} miktarda *b* malzemesine olan isteğinin yalnız bu alıcı düğüme uğrayan tüm taşıtların getirdikleri *b* malzeme çeşidinin miktarları toplamına eşit olduğunu anlatıyor.

Yukarıda anlattığımız problem Çok Depolu Taşıt Yönlendirme Problemi (ÇDTYP) [28] ve Parçalanmış Teslimli Çok Depolu Taşıt Yönlendirme Problemi (PTÇDTYP)[29] problemlerinin daha genel şeklidir. (3.1)-(3.3) denklemlerindeki parametreleri aşağıdakı değerleri $p_a=1$, $v_k=1$, $r_{kc}=1$, $mc_k=0$ aldıkta denklem (3.2)

$$\text{En Aza İndirge } Z(dem_a) = \sum_{i \epsilon CUD} \sum_{j \epsilon CUD} dis_{ij} \sum_{k=D} x_{ijk} \qquad \textbf{(3.4)}$$

şeklinde yapıya sahip olur ki, bu ÇDTYP problemidir.

Parametreler aşağıdakı değerleri $p_a=1$, $v_k=1$, $mc_k=0$ aldıkta ise denklem (3.2)

$$\text{En Aza İndirge } Z(dem_a) = \sum_{i \epsilon CUD} \sum_{j \epsilon CUD} dis_{ij} \sum_{k=D} \sum_{l=1}^{v_k} (x_{ijkl}) \qquad \textbf{(3.5)}$$

şeklinde yapıya sahip olur ki, bu PTÇDTYP problemidir.

Anlatılanlardan yeni tasarlanmış taşıt yönlendirme problemini çözmek için oluşturulmuş algoritma, ÇDTYP ve PTÇDTYP problemlerini de çözmek özelliğini taşıyacaktır.

3.2. GÇDTYP için Algoritmanın Oluşturulması

Burada biz tüm Z(dem) fonksiyonlarını en aza indirgeyen rotaların bulunması yöntemini kullanacağız. Bilindiği gibi bu tip problemler için kısa zaman diliminde kesin çözüm bulunan algoritmalar henüz bulunmamıştır. Bu yüzden bu algoritmada ileride açıklanacak bazı sezgisel metotlar kullanılmıştır.

Algoritma aşağıda Şekil 3.1 üzerinden adım-adım anlatılmıştır. Sürecin başlangıcında tüm satıcı ve alıcı düğümler hakkında bazı bilgilerin (düğümlerin coğrafi konumu, düğümler arasındaki mesafe, alıcıların istekleri ve b.) edinilmesi gerekiyor. Algoritmada kullanılan bazı fonksiyonların işlevleri, ana fonksiyondan sonra anlatılmıştır.

```
function GÇDTYP (AllDemands, AllDepots)//Geliştirilmiş Çok
Depolu Taşıt Yönlendirme Problemi Algoritması
  DemandTypes=getDemandTypes()//farklı çeşitli isteklerin
     listesi
  dmc=count(DemandTypes)// farklı çeşitli isteklerin sayısı
  clear RouteList//taşıtların rotaları, başlangıcta boştur
  for mi = 1 to dmc do// tüm farklı çeşitli istekler
     incelenmelidir
    ActiveDepots=getListOfActiveDepots(mi)//mi çeşitli isteğin
     bulunduğu aktif depolar listesi
    foreach depot i in ActiveDepots do
      ActiveVehiclesOfDepot[i]=
     getActiveVehicleListOfDepot(ActiveDepots,i,mi)//i
     deposunun mi çeşitli malzemesini taşıya bilecek aktif
     taşıtları
     VehicleList[i]=
     SpecialSortByDescending(ActiveDepotVehicles[i])//"taşıt
          kapasitesi/taşıt maliyeti" değerine göre azalan
          sırada sıralanmış taşıt listesi
     VehicleIndex[i] = 1;// VehicleList[i] listesindeki ilk
          taşıt
    end foreach
    PassedPoints={};//herhangi bir depoya atanmış istekçi
     düğümler kümesi, başlangıcta boştur
    while any demand point i not in PassedPoints do
      for i=1 to N do //tüm istekçi düğümler
        if i is not in set PassedPoints then

     (d1,dm1)=findFirstMinimalDiValueAndItsDepotIndex(i);

     (d2,dm2)=findSecondMinimalDiValueAndItsDepotIndex(i);
          r=d1/d2
          isteğe bağlı sigma değeri seç, 0<sigma<1
          if r<sigma then
             i istekçi düğümünü dm1 deposuna ata
             i değerini PassedPoints kümesine ekle
          end if
        end if
      end for
      foreach i not in PassedPoints do
         j=getDepotIndexByMinimalDjkValue(i)
         i istekçi düğümünü j deposuna ata
      end foreach
     {Şu anda her bir istekçi düğüm uygun bir depoya atanmış
     durumdadır}
      for i =1 to D do //tüm depolar
        DemPointsOfDepot[i] oluştur//i deposuna atanmış olan
          istekçi düğümlerin listesi
        Route=CreateRouteOfVehicle(DemPointsOfDepot[i],
          VehicleList[i],VehicleIndex[i])//i deposunun
          VehicleIndex[i]. taşıtının rotası
        if (Route boştursa) and (DemPointsOfDepot(i)
          listesinde herhangi bir rotaya atanmamış istekçi
          düğümler varsa) then
```

Şekil 3.1: GÇDTYP algoritmasının sözde-kodu

```
            DemPointsOfDepot[i] listesindeki ilk istekçi
            düğümün isteğini iki kısıma böl
                a: birinci kısım VehicleIndex[i]. taşıtın
                 kapasitesine eşittir.
                b: ikinci kısım toplam istekden birinci kısımı
                 çıkardıkta geri kalan istek kısımına eşittir.
            Birinci kısımı VehicleIndex[i]. taşıtın rotasına bu
            şekilde ekle : taşıt i deposundan kalkıyor, ilk
            istekçi düğüme tüm kapasitesi ile hizmet ediyor ve
            geriye aynı depoya dönüyor.
          end if
          VehicleIndex[i]= VehicleIndex[i]+1;//sonrakı devirde
            başka taşıt kullan
          if (VehicleIndex>Length(VehicleList[i])then
            VehicleIndex=1;//sonrakı devirde birinci taşıtı
            yeniden kullan
          VehicleIndex[i]. taşıtın rotasına atanmış tüm istekçi
          düğümleri PassedPoints kümesine ekle
          (mi istek çeşidini, i depo indeksini, VehicleIndex[i]
          ve Route değerlerini) RouteList listesine ekle
        end for
      end while
    end for
    return RouteList
end function

function
findFirstMinimalDiValueAndItsDepotIndex(demandPointIndex)
 d(i)formülüne en küçük değeri veren i depo indeksini bul:
  d(i)=vc·dis+mc·ma
  vc –i deposunun uygun VehicleList[i] listesinden seçilmiş
     VehicleIndex. taşıtı
  dis – i depose ile demandPointIndex istekçi düğümü
     arasındakı mesafe
  mc – i deposunun mi malzeme çeşidinin satış fiyatı
  ma – i istekçi düğümünün malzeme isteği
return d(i),i
end function

function
findSecondMinimalDiValueAndItsDepotIndex(demandPointIndex)
 findFirstMinimalDiValueAndItsDepotIndex fonksiyonu ile
 aynıdır, sadece ikinci en küçük değeri veren d(i) ve i
 değerlerini geri döndürür.
end function

function getDepotIndexByMinimalDjkValue(i)
 ActiveDepots listesindeki tüm depolar içinde herhangi bir
 depoya atanmış her bir (j, k) istekçi düğümler cifti için
 djk(i) formülüne en küçük değeri veren depotindex depo
 indeksini bul,
  djk(i)=vc·(disij+disik-disjk)
  vc – j ve k istekçi düğümlerinin atanmış olduğu deponun
```

Şekil 3.1 (devam): GÇDTYP algoritmasının sözde-kodu

```
  VehicleIndex. taşıtı
  dis_ij – i ve j istekçi düğümleri arasındakı mesafe
return depotindex
end function

function
CreateRouteOfVehicle(DemPointsOfDepot[i],VehicleList[i],
VehicleIndex[i])
 DemPointsOfDepot[i] kümesini ve i deposunun VehicleIndex[i].
 taşıtının özelliklerini kullanarak Değiştirilmiş Clack–Wright
 Savings algoritması ile aynı taşıtın rotasını hesapla
return Route
end function
```

Şekil 3.1 (devam): GÇDTYP algoritmasının sözde-kodu

3.3. Değiştirilmiş Clack – Wright Savings Algoritması[30]

Ortamda N tane alıcı düğüm olduğunu düşünelim. Alıcı düğümlerin istek miktarı Q_i (i = 1, 2, . . . , N) değerine eşittir(istekler aynı çeşitlidir ama farklı miktarda olabilirler). İstekler satıcı D düğümü tarafından ödenilir.

Bu problemin amacı, satıcı D düğümünden başlayıp farklı alıcı düğümlerden geçen taşıt için geçtiği yolların toplam uzunluğunu en aza indirgeyen rota bulmaktır. Burada alıcı düğümlerin istek miktarlarının Q_i (i = 1, 2, . . . , n) taşıtın kapasitesini geçmemesi zorunluluğu vardır. Taşıt kapasitesine uygun olarak alıcı düğüm altkümesine hizmet edebilir.

Algoritma bu şekilde çalışır:

1. ∀(i,j) alıcı çifti için Tasarrufları (Savings) hesapla

   $s(i,j) = dis_{Di} + dis_{Dj} - dis_{ij}$

   ve azalan sırada sırala. Bu "Tasarruflar Listesini"(Savings List) oluşturur. Bu listeni en üstteki elemanından(en büyük s(i,j) tasarrufu) başlayarak işleme tabi tut.
2. Eger carı s(i,j) tasarrufu için rota kısıtlamalarının (taşıtın rotası üzerinde bulunan alıcı düğümlerin istek miktarlarının toplamı taşıtın kapasitesini geçmiyorsa, i ve j alıcı düğümleri arasında yol varsa, çünkü i ve j düğümleri arasında direkt yol olmadıkta bu düğümler birleştirilmemelidir) hiçbir ihlali yoksa o zaman (i,j) yolunu rotaya dahil et ve eger:
   a. ∀i, j alıcı düğümleri ve R rota kümesi için, $i \notin R \wedge j \notin R$ ise o zaman i ve j alıcı düğümlerini içeren yeni bir rota oluştur.

b. ∀i, j alıcı düğümleri ve R rota kümesi için, $i \notin R \wedge j \notin R$ ise ve bu düğüm bu rotada iç düğüm değilse (bir düğümün dahil olduğu rotada iç düğüm olması için rota üzerinde bu düğümden D satıcı düğüme direkt yol olmaması gerekir), bu durumda (i,j) yolu aynı rotaya ilave olunur.

c. ∀i, j alıcı düğümleri farkli rotalara atanmış durumdadır ve hiç biri iç düğüm değilse, bu zaman rotalar birleştirilir.

3. Eger Tasarruflar Listesi bitmemiş ise bu zaman 2. Adıma geri dön ve listeden bir sonrakı kayıtı işleme tabi tut, aksi halde dur. Taşıt Yönlendirme Probleminin çözümü 2. Adımda hesaplanmış rotalardan oluşur. (2. Adımda rotaya atanmamış her bir alıcı düğüm için rotası D satıcı düğümünden başlayıp bu alıcı düğüme uğrayan ve geri D satıcı düğümüne dönen taşıtlar ile hizmet verilir.)

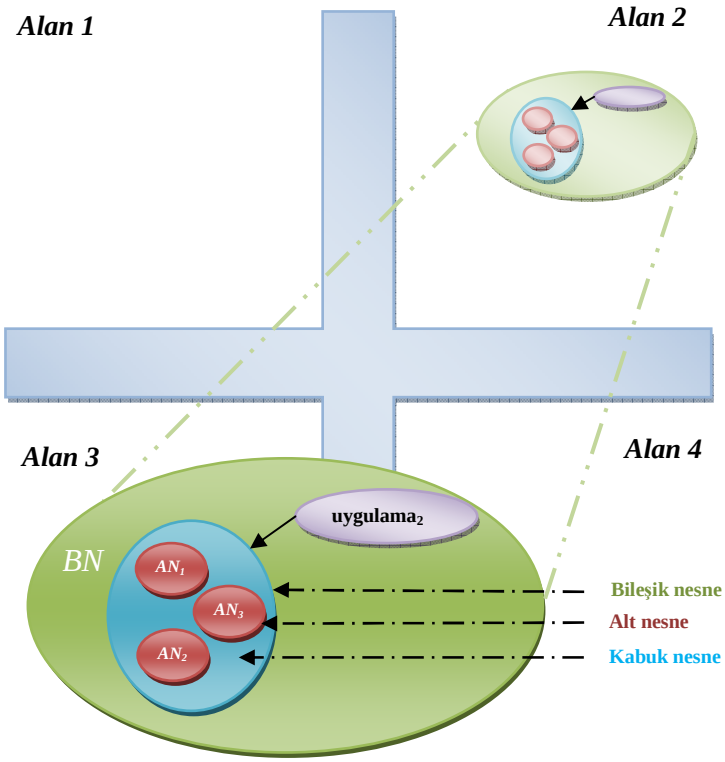
4. DBN MODELİ VE DBNTO ORTAMI

İşbirliğiyle yürütülen dağıtılmış uygulamalar genel olarak iri ya da orta taneli nesneler üzerinde çalışmalarına rağmen, bu tip uygulamaların kullanıcıları genelde nesnelerin daha küçük bir parçasıyla ilgilenmektedirler. Bundan dolayı, eğer büyük bir nesne daha küçük alt nesnelere parçalanabilirse ve istekçi alana nesnenin yalnızca kullanılacak olan ilgili parçası gönderilirse, erişilecek ve farklı alanlar arasında aktarılacak olan verinin miktarı azalacağından, çalışan uygulamanın başarımı da yükselecektir. Kopyalamaya dayalı yeni bir model olan DBN modeli bu noktadan yola çıkarak geliştirilmiştir. Bu modelinin ardında yatan temel fikir, çok sayıdaki alt nesnenin bileşiminden meydana gelen bir bileşik nesnenin farklı alanlar üzerinde fiziksel olarak dağıtılmasıdır.

4.1. DBN Modelinin Haberleşme Şekli ve Nesne Yapısı

Önerilen modelde, dağıtılmış ortamdaki uygulamalar bileşik nesneler üzerinden haberleşirler ve birbirleri ile etkileşimde bulunurlar. Bu amaçla, her bir paylaşılan nesne bir arayüz üzerinden kullanılabilen bir metotlar kümesi sunar. Nesneler pasiftir. Etkinlik, nesneleri paylaşan ve onların metotları üzerinde eşzamanlı olarak çağrı yapabilen uygulama programları tarafından sağlanır.

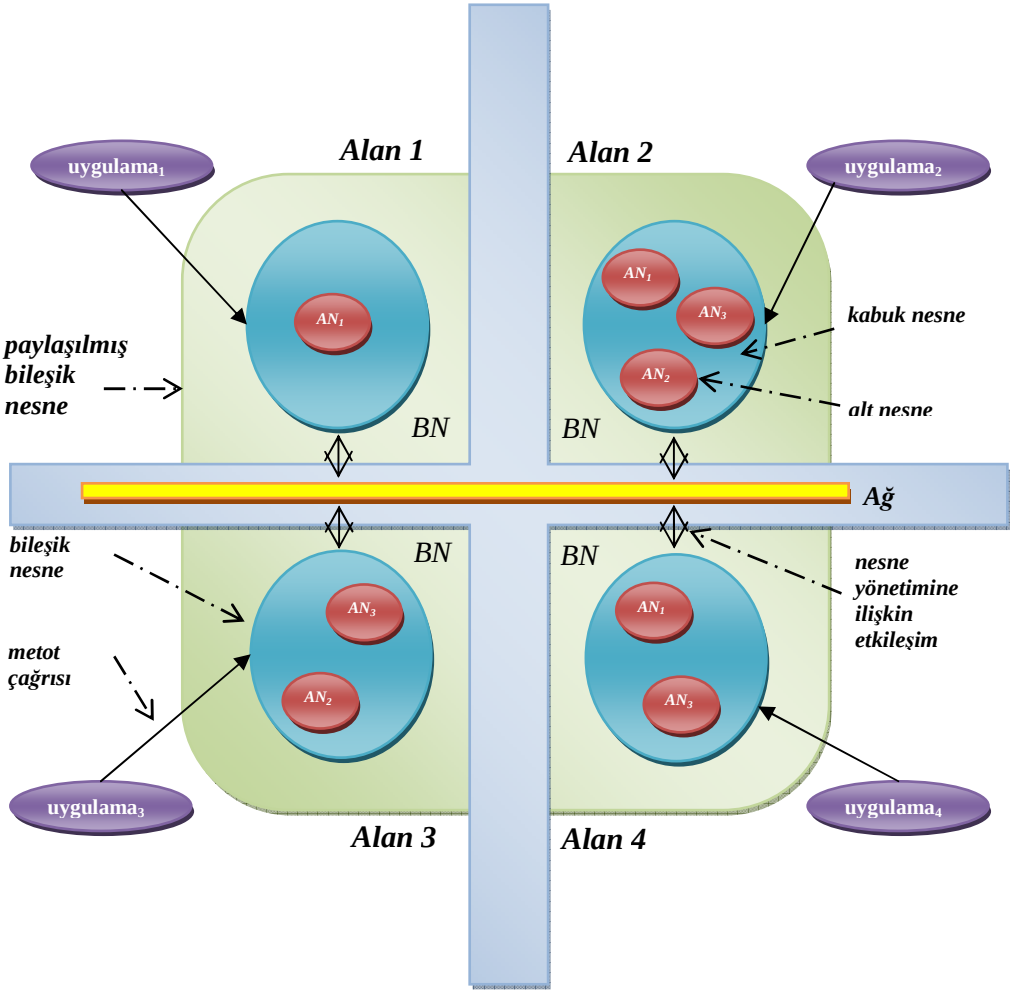
Modele göre, bir dağıtılmış bileşik nesne, Şekil 4.1'de sunulduğu gibi, öncelikle tekil bir adres alanı üzerinde bir veya daha fazla sayıda *alt nesne (AN)*'nin bir kabuk nesne altında birleştirilmesi suretiyle bir *bileşik nesne (BN)* olarak yaratılır. Bileşik nesne diğer adres alanları üzerine, yalnızca o alanlarda "çalışan uygulamaların gerektirdiği alt nesneler" ile birlikte kopyalandıktan sonra, dağıtılmış bileşik nesne yapısı oluşur.



Şekil 4.1: Tekil bir adres alanında üç alt nesnesi ile birlikte yaratılmış bileşik nesne

4.2. DBN Modelinde Nesnelerin Alanlar Üzerinde Paylaşımı

Şekil 4.1'de, Alan 2 üzerinde üç alt nesnesi ile yaratılan BN bileşik nesnesi diğer adres alanlarında bulunan uygulamaların yürüttükleri metot çağrıları sonucu o alanlar üzerinde, yalnızca metot çağrısının gerektirdiği alt nesneleri ile birlikte kopyalanır ve böylece, Şekil 4.2'de görülen nesne yapısı oluşur. Bu yapı incelendiğinde, Alan 1'de bulunan uygulamanın BN üzerinde yürütmüş olduğu metot çağrıları sonucunda BN bu alan üzerine yalnızca AN_1 alt nesnesi ile kopyalanmıştır. Benzer şekilde, Alan 3'de bulunan uygulamanın yürütmüş olduğu metot çağrıları sonucunda AN_2 ve AN_3 alt nesneleri ile birlikte ve Alan 4'de bulunan uygulamanın yürütmüş olduğu metot çağrıları sonucunda da AN_1 ve AN_3 alt nesneleri ile birlikte kopyalanmıştır. Bu suretle ortaya bir "dağıtılmış bileşik nesne" yapısı çıkmıştır.



Şekil 4.2: Dört farklı adres alanı üzerine yayılmış bir dağıtılmış bileşik nesne

4.3. DBN Modelinde Nesne Kontrolü Ve Saydamlık

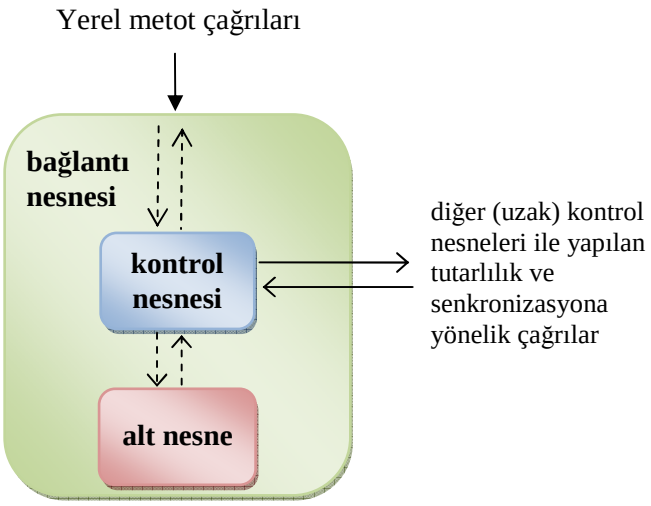
DBN modelinde, dağıtılmış bileşik nesneyi oluşturan alt nesnelerin farklı alanlar üzerinde rahatlıkla bağlanabilmesi amacıyla bir bağlantı nesnesi ve alt nesne kopyalarının senkronizasyon ve tutarlılığının sağlanabilmesi amacıyla da bir kontrol nesnesi her bir alt nesnenin önüne eklenmiştir. Bağlantı nesnesi ile dinamik bağlanma sağlanırken, kontrol nesnesi ile, dağıtılmış bileşik nesnenin alt nesneleri bazında düzenlenebilen çeşitli tutarlılık ve senkronizasyon mekanizmaları kurulabilmektedir. Ayrıca, dağıtılmış bileşik nesneyi oluşturan alt nesneler ve bunların yapılarını yalnızca bu nesnenin tasarımcısı bilmektedir. Tüm diğer istekçiler, nesne hakkında kendilerine sunulan istekçi arayüzünün haricinde başka bir bilgiye sahip değildirler. Bu istekçiler aynı zamanda bir metot çağrısı yürüttüklerinde bir alt nesnenin kendi alanlarına yüklenip/yüklenmediğini dahi bilmemektedirler.

Bir DBN'nin iki farklı görünüş şekli vardır: DBN'yi kullanan uzak uygulamaların bakış açısına göre; o diğer nesnelerden farkı olmayan tekil paylaşılan bir nesnedir. Farklı adres alanlarında bulunan çok sayıdaki uygulama tarafından paylaşılmaktadır. Bu nesneye programcı tanımlı bir arayüz üzerinden erişilir. Bileşik nesneyi oluşturan alt nesneler ve özellikle de sisteme nasıl dağıldıkları görünemez. Bileşik nesnenin tasarımcısının bakış açısına göre ise; bir DBN birbirleriyle birlikte çalışan bir alt nesneler kümesinden oluşmaktadır. Bu yapıdaki her bir alt nesne temel bir nesnedir. Diğer bir deyişle, her bir alt nesne merkezi bir gösterime sahiptir. DBN'nin bir parçasını gerçekleyen her bir alt nesne başlangıçta tekil bir adres alanına yerleştirilir. Daha sonra, çalışma içerisinde farklı adres alanlarında kopyalanan alt nesneler, DBN'nin tutarlı bir görüntüsünü oluşturmak için diğer adres alanındaki alt nesnelerle haberleşirler.

4.4. Bir Bileşik Nesnenin İç Yapısı

Amaçlanan DBN yapısının, bir önceki bölümde ele alındığı şekliyle, sadece çok sayıdaki alt nesnenin bir kabuk nesnesi içerisine yerleştirilmesi suretiyle elde edilemeyeceği açıktır. Bir alt nesnenin gerektiğinde bir diğer adres alanı üzerinde kopyalanarak o alana daha önce taşınmış olan kabuk nesneye bağlanması gerekir. Ayrıca, farklı alanlar üzerinde kopyalara sahip olan bir alt nesneye, nesnenin bütünlüğünü bozmaksızın çağrıda bulunulması ve bir alt nesne kopyasının güncellenmesi durumunda diğer kopyaların da o kopya ile tutarlılığının sağlanması gerekir. İşte bu sebeplerden dolayı nesne yapısına bazı eklentilerin yapılması zorunlu olmaktadır. Çözüm olarak, bileşik nesnenin yaratıldığı kabuk nesnesi ile her bir alt nesne arasına, "bağlantı" ve "kontrol" nesneleri olarak adlandırılan iki ara nesne eklenmiştir.

Bir bileşik nesneyi oluşturan her bir alt nesnenin önüne eklenen bağlantı ve kontrol nesneleri ile birlikte, Şekil 4.2'de yer alan her bir AN_i nesnesi aslında Şekil 4.3'teki gibi bir üçlü bir nesne grubu şeklinde gerçeklenmektedir.



Şekil 4.3: Alt nesne erişim düzeni

Bağlantı ve kontrol nesneleri bir bileşik nesnenin dağıtılmış bileşik nesne olarak kullanılabilmesini sağlayan ara nesnelerdir. Dinamik bağlama, senkronizasyon ve tutarlılığı sağlama gibi karmaşık ve gerçeklemesi zor olan dağıtılmış sistemle ilgili ayrıntıları kullanıcıdan gizleyen bu nesneler, ilgili alt nesnelerin arayüz tanımlamaları kullanılarak otomatik olarak üretilirler.

4.4.1. Bağlantı Nesnesi

Herhangi bir DBN üzerinde bir metot çağrısı yürütüldüğünde, çağrının gerektirdiği alt nesnenin, eğer daha önceden yüklenmemiş ise, o adres alanına yüklenmesi gerekmektedir. Eğer çağrı DBN'nin yaratıldığı alandan yapılmış ise, zaten tüm alt nesneler bu alana yereldirler ve herhangi bir uzaktan yüklenme işlemi gerekmez. Ancak, DBN için bir arama işlemi yürüterek kabuk nesneyi kendi adres alanına bağlamış olan bir uzak uygulamanın bir metot çağrısı yürütmesi durumunda yüklenme gereği ortaya çıkar.

Bir alt nesnenin yürütme anında dinamik olarak istekte bulunulan adres alanı üzerine yüklenebilmesini sağlamak amacıyla *bağlantı nesnesi* olarak adlandırılan ve kabuk nesne ile kontrol nesnesi arasında yer alan bir ara nesne kullanılır. Eğer kontrol nesnesi o adres alanı üzerinde bulunuyor ise, bağlantı nesnesi içerisinde bulunan nesne referansı kontrol nesnesine işaret eder. Eğer referans değeri *null* ise, bu alt nesnenin henüz o adres alanına yüklenmediğini gösterir. Bu durumda, bağlantı nesnesi DBNTO sistem çağrılarını kullanarak yükleme işlemini gerçekleştirir. Tüm alt nesne erişimleri bağlantı nesnesi üzerinden sağlandığı için, bağlantı nesnesi alt

nesnenin sunduğu arayüzün aynısını sunmaktadır. Yüklenme işlemi tamamlandıktan sonra, metot çağrısı bir yerel çağrı şeklinde yürütülür.

4.4.2. Kontrol Nesnesi

Bir alt nesnenin farklı alanlar üzerindeki kopyalarına birden fazla uygulamadan eşzamanlı olarak metot çağrıları gelebilir. Eğer gelen bu çağrılar birbirleri ile *çelişmeyen (non-conflicting)* çağrılar ise (örneğin, hepsi alt nesnenin durumu üzerinde okuma işlemi gerçekleştirecek olan çağrılar ise), problem yoktur. Ancak, çağrılardan en az biri alt nesnenin durumu üzerinde güncelleme işlemi yürütecek bir çağrı ise, bu durumda *çelişme (conflict)* söz konusudur ve alt nesnenin bütünlüğü bozulabilir. Bu yüzden, farklı alanlardaki kopyalara yönelen eşzamanlı metot çağrıları birbirleri ile çelişiyor iseler, ardışıl olarak yürütülmelidirler. Ayrıca, yapılan bir metot çağrısı o alanda bulunan alt nesne kopyasını diğerlerinden farklı kıldıysa, bu durumda kullanılan tutarlılık protokolüne göre, diğer kopyalarında güncellenmesi veya geçersiz kılınmaları gerekmektedir. İşte bu iki sebepten dolayı alt nesnenin hemen önüne sözü edilen senkronizasyon ve tutarlılık işlemlerini gerçekleyecek ve yapılan tüm erişimleri denetleyecek bir *kontrol nesnesi* yerleştirilir.

4.5. DBN Modelinin Avantajları

Dağıtılmış ortamdaki nesnenin alt nesnelere bölünüp, fiziksel olarak farklı adres alanlarında kopyalanmasını öngören DBN modeli ile kazanılan yararlar şu şekilde sıralanabilir:

- Kopyalama sonucu hataya hoşgörülü sistemlerin oluşturulabilmesi.
- Yalnızca alt nesnelerin kopyalanması sonucu, ağda taşınan veri miktarının azalması.
- Yalnızca hedeflenen alt nesnelerin kopyalanması sonucu, verimli bellek kullanımı.

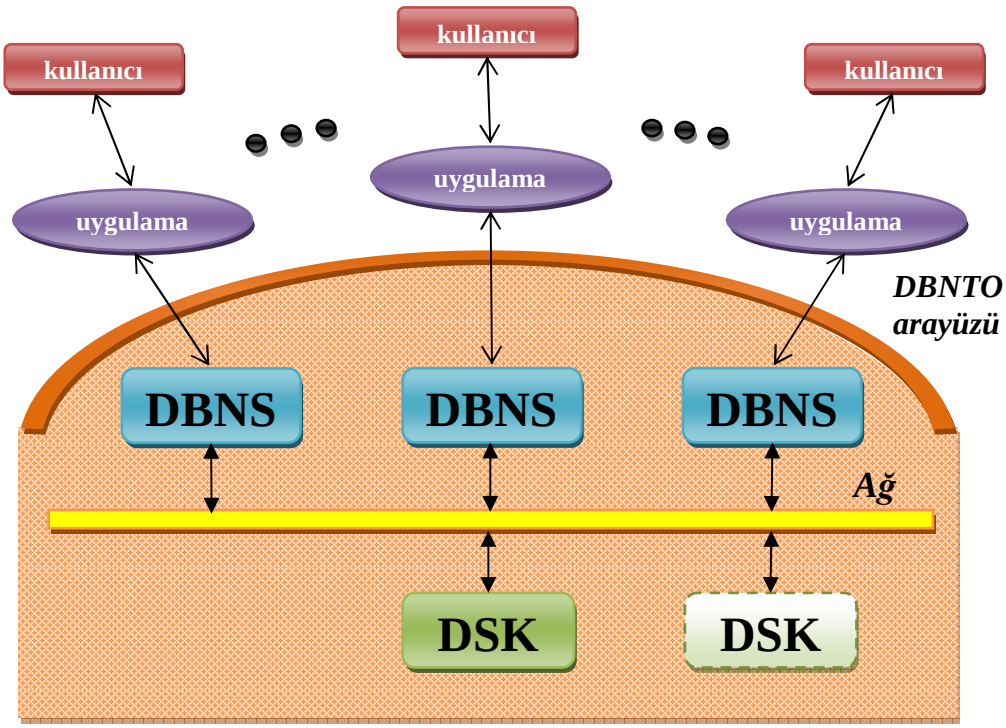
4.6. Dağıtılmış Bileşik Nesne Tabanlı Ortam (DBNTO)

Dağıtılmış bileşik nesne tabanlı ortamın tasarımında, nesne bileşimi yönteminden yola çıkarak, mevcut Java nesne modelinin[31] kopyalanarak fiziksel olarak dağıtılmış paylaşılan nesnelere genişletilmesi için gerekli alt yapının nasıl geliştirilebileceği konusu üzerinde yoğunlaşılmıştır. Özellikle, internet üzerinde

dağıtılmış müşterek grup çalışmalarının dağıtılmış bileşik nesneler kullanılarak kolaylıkla gerçeklenebilmesini sağlayacak uygun mekanizmalar aranmıştır.

Dağıtılmış bileşik nesne modelini kullanan bir müşterek uygulamanın genel yapısı Şekil 4.4'te görülmektedir. Bu yapıda, dağıtılmış ortamda müşterek çalışmalar yürüten kullanıcılara ait uygulamalar DBNTO arayüzü üzerinden dağıtılmış bileşik nesneleri paylaşırlar.

Burada yer alan her bir uygulama, Dağıtılmış Bileşik Nesne Sunucusu (DBNS) olarak adlandırılan bir sistem birimi aracılığıyla dağıtılmış bileşik nesne ortamına erişir. Ayrıca, DBNS'lerin ortama giriş/çıkış işlemlerini denetleyen, onlara sistem çapında tekil bir tanımlayıcı atayan ve yeni yaratılan bir DBN'yi oluşturan her bir alt nesne için tekil bir nesne tanımlayıcısı sağlayan, bir DBNTO Sistem Koordinatörü (DSK) yer alır.



Şekil 4.4: DBNTO sisteminin genel yapısı

Sistemin hata-hoşgörü oranını arttırmak için, Şekil 4.4'te kesikli çizgilerle gösterilmiş olan ikinci bir DSK, yedek sistem koordinatörü olarak görev yapmaktadır. Herhangi bir şekilde asıl DSK'nın devre dışı kalması durumunda, yedek DSK üzerinden sistem problemsiz bir şekilde çalışmasına devam edebilmektedir.

4.6.1. DBNTO Sisteminin Temel Fonksiyonları

DBNTO sistemi kullanıcılarına şu temel fonksiyonları sunmaktadır:

- **İnternet uygulamaları için destek sağlama:** DBNTO özellikle, internet üzerine yayılmış dağıtılmış nesne-tabanlı uygulamaların geliştirilmesi için tasarlanmış olan bir *arakatman (middleware)* yazılımıdır.
- **Dağıtılmış bileşik nesne modeli:** DBNTO, DBN nesne modelini esas almaktadır. DBN'nin tasarımındaki temel fikir, bir bileşik nesnenin alt nesnelerinin kopyalanması suretiyle fiziksel olarak dağıtılmış olmasıdır. Diğer bir deyişle, nesnenin durumu, aynı anda farklı adres alanları üzerine yerleştirilmiş birçok alt nesnenin durumlarından oluşmaktadır.
- **Kopyalama:** Dağıtılmış ortamda birlikte yürütülen uygulamalar için başarımı etkileyen en önemli faktörlerden biri de kopyalamadır. DBNTO, dağıtılmış bileşik nesnenin paylaşılan alt nesnelerinin birlikte iş yürütmekte olan farklı alanlar üzerinde kopyalanmasına izin verir. Böylece, dağıtılmış nesneler üzerinde metot çağrıları, uzak-nesne modelinden farklı olarak, birer yerel çağrı şeklinde yürütülürler ve bu şekilde iletim gecikmesi süresi de azaltılır. Ayrıca, kopyalama sırasında bileşik nesne yeni alan üzerinde tümüyle kopyalanmaz. Hangi alt nesnelerin ilgili alan üzerinde kopyalanacağını o alanda yürütülen metot çağrılarının şekli belirler. Yani, her bir alt nesne, ihtiyaç duyuldukça yeni alan üzerinde kopyalanır.
- **Tutarlılık:** Bir dağıtılmış bileşik nesnenin farklı alanlar üzerinde kopyalanmış olan alt nesnelerinin tutarlılığının sağlanması amacıyla tasarımcıya iki farklı yöntem sunulmaktadır. Tasarımcı bir alt nesne için, nesnenin özelliklerini göz önünde bulundurarak, *yazma-güncelleme (write-update)* ya da *yazma-geçersiz kılma (write-invalidate)* yöntemlerinden birini seçer. Yazma-güncelleme yöntemine göre, bir alt nesne kopyasının durumu üzerinde değişiklik yapıldığında, diğer kopyalar da aynı şekilde güncellenir. Yazma-geçersiz kılma yöntemine göre ise, tüm diğer kopyalar geçersiz kılınırlar. Daha sonra, geçersiz bir kopyaya sahip bir alandan nesne erişim isteği yapıldığında, son güncel değer ilgili alandan alınmak suretiyle, erişim gerçekleştirilir.
- **Adlandırma ve bağlama hizmeti:** DBNTO sistemi, dağıtılmış bileşik nesnelerin adlandırılması ve yerleşkelerinin bulunması amacıyla da bir yöntem sunmaktadır.

Her bir bileşik nesne kullanıcı tanımlı bir ad ile kaydedilir ve bir bileşik nesneyi kendi adres alanına yüklemek isteyen bir kullanıcı yürüteceği bir arama komutu ile bileşik nesneyi ifade eden kabuk nesneyi kendi adres alanına yükler. Daha sonra, yapılan bir metot çağrısı için gerekli olan alt nesne ya da nesnelerin de o adres alanı üzerinde bir kopyası oluşturulur.

- **Dinamik yayılma:** Bir DBN erişim isteğinde bulunulan düğümlere, kendisine sahiplik eden düğümden (gerekli olan alt nesneleri ile birlikte) yürütme sırasında dinamik olarak kopyalanır. Böylece, kullanıcılar uygulamayı çalıştırmaya başlamadan önce, paylaşılan nesnelere ilişkin herhangi bir bilgiyi kendi alanlarına yüklemek zorunda kalmazlar.
- **Dinamik uyarlama:** Büyük ölçekli geniş alana yayılmış olan bir uygulama dinamik olarak yeni bileşenlerin eklenmesi ve var olanların çıkarılabilmesine olanak tanımalıdır. DBNTO, mevcut durumda nesneye bağlantı sağlamış olan istekçileri etkilemeksizin, DBN'ye yeni bir alt nesne eklenmesine, var olanların çıkarılmasına ya da değiştirilmesine imkan tanır.
- **Saydamlık:** DBNTO sisteminde, uygulamanın hem geliştirilme, hem de yürütülme aşamalarında dağıtımın getirdiği problemlerin etkisinin en aza indirilmesi amaçlanmıştır. Dağıtılmış ortamdaki çok sayıdaki kullanıcı tarafından ortaklaşa yürütülen bir uygulama merkezi bir yapıda çalışılıyormuş gibi geliştirilebilir. Dağıtım, senkronizasyon ve tutarlılık ile ilgili konular uygulama kodundan bağımsız olarak çözümlenir. Bu aynı zamanda, mevcut merkezi uygulamaların dağıtılmış ortamda çalışabilmelerini de mümkün kılar.

Dağıtılmış bileşik nesne yapısı özellikle internet üzerinden veri paylaşımına izin veren bir çok dağıtılmış sistem uygulamalarında etkinlikle kullanılabilecek bir nesne yapısıdır. Bir sonrakı bölümde bu özellikleri kendinde bulunduran güncel bir lojistik uygulamasının tasarım aşamaları anlatılacaktır.

5. LOJİSTİK UYGULAMA ORTAMININ TASARIMI VE GERÇEKLENMESİ

İnternet üzerinden çalışan lojistik uygulamanın geliştirilmesinin amacı, coğrafik olarak birbirlerinden uzak birden fazla lojistik firma çeşitleri arasında ortak bir haberleşme ve yönetim ortamı sunmaktır. Güncel bir lojistik uygulama paylaşımlı ortamda dinamik yönetilebilme, dayanıklılık, ölçeklenebilirlik, isteklerin hızlı yanıtlaması ve lojistik maliyeti minimize etmeğe yönelik optimum çözümler sunabilme gibi birçok özellikleri kendisinde bulunduracak şekilde tasarlanmak durumundadır[32]. Bölüm 4'de incelenen DBNTO sistemi böyle bir dağıtılmış ortam için gerekli alt yapıyı sunmaktadır. Lojistik uygulama için bu sistemin seçilmesindeki amaç daha önce anlattığımız özellikleri taşıyan bir uygulama ortamına sahip olmaktır. Lojistik ortamdaki malzemeler ve taşıtlar bir nesne olarak düşünüldüğünde, onların aslında tekil bir büyüklük olmayıp, çok sayıdaki birbirinden bağımsız bölümlerden ve her bir bölümünde kendi içinde alt bölümlerden oluştuğu parçalı bir yapıya sahip olduğu görülür. Bu parçalı görüntü, onların bir "dağıtılmış bileşik nesne" olarak modellenebilmesinin mümkün olabileceğini göstermektedir.

Bu amaçla, malzemelerin ve taşıtların her bir özelliği çeşitleri, satış fiyatı, taşıt kapasitesi, maliyeti bu alt nesnelerin bir araya gelmesinden oluşan bir bileşik nesne olarak gerçeklenmiştir. Böylece, kullanıcı uygulamaların tek bir nesneye erişiyor görüntüsü altında, aslında yalnızca kendilerini ilgilendiren alt nesneye erişmeleri ile hem birden fazla uygulamanın aynı anda gerekli bölümleri üzerinde çalışabilmeleri, hem de iletim sırasında ağ üzerinden aktarılacak veri miktarının daha az olması sağlanmıştır. Paralelliğin artması ve ağ iletim sürelerinin kısalması olarak kazanılan bu avantajlar, doğal olarak çalışma ortamının başarım ve erişilebilirliğin de artmasını sağlamıştır.

5.1. Lojistik Uygulama Ortamında DBNTO Sisteminin Kullanım Avantajları

Burada DBNTO sisteminin lojistik uygulama tasarımına izin veren bazı önemli özellikleri açıklanmıştır:

- DBNTO sistemi *bileşik nesne yönetimini* desteklemektedir. Bu özellik bize lojistik uygulamada yer alan taşıtları, malzemeleri ve onların birtakım niteliklerini birer nesne olarak modelleme olanağı sunar. İstekte bulunan düğümlere yönelik hesaplamalarda sadece gerekli bileşik nesnelerin internet ortamında iletilmesinin mümkün olması veri yoğunluğunda azalma, paralel olarak ağ yükünün düşmesi ile daha az verinin hızlı şekilde gönderilebilmesi avantajını sağlar.
- DBNTO sistemi *İnternet uygulamaları için destek sağlar* ve bu özellik lojistik uygulamanın yalnız yerel ortamda değil aynı zamanda internet ortamında da çalışmasına izin verir.
- DBNTO sistemi *"yazma-güncelleme"* ve *"yazma-geçersiz kılma"* olarak adlandırılan iki tutarlılık protokolü desteklemektedir. Sistemde bu protokol tiplerinden *"yazma-güncelleme"* kullanıldığında, sunuculardaki herhangi bir nesnede değişiklik olursa, bu nesnenin sistemdeki tüm kopyaları bu değişikliği yansıtacak şekilde eşzamanlı olarak güncellenir. *"yazma-geçersiz kılma"* protokolü kullanıldığında ise sistemdeki herhangi bir nesnedeki değişiklik bu nesnenin kopyalarını kullanan tüm sunuculara bildirilir; sunucu ancak gerek duyulduğunda bu nesnenin en güncel şeklini uygun adresten yükler.
- DBNTO sisteminin *dinamik uyarlama* özelliği vardır. Bu özellik geniş alana yayılmış lojistik uygulama ortamına dinamik olarak yeni arayüzlerin ve onların bileşenlerinin eklenmesine, var olanların çıkarılabilmesine olanak tanır.

Bu özellikler bize lojistik uygulamada optimal değerleri hesaplarken, kullanıcı arayüzler arasında bilgi transferi zamanı düğümlerdeki (sunuculardaki) verilerin en son hallerine hızlı bir şekilde ulaşmak imkanı tanır sağlar. Bu nedenle DBNTO hız artımında ve sistem kaynaklarının daha az kullanılmasında önemli katkılar sağlamaktadır. Lojistik ortamın tasarımında nesne tabanlı yaklaşım kullanmak ise başka bir avantajdır[33].

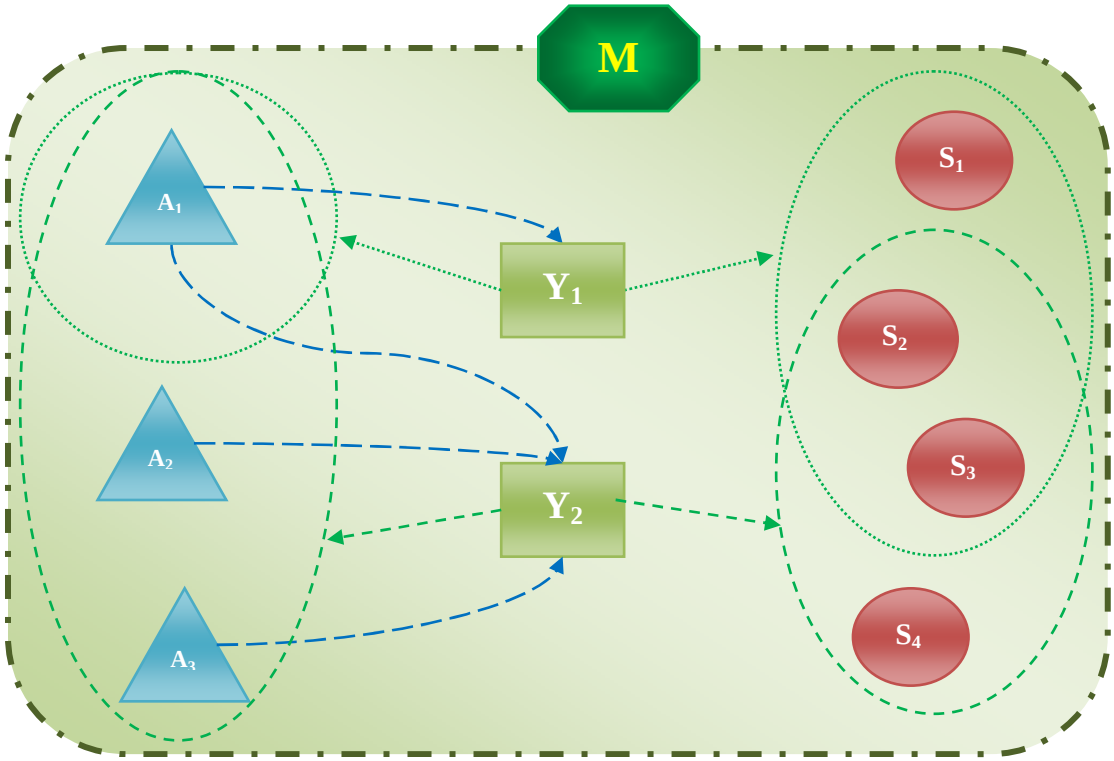
5.2. Lojistik Uygulama Ortamının Genel Görünümü

Lojistik uygulamanın DBNTO ortamı üzerinde tasarımı yapılarken bu ortamın yapısına uygun ve uygulama performansını en yüksek dereceye taşıyacak bir yol izlenmesi gerekir.

Lojistik uygulamanın çalışma ortamı DBNTO sistemidir. DBNTO sisteminin yapısından anlaşılacağı gibi bir düğüm üzerinde DBNTO Sistem Koordinatörü (DSK) ve diğer düğümler üzerinde DSK'nın koordine ettiği Dağıtılmış Bileşik Nesne Sunucuları (DBNS) vardır. Uygulamanın tasarımında DBNS'ler düğümlerde çalışan farklı kullanıcı uygulamaları, DSK ise tüm sistemi kontrol eden ve merkezi bir düğümde çalışan sistem birimini temsil ediyor. Mutlak gereksinim olmamakla beraber DSK kullanıcı programlarından her hangi birinin çalıştığı düğüm üzerinde yerleştirilebilir (biz farklı düğümde çalışan versiyonunu inceleyeceğiz).

Burada biz Bölüm 2.1 de anlatılan ve Şekil 2.2 üzerinde resmedilen lojistik ortamın DBNTO sistemi kullanılarak tasarımını yapacağız.

Şekil 5.1 de sistem koordinatörü vazifesini üstlenen M merkezi düğümü resmedilmiştir. Diğer tüm düğümlerde kullanıcı uygulamalar çalışmakta ve ağ (internet) üzerinden M merkezi düğümü ile bağlantıları bulunmaktadır. Şekildeki kesik-noktalı çizgi tüm düğümlerin M merkezi düğümünün kontrolü altında olduğunu gösterir. Burada Y_1, Y_2 alıcı ile satıcı firmaları yöneten aracı firma kullanıcı uygulamalarını, S_1, S_2, S_3, S_4 satıcı firma kullanıcı uygulamalarını, A_1, A_2, A_3 alıcı firma kullanıcı uygulamalarını göstermektedir. Firmalar arasındakı anlaşmalar Bölüm 2.1 de anlatıldığı gibidir.



M-merkezi düğüm
Y_i – aracı , S_i – satıcı, A_i– alıcı firma uygulamaları

Şekil 5.1: Lojistik uygulama ortamında kullanıcı uygulamaların haberleşme yapısı

5.3. Uygulamanın Gerçekleme Ayrıntıları

İlerleyen bölümlerde, daha önce anlattığımız lojistik uygulama ortamının tasarımını temel alarak, Java programlama dili ile geliştirilmiş lojistik uygulamanın gerçekleme adımları anlatılmıştır. Farklı düğümlerde çalışacak olan kullanıcı uygulamaları için DBNTO sistemine uygun yapının oluşturulması, kullanıcılara sunulan arayüzlerin gerçeklenmesi aşamaları incelenmiş ve onlar arasındaki etkileşim adımları gösterilmiştir.

Lojistik uygulama ortamında, her bir düğümde yer alan farklı malzeme ve taşıt çeşitlerinin her biri ait olduğu düğümün birer parçası olarak düşünülmüş ve bu parçalar birer alt nesne olarak tasarlanmıştır. Daha sonra, bu alt nesneleri bir kabuk nesne içinde birleştiren, her bir düğüm çeşidine uygun bir bileşik nesne sınıfı oluşturulmuştur. Uygulama ortamında satıcı, aracı ve alıcı gibi üç düğüm çeşidi olduğundan, bunların her biri için tasarlanmış SatıcıDüğüm, AracıDüğüm, AlıcıDüğüm isimli bileşik nesne sınıflarının her biri ayrı-ayrılıkta incelenmiştir. Başlangıçta kullanıcı uygulamaların çalıştığı her bir düğüm için uygun sınıfından bir bileşik nesne türetilmekte, gereken alt nesneler bu bileşik nesneye eklenmekte ve Daha sonra, bu kullanıcı arayüzler kullanılarak bileşik nesneler arasında gereken

işlemler yürütülmektedir. Yani, çalışma sırasında bileşik nesnelere yeni alt nesneler eklenip, çıkarılmaktadır.

5.4. Kullanıcı Uygulamalarda Bileşik Nesne Yapısının Kullanımı

Bu bölümde, kullanıcı uygulamaların çalıştığı her bir düğüme uygun bileşik nesnelerde yer alan metotların yapısı incelenecektir. Her bir kullanıcı uygulama çeşidi için bu metotlar altbölümler şeklinde anlatılacaktır.

Lojistik ortamdaki satıcı düğümlerden farklı özellikleri olan taşıtlar vasıtası ile farklı satış fiyatları olan malzeme çeşitleri alıcı düğümlere taşınmaktadır. Taşıma planını ise aracı düğümler hesaplamakta ve yerine getirmektedir. Bu yüzden her bir taşıt ve malzeme sınıfının hesaplamalarda gerekli olan bazı özellikleri vardır. Bu sınıflar SatıcıDüğüm bileşik nesne sınıfının altsınıflarını oluştururlar. Aracı düğüm optimal gönderim planını hesapladığı zaman malzeme ve taşıt nesnelerinin özellikleri hakkında bilgi edinmek için onlardan gerekli olanlarını kendi alanına yükleyerek hesaplama işlemlerini yapıyor. Hesaplama sonucu hem alıcı, hem de satıcı düğüme gönderilecek olan bilgiler Bilgi isimli sınıftan yaratılan altnesnelere eklenir. Gerekli olduğunda bileşik nesneni oluşturan bu alt nesnelerin uygun olanları aracı düğümün uyarması sonucunda alıcı ve satıcı düğümler tarafından kendi alanlarına kopyalanarak kullanılır. Sistemde ayrıca alıcı ve satıcı düğümlerle aracı düğüm arasında anlaşma yapmak/bozmak için Mesaj isimli sınıf da tasarlanmıştır.

DBNTO ortamında bileşik nesneni oluşturan nesneler daha önce Şekil 4.3 üzerinde incelendiği gibi kendi içlerinde bağlantı nesnesi, kontrol nesnesi ve alt nesneden oluşan üçlü bir nesne grubu şeklinde gerçeklenmektedir. Bağlantı ve kontrol nesneleri DBNTO sistemi tarafından altnesnelerin yapısına uygun otomatikman oluşturulduğuna göre, biz burada sadece kendi oluşturduğumuz alt nesneleri inceleyeceğiz.

Aşağıda ilk önce bir bileşik nesneni oluşturan Malzeme, Taşıt, Bilgi ve Mesaj altsınıflarının tasarımı incelenmiş, Daha sonra, her bir düğüme uygun bu sınıflardan oluşan nesneleri içeren SatıcıDüğüm, AracıDüğüm ve AlıcıDüğüm bileşik nesneleri anlatılmıştır.

5.4.1. Alt Nesnelerin Gerçeklenmesi

Malzeme altsınıfının tasarımı Şekil 5.2 de verilmiştir. Burada malzemelerin isim(çeşit), miktar, birim başına satış fiyatı özellikleri ve onlara uygun metotlar tanımlanmıştır.

```
public class Sub_Malzeme implements java.io.Serializable {

      void put_ad(String ad)
      void put_miktar(double miktar)
      void put_maliyet(double maliyet)
      String get_ad()
      double get_miktar()
      double get_maliyet()
}
```

Şekil 5.2: Malzeme alt sınıfının sınıf kodu

Aynı şekilde Taşıt altsınıfı da Şekil 5.3 da gösterildiği gibi oluşturulmuştur. Burada taşıtların isim(plaka), çeşit(kamyon, tır ve b.), birim uzunluğa göre hız, birim uzunluğa göre maliyet, kapasite özellikleri ve onlara uygun metotlar tanımlanmıştır.

```
public class Sub_Taşıt implements java.io.Serializable {
      void put_ad(String ad)
      void put_çeşit(String çeşit)
      void put_maliyet(double maliyet)
      void put_hız(int hız)
      void put_kapasite(double kapasite)
      String get_ad()
      String get_çeşit()
      int get_hız()
      double get_maliyet()
      double get_kapasite()
}
```

Şekil 5.3: Taşıt alt sınıfının sınıf kodu

Daha sonra, Bilgi altsınıfı Şekil 5.4 üzerinde gösterilmiştir. Burada optimal gönderim planı hesaplandıktan sonra hem alıcı, hem de satıcı tarafa gönderilecek olan bilgilerin özellikleri ve onlara uygun metotlar tanımlanmıştır.

```
public class Sub_Bilgi implements java.io.Serializable {
      void put_satıcıadı(String satıcıadı)
      void put_aracıadı(String aracıadı)
      void put_alıcıadı(String alıcıadı)
      void put_malzemeadı(String malzemeadı)
      void put_malzememiktarı(double malzememiktarı)
      void put_malzememaliyeti(double malzememaliyeti)
      void put_taşıtadı(String taşıtadı)
      void put_taşıtçeşidi(String taşıtçeşidi)
      void put_taşıtkapasitesi(double taşıtkapasitesi)
      void put_taşıthızı(double taşıthızı)
      void put_taşıtmaliyeti(double taşıtmaliyeti)
      void put_taşıtrotası(String taşıtrotası)
      void put_kabuldurumu(String kabuldurumu)

      String get_satıcıadı()
      String get_aracıadı()
      String get_alıcıadı()
      String get_malzemeadı()
      double get_malzememiktarı()
      double get_malzememaliyeti()
      String get_taşıtadı()
      String get_taşıtçeşidi()
      double get_taşıtkapasitesi()
      double get_taşıthızı()
      double get_taşıtmaliyeti()
      String get_taşıtrotası()
      String get_kabuldurumu()
}
```

Şekil 5.4: Bilgi alt sınıfının sınıf kodu

Son olarak alıcı ve satıcı düğümlerle aracı düğüm arasında anlaşma yapmak/bozmak için kullanılan Mesaj altsınıfının metotları Şekil 5.5 üzerinde incelenmiştir.

```
public class Sub_Mesaj implements java.io.Serializable {
      void put_mesaj(String mesaj)
      void put_satıcıadı(String satıcıadı)
      void put_aracıadı(String aracıadı)
      void put_alıcıadı(String alıcıadı)

      String get_mesaj()
      String get_satıcıadı()
      String get_aracıadı()
      String get_alıcıadı()
}
```

Şekil 5.5: Mesaj alt sınıfının sınıf kodu

Diğer adres alanlarında bulunan kullanıcı uygulamaların yürüttükleri metot çağrıları sonucu o alanlar üzerinde, bu şekildeki sınıflardan türetilen nesnelerden yalnız gerekli olanları kopyalanır ve kullanılır.

5.4.2. Bileşik Nesnelerin Gerçeklenmesi

Kullanıcı uygulamalarda bileşik nesnenin türetileceği bileşik nesnenin sınıf tanımlamasının hazırlanması için kullanılan her bileşik nesne gizli(private) ve açık(public) metotlar içerir. Gizli metotlar kullanıcılara bir grafik kullanıcı arayüzü (graphical user interace-GUI) üzerinden sunulan metotların yapısını oluşturur. Kullanıcı arayüzünden genellikle her düğüme uygun bileşik nesne metotlarını çağırılmakta ve onlar üzerinden gereken işlemleri yapılmaktadır.

Açık metotlar ise DBNTO ortamında paylaşılmış nesnelerin farklı düğümlerden çağırılarak kullanılması içindir. Bu şekildeki her bir çağrı sonucu bileşik nesnenin uygun altnesnesi çağrı yürütülen tarafa kopyalanmakta ve kullanılmaktadır. Böylece, lojistik uygulamanın DBNTO yapısına uygun çalışma şekli oluşmaktadır.

5.4.2.1 Satıcı Düğümlerdeki Bileşik Sınıf Yapısının Gerçeklenmesi

Satıcı düğümlerdeki kullanıcı uygulamalarda bileşik sınıf tanımlamaları Şekil 5.6'daki gibi olan bu metotların işlevleri Çizelge 5.1'de verildiği gibidir.

```
public class SatıcıDüğüm implements java.io.Serializable {
      Vector<Malzeme> vmalzeme;
      Vector<Taşıt> vtaşıt;
      Vector<Bilgi> vbilgi;
      Vector<Mesaj> vmesaj;
      ...
      //gizli metotlar
      void malzeme_çeşidi_ekle(String malzeme)
      void taşıt_çeşidi_ekle(String taşıt)
      void malzeme_ekle(Malzeme malzeme)
      void taşıt_ekle(Taşıt taşıt)
      boolean malzeme_al(Malzeme malzeme, double miktar)
      boolean taşıt_al(Taşıt taşıt)
      void optimal_sonuca_yanıtı_aracılara_gönder(Bilgi bilgi)
      void kabul_durumuna_yanıtı_aracılara_gonder(Bilgi bilgi)
      void anlaşma_isteği_gönder(Mesaj mesaj)

      //açık metotlar
      String satıcı_adı_döndür()
      Vector<Taşıt> taşıt_listesi_döndür(malzemeadı)
      Vector<Malzeme> malzeme_listesi_döndür()
      Vector<Bilgi> optimal_sonuca_yanıtı_satıcıdan_al(String
            aracıismi)
      Vector<Bilgi> kabul_durumuna_yanıtı_satıcıdan_al(String
            aracıismi)
      Vector<Mesaj> anlaşma_isteğini_satıcıdan_al(String aracıismi)
}
```

Şekil 5.6: SatıcıDüğüm bileşik sınıfının sınıf kodu

Çizelge 5.1: SatıcıDüğüm bileşik sınıfında yer alan metotların işlevleri

Metot Adı	**İşlevi**
malzeme_çeşidi_ekle	Yeni bir malzeme çeşidi(kum, çimento ve b.) ekler.
taşıt_çeşidi_ekle	Yeni bir taşıt çeşidi(kamyon, tır ve b.) ekler.
malzeme_ekle	Var olan malzeme çeşidine miktar kadar ekler.
taşıt_ekle	Var olan taşıt çeşidinden(kamyon, tır ve b.) yeni taşıt nesnesi oluşturur.
malzeme_al	Var olan malzeme çeşidinden miktar kadarını alıcılara göndermek için çıkarır.
taşıt_al	Var olan taşıtlardan gerekli olanı alıcılara malzeme göndermek için çıkarır.
optimal_sonuca_yanıtı_aracılara_gönder	Aracıların gönderdiği Optimal gönderim planına uygun satıcının yanıtını(“kabul” veya “ret”) içeren nesne oluştur.
kabul_durumuna_yanıtı_aracılara_gönder	Aracıların gönderdiği Optimal gönderim planının kabul durumuna uygun satıcının yanıtını(“kabul” veya “ret”) içeren nesne oluştur.
anlaşma_isteğini_gönder	Aracı düğüm ile anlaşma yapmak ve ya anlaşmayı bozmak için Mesaj sınıfından nesne oluştur.
satıcı_adı_döndür	Satıcı düğümün adını geri döndürür.
taşıt_listesi_döndür	Tüm aktif taşıtların nesne listesini geri döndürür. Malzeme parametresi girildikte, bu malzemeyi taşıya bilecek tüm aktif taşıtların nesne listesini geri döndürür.
malzeme_listesi_döndür	Tüm malzemeleri nesne listesi olarak geri döndürür.
optimal_sonuca_yanıtı_satıcıdan_al	Aracıların gönderdiği Optimal gönderim planına satıcının yanıtını aracı ismine göre aracılara gönder.
kabul_durumuna_yanıtı_satıcıdan_al	Aracıların gönderdiği Optimal gönderim planının kabul durumuna satıcının yanıtını aracı ismine göre aracılara gönder
anlaşma_isteğini_satıcıdan_al	“Anlaşma yap” veya “anlaşmayı boz” isteğini aracı ismine göre aracılara gönder

5.4.2.2 Aracı Düğümlerdeki Bileşik Sınıf Yapısının Gerçeklenmesi

Aracı düğümlerdeki kullanıcı uygulamalarda bileşik sınıf tanımlamaları Şekil 5.7'deki gibi olan bu metotların işlevleri Çizelge 5.2'de verildiği gibidir.

```
public class AracıDüğüm implements java.io.Serializable {
      Vector<Bilgi> vbilgi;
      Vector<Mesaj> vmesaj;
      ...
      //gizli metotlar
      Vector<Malzeme>malzeme_listesi_al(String satıcıadı)
      Vector<Taşıt>taşıt_listesi_al(String satıcıadı)
      Vector<String>satıcı_listesi_al()
      Vector<String>alıcı_listes_al()
      void optimal_transfer_planı_hesapla()
      void optimal_sonucu_satıcılara_gönder(Bilgi bilgi)
      void optimal_sonucu_aracılara_gönder(Bilgi bilgi)
      void optimal_kabul_durumunu_satıcılara_gönder(Bilgi bilgi)
      void optimal_kabul_durumunu_alıcılara_gönder(Bilgi bilgi)
      void anlaşma_isteğine_yanıt_gönder_s(Mesaj mesaj)
      void anlaşma_isteğine_yanıt_gönder_a(Mesaj mesaj)

      //açık metotlar
      String aracı_adı_döndür()
      Vector<Bilgi> optimal_sonucu_aracıdan_al_s(String satıcıadı)
      Vector<Bilgi> optimal_sonucu_aracıdan_al_a(String alıcıadı)
      Vector<Bilgi> optimal_sonuca_yanıtı_aracıdan_al_s(String
            satıcıadı)
      Vector<Bilgi> optimal_sonuca_yanıtı_aracıdan_al_a(String
            alıcıadı)
      Vector<Mesaj> anlaşma_isteğine_yanıtı_aracıdan_al_s(String
            satıcıismi)
      Vector<Mesaj> anlaşma_isteğine_yanıtı_aracıdan_al_a(String
            alıcıismi)
}
```

Şekil 5.7: AracıDüğüm bileşik sınıfının sınıf kodu

Çizelge 5.2: AracıDüğüm bileşik sınıfında yer alan metotların işlevleri

Metot Adı	**İşlevi**
malzeme_listesi_al	Satıcı ismine göre tüm malzemelerin listesini al.
taşıt_listesi_al	Satıcı ismine göre tüm taşıtların listesini al.
satıcı_listesi_al	Tüm satıcıların listesini al.
alıcı_listesi_al	Tüm alıcıların listesini al.
optimal_transfer_planı_hesapla	Optimal transfer planı(Optimal sonuç) hesapla.
optimal_sonucu_satıcılara_gönder	Bilgi sınıfından satıcılara göndermek için Optimal sonucu içeren nesneler oluştur.

Çizelge 5.2 (devam): AracıDüğüm bileşik sınıfında yer alan metotların işlevleri

optimal_sonucu_aracılara_gönder	Bilgi sınıfından alıcılara göndermek için Optimal sonucu içeren nesneler oluştur.
optimal_kabul_durumunu_satıcılara_gönder	Optimal sonucun “kabul” veya “ret” edildiğini satıcılara bildirmek için Bilgi sınıfından nesne oluştur.
optimal_kabul_durumunu_alıcılara_gönder	Optimal sonucun “kabul” veya “ret” edildiğini alıcılara bildirmek için Bilgi sınıfından nesne oluştur.
anlaşma_isteğine_yanıt_gönder_s	Satıcı düğümün gönderdiği “anlaşma yap” veya “anlaşmayı boz” mesajına cevap için Mesaj sınıfından nesne oluştur.
anlaşma_isteğine_yanıt_gönder_a	Alıcı düğümün gönderdiği “anlaşma yap” veya “anlaşmayı boz” mesajına cevap için Mesaj sınıfından nesne oluştur.
aracı_adı_döndür	Aracı düğümün adını geri döndürür.
optimal_sonucu_aracıdan_al_s	Optimal sonucu satıcı ismine göre satıcılara gönder.
optimal_sonucu_aracıdan_al_a	Optimal sonucu alıcı ismine göre alıcılara gönder.
optimal_sonuca_yanıtı_aracıdan_al_s	Optimal sonuca yanıtı satıcı ismine göre satıcılara gönder.
optimal_sonuca_yanıtı_aracıdan_al_a	Optimal sonuca yanıtı satıcı ismine göre satıcılara gönder.
anlaşma_isteğine_yanıtı_aracıdan_al_s	“Anlaşma yap” veya “anlaşmayı boz” yanıtını satıcı ismine göre satıcılara gönder
anlaşma_isteğine_yanıtı_aracıdan_al_a	“Anlaşma yap” veya “anlaşmayı boz” yanıtını alıcı ismine göre alıcılara gönder

5.4.2.3 Alıcı Düğümlerdeki Bileşik Sınıf Yapısının Gerçeklenmesi

Alıcı düğümlerdeki kullanıcı uygulamalarda bileşik sınıf tanımlamaları Şekil 5.8'deki gibi olan bu metotların işlevleri Çizelge 5.3'de verildiği gibidir.

```
public class AlıcıDüğüm implements java.io.Serializable {
      Vector<Bilgi> vbilgi;
      Vector<Mesaj> vmesaj;
      ...
      //gizli metotlar
      void malzeme_isteği_gönder(Bilgi bilgi)
      void optimal_sonuca_yanıtı_aracılara_gönder(Bilgi bilgi)
      void kabul_durumuna_yanıtı_aracılara_gönder(Bilgi bilgi)
      void anlaşma_isteği_gönder(Mesaj mesaj)

      //açık metotlar
      String alıcı_adı_döndür()
      Vector<Bilgi> malzeme_isteğini_alıcıdan_al(String aracıadı)
      Vector<Bilgi> optimal_sonuca_yanıtı_alıcıdan_al(String
            aracıadı)
      Vector<Bilgi> kabul_durumuna_yanıtı_alıcıdan_al(String
            aracıadı)
      Vector<Mesaj> anlaşma_isteğini_alıcıdan_al(String aracıismi)
}
```

Şekil 5.8: AlıcıDüğüm bileşik sınıfının sınıf kodu

Çizelge 5.3: AlıcıDüğüm bileşik sınıfında yer alan metotların işlevleri

Metot Adı	**İşlevi**
malzeme_istegi_gönder	Farklı malzeme isteklerini aracı düğümlere göndermek için Bilgi sınıfından nesneler oluştur.
optimal_sonuca_yanıtı_aracılara_gönder	Aracıların gönderdiği Optimal gönderim planına uygun alıcının yanıtını("kabul" veya "ret") içeren nesne oluştur.
kabul_durumuna_yanıtı_aracılara_gönder	Aracıların gönderdiği Optimal gönderim planının kabul durumuna uygun alıcının yanıtını("kabul" veya "ret") içeren nesne oluştur.
anlaşma_isteği_gönder	Aracı düğüm ile anlaşma yapmak ve ya anlaşmayı bozmak için Mesaj sınıfından nesne oluştur.
alıcı_adı_döndür	Alıcı düğümün adını geri döndürür.
malzeme_isteğini_alıcıdan_al	Farklı malzeme isteklerini aracı ismine göre aracılara gönder.

Çizelge 5.3 (devam): AlıcıDüğüm bileşik sınıfında yer alan metotların işlevleri

optimal_sonuca_yanıtı_alıcıdan_al	Aracıların gönderdiği Optimal gönderim planına alıcının yanıtını aracı ismine göre aracılara gönder.
kabul_durumuna_yanıtı_alıcıdan_al	Aracıların gönderdiği Optimal gönderim planının kabul durumuna alıcının yanıtını aracı ismine göre aracılara gönder
anlaşma_isteğini_alıcıdan_al	"Anlaşma yap" veya "anlaşmayı boz" isteğini aracı ismine göre aracılara gönder

5.5. Kullanıcı Uygulamalara Sunulan Arayüzler

Bu bölümde, kullanıcılara bir grafik kullanıcı arayüzü (graphical user interace-GUI) üzerinden sunulan metotların yapısı incelenecektir. Bu metotlar genellikle her düğüme uygun yukarıdaki bölümde incelediğimiz bileşik nesne metotlarını çağırmakta ve onlar üzerinden gereken işlemleri yapmaktadırlar. Gereklilik açısından aşağıda sadece bileşik nesne metotları üzerinden işlem yapan kullanıcı uygulama metotları incelenmiştir.

Lojistik sistemde kullanıcı uygulamalar arasındaki çoğu işlemler otomatik gerçeklendiğinden dolayı kullanıcı uygulamalar daha az işlemi manüel olarak yapmaktadır. Sistemdeki her bir kullanıcı uygulama arayüzünde sistemin kullanıcıya ait işleyişi gözükmekte ve kontrol edilmektedir.

5.5.1. Satıcı Düğümlerdeki Kullanıcı Uygulamalara Sunulan Arayüzler

Satıcı düğümlerdeki kullanıcı uygulamaların yerine getirdiği işlemleri Şekil 5.9 üzerinde inceleyelim.

```
void malzeme_çeşiti_ekle(…)
void taşıt_çeşiti_ekle(…)
void malzeme_ekle(…)
void taşıt_ekle(…)
void malzeme_çıkar(…)
void taşıt_çıkar(…)
void optimal_sonuca_yanıt_gönder(…)
void anlaşma_istegi_gönder(…)
```

Şekil 5.9: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotlar

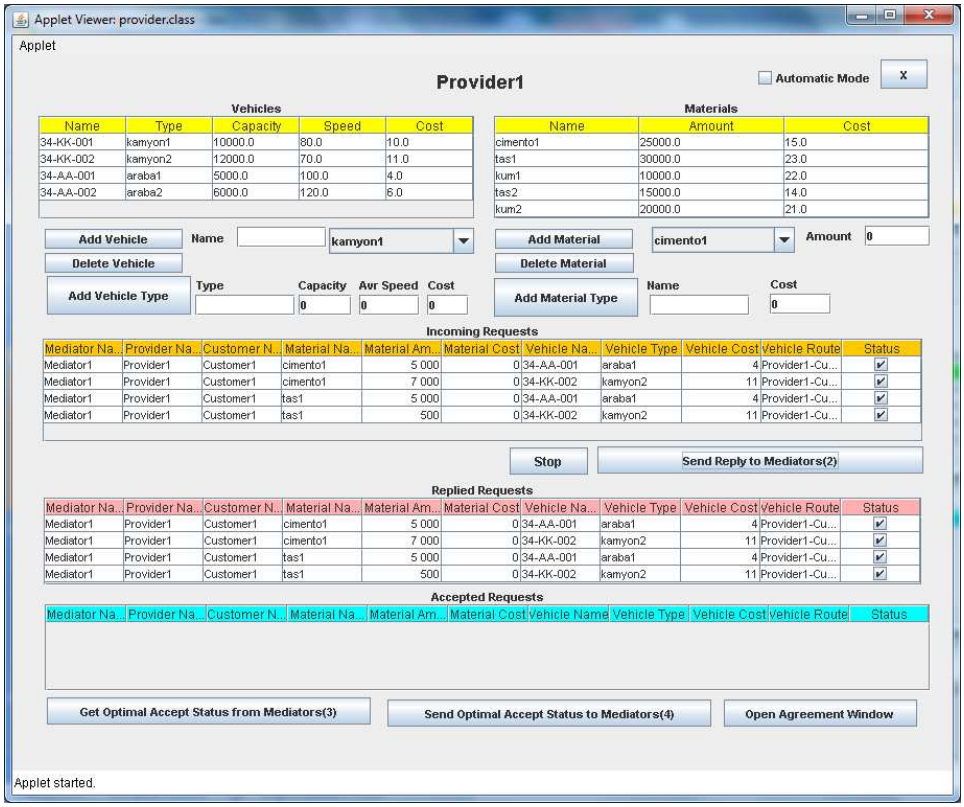
Bu metotların isimleri daha önce anlattığımız SatıcıDüğüm bileşik nesnesinin metotlarına çok benzemektedir, çünkü buradaki her bir metot SatıcıDüğüm bileşik

nesnesinin metotlarını kullanmaktadır. Metotların işlevleri Çizelge 5.4'de verildiği gibidir.

Çizelge 5.4: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotların işlevleri

Metot Adı	**İşlevi**
malzeme_çeşidi_ekle	Yeni malzeme çeşidi ekle
taşıt_çeşidi_ekle	Yeni taşıt çeşidi ekle
malzeme_ekle	Var olan malzeme çeşidine miktar kadar ekle
taşıt_ekle	Var olan taşıt çeşidinden bir taşıt ekle
malzeme_çıkar	Var olan malzeme çeşidinden miktar kadar çıkar
taşıt_çıkar	Var olan taşıt çeşidinden gerekli olan taşıtı çıkar
optimal_sonuca_yanıt_gönder	Aracılar tarafından gönderilen Optimal gönderim planına "kabul" veya "ret" yanıtı gönder
anlaşma_isteği_gönder	Aracılarla anlaşma yapmak için "anlaşma yap" veya anlaşmayı bozmak için "anlaşmayı boz" isteğini gönder

Şekil 5.10 üzerinde Satıcı düğümlerdeki kullanıcı uygulama arayüzü gösterilmiştir. Görüldüğü gibi, satıcı kullanıcı uygulaması malzeme ve taşıtların eklenmesi ve çıkarılması ile bağlı birçok işlemleri yerine getirmektedir. Ayrıca optimal gönderim planının kabul veya ret edilmesi, aracı düğümlerle anlaşma yapılması veya anlaşmanın iptali da satıcı kullanıcı uygulamanın yaptığı işlemlerdir.



Şekil 5.10: Satıcı düğümlerdeki kullanıcı uygulama arayüzü

2. bölümde Şekil 2.3 üzerinde anlatılan protokol yapısının satıcı düğüm tarafından gerçeklenmesi gereken diğer kısımları otomatik olarak yerine getirilmektedir.

5.5.2. Aracı Düğümlerdeki Kullanıcı Uygulamalara Sunulan Arayüzler

Aracı düğümlerdeki kullanıcı uygulamaların yerine getirdiği işlemleri Şekil 5.11 üzerinde inceleyelim.

```
void anlaşma_isteğine_yanıt_gönder_s(Mesaj mesaj)
void anlaşma_isteğine_yanıt_gönder_a(Mesaj mesaj)
```

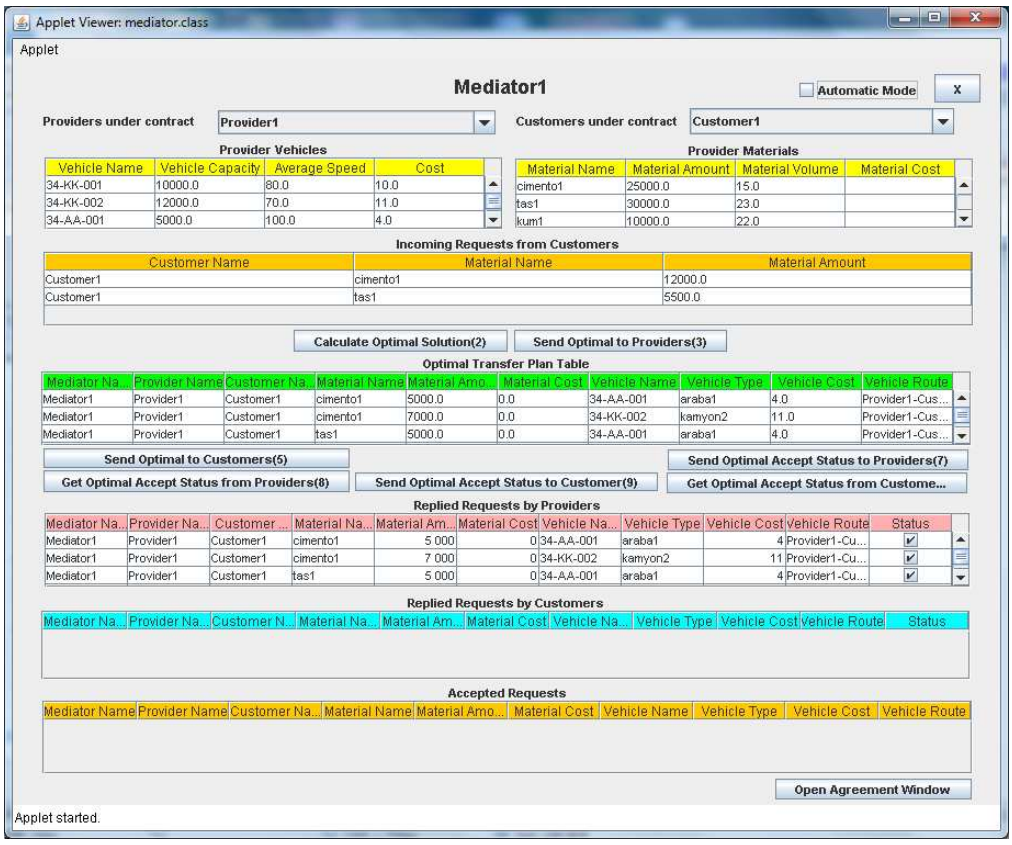
Şekil 5.11: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotlar

Görüldüğü gibi, burada kullanıcılara sunulan sadece iki metot vardır. Metot sayısının az olması, sistemde geri kalan kısımların otomatik olarak yerine getirilmesinden kaynaklanmıştır. Metotların işlevleri Çizelge 5.5'de verildiği gibidir.

Çizelge 5.5: Aracı düğümlerdeki kullanıcı uygulamalarda yer alan metotların işlevleri

Metot Adı	**İşlevi**
anlaşma_isteğine_yanıt_gönder_s	Satıcı düğümlerden gelen "anlaşma yap" veya "anlaşmayı boz" isteklerine aracı düğümün yanıtı.
anlaşma_isteğine_yanıt_gönder_a	Alıcı düğümlerden gelen "anlaşma yap" veya "anlaşmayı boz" isteklerine aracı düğümün yanıtı.

Şekil 5.12 üzerinde Aracı düğümlerdeki kullanıcı uygulama arayüzü gösterilmiştir. Burada, aracı kullanıcı uygulaması anlaşmada olduğu alıcı ve satıcı düğümler arasındaki kendi yönettiği tüm olayları izleye bilmektedir. Ayrıca satıcı ve alıcı düğümlerin gönderdiği anlaşma istekleri sonucu onlarla anlaşma yapılması veya anlaşmanın iptali da aracı kullanıcı uygulamanın yaptığı işlemlerdir.



Şekil 5.12: Aracı düğümlerdeki kullanıcı uygulama arayüzü

5.5.3. Alıcı Düğümlerdeki Kullanıcı Uygulamalara Sunulan Arayüzler

Alıcı düğümlerdeki kullanıcı uygulamaların yerine getirdiği işlemleri Şekil 5.13 üzerinde inceleyelim.

```
void malzeme_istegi_gönder(…)
void optimal_sonuca_yanıt_gönder(…)
void anlaşma_isteği_gönder(…)
```

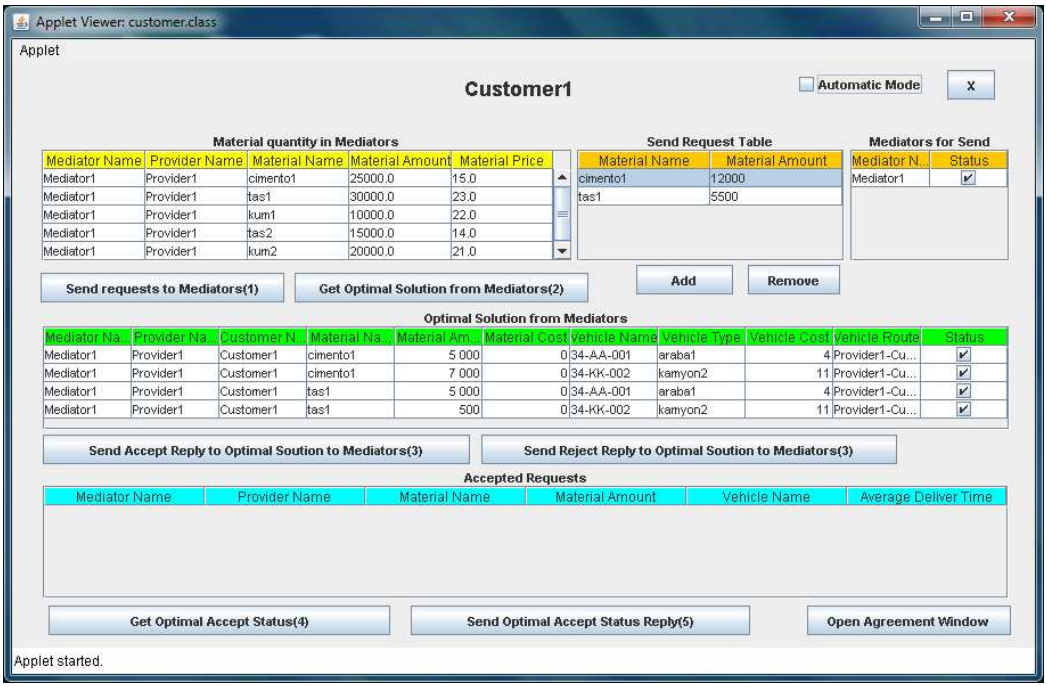
Şekil 5.13: Satıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotlar

Buradaki her bir metot AlıcıDüğüm bileşik nesnesinin metotlarını kullanmaktadır. Metotların işlevleri Çizelge 5.6'de verildiği gibidir.

Çizelge 5.6: Alıcı düğümlerdeki kullanıcı uygulamalarda yer alan metotların işlevleri

Metot Adı	**İşlevi**
malzeme_isteği_gönder	Farklı malzeme isteklerini aracı düğümlere gönder.
optimal_sonuca_yanıt_gönder	Aracılar tarafından gönderilen Optimal gönderim planına "kabul" veya "ret" yanıtı gönder.
anlaşma_isteği_gönder	Aracılarla anlaşma yapmak için "anlaşma yap" veya anlaşmayı bozmak için "anlaşmayı boz" isteğini gönder

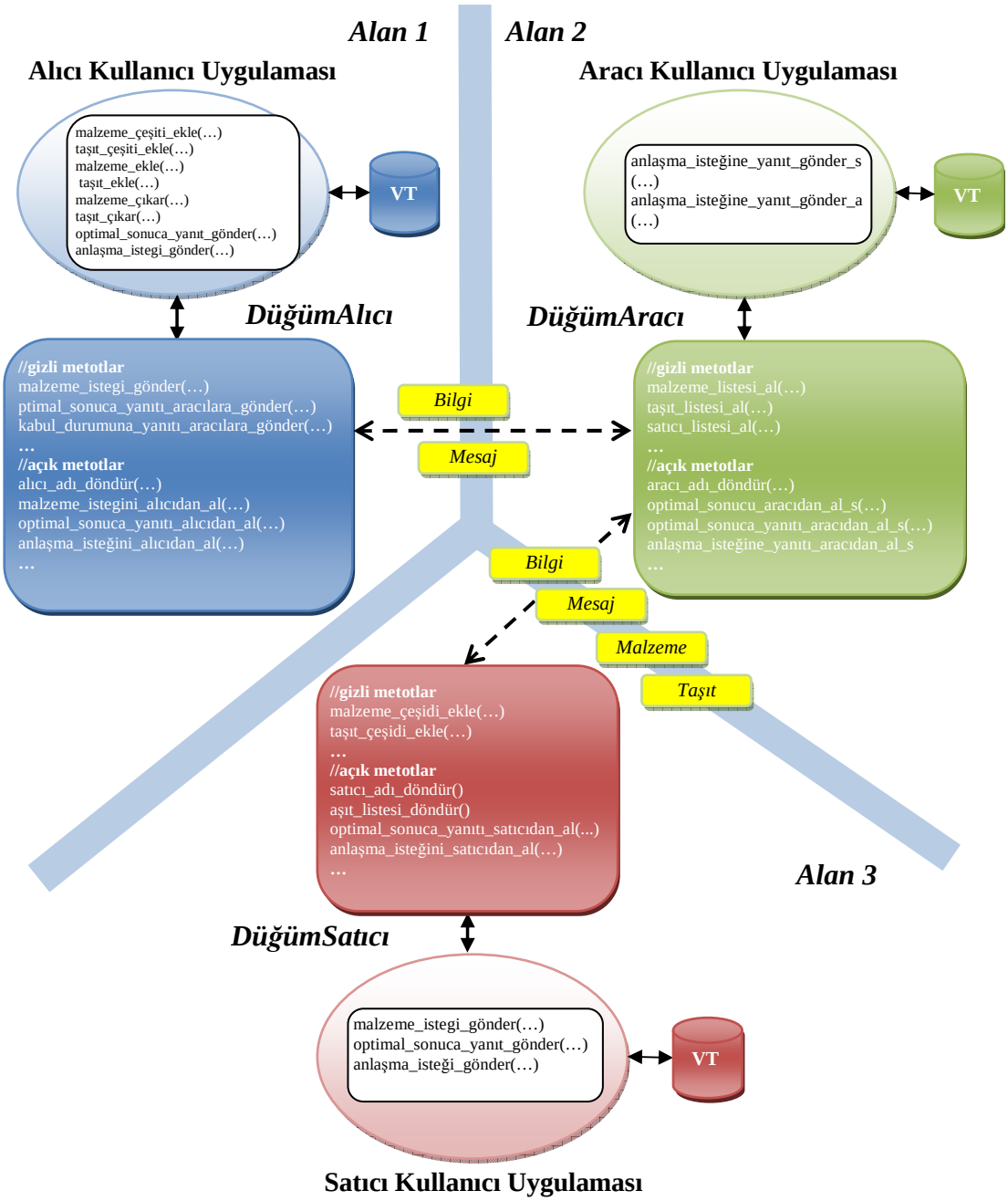
Şekil 5.14 üzerinde Alıcı düğümlerdeki kullanıcı uygulama arayüzü gösterilmiştir. Burada, alıcı kullanıcı uygulaması anlaşmada olduğu aracı düğümlere malzeme istekleri göndermekte, optimal sonuca yanıt vermekte, anlaşma yapılması veya anlaşmanın iptali için aracı kullanıcı uygulamalarla istek göndermektedir. Ayrıca, arayüz üzerinde alıcıya ait tüm olaylar izlene bilmektedir.



Şekil 5.14: Alıcı düğümlerdeki kullanıcı uygulama arayüzü

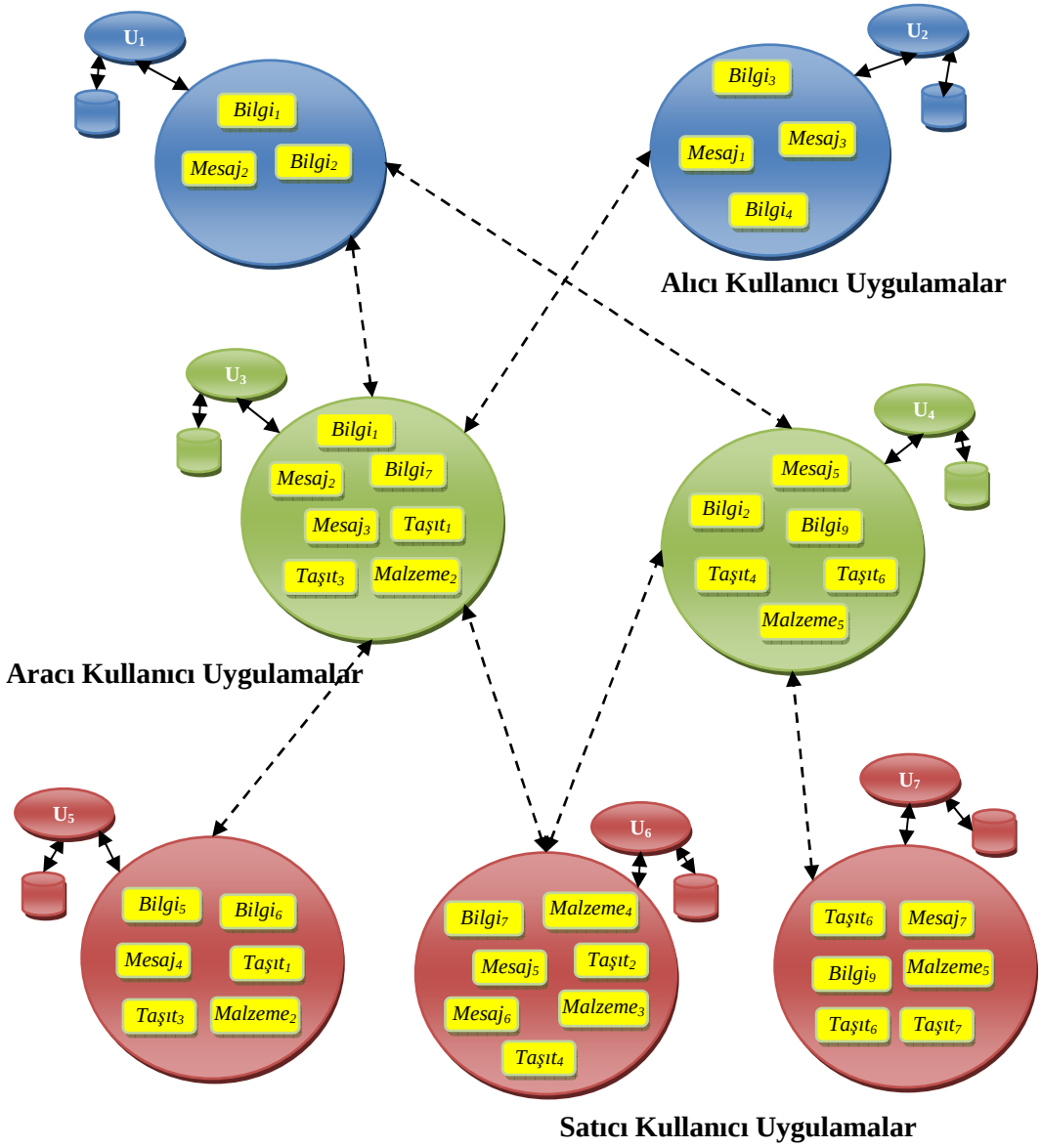
5.5.4. Kullanıcı Uygulamalar Arasındaki Etkileşim

Kullanıcı uygulamalar arasındaki etkileşim Şekil 5.15 üzerinden anlatılmıştır. Burada her bir kullanıcı uygulama tarafından çağrılan metotlar kendine uygun bileşik nesnenin ya gizli ya da açık metotlarını çağırmaktadır. Gizli metot çağrıları sadece bileşik nesnede var olan altnesneler üzerinde işlem yapmaktadır, açık metotlar ise bu bileşik nesneni kendi alanına yükleyerek işlem yapa bilen kullanıcı uygulamalar tarafından kullanılmaktadır. Şekilden de anlaşıldığı gibi, aracı düğüm bileşik nesnesi hem alıcı, hem de satıcı düğüm bileşik nesneleri ile haberleşmekte, alıcı ve satıcı düğüm bileşik nesneleri ise sadece aracı düğüm bileşik nesnesi ile haberleşmekte ve etkileşimde bulunmaktadır. Kullanıcı uygulamalardaki bileşik nesne yapısındaki bilgiler her kullanıcının veritabanında tutulmakta, nesneler üzerindeki herhangi bir değişim veritabanına da yansımaktadır. Ayrıca lojistik sistem sonlandırılıp, yeniden başlatıldığında her bir kullanıcının bileşik nesne yapısı kendi veritabanından uygun şekilde yüklenmektedir.



Şekil 5.15: Alıcı, Aracı ve Satıcı düğümlerdeki kullanıcı uygulamalar arasındaki etkileşim

Bu etkileşim sonucunda, zamanla lojistik ortamdaki kullanıcı uygulamaların yerleştiği düğümlerde farklı alt nesnelerin bileşik nesne ortamına uygun dağılımı oluşmaktadır. Böyle bir bileşik nesne ortamı örneği Şekil 5.16'da gösterilmiştir.



Şekil 5.16: Alıcı, Aracı ve Satıcı kullanıcı uygulamalar arasındaki dağıtılmış bileşik nesneler

Burada, 2 alıcı, 2 aracı ve 3 satıcı düğüm arasında alt nesnelerin paylaşılmış ortamı resmedilmiştir. Düğümler arasındaki oklar, onlar arasında anlaşma olduğunu gösteriyor. Anlaşıldığı gibi, anlaşmada olan düğümler arasında karşılıklı kullanılan altnesnelerin kopyaları oluşmakta ve kullanılmaktadır. Aynı bir düğümün farklı düğümlerle anlaşmada olması mümkün olduğundan, o düğümdeki bir altnesnenin farklı düğümlerde kopyalarının olması olasıdır.

5.6. Lojistik Uygulamanın Çalışmaya Hazır Duruma Getirilmesi

Uygulamamızın DBNTO sistemi üzerinde çalışması için ilk önce dağıtılmış bileşik nesne ortamının hizmete hazır duruma getirilmesi gerekir. Bunun için ilk olarak

çalıştırılacak olan birim DBNTO Sistem Koordinatörü (DSK)'dür. Koordinatör ilk olarak Daha sonra, yaratılacak olan sunucuların da önceden bileceği belirli bir düğümde ve belirli bir *iskele (port)* numarası ile başlatılır. Daha önce de anlatıldığı gibi, sistemin hata-hoşgörü oranını arttırmak üzere Yedek Sistem Koordinatörü de aynı anda belirli bir port numarası ile çalıştırılır.

Bir sonraki adımda sisteme katılacak olan her bir kullanıcı uygulama DBNTO sistemine giriş için

```
new DBNTO_Server();
```

komutunu yürütür. Yeni bir satıcı düğüm uygulaması başlatılacak ise,

```
(1)  SatıcıDüğüm satıcıdüğüm = new SatıcıDüğüm();
(2)  register_class(SatıcıDüğüm,
     "http://sınıf/tanımlamalarının/bulunduğu/adres")
(3)  register_DBN(satıcıdüğüm, "Satıcı Düğüm");
```

komutları yürütülür. Bu komutlardan, (1) "`SatıcıDüğüm`" sınıfından bir "`satıcıdüğüm`" bileşik nesnesi yaratır. (2) uzaktan dinamik yüklemeyi sağlamak üzere sınıf tanımlamalarının bulunduğu URL'yi DBNTO sistem koordinatörüne kaydettirir. (3) `satıcıdüğüm` nesnesini "`Satıcı Düğüm`" adı ile DSK'da kaydettirir. Aracı ve alıcı düğüm uygulamaları için de uygulamanın başlatılma şekli benzerdir.

Her bir aracı düğüm anlaşmada olduğu satıcı düğümle haberleşmek veya etkileşimde bulunmak için satıcı düğüm tarafından yaratılmış ve yukarıda anlatıldığı gibi sisteme kaydedilmiş olan `satıcıdüğüm` nesnesini ilk olarak aşağıdakı şekilde kendi alanına yükler:

```
(1)  load_class(SatıcıDüğüm);
(2)  SatıcıDüğüm satıcıdüğüm = lookup("Satıcı
     Düğüm");
```

(1) `SatıcıDüğüm` sınıfını dinamik olarak uzak alandan yükler ve (2) `satıcıdüğüm` nesnesine ilişkin kabuk nesneyi uygulamanın adres alanına yükler. Daha sonra, yüklenen nesnenin gerekli açık metotlarını kullanarak işleyişine devam eder. Lojistik sistemde tüm kullanıcı uygulamalar için satıcı-aracı, aracı-alıcı ve alıcı-aracı gibi haberleşme ve etkileşimde bulunma şekli yukarıda anlatılan aracı-satıcı şekline benzemektedir.

Görüldüğü gibi, uzaktan sınıf yüklemesi gerçekleştirmek isteyen bir uygulamanın yalnızca yükleyeceği sınıfın adını vermesi yeterlidir. Geri kalan tüm ara işlemler sistem tarafından çözümlenmektedir.

Lojistik sistemde herhangi bir kullanıcı uygulama başlatıldıktan ve uygulamanın DBNTO sistemine uygun ortamı hazır olduktan sonra, bu kullanıcı uygulamanın veritabanından gerekli veriler bileşik nesne yapısına uygun şekilde DBNTO sistemine yüklenir. Sistemdeki kullanıcı uygulamalar kendilerine has bir kullanıcı arayüzü üzerinden çalışırlar. Satıcı firmalarda çalışan kullanıcı uygulamalar o firmada bulunan taşıtları, malzemeleri kontrol etmek, yenilerini eklemek, lojistik firmalardan gelen istekleri incelemek, kabul/ret etmek, yeni lojistik firmaya üye olmak, üyelikten ayrılmak gibi işlemler yürütürler. Alıcı firmalardaki kullanıcı uygulamalar ise, sistemde yer alan lojistik firmaları görür, onlara isteklerini göndererek en uygun olanları ile anlaşma yapabilirler. Lojistik firmalardaki kullanıcı uygulamalar alıcı tarafın isteklerini kabul etmek, optimal sonucu bulmak, bu sonuca uygun satıcı firmalara istek göndermek, yeni satıcı ve alıcı firmalarla anlaşma yapmak v.b. gibi işlemler gerçekleştirirler.

Sisteme eklenecek herbir yeni kullanıcı uygulama DSK üzerinde kayıt yapmak zorundadır. Sisteme kayıt yapan ve sistemden ayrılan her kullanıcı bilgisi DSK tarafindan tüm sistem kullanıcılarına bildirilir.

6. SONUÇLAR VE DEĞERLENDİRMELER

Bu bölümde, tezde önerilen optimal gönderim planının hesaplanmasında kullanılan GÇDTYP algoritmasının başarımı değerlendirilmiş, daha sonra Lojistik Ortamın başarımını benzer özellikler taşıyan diğer sistemlerle karşılaştırmak amacıyla gerçekleştirilen testler ve elde edilen sonuçlar incelenmiştir.

6.1. GÇDTYP Algoritmasının Değerlendirilmesi

Burada, optimal gönderim planının hesaplanmasında kullanılan GÇDTYP algoritmasının başarımını değerlendirmek için simülasyon uygulaması tasarlanmıştır. Uygulama İntel mimarisini destekleyen bilgisayarda(Core 2 Duo, 2.2 GHz, 4GB RAM) Windows 7 ortamında test edilmiştir.

Çizelge 6.1 deki sonuçlar, farklı giriş değerlerinde(satıcı, alıcı sayısı ve b.) algoritmanın optimal gönderim planını hesaplama zamanını gösterir. Bulunan sonuçlara göre, bu algoritma gerçek zaman problemlerinin çözümünde çok kullanışlı olabilir.

Çizelge 6.1: Test Çizelgesi

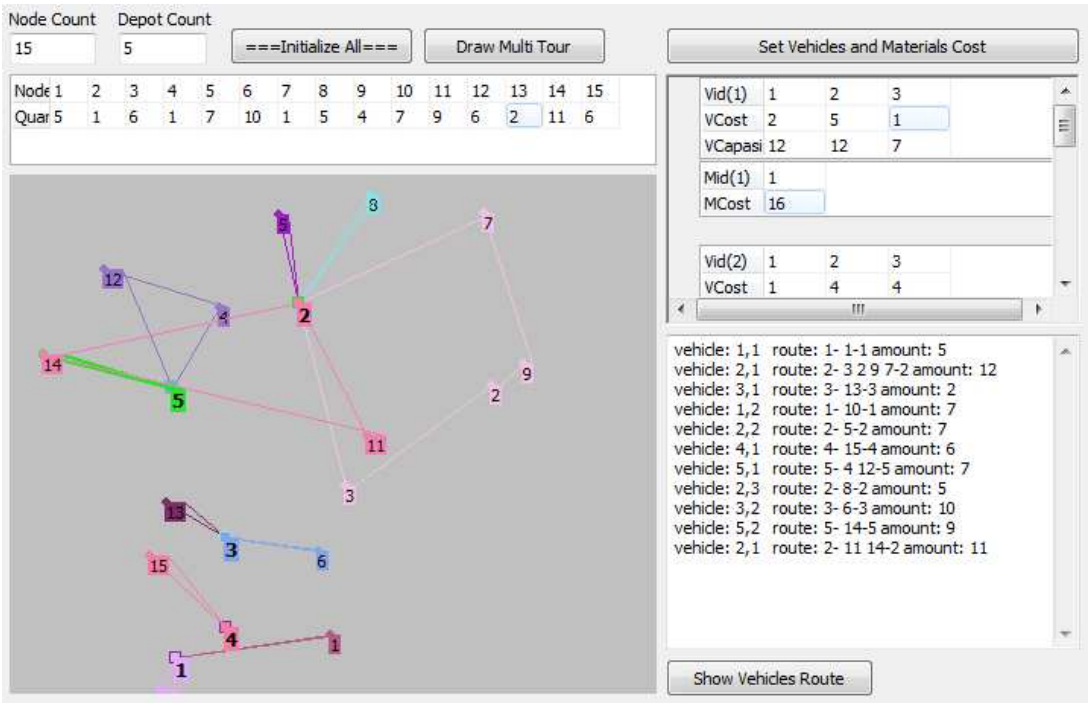
Depo Sayısı	İstekçi Düğüm Sayısı	Deponun Taşıt Sayısı	Malzeme Çeşidi Sayısı	Hesaplama Zamanı (ms)
5	20	3	3	6.54
5	50	5	3	16 .96
10	20	5	3	12.08
10	50	7	3	19.32
20	50	7	5	36.18
20	100	9	5	54.62
50	100	9	5	132.81
50	200	10	5	197.24
50	400	10	5	293.11
100	800	10	7	878.65

Şekil 6.1 da, Çizelge 6.2 de gösterilen giriş parametreleri ile bir örnek incelenmiştir: Giriş için 15 alıcı düğüm ve her birinin 3 taşıtı olan 5 satıcı düğüm verilmiştir. Kolay anlaşılması için yalnız bir malzeme türü seçilmiştir.

Çizelge 6.2: Giriş için Rastgele Değerler

Depo No	Taşıt No	Taşıt Maliyeti	Taşıt Kapasitesi	Malzeme 1 Maliyeti
1	1	2	12	16
	2	5	12	
	3	1	7	
2	1	1	13	14
	2	4	10	
	3	4	7	
3	1	2	8	12
	2	4	12	
	3	5	7	
4	1	2	7	14
	2	3	8	
	3	4	10	
5	1	5	9	13
	2	5	9	
	3	3	9	

İstekçi Düğüm	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Malzeme 1 için İstek Miktarı	5	1	6	1	7	10	1	5	4	7	9	6	2	11	6



Şekil 6.1: Simülasyon Uygulaması

Uygulama 3.2. bölümde anlatılan algoritmayı kullanarak optimal gönderim planını bulmuştur. Hesaplanmış plana uygun olarak nakliyatta kullanılan taşıtlar, her bir taşıtın rotası ve taşıdığı malzeme miktarı Çizelge 6.3 te gösterilmiştir.

Çizelge 6.3: Taşıt rotaları ve taşınan malzeme miktarları

Depo No, Taşıt No	Taşıt rotası (depo– istekçi düğümler - depo)	Taşınan Malzeme Miktarı
1,1	1 -1-1	5
2,1	2- 3 2 9 7-2	12
3,1	3- 13-3	2
1,2	1- 10-1	7
2,2	2- 5-2	7
4,1	4- 15-4	6
5,1	5- 4 12-5	7
2,3	2- 8-2	5
3,2	3- 6-3	10
5,2	5- 14-5	9
2,1	2- 11 14-2	11

Şekil 6.1'deki grafiksel anlatım ve Çizelge 6.3'deki sonuçlar alıcı düğümlerin malzeme isteklerinin uygun taşıtların ilk ugramasında ödendiğini gösterir. Bu örnekde ancak 14. alıcı düğümünün malzeme isteği parçalı teslim olarak yerine getirilmiştir. İlk olaraq 5.satıcı düğümünün 2.taşıtı 9 birim miktarda malzeme ile 14. alıcı düğüme hizmet vermiş, Daha sonra, 2.satıcı düğümünün 1.taşıtı artakalan 2 birim miktarda malzemeni 11. alıcı düğüme hizmet verdikten sonra buraya ulaştırmıştır.

Buradan anlaşılıyor ki, algoritma optimal gönderim planını kısa zamanda hesapliyor. Böylece, bize bu algoritmayı test edilmiş DCOBE sistemi ile entegrasyon etmek gerekiyor.

6.2. Lojistik Uygulamanın Değerlendirilmesi

Lojistik uygulamanın başarımını değerlendirmek amacıyla sistem yapısı, çalışma ortamı ve uygulama alanı bakımından kendisi ile benzerlik gösteren çalışmalar karşılaştırılabilir. Ancak, literatürde sözü edilen çalışmalar üzerinde uygulanmış testler ve bu testlere ilişkin sonuçlar ve değerlendirmeleri bulunmamaktadır. Gerçekçi bir karşılaştırma yapabilmek için, karşılaştırılacak olan sistemlerin aynı platform üzerinde, aynı şartlarda ve aynı kriterlerle test edilmeleri gerekir. Aksi takdirde, elde edilecek sonuçlar yanlış değerlendirmelerin yapılmasına neden olabilir. Bu amaçla, Lojistik uygulamanın çalışma ortamı olan DBNTO sistemi ile

Java'nın RMI yapısı [34] karşılaştırılmış, aynı test ortamında, aynı koşullarda ve aynı bileşik nesne örneği üzerinde yapılmış ölçümleri değerlendirilmiştir.

Test sonuçları incelendiğinde, Java RMI'da istekte bulunulan alana uzak nesneye ilişkin erişim nesnesi, DBNTO'da ise bileşik neneye ilişkin kabuk nesne ve bağlantı nesneleri yüklenir. Sonuçta bu işlemler her uygulama için yalnızca bir kez yürütüleceğinden dolayı sistem başarımı üzerinde önemli bir etkisinin olmadığı açıktır.

Yürütülen bir DBNTO metot çağrısı, eğer henüz o alanda bir kopyası çıkarılmamış olan bir alt nesneye ilişkin ise, bu durumda o alt nesne ve ilgili kontrol nesnesinin birer kopyası çağrının yapıldığı alanda oluşturulacaktır. Kontrol nesnesi ilgili alanda kopyalandığı andan itibaren, dağıtılmış bileşik nesne uygulaması sonlanana kadar o alanda kalacaktır. Bundan dolayı, kontrol nesnesi ve alt nesnenin yüklenme sürecinin sistem başarımı üzerinde önemli bir etkisi yoktur.

Java RMI'da metot çağrısı sırasında aktarılan ve geri dönen parametrelerin büyüklüğü doğrudan çağrının sonlanma zamanını etkilemektedir. Örneğin, iki matrisi çarpıp, sonuç matrisini geri döndüren bir uzak nesne düşünüldüğünde, her çağrı için parametre olarak iki matris uzak alana ağ üzerinden aktarılacak ve sonuç matrisi de geri döndürülecektir. Oysa DBNTO sisteminde tüm çağrılar yerel olarak yürütüldüğünden, aynı işlevi gerçekleştiren bir DBN'de parametre büyüklükleri sistem başarımını etkileyen bir etken olmayacaktır.

DBNTO sisteminin özellikle okuma çağrılarının yazma çağrılarına oranla ağırlıklı olduğu uygulamalarda çok iyi bir performans vereceği görülmektedir. Önerilen dağıtılmış bileşik nesne ortamındaki test değerlendirmesi sonucunda kullanıcı uygulamalar arasında bu nesnelerin işlem hızı ölçülmüş ve Şekil 6.4'te gösterilen sonuçlara ulaşılmıştır.

Ayrıca, bir DBNTO çağrısı sonlanma süresi ile Java RMI çağrısının sonlanma süresi arasında yaklaşık olarak on bin kat fark olduğu görülmüştür. Bunun nedeni, DBNTO sisteminde bir alan için gerekli olan nesneler üzerinde ilk defa metot çağrısı yürütüldükten sonra, zamanla bu alanda o nesnelerin güncellenmiş kopyalarının oluşması ve çağrıların uzak bir nesne metotlarına olduğu gibi kopyalanmış nesneler üzerinden yerel olarak yapılmasıdır. Java RMI çağrıları ise uzak metot çağrısı şeklinde yapılmaktadır.

Çizelge 6.4: DBNTO üzerinde yapılan testlerden elde edilen sonuçlar

İşlem	**Süre**
Müşterek bir uygulamanın bileşik nesneye ilişkin kabuk nesne ve bağlantı nesnelerini kendi alanına yüklemesi	4.62 ms
Eğer bir alt nesne üzerinde ilk çağrı yürütülüyorsa, kontrol nesnesi ve alt nesnenin istekçinin bilgisayarına yüklenme süresi	0.04 ms
Eğer bir alt nesne üzerinde ikinci ve sonraki çağrılar yürütülüyorsa	9334 ns

7. SONUÇLAR

Bu çalışmada, dağıtılmış nesneye-yönelik sistemler için kopyalamaya dayalı bir model olan “dağıtılmış bileşik nesne” yöntemi ve bu yöntemi destekleyecek bir ara katman yazılımı olan “dağıtılmış bileşik nesne tabanlı ortam” sunulmuştur. Önerilen sistem yaklaşımı, farklı alanlarda bulunan çok sayıdaki kullanıcı arasında bilgi paylaşımına ve müşterek iş yürütmeye imkan sağlayan uygulamalar için paylaşılan nesneler kümesi olarak düzenlenmiş, dağıtılmış haberleşme ortamı sunmaktadır.

Daha sonra, bu ortam üzerinde dağıtılmış paylaşımlı sistemlere örnek olarak güncel bir lojistik uygulamasının genel yapısı anlatılmıştır. Farklı firma çeşitleri, onların görevleri ve birbirileri arasındakı etkileşim adımları anlatılmıştır. Sistemde optimal gönderim planını hesaplamak için Geliştirilmiş Taşıt Yönlendirme Problemi oluşturulmuş ve bu problemi çözmek için algoritma tasarlanmıştır. DBNTO sisteminin bileşik yapısı ve kopyalamaya dayalı nesne tutarlılık yönetimi, yürütme zamanını azaltmak üzere istekçiye bileşik nesnenin yalnızca kullanılacak olan ilgili kısmının gönderilmesi sonucu çalışan uygulamanın başarımını yükseltmeğe yönelik konular incelenmiştir. Ayrıca, lojistik uygulamanın tasarım ve gerçekleme konuları anlatılmış, farklı düğümlerde çalışan kullanıcı uygulamaların tipleri ve özellikleri açıklanmıştır. Sistemde maliyeti optimize etmek için kullanılacak olan algoritmanın simulasyon ortamı hazırlanmış ve test değerlendirmeleri yapılmıştır.

KAYNAKLAR

[1] **Marshall, D. (**1999). Remote Procedure Calls (RPC) - A tutorial on ONC RPC, Cardiff University,

[2] **Wilkinson, B. and Allen, M. (**2004). Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., Pearson Education Inc.

[3] **Yilmaz and Erdogan.** (2005). DCOBE: Distributed Composite Object-Based Environment, *The Computer Journal*, 48: 273-291

[4] **Yılmaz, G. ve Erdoğan, N.** (2002). Dağıtılmış nesneye dayalı sistemler için dağıtılmış bileşik nesne modeli, *itüdergisi/d mühendislik*, **1(1)**, Ağustos

[5] **Andrew, S. T., Henri, E. Bal, Saniya, B. H.** (1994). An Object-Based Approach to Programming Distributed Systems, Dept. of Mathematics and Computer Science, Vrije Universiteit Amsterdam, The Netherlands, M. Frans Kaashoek Laboratory for Computer Science, M.I.T. Cambridge, MA

[6] **Applied Computing, Mathematics and Statistics Group.** (2001). Advantages of using the object-oriented paradigm for designing and developing software, Applied Management and Computing Division, Box 84, Lincoln University, Lincoln, Canterbury, New Zealand

[7] **Sunderam, V**., (1990). PVM: A Framework for Parallel Distributed Computing, Concurrency: Practice and Experience, **24(4)**, pp. 315-339, December.

[8] **Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J.,** (1996). MPI: The Complete Reference, MIT Press, Cambridge, MA.

[9] **Rosenberry, D., Kennedy, D. and Fisher, G.,** (1992). Understanding DCE, O'Reilly and Associates, Sebastopol , California.

[10] **Levy, E. and Silberschatz, A.,** (1990). Distributed File Systems: Concepts and Examples, ACM Computing Surveys, **21(4),** pp. 321-375, December.

[11] **Eddon, G., Eddon, H.,** (1998). Inside Distributed COM, Microsoft Press, Redmond, WA.

[12] **Grimshaw, A. and Wulf, W.,** (1997). The Legion Vision of A Worldwide Virtual Computer, Communication of the ACM, **40(1)**, pp. 39-45, January.

[13] **Globus Group,** (t.y.). Globus Project and Approach, Alındığı tarih: 10.06.2011, adres: http://www.globus.org/

[14] **Object Management Group,** (1999). The Common Object Request Broker: Architecture and Specification. Revision 2.3.1. OMG Document formal/99-10-07, Object Management Group, Framingham, MA, October.

[15] **Mitchell, J.G., Gibbson, J.J., Hamilton, G. and Kessler, P.B., (**1994). An Overview of the Spring System, In Proc. Compcon Spring, IEEE, February.

[16] **Shapiro, M., Gourhant, Y., Habert, S., Mosseri, L., Ruffin, M. and Valot, C.,** (1989). SOS: An Object-Oriented Operating System - Assesment and Perspectives, Computing Systems, **2(4)**, pp.287-338, December.

[17] **Makpangou, M., Gourhant, Y., LeNarzul, J.P. and Shaphiro, M.,** (1994). Fragmented Objects for Distributed Abstractions, In T.L. Casavant and M. Singhal, (eds.), Readings in Distributed Computing Systems, pp. 170-186, IEEE Computer Society Press.

[18] **Steen, M.V., Homburg, P. and Tanenbaum, A.,** (1999). Globe: A Wide-Area Distributed System, IEEE Concurrency, **7(1)**, pp. 70-78, January-March.

[19] **Pacull, F., Sandoz, A. and Schiper, A.,** (1994). Duplex: A Distributed Collaborative Editing Environment in Large Scale,

[20] **Hauck, F., Becker, U., Geicer, M., Meier, E., Rastofer, U. and Steckermeier, M.,** (1998). AspectIX: An Aspect-Oriented and CORBA-Compliant ORB Architecture, Technical Report TR-14-08-08, IMMD IV, Univ. Erlange-Nürnberg, September.

[21] **Sophie, N. P., Karl, F. D. and Richard, F. H.** (2008). A survey on pickup and delivery problems, Institut f¨ur Betriebswirtschaftslehre, Universit¨at Wien Br¨unnerstr. 72, 1210 Wien, Austria, February 19

[22] **Rushton, A., Croucher, P. and Baker, P.** (2006). The Handbook of Logistics and Distribution Management, Chartered Institute of Logistics and Transport, Jul 28

[23] **Nilsson, F. and Waidringer, J**. (2004). "Logistics Management from a Complexity Perspective", *The ICFAI Journal of Operations Management*, **3(2)**, pp. 59-73

[24] **Perona, M. and Miragliotta, G.** (2004). Complexity management and supply chain performance assessment, A field study and a conceptual framework, *International Journal of Production Economics,* **90(1),** pages 103-115

[25] **Michael, G.K.** (2004). Protocol Design for a Public Logistic Network Department of Industrial Engineering North Carolina State University, Raleigh, NC 27695-7906

[26] **Zhao, L.** (2010). On Optimization of Logistics System in Manufacturing Enterprises, School of Economy & Management Zhongyuan University of Technology, ZhengZhou, P. R. China, 450007

[27] **Jonsson, P.** (2008). Environmental Aspects of Logistics, McGraw Hill Education, MHE0122BC

[28] **Richard, C. L. and Amedeo, R. O.** (1981). URBAN OPERATIONS RESEARCH Logistical and transportation Planning Methods, Prentice-Hall

[29] **Archetti, C. and Speranza, M. G.** (2008). The Split Delivery Vehicle Routing Problems: A Survey, Department of Quantitive Methods, University of Brescia, Contrada Santa Chiara 50, I-25122 Brescia, Italy

[30] **Clarke, G. and Wright, J. W.** (1964). Scheduling of vehicles from a central depot to a number of delivery points, Operations Research, 12:568–81

[31] **Wollrath, A., Riggs, R. and Waldo, J.** (1996). A Distributed Object Model for the Java System, Sun Microsystems, Inc.

[32] **Features of Logistics planning software**. (t.y.). Alındığı tarih: 25.02.2011, adres: http://www.logistics-management-software-guide.com/

[33] **Applied Computing, Mathematics and Statistics Group.** (2001). Advantages of using the object-oriented paradigm for designing and developing software, Applied Management and Computing Division, Box 84, Lincoln University, Lincoln, Canterbury, New Zealand

[34] **SUN Microsystems, Inc.** (t.y.). Java Remote Method Invocation Specification. Alındığı tarih: 07.03.2011, adres: java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf

ÖZGEÇMİŞ

Ad Soyad: Jamshid Nakhchivanski

Doğum Yeri ve Tarihi: Oğuz/Azerbaycan, 25 şubat 1985

Adres: H.Nahçivanlı cad, ev 1, Oğuz/Azerbaycan

E-Posta: nakhchivanski@itu.edu.tr; jamshid_ce@yahoo.com

Lisans: Uygulamalı Matematik ve Enformatik Bölümü, Bakü Devlet Üniversitesi

TEZDEN TÜRETİLEN YAYINLAR/SUNUMLAR

- **Nakhchivanski, J.,** Yılmaz, G. ve Erdoğan, N. (2010). Dağıtılmış Bileşik Nesne Tabanlı Ortam üzerinde Lojistik Uygulaması. *2.Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu*, İstanbul, 3-4 Aralık

- **Nakhchivanski, J.,** Yilmaz, G. and Erdogan, N. (2011). Logistics Application on Distributed Composite Object Based Environment., *4th International Conference on Computer and Electrical Engineering,* Singapore, October 14-15