itüdergisi/d
mühendislik
Cilt:1 Sayı:1 Ağustos 2002

Dağıtılmış nesneye dayalı sistemler için dağıtılmış bileşik nesne modeli

Güray YILMAZ*, Nadia ERDOĞAN

İTÜ Elektrik Elektronik Fakültesi, Kontrol ve Bilgisayar Mühendisliği Bölümü, 80626, Maslak, İstanbul

Özet

Dağıtılmış sistemleri çekici kılan en önemli tasarım konusu, 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 çalışmada Dağıtılmış Bileşik Nesne olarak adlandırılan yeni bir nesne modeli geliştirilmiştir. Aynı zamanda, Java programlama dilini kullanarak dağıtılmış uygulamalar geliştiren yazılımcılar için tekdüze bir arayüz sağlayan, Dağıtılmış Bileşik Nesne Tabanlı Ortam ara katman yazılımı da tasarlanmış ve gerçeklenmiştir. Bu katman uygulama programcılarına bir çok dağıtılmış uygulama için gerekli olan haberleşme, verilerin kopyalanması, parçalanma, tutarlılık, uygulamaların dinamik olarak yüklenmesi v.b. gibi temel mekanizmaları sağlar.
Anahtar Kelimeler: *Dağıtılmış bileşik nesne, nesneye yönelik programlama, dağıtılmış nesneye-dayalı sistemler, bilgisayar destekli birlikte iş yürütme.*

Distributed composite object model for distributed object-based systems

Abstract

A key design issue that makes distributed systems attractive is that all aspects related to the distribution are transparent to users. Unfortunately, general-purpose wide area distributed systems that allow users to share and manage arbitrary resources in a transparent way hardly exist. Constructing wide area applications, such as sharing data across the Internet, often requires a substantial development effort. This is mainly caused by the lack of proper communication facilities as offered by the underlying operating systems and middleware solutions. In this study, a new object model, called "Distributed Composite Object" is proposed. An environment that facilitates the development of internet-wide distributed applications, Distributed Composite Object Based Environment, is also developed. The fundamental idea behind the design of the distributed composite object is that it is physically distributed *over multiple sites. This implies the state of the root object becomes partitioned. Proposed model allows programmers to describe applications in terms of a single composite object which implementation details are embedded and encapsulated in different types of subobjects. Instead of viewing a distributed object as an entity running on a single machine, we view a distributed object as a conceptual object, distributed over multiple machines with its several subobjects.*
Keywords: *Distributed composite object, object-oriented programming, distributed object-based systems, computer supported collaborative work.*

---

*Yazışmaların yapılacağı yazar: Güray YILMAZ. g.yilmaz@hho.edu.tr; Tel: (212) 663 24 90 dahili: 4321.
Bu makale, birinci yazar tarafından İTÜ Elektrik Elektronik Fakültesi'nde tamamlanmış "Dağıtılmış nesneye dayalı sistemler için dağıtılmış bileşik nesne modeli" adlı doktora tezinden hazırlanmıştır. Makale metni 21.01.2002 tarihinde dergiye ulaşmış, 25.01.2002 tarihinde basım kararı alınmıştır. Makale ile ilgili tartışmalar 31.12.2002 tarihine kadar dergiye gönderilmelidir.

Giriş

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. Fakat ne yazık ki, sistem kaynaklarını kullanıcılar arasında saydam bir şekilde paylaştıran ve yöneten genel amaçlı, geniş alana yayılmış dağıtılmış sistemler neredeyse yok denecek kadar azdır. Var olan bazı sistemler de kısıtlı ölçüde kolaylıklar sağlamaktadırlar. Dağıtılmış sistemlerde saydamlık genelde yerel alan sistemleri için sağlanmıştır. Fakat, bu sistemler için bulunan çözümler geniş alana ölçeklenememektedirler (Neuman, 1994).

Bu güne kadar geliştirilen dağıtılmış sistemler genelde geniş alan ağı ortamında, bilinen dosya işlemleri sunmuşlardır (Levy v. diğ, 1990). Bu uygulamalar çoğunlukla düşük-düzeyli haberleşme ilkelerini kullanarak gerçeklenmiş, basit ve tekdüze bir kavramsal modele sahiptirler. Var olan bu ilkeller; *uzaktan yordam çağırma (Remote Procedure Call-RPC), noktadan-noktaya mesaj iletimi (point-to-point message passing), akışa-yönelik haberleşme (stream-oriented communication), dağıtılmış paylaşılan bellek (distributed shared memory)* kullanımı gibi yöntemlerdir.

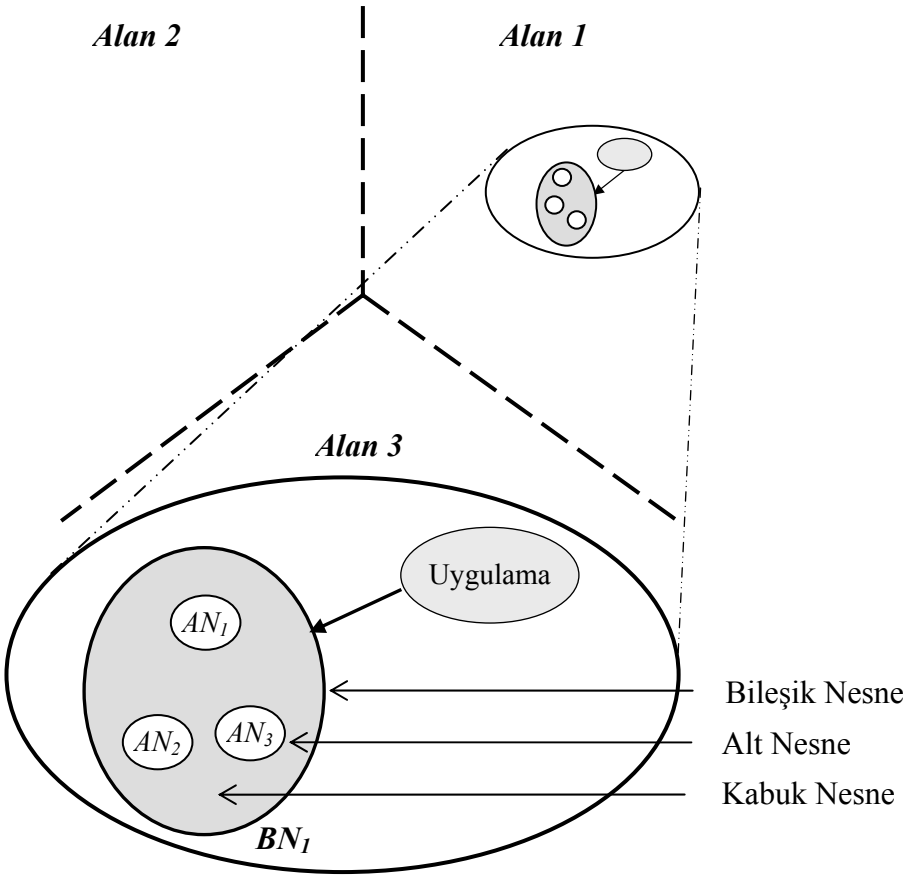
Nispeten düşük düzeyli olan bu haberleşme modelleri uygulamada bazı problemlerin de kaynağı durumundadırlar. Öncelikle, bu modeller *kopyalama (replication)* ve *verinin göç ettirilmesi (data migration)* gibi, tüm dağıtılmış sistemler için geçerli olan haberleşme problemlerine çözüm sağlayamamaktadırlar. Sonuçta, her uygulama amacına yönelik kendine ait bir çözüm aramak zorunda kalmaktadır. İkinci olarak, sunulan kolaylıkların çeşitliliği farklı gerçeklemeleri gerektirdiği için, uygulamaları tek bir çatı altında bir araya getirmek, ya da uygulamalar arasında bilgi alışverişini sağlamak zorlaşmaktadır. Ayrıca, bu haberleşme modelleri semantikleri ve arayüzleri açısından incelendiğinde, genelde bir standardın var olmadığı da görülür. Sonuç olarak, dağıtılmış bir uygulamanın başarımını etkileyen en önemli tasarım konusu olan bilgi alışverişi ve paylaşımı problemi, uygulamanın bir parçası olarak ele alınmakta ve her tasarımcı farklı çözümler geliştirdiği için, uygulamalar arasında heterojenlik problemleri ortaya çıkmaktadır (Homburg v.diğ., 1996; Steen v. diğ., 1999).

Geniş alan dağıtılmış sistemleri için bilgi alışverişi ve paylaşımı konusunda karşılaşılan haberleşme problemlerini çözümleyecek bir modele ihtiyaç olduğu görülmektedir. Model, basit haberleşme ilkelerine oranla daha soyut düzeyde ve aynı zamanda uygulamadan da bağımsız olmalıdır. Özellikle internet üzerinde ortaklaşa iş yürütmeyi destekleyen, farklı yapılardaki bir çok geniş alan uygulamasının geliştirilebilmesine olanak sağlayan genel ve basit bir gerçekleme ortamı sunmalıdır.

Bu çalışmada *Dağıtılmış Bileşik Nesne* (DBN) modeli olarak adlandırılan yeni bir dağıtılmış paylaşılan nesne modeli önerilmektedir. Ayrıca, nesne modeli ile birlikte, Java'da dağıtılmış uygulamalar geliştiren yazılımcıların kullanımı için tekdüze bir programlama arayüzü sağlayan, *Dağıtılmış Bileşik Nesne Tabanlı Ortam* (DBNTO) ara katman yazılımı da gerçeklenmiştir (Yılmaz v. diğ., 2000, 2001a, 2001b). DBNTO programcılara bir çok dağıtılmış uygulama için temel oluşturacak olan; haberleşme, verilerin kopyalanması, parçalanma, tutarlılık, uygulamaların dinamik olarak yüklenmesi gibi mekanizmalar sunmaktadır.

Dağıtılmış bileşik nesne modeli

İnternet ortamında işbirliğiyle yürütülen dağıtılmış uygulamalar genellikle 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.

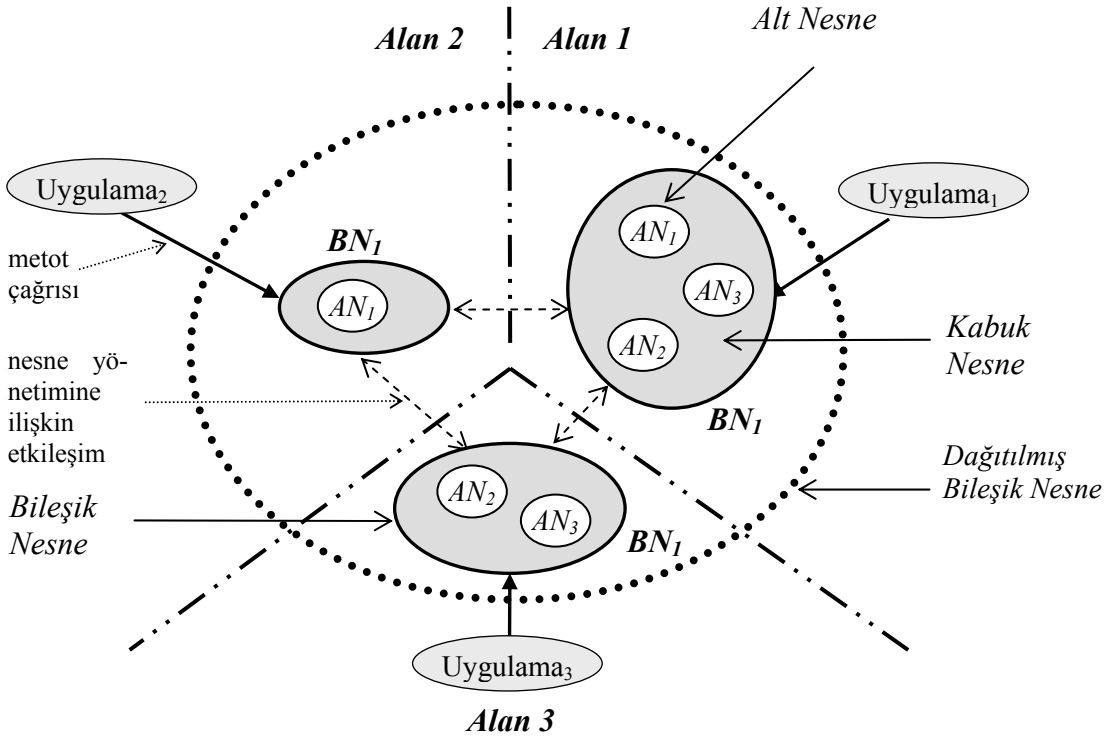


Şekil 1. Tekil bir adres alanı üzerinde üç alt nesnesi ile birlikte yaratılmış bir bileşik nesne

DBN modelinin ardında yatan temel fikir, çok sayıdaki alt nesnenin bileşiminden meydana gelen bir bileşik nesnenin farklı alanlar üzerinde fiziksel olarak dağıtılmasıdır. Önerilen modele göre, 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 1'de sunulduğu gibi, öncelikle tekil bir adres alanı üzerinde bir veya daha fazla sayıda *alt nesnenin (AN)* bir kabuk nesne altında birleştirilmesi suretiyle bir *bileşik nesne (BN)* olarak yaratılır.

Şekil 1'de, *Alan 1* üzerinde üç alt nesnesi ile yaratılan BN_1 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 2'de görü-len nesne yapısı oluşur. Bu yapı incelendiğinde, *Alan 2*'de bulunan uygulamanın BN_1 üzerinde yürütmüş olduğu metot çağrıları sonucunda BN_1 bu alan üzerinde yalnızca AN_1 alt nesnesi ile ve benzer şekilde, *Alan 3*'de bulunan uygulamanın yürütmüş olduğu metot çağrıları sonucunda da AN_2 ve AN_3 alt nesneleri ile birlikte kopyalanmıştır. Bu suretle ortaya bir *dağıtılmış bileşik nesne* yapısı çıkmıştır.

Şekil 2'deki dağıtılmış bileşik nesne yapısı incelendiğinde, *Alan 2*'de bulunan uygulamanın BN_1 üzerinde yürütmüş olduğu metot çağrıları sonucunda BN_1 bu alan üzerinde yalnızca AN_1 alt nesnesi ile ve benzer şekilde, *Alan 3*'te bulunan uygulamanın yürütmüş olduğu metot çağrıları sonucunda da AN_2 ve AN_3 alt nesneleri ile birlikte kopyalanmıştır. Bu suretle bir *dağıtılmış bileşik nesne* yapısı ortaya çıkmıştır.



Şekil 2. Üç adres alanı üzerine yayılmış bir dağıtılmış bileşik nesne

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.

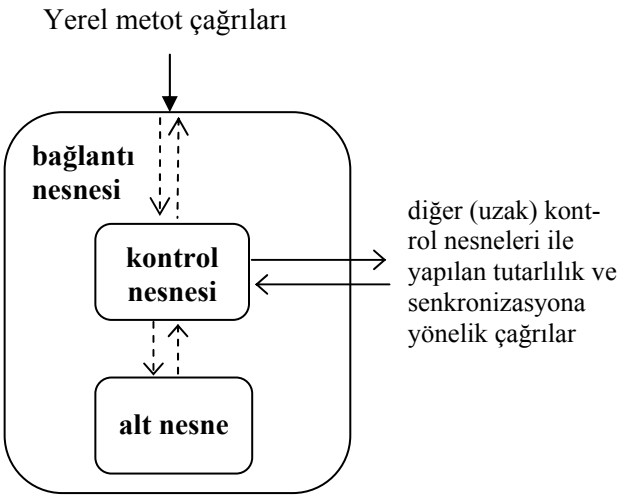
Dağıtılmış bileşik nesnenin yönetilmesi

Bir dağıtılmış bileşik nesnenin iki görünüşü vardır: İstekçilerin bakış açısına göre tekil paylaşılan bir nesnedir. Farklı adres alanlarında bulunan çok sayıdaki istekçi uygulamaları tarafından paylaşılmaktadır. Bu nesneye tanımlı bir arayüz üzerinden erişilir. Nesnenin bileşenleri ve özellikle de nasıl dağıtıldıkları görünemez. Tasarımcısının bakış açısına göre ise bir dağıtılmış bileşik nesne, birlikte çalışan bir alt nesneler kümesinden oluşmaktadır. Bu yapıdaki her bir alt nesne, merkezi yapıda olduğu gibi, temel bir nesnedir. Diğer bir deyişle, her bir alt nesne merkezi bir gösterilime sahiptir.

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 2'de yer alan her bir AN_i nesnesi aslında Şekil 3'teki gibi üçlü bir nesne grubu şeklinde gerçeklenmektedir.

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 sis-



Şekil 3. Alt nesne erişim düzeni

temle ilgili ayrıntıları kullanıcıdan gizleyen bu nesneler, ilgili alt nesnelerin arayüz tanımlamaları kullanılarak otomatik olarak üretilirler. Bu amaçla, ayrıntıları ileriki bölümlerde açıklanan, bir *Otomatik Sınıf Üreteci (OSÜ)* geliştirilmiştir.

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'in 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 yüklemiş olan bir uzak uygulamanın bir metot çağrısı yürütmesi durumunda yüklenme gereği ortaya çıkar.

Kontrol nesnesi
Bir alt nesnenin yürütme anında dinamik olarak istekte bulunulan adres alanına 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 kontrol nesnesi ve alt nesnenin henüz o adres alanına yüklenmediğini gösterir. Bağlantı nesnesi bu durumda 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.

Bir alt nesnenin farklı alanlar üzerindeki kopyalarına farklı alanlardaki uygulamalardan 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 yürütülmeleri gerekir. Ayrıca, yapılan bir metot çağrısı o alanda bulunan alt nesne kopyasını diğerlerinden farklı kıldıysa, bu durumda kullanılan tutarlılık protokolüne göre, diğer kopyaların da 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.

Bir bileşik nesnenin hazırlanması

Bir bileşik nesne tekil bir adres alanında çalışmakta olan bir uygulama tarafından yaratılır ve diğer adres alanları üzerinde çalışmakta olan istekçi uygulamaların yürüttükleri metot çağrılarının gerektirdiği alt nesneleri ile birlikte kopyalanması suretiyle dağıtılmış bileşik nesne dinamik olarak oluştulur. Bu bölümde dağıtılmış müşterek bir uygulama için bir bileşik nesnenin yaratılma aşamaları anlatılacaktır.

DBNTO ara katman yazılımı tasarımcıya merkezi bir yapıda tek bir uygulama tarafından kullanılacakmış görüntüsü altında, dağıtım ile ilgili ayrıntılarla ilgilenmesine gerek kalmadan, dağıtılmış bileşik nesneyi geliştirmesine olanak sağ-

layacak bir ortam hazırlar. Bu nedenle, tasarımcının sadece bileşik nesneyi oluşturan fark-lı tiplerdeki alt nesnelerin üretileceği sınıf tanımlamalarını ve alt nesnelerin bileşik nesne içerisinde bir araya getirilmesini sağlayan kabuk nesnenin sınıf tanımlamasını hazırlaması yeterlidir. Daha sonra, dağıtım ile ilgili bölümleri içeren bağlantı ve kontrol nesnelerinin sınıfları, alt nesne sınıflarına ait arayüz tanımlamaları kullanılarak bir sınıf üreteci tarafından otomatik olarak üretilir. Bu amaçla kullanılabilecek örnek bir arayüz dosyası Şekil 4'te görüldüğü gibi olabilir.

Şekil 4 incelendiğinde her bir metot adının başında W_ veya R_ öneklerinin yer aldığı görülmektedir. Bunun nedeni, alt nesnenin farklı alanlarda bulunan kopyaları üzerinde yürütülecek olan metot çağrıları arasında gerekli eşza-manlılık kontrollerinin yapılması ve tutarlılığın sağlanması gerekliliğidir. Kontrol nesnesi tarafından, eğer güncelleme erişimi yapılacaksa yazma, okuma işlemi yapılacaksa okuma izninin alınması gerekir. Alt nesnenin tasarımcısı erişim türünü belirlemek üzere, her bir metot için, o metot adının başına güncelleme erişimi için W ve okuma erişimi için R öneklerini ekler. Böylece, kontrol nesnesi sınıfı üretilirken sınıf yapısının içine erişim türü için gerekli olan denetim kodu eklenir.

DBN modeli içinde yer alan bağlantı ve kontrol nesneleri otomatik olarak üretileceği için, alt nesne sınıfları ve arayüz tanımlamalarını içeren dosyaların belli bir kurala göre adlandırılmaları gerekir. Bu nedenle, alt nesnelere ait sınıf dosyalarının Sub_ öneki ile başlamaları ve arayüz tanımlama dosyalarının da _itf soneki ile sonlanacakları kabul edilmiştir.

Bileşik nesne üzerinde bir metot çağrısının yürütülmesi

Kabuk nesne üzerinde yürütülen bir metot çağrısı öncelikle bağlantı nesnesi üzerinde bir çağrıya dönüşür. Eğer bağlantı nesnesi içinde bulunan nesne referansı *null* ise, bu ilgili alt nesneye ait kontrol nesnesinin henüz o adres alanına yüklenmediğini gösterir. Bu durumda, ara katman yazılımının gerekli sistem çağrısını yürüten bağlantı nesnesi kontrol nesnesini kendi adres alanına yükler ve ardından metot çağrısını kontrol nesnesine aktarır.

Kontrol nesnesi gelen metot çağrısını alt nesneye iletmeden önce, erişim izni almak zorundadır. Çünkü, aynı alt nesnenin farklı alanlardaki kopyaları üzerinde eşzamanlı metot çağrı-larının yürütülmesi mümkündür. Eğer bu eşza-manlı çağrılar salt oku düzeninde ise, eşzamanlı yürütülmeleri sistemin başarımını olumlu etkiler. Ancak, bu çağrıların alt nesnelerin bütünlüğünü bozabilecek çağrılar olmaları durumunda (okuma/yazma ya da yazma/yazma çağrıları gibi), birbirleriyle çelişen çağrılar adını alırlar ve ardışıl olarak yürütülmeleri gerekir. Bundan dolayı, nesne durumu tutarlılığının sağlanması amacıyla, bir alt nesne için *sahiplik (ownership)* tanımı yapılmıştır. Bu tanıma göre, alt nesnenin durumu üzerinde en son güncelleme işlemi yürütmüş olan alandaki kontrol nesnesi o alt nesnenin sahibidir ve sahiplik zaman içerisinde bir alandaki kontrol nesnesinden bir diğer alandakine aktarılır.

Kontrol nesnesi alt nesne üzerinde metot çağrısını yürütmeden önce, erişim türünü (okuma/yazma şeklinde) belirterek, izin isteğinde bulunur. Eğer alt nesnenin sahipliği yerel kontrol nesnesinde ise, izin işlemleri yerel kontrol nesnesi tarafından yürütülür. Aksi durumda, istek sahipliği elinde bulunduran uzak kontrol nesnesine

```
public interface Person {
      public void W_put_lastname(String name);
      public String R_get_lastname();
      public void W_put_firstname(String name);
      public String R_get_firstname();
}
```

Şekil 4. Örnek bir arayüz tanımlama dosyası

iletilir. Eğer o an bir yazma çağrısı yürütülmüyor ve bekleyen istekler kuyruğu da boş ise, erişim isteğine izin verilir. Aksi takdirde, istek bekleyenler kuyruğuna eklenir. Erişim izninin gelmesinin ardından, kontrol nesnesi alt nesne üzerinde metot çağrısını yürütür. Yapılan çağrı sonucunda alt nesnenin durumu üzerinde güncelleme yapılmış ise, sahiplik yerel kontrol nesnesine geçecek, aksi durumda sahiplik değişmeyecektir. Erişimin tamamlanması ile birlikte, erişimin sona erdiği bilgisi sahipliği elinde tutan kontrol nesnesine bildirilir. Kontrol nesnesi bu arada istekte bulunmuş, fakat yürütülen metot çağrısı ile çeliştiği için beklemeye alınmış olan istekler varsa, o isteklerden mümkün olanlara izin verir.

Alt nesnelerin yönetilmesi

Alt nesne kopyalarının yönetiminde çözümlenmesi gereken iki ana konu vardır. Bunlar; aynı anda farklı alanlardaki kopyalara yapılan erişimlerin alt nesnenin bütünlüğünü bozmadan gerçeklenmesini sağlayacak şekilde eşzamanlılık kontrolü ile, kopyalar arasında tutarlılığın sağlanabilmesi için alt nesneler üzerinde geçersiz kılma veya güncelleme gibi işlemlere izin veren tutarlılık mekanizmalarının tasarlanmasıdır. Bu mekanizmalar, aynı zamanda, uygulama programcısından da gizli olarak çalıştırılmalıdır. Uygulama programcısı her bileşik nesneyi bir yerel nesne gibi görmeli ve sadece nesne referanslarını kullanarak işlemlerini yürütmelidir.

Sözü edilen bu mekanizmalar tüm alt nesne erişimlerini denetleyen kontrol nesnesi içine yerleştirilmişlerdir. Böylece, eşzamanlılık kont-rolü ve tutarlılık sağlama mekanizmaları bir bütün olarak nesne modelinin bir parçasını oluşturmaktadırlar. Bu yapının avantajı, sadece kontrol nesnesi sınıfını üreten OSÜ üzerinde yapılacak bazı değişiklikler ve eklemelerle, DBNTO sistemini oluşturan diğer modüller üzerinde hiçbir değişiklik yapmaksızın, bu mekanizmalara eklentiler yapılabilmesi ya da tamamıyla farklı yeni mekanizmaların geliştirilebilmesidir.

Eşzamanlılık Kontrolü ve Tutarlılığı Sağlama

DBN modelinde *çoklu okuyucu tek yazıcı (multiple reader single writer - MRSW)* senkronizas-yon modeli benimsenmiştir. Bu modele göre, bir alt nesnenin farklı alanlardaki kopyalarına aynı anda yapılan çağrılardan en az biri yazma erişimli bir çağrı ise, bu çağrılar ardışıl olarak yürütülürler.

Alt nesne kopyaları arasında tutarlılığı sağlayan mekanizma için ardışıl tutarlılık yöntemi benimsenmiştir (Mosberger, 1993). Bu yöntem basit, anlaşılır ve tutarlılık modelleri arasında yapısı MRSW senkronizasyon modeli ile birlikte gerçek-lenmeye en uygun olan modeldir. Bu modele göre, kontrol nesnesi alt nesneyi erişim sırasında mutlaka tutarlı olarak görür.

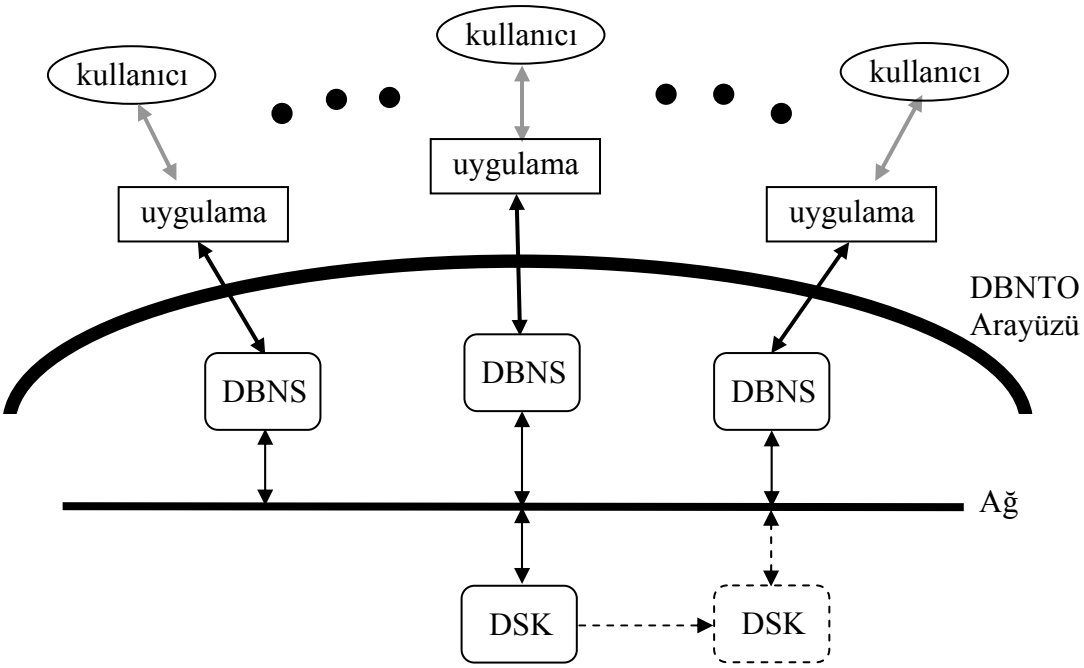
Tutarlılık mekanizması içinde, bileşik nesne tasarımcısının seçimine bağlı iki farklı protokol sunulmuştur. Bunlar; *yazma-güncelleme* ve *yazma-geçersiz kılma* protokolleridir. Tasarımcı, sunulan protokollerden hangisinin kullanılacağına kontrol nesnesinin üretilmesi aşamasında karar verir ve kontrol nesnesi sınıfı bu tercihe bağlı olarak üretilir. Tasarımcı, uygulama gerek-tirdiği takdirde, başarımı yükseltmek amacıyla aynı DBN'nin farklı alt nesneleri için farklı protokoller de belirleyebilir (Gharachorloo v. diğ., 1990; Mosberger, 1993).

Dağıtılmış bileşik nesne tabanlı ortam

Dağıtılmış bileşik nesne tabanlı ortamın tasarımında, nesne bileşimi yönteminden yola çıkarak, mevcut Java nesne modelinin kopyalanarak fiziksel olarak dağıtılmış paylaşılan nesnelere genişletilmesi için gerekli alt yapının nasıl geliştirilebileceği konusu üzerinde yoğunlaşıl-mıştır. Özellikle, internet üzerinde dağıtılmış müşterek grup çalışmalarının dağıtılmış bileşik nesneler kullanılarak kolaylıkla gerçeklenebil-mesini sağlayacak uygun mekanizmalar aran-mıştır.

Dağıtılmış bileşik nesne modelini kullanan bir müşterek uygulamanın genel yapısı Şekil 5'te görülmektedir.

Bu yapıda, dağıtılmış ortamda müşterek çalışmalar yürüten kullanıcılara ait uygulamalar DBNTO arayüzü üzerinden dağıtılmış bileşik nesneleri paylaşırlar.



Şekil 5. DBNTO sisteminin genel yapısı

Şekil 5'te 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. Sistemin hata-hoşgörü oranını arttırmak için, Şekil 5'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.

Herhangi bir uygulama programı DBNS tarafından kendisine sunulan komut kümesini kullanarak dağıtılmış bileşik nesne ortamına giriş/çıkış yapma, yeni bir DBN yaratma ve *kaydettirme (registration)*, var olan bir DBN'yi sistemde *arama (lookup)* ve kendi adres alanına yüklen-me gibi işlemleri gerçekleştirir. DBN üzerinde bir metot çağrısı yürütüldüğünde, ilgili alt nesnenin/nesnelerin çağrının yapıldığı adres alanına taşınması, bütünlüğü bozmayacak şekilde çağrının yerel olarak gerçekleştirilmesi ve uzak kopyalarla tutarlılığın sağlanmasına ilişkin ara işlem adımları da bağlantı nesnesi, kontrol nesnesi ve ilgili DBNS'ler tarafından arka planda yürütülür.

DBNTO sistemi farklı kullanıcı gruplarının, farklı uygulamalarına aynı anda hizmet veriyor olabilir. Fakat, her ne kadar uygulama alanları farklı olsa da, bu gruplar çalışmaları sırasında DBNTO sisteminin altı temel hizmetini kullanırlar. Bu hizmetler;

- Yeni bir uygulamanın sisteme katılması,
- Yeni bir DBN'nin yaratılması ve kayıt ettirilmesi,
- DBN üzerinde bir metot çağrısının yürütülmesi,
- Uzak bir uygulamanın bir DBN'yi yüklemesi,
- Bir uygulamanın sona ermesi,
- Bir DBN'e yeni alt nesne(ler) eklenmesi veya var olanlardan bir kısmının çıkarılmasıdır.

Test sonuçları ve değerlendirme

Dağıtılmış bileşik nesne modelinin başarımını değerlendirmek amacıyla DBNTO, uzak-nesne modeli sınıfına giren Java'nın RMI yapısı (Sun, 2000) ile karşılaştırılmıştır. Ölçümler aynı test ortamında, aynı koşullarda ve aynı bileşik nesne örneği üzerinde uygulanmıştır.

DBNTO ve Java RMI için yapılan tüm başarım ölçümleri bir yerel alan ağı üzerinde gerçekleştirilmiştir. Yerel alan ağının iş yükü günün değişik saatlerinde farklılıklar gösterdiği için ölçümler de günün farklı saatlerinde 10'ar kez tekrarlanarak uygulanmıştır. Bu bölümde veri-len tüm ölçüm değerleri bu testlerden elde edilen sonuçların aritmetik ortalamaları alınarak elde edilmiş olan sonuçlardır.

Java RMI'dan elde edilen test sonuçları

RMI'da uzaktan metot çağrısı yapılacak olan nesnenin bir sunucu bilgisayar üzerine yerleştirilmesi gerekir. Bu nesneye uzak erişim için hem sunucu, hem de istekçi bilgisayar üzerinde birer vekil nesnesi bulunur (stub, skeleton). Ayrıca, uzak erişimin sağlanacağı nesne bulunduğu bilgisayarda, kullanıcı tanımlı bir "ad" ile kaydedilir. Bu nesne üzerinde uzaktan metot çağrısı yürütmek isteyen diğer alanlardaki istekçi programlar, sunucu bilgisayarın kayıt alanı üzerinde nesnenin adını vererek yaptığı bir arama işlemi ile, istekçi taraf vekil nesnesini (stub) kendi alanlarına yüklerler. Bu andan itibaren nesne üzerinde uzak metot çağrıları yürü-tülebilir.

Java RMI üzerinde uygulanan testlerde, yaratılan uzak nesne öncelikle yerel sunucu bilgisayar üzerinde kaydedilmiş, diğer bilgisayarlarda bulunan istekçi programlar sunucunun kayıt alanı üzerinde nesne adını vererek gerçekleştirdikleri bir arama işlemi ile vekil nesnesini kendi alanlarına yüklemişler ve uzak nesne üzerinde çağrılar yürütmüşlerdir. Bu işlemler için yapılan ölçümlerde elde edilen sonuçlar Tablo 1'de sunulduğu gibidir.

Tablo 1'de verilen işlemlerden uzak nesnenin kaydedilmesi işlemi sunucu tarafında ve uzak nesneye ait vekil nesnesinin yüklenmesi işlemi de istekçi tarafında birer kez yürütülen işlemlerdir. Ancak, istekçinin uzak nesne üzerinde yürüteceği her bir metot çağrısı tablodan da görülebildiği gibi hemen hemen 0.5 ms sürmek-tedir. Aynı nesne üzerinde yerel olarak yürütü-len metot çağrılarına ilişkin ölçüm sonuçları ise, parametreli çağrı için 610 ns ve parametresiz çağrı için 600 ns olarak ölçülmüştür. Buradan çıkan sonuç; bir yerel metot çağrısının sonlanma süresi ile uzak metot çağrısının sonlanma süresi arasında yaklaşık olarak on bin kat fark olduğudur.

DBNTO'dan elde edilen test sonuçları

Bileşik nesne yaratıldıktan sonra,

- Uygulama programı tarafından sunucu ve koordinatör üzerinde kullanıcı tanımlı bir ad ile kaydedilmesi,
- Bileşik nesne üzerinde metot çağrısı yürütmek isteyen bir uygulamanın bir arama çağrısı ile nesneye ilişkin kabuk nesne ve bağ-lantı nesnelerini kendi alanına yüklemesi,
- Eğer daha önce yüklenmemiş ise, kontrol nesnesi ve alt nesnelerin çağrının yapılacağı alana yerel olarak yüklenmeleri,
- Bir okuma veya yazma çağrı isteğinin yürütülmesi ve kullanılan tutarlılık protokolüne göre geçersiz kılma veya güncelleme işlemleri

için ölçümler alınmıştır. Sonuçlar Tablo 2'de

Tablo 1. Java RMI üzerinde yapılan testlerden elde edilen sonuçlar

İşlem	Süre
Uzak nesnenin sunucu bilgisayarda bir ad ile kaydedilmesi	2.17 ms
İstekçi programın sunucu bilgisayarın kayıt alanından yaptığı arama işlemi ile vekil nesneyi yüklemesi	1.75 ms
Bir uzak metod çağrısı (parametreli)	0.49 ms
Bir uzak metod çağrısı (parametresiz)	0.38 ms

sunulduğu gibidir.

Elde edilen sonuçların değerlendirmesi

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 nesnenin varlığı 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 parametrelerin büyüklükleri sistem başarımını etkileyen bir faktör olmayacaktır.

DBNTO sisteminin özellikle okuma çağrılarının yazma çağrılarına oranla ağırlıklı olduğu uygulamalarda çok iyi bir performans vereceği görül-mektedir. Bir DBNTO çağrısı 600 ns'de sonla-nırken, Java RMI çağrısı yaklaşık 1000 kat daha uzun bir sürede sonlanmaktadır. Burada ortaya çıkan sonuç oldukça dikkat çekicidir.

DBNTO sistemi yazma çağrılarının ağırlıklı olduğu bazı özel uygulamalarda da iyi sonuç vermektedir. Örneğin, belirli zaman aralıkların-

Tablo 2. DBNTO üzerinde yapılan testlerden elde edilen sonuçlar

İşlem	Süre
Yaratılan bileşik nesnenin kullanıcı tanımlı bir ad ile sunucu ve koordinatör üzerinde kaydedilmesi	1.25 ms
Müşterek bir uygulamanın bileşik nesneye ilişkin kabuk nesne ve bağlantı nesnelerini kendi alanına yüklemesi	3.15 ms
Eğer bir alt nesne üzerinde ilk çağrı yürütülüyorsa, kontrol nesnesi ve alt nesnenin istekçinin bilgisayarına yüklenme süresi	2.65 ms
Okuma erişimli metot çağrısı için;	
• Yerel alandan izin alma süresi	20 ns
• Uzak alandan izin alma süresi	0.75 ms
• Okuma çağrısının tamamlanması	600 ns
Yazma erişimli metot çağrısı için;	
• Yerel alandan izin alma süresi	20 ns
• Uzak alandan izin alma süresi;	0.75 ms
• Geçersiz kılma tutarlılık protokolü uygulanıyor ise	0.72 ms
• Geçersiz kılma mesajı gönderilecek ilave her bir alan için	0.75 ms
• Güncelleme tutarlılık protokolü kullanılıyor ise	0.75 ms
• Yazma çağrısının tamamlanması	610 ns
Yazma çağrısının sonlanmasının ardından (güncelleme tutarlılığı için),	
• Uzak bir alt nesnenin güncellenmesi	1 ms
• Güncelleme yapılacak ilave her bir alan için	0.95 ms

da bazı alanların diğerlerine oranla daha sık yazma işlemi yürütmeleri durumunda, alt nesne üzerinde erişim izni doğrudan yerel kontrol nesnesinden geleceğinden, Java RMI'ya oranla çok daha iyi bir başarım elde edilecektir.

Sonuçlar ve tartışma

Bu çalışmada, dağıtılmış nesneye-yönelik sistemler için yeni bir model olan "dağıtılmış bileşik nesne" modeli ve bu modeli destekleyecek ara katman yazılımı olan "dağıtılmış bileşik nesne tabanlı ortam" sunulmuştur (Yılmaz v. diğ., 2000, 2001a, 2001b). Model, farklı alanlarda çalışan müşterek iş yürütme uygulamalarını destekleyen, paylaşılan nesneler kümesi olarak düzenlenmiş bir haberleşme ortamı sunmaktadır.

Önerilen modelde, büyük bir nesnenin çok sayıdaki alt nesnelerin birleşiminden oluşan bir bileşik nesne olarak yaratılması ve istekte bulunulan alanlar üzerinde nesnenin sadece yapılan metot çağrısının gerektirdiği ilgili alt nesnelerinin kopyalanması öngörülmüştür. Bu arada, ortaya çıkan parçalı nesne yapısının istekçilere gösterilmeden, onların yine tekil bir nesne ile işlem yapıyorlarmış görüntüsü altında erişimlerini gerçeklemelerini sağlayacak bir saydam model amaçlanmıştır.

Dağıtılmış bileşik nesneyi oluşturan alt nesnelerin farklı alanlar üzerinde dinamik olarak bağlanabilmesi amacıyla bir bağlantı nesnesi ve alt nesne kopyalarının senkronizasyon ve tutarlılığının sağlanabilmesi amacıyla da bir kontrol nesnesi her bir alt nesnenin önüne eklenmiştir. Ayrıca, bu ara nesnelerin gerçekleme ayrıntıları kullanıcıya saydam olarak gerçekleştirilmektedir.

Bağlantı ve kontrol nesneleri sınıfları alt nesne sınıflarına ait arayüz tanımlamalarından yararlanılarak bir sınıf üreteci tarafından otomatik olarak üretilmektedir. Otomatik sınıf üretme işleminde tasarımcıya düşen görev, sadece bir alt nesne metodundaki erişim düzeninin hangi tipte (okuma/yazma) olduğunu belirlemek ve arayüz tanımlama dosyasını buna göre hazırlamaktır.

Geliştirme önerileri

Bu çalışmada, çok sayıdaki istekçi arasında paylaşımlı olarak kullanılan dağıtılmış bileşik nesnenin alt nesneleri arasında nesne bazında uygulanabilen, "yazma-geçersiz kılma" ve "yazma-güncelleme" şeklinde iki genel tutarlılık protokolü sunulmuştur. Ancak, farklı uygulama alanları için sistem başarımını artırmaya yönelik, daha farklı tutarlılık protokolleri de sunulabilir. Tasarım aşamasında bu nokta dikkate alınarak esnek bir yapı oluşturulmuştur. Öyle ki, yeni protokollerin desteklenmesi, yalnızca kontrol nesnesinin yapısına bazı eklemelerin yapılması ile mümkündür. Nesne modeli ve DBNTO ara katman yazılımında herhangi bir değişiklik yapılmasına gerek yoktur.

Sistem mevcut haliyle haberleşme ağında meydana gelebilecek arızalara karşı korumalı değildir. Her ne kadar, DSK'nın yedeği mevcut ise de, sistemdeki bir DBNS'e erişilememesi durumunda sistem çalışamaz hale gelebilir. Buna çözüm olarak, sisteme bir *izleme (monitoring)* mekanizması eklenebilir. Bu mekanizma bir DBNS'in devre dışı kaldığını belirlediğinde, DBNS'lerin sistemi terk ederken yaptıkları işlemleri o DBNS adına uygulayarak normal bir sonlanmanın gerçekleşmesini sağlayabilir.

Kaynaklar

Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A. ve Hennessy, J., (1990). Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, *Computer Architecture News,* **18**, 2, 15-26.

Homburg, P., Steen, M. V. ve Tanenbaum, A. S., (1996). An Architecture for A Scalable Wide Area Distributed System, *7. ACM SIGOPS European Workshop*, Connemara, Ireland.

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

Mosberger, D., (1993). Memory Consistency Models, *Operating Systems Review,* **17**, 1, 18-26.

Neuman, B. C., (1994). Scale in Distributed Systems, *Readings in Distributed Computing Systems,* IEEE Computer Society Press.

Steen, M. V., Homburg, P. ve Tanenbaum, A. S., (1999). Globe: A Wide-Area Distributed System, *IEEE Concurrency,* **7**, 1, 70-78.

SUN Microsystems Inc., **(**2000). Java Remote Method Invocation Specification. http://java.sun.com/products/jdk/1.3/docs/guide/rmi

Yılmaz, G. ve Erdoğan, N., (2000). Geniş Alanlı Ağlar İçin Bir Dağıtılmış Kopyalı Nesne Modeli, *IEEE SIU-2000 8. Sinyal İşleme ve iletişim Uygulamaları Kurultayı*, 14-17 Haziran, Antalya.

Yılmaz, G. ve Erdoğan, N., (2001a). İnternet Ortamında Ortaklaşa İş Yürütmeyi Destekleyen Yeni Bir Nesne Modeli ve Programlama Arayüzü, *I. Ulusal İletişim Sempozyumu,* 17-21 Ekim, Ankara.

Yılmaz, G. ve Erdoğan, N., (2001b). A New Distributed Composite Object Model For Collaborative Computing, *16th International Symposium on Computer and Information Sciences (ISCIS XVI),* 6-8 Kasım, Antalya.