GENİŞ ALANLI AĞLAR İÇİN BİR DAĞITILMIŞ KOPYALI NESNE MODELİ

Güray YILMAZ Hava Harp Okulu Bilgisayar Müh. Bölümü Yeşilyurt, 34807, İstanbul e-mail : g.yilmaz@hho.edu.tr	Nadia ERDOĞAN İstanbul Teknik Üniversitesi Bilgisayar Müh. Bölümü Ayazağa, 80626, İstanbul e-mail: erdogan@cs.itu.edu.tr

Özetçe : Dağıtılmış nesneye dayalı bir ortamda daha etkin kullanılabilirlik ve güvenilebilirliği sağlamak amacıyla düşünülebilecek ilk çözüm yöntemi nesnelerin kopyalanmasıdır. Fakat kopyalanan nesnenin kopyaları arasında da tutarlılığın sağlanmış olması gerekir. Kopyalanmış olan nesnenin durumunun tutarlılığı ile ilgili olarak farklı uygulamalar, farklı yöntemler gerektirirler. Şimdiye kadar yapılan çalışmalarda üç yaklaşım göze çarpmaktadır. Bu yöntemler problemin sadece bir kısmına çözüm bulmaktadırlar. Bizim amacımız, değişik durumlara uyum sağlayabilen kopyalı nesneler için bir altyapı hazırlamaktır.

1. GİRİŞ

Dağıtılmış nesneye dayalı bir ortamda daha etkin kullanılabilirlik ve güvenilebilirliği sağlamak amacıyla düşünülebilecek ilk çözüm yöntemi **nesnelerin** kopyalanmasıdır. Kopyalama ile nesne çok sayıdaki alan üzerinde çoğullanmış olur. Fakat kopyalanan nesnenin kopyalarının da birbirleri ile tutarlı olmaları gerekmektedir. Bu da hangi tür uygulamalar için hangi tutarlılık yönteminin daha uygun olduğu konusunu ortaya çıkarmaktadır. Sonuçta seçilecek olan gerçekleme performansta önemli bir kayıba sebep olmamalıdır.

Kopyalanmış olan nesnenin durumunun tutarlılığı ile ilgili olarak farklı uygulamalar, farklı yöntemler gerektirirler. Diğer bir deyişle, performans ve tutarlılık arasında farklı dengeleme durumları söz konusudur. Ayrıca, seçilen bir yöntemin farklı haberleşme ortamlarında ya da uygulamadaki değişmelere bağlı olarak değişmesi gerekebilir.

Şimdiye kadar yapılan çalışmalarda üç yaklaşım göze çarpmaktadır. Bunlardan ilki, uygulama programcıları sistem tarafından desteklenen tutarlılık yönetim protokollerinden birini seçerler [5]. İkincisi, programcılar uygun bir tutarlılık yönetim protokolünü kolaylıkla gerçekleyebilmek için temel geliştirme bloklarını (building blocks) kullanırlar [3]. Üçüncüsü ise, programcılar en iyi tutarlılık yönetimini kendi yapısal modellerine uygun olarak kendileri gerçekleyebilirler [7].

Bu yöntemler problemin sadece bir kısmına çözüm bulmaktadırlar. İlki performans üzerinde odaklanmıştır, fakat programcının seçimini önceden tanımlanmış olan bir protokoller kümesi içinde kısıtlamıştır. Son ikisi ise, uygulama programcılarını kendi tutarlılık yöntemlerini hazırlama yükü ile karşı karşıya bırakmaktadırlar.

Biz bu çalışmada değişik durumlara uyum sağlayabilen kopyalı nesneler için bir altyapı hazırladık. Bölüm 2'de ileriki bölümlerde sözü edilen bazı terimler tanımlanmıştır. Bölüm 3'de kurduğumuz kopyalı nesne modelini, bölüm 4'de ise, mimariyi ele aldık. Son olarak bölüm 5'de konunun kısa bir değerlendirmesi ve sonuç yer almaktadır.

2. TANIMLAMALAR

İstekçi: Kopyalı nesneye çağrı yapan varlıktır. Bu bir thread, bir proses, bir hareket işlemi ya da doğrudan doğruya bir kullanıcı olabilir. Ayrıca, bu istekçi nesnenin başka istekçiler tarafından da erişilebilir olduğunu bilmektedir.

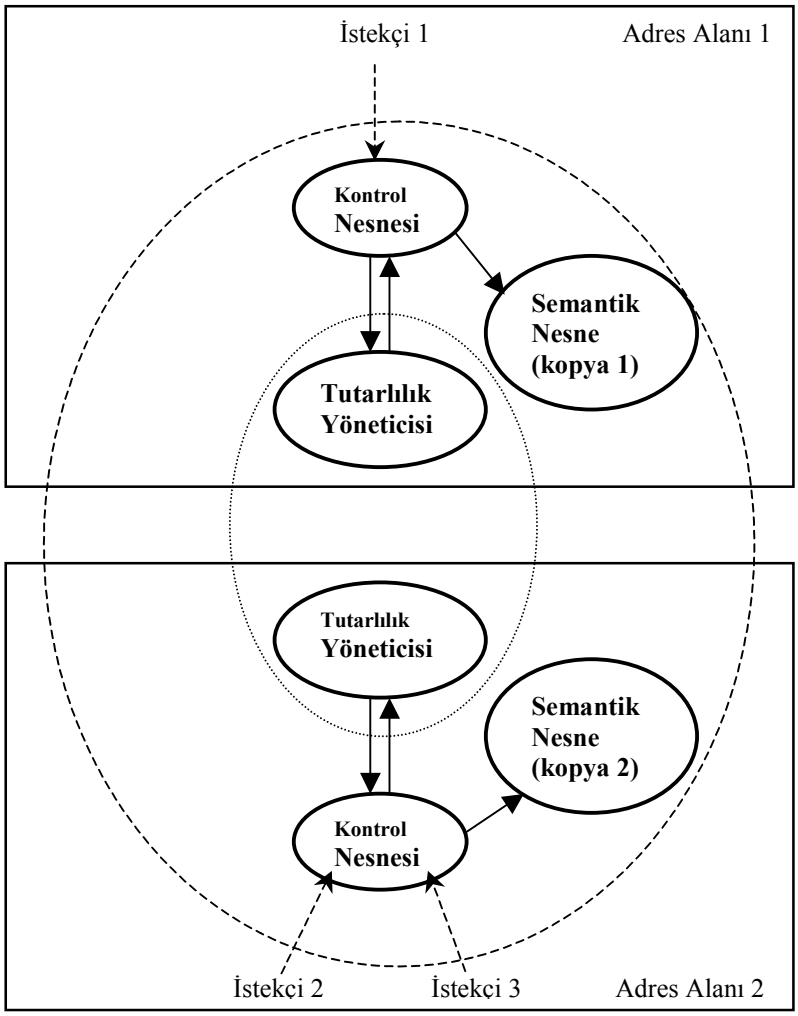
Hareket: Belli bir zaman aralığında yapılan veri erişimi kümesidir. Örneğin çoklu yazışma ortamındaki bir hareket, bölümler ile ayrılmış olan bir dokümanda birinci bölümden üçüncü bölüme kadar yapılan bir değişiklik olabilir.

Hareket Alanı: Hareketin erişip, üzerinde işlem yaptığı alanlardan oluşan kümedir. Yukarıda verilen örnek için bir, iki ve üçüncü bölümler hareket alanı olmaktadırlar.

İstek: İstekçinin kopyalı bir nesne üzerinde yapmayı planladığı bir hareketi tanımlar ve beli bir zaman aralığında yapılan bir veri erişim kümesidir. Diğer bir deyişle, bir istek kullanıcıyı, hareket alanını ve bu kullanıcının belirlenen alanda gerçekleştirmek istediği işlem tiplerini tanımlar.

3. KOPYALI NESNE MODELİ

Bizim bu çalışmada amacımız, değişik durumlara uyum sağlayabilen kopyalı nesneler için bir altyapı hazırlamaktır. Esnek bir yapı oluşturmak için, kopyalı bir nesne için üç bileşenden oluşan bir yapı önereceğiz (şekil 1). Bunlar;



Şekil 1: İki adres alanı üzerine dağıtılmış kopyalı nesnenin görünüşü (kesikli çizgiler dağıtılmış nesnenin mantıksal sınırlarını göstermektedir).

- Her bir alanda bulunan ve nesnenin semantiğini gerçekleyen **semantik nesnesi**,
- Erişim protokolünü gerçekleyen **kontrol nesnesi,**
- Tutarlılık protokolünü gerçekleyen bir dağıtılmış **tutarlılık yöneticisidir.**

Semantik nesne, kopyalanmış olan durumun yerel kopyasını tutar ve bu kopyanın durumu üzerindeki işlemleri gerçekleştirecek olan metodlar için bir arayüzden oluşur.

Kontrol nesnesi, semantik nesneye yapılan erişimleri denetleyen bir nesnedir, semantik nesne için bir bakıma örtü görevini görür (semantik nesneye erişmenin tek yolu kontrol nesnesi üzerindendir).

Tutarlılık yöneticisi ise, diğer alanlardaki tutarlılık yöneticileri ve kendi alanındaki kontrol nesnesi ile koordineli olarak çalışmak suretiyle, kopyalanmış olan nesne durumunun tutarlılığını sağlar.

Kontrol nesnesi ve semantik nesne farklı tiplerde olabilirken, tutarlılık yöneticisinin yapısı geneldir. Yani, bir tutarlılık yöneticisinin iç yapısı kopyalı nesnenin tipi ile ilgili herhangi bir bilgi içermez. Aynı zamanda, tutarlılık yöneticisi eşzamanlılık kontrolünü de sağlar. Bu amaçla, genelde kullanılmakta olan tutarlılık yöntemlerini gerçekleyen tutarlılık yöneticileri sınıfları oluşturulacaktır.

3.1. Bileşenlerin Daha Ayrıntılı Tanımlanması

3.1.1. Kontrol Nesnesi

Kontrol nesnesinin, semantik nesneye yapılan erişimleri denetleyen ve semantik nesne için bir bakıma örtü görevini gören bir nesne olduğunu daha önce söylemiştik Her bir kontrol nesnesinin iki arayüzü vardır. Bunlar; yerel istekçi tarafından kullanılacak olan bir **istekçi arayüzü** ve tutarlılık yöneticisi tarafından uzak değişikliklerin bildirilmesi için kullanılacak olan bir **güncelleme arayüzü'**dür.

Kullanıcı arayüzü de iki kısma ayrılır; **eşzamanlılık kontrolü arayüzü** ve **hizmet arayüzü.** Hizmet arayüzü semantik nesneninki ile aynıdır. Eşzamanlılık kontrolü arayüzünün kullanılmasının sebebi; bir istekçi tek bir istek altında birkaç çağrının peş peşe yürütülmesini isteyebilir. Bu durumda *harekete_başla* ve *hareketi_sonlandır* arasında kalan çağrıların başka çağrılar tarafından kesintiye uğramaksızın yapılmasını isteyebilir.

Kontrol nesnesinin kullanıcı arayüzünün herbir metodu aşağıdaki erişim protokollerini gerçekler.

1. Yapılan çağrının erişim özelliklerine ve istekçi tanımlayıcısına bakarak bir **istek nesnesi** canlandırır.
2. Daha sonra bir önce yaratılmış olan isteği de aktararak tutarlılık yöneticisini çağırır ve kilit isteğinde bulunur (**hareketi_başlat**).
3. Hareket, tutarlılık yöneticisinden alacağı izin ile semantik nesnenin ilgili metoduna çağrıda bulunur.
4. Eğer metod(lar)a yapılan çağrı sonucu yeni bir nesne durumu oluşmuş (yani veri güncellenmiş) ise, kontrol nesnesi güncellemeleri gösteren bir mesaj oluşturur ve bu mesaj tutarlılık yöneticisi üzerinden diğer kopyalara aktarılır.
5. **hareketi_sonlandır** ile tutarlılık yöneticisine hareketin sona erdiği bildirilir.

3.1.2. Tutarlılık Yöneticisi

Tutarlılık yöneticisi, tutarlılık mekanizmalarını gerçekler. Genel olarak kilitlerin elde edilmesi ve güncelleme isteklerinin diğer alanlara aktarılması gibi iki önemli görevi vardır.

Tutarlılık yöneticisi geneldir ve bir kopyalı nesnenin tutarlılığının o nesnenin tipine bağlı olmaksızın etkin bir şekilde yönetilebilmesini sağlar. Her bir tutarlılık yöneticisi kendi kontrol nesnesine erişebilmek için bir referans tutar.

Tutarlılık Yöneticisi aşağıdaki sınıf yapısında gösterilen üç metod sunar.

```
Class Tutarlılık_Yöneticisi {
        public void güncelle(Hareket_id , Güncelenecek_blok);
        public Hareket_id harekete_başla(İstek_id);
        public void hareketi_sonlandır(Hareket_id);
}
```

1. **harekete_başla :** Yeni hareketin istekte bulunulan işlemlerini bildirir ve bu hareketi başlatmak için izin ister. Bu metod çağrıldığında, tüm hareket için izin verilinceye kadar istek bloklanır. Ayrıca geri dönüş parametresi olarak yeni başlatılmış olan hareket için bir tanımlayıcı (Hareket_id) döndürülür.

2. **Güncelle :** Tutarlılık yöneticisini yerel kopyanın güncellendiği konusunda haberdar eder. İlk argüman değişiklikleri yapacak olan hareketi, ikincisi ise bu değişiklikleri tanımlar.

   Güncelleme bilgileri ile birlikte, tutarlılık yöneticisi kopyalar arasında bir kanal gibi davranır. Yönetici güncelleme bilgilerini tüm kontrol nesnelerine (onların güncelleme arayüzleri üzerinden) aktarır. Kanal tarafından gerçeklenen protokol seçilen tutarlılık mekanizmasına bağlıdır.

3. **hareketi sonlandır:** Hareketin sonlandığını bildirir. İstekçinin kopyalı nesneye yapmış olduğu erişim bu metodun çağrılmasından sonra tamamlanır.

3.1.3. Genel Eşzamanlılık Kontrolü

harekete_başla çağrısının yapılması üzerine, tutarlılık yöneticisi istenilen harekete katılacak olan yöneticiler grubunu saptamalıdır. Eğer isteğin yüklemi **zayıf** (*isWeak*) ise, bu durumda yerel tutarlılık yöneticisi tek başına karar verebilir. Aksi takdirde, tüm tutarlılık yöneticileri karara katılırlar.

Katılan her bir tutarlılık yöneticisi öncelikle askıda bekleyen yerel istekleri kontrol ederler. Eğer askıda bekleyen bir istek yoksa, ve şu an yapılan istek halen devam eden hareketler ile çatışmıyor ise (*isConflict*), istek onaylanır. Eğer askıda bekleyen herhangi bir istek varsa ve eğer gelen istek de o anda devam eden isteklerden birinin bir alt isteği ise (*isNested*) ve onlardan herhangi biri ile zıtlık oluşturmuyor ise, yine istek onaylanır. Aksi durumda (yani yukarıdaki koşullardan hiçbirinin sağlanmaması durumunda), gelen istek askıda bekleyen istekler kuyruğuna eklenir.

4. MİMARİ

Kurulacak olan altyapı nesne kopyalarına yapılacak olan tüm erişimleri yakalamalı ve böylece kullanılan tutarlılık mekanizmasına göre onların sürekli olarak tutarlı kalmalarını sağlamalıdır. Bu iki önemli fonksiyon kontrol nesnesi ve tutarlılık yöneticisi tarafından sağlanacaktır.

Herhangi bir istekçinin bir kopyalı nesneye erişebilmesi için, öncelikle onun bir kopyasına kendi adres alanı içinde sahip olması gerekir. Eğer sahip değilse, bir kopya yaratılıp o alanda canlandırılmalıdır. İstekçi kopyalı nesneye onun kontrol nesnesi üzerinden erişebilir. İstekçiden nesneye erişim isteği geldiğinde, kontrol nesnesi öncelikle o nesne üzerinde bir kilit elde eder. İkinci olarak yerel kopya üzerinde çağrıyı gerçekleştirir ve tutarlılık yöneticisini herhangi bir değişiklik durumundan haberdar eder. Son olarak da, kilidi serbest bırakır.

İstekçi uygun tutarlılık yöneticisini canlandırmak suretiyle tutarlılık mekanizmasını seçer. Tutarlılık yöneticisi, kilitlerin elde edilmesi ve güncellemelerin diğer alanlara aktarılması kararlarını verecek, seçilmiş olan tutarlılık yöntemini gerçekler.

İstekçiler tarafından yürütülen istekler bir **istek sınıfı'**ndan üretilecektir. Her istek sınıfı temel bir arayüz gerçekler. Bu arayüzde aşağıda yer alan metodlar bulunur.

isWeak: ilgili hareket **zayıf** (weak) bir harekat ise, true döndürür. Hareketleri iki kısıma ayıracağız; zayıf ve **kuvvetli** (strong). Zayıf hareketler birbirleriyle çatışma oluşturmayan hareketlerdir, yani bu hareketlerin farklı nesne kopyaları arasındaki işlem sırası önemsizdir. Kuvvetli hareketler ise, potansiyel olarak, birbirleriyle ya da bir zayıf hareket ile çatışma yaratabilecek olan hareketlerdir.

isConflict: Eğer yapılacak olan isteğin argümanları halihazırda yürütülmekte olan istek(ler)in argümanları ile çatışıyor ise, true döndürür. Buradaki mukayese üç boyutta yapılır. Bunlar; işlem tipi, hareket alanı ve istekçi tanımlayıcısıdır. Yukarıda verilen çoklu yazışma ortamındaki uygulama örneği tekrar ele alınacak olursa, böyle bir ortamdaki herhangi iki istek aşağıdaki koşullarda birbirleriyle çatışırlar:
Eğer erişim alanları örtüşüyor ise (her ikiside doküman içindeki aynı bölüme erişmek isteyebilir), hareket alanında. Eğer değişme özelliği gösteren işlemler değil iseler, işlem tipinde ve son olarak, farklı iki istekçi tarafından yürütülmüş iseler, istekçi boyutunda çatışma oluşur.

i**sNested:** Eğer hareket iç içe hareketlerden oluşuyor ise, true döndürür. A1 hareketi aşağıdaki koşul altında A2 hareketinin bir alt hareketidir. A1 ve A2 aynı istekçi tarafından başlatılmış, ve A1'in hareket alanı A2'nin hareket alanının içinde bulunuyor ise.

4.1. Etkin İstekler

Eşzamanlılığı ve performansı arttırmanın bir yolu nesnenin semantiğinden yararlanma [4,6]ve diğer yolu ise, ince-taneli (fine-grain) eşzamanlılık kontrolüdür [2]. Biz bu yöntemlerden her ikisinden de yararlanacak bir yapı öneriyoruz.

Bir istek eşzamanlılığın derecesini arttırmak için **etkin** yapılabilir. Bir **etkin istek** nesnenin mantıksal yapısını göz önünde bulundurmanın yanında, nesneye-özel işlem tiplerini de ele alır. İstekler nesnenin yapısı ve semantiğini birlikte dikkate aldığı ölçüde, daha fazla istekçi eşzamanlı olarak çalışabilirler. Bu konuyu iki örnekle açıklayalım.

1. İki yazıcı prosesin bir dokümanın farklı bölümlerini değiştirme isteğinde bulunduğunu düşünelim. Bu durumda her iki yazıcı da istenen hareketi tanımlayan birer istek yaratırlar.

   Eğer istekler dokümanı bölümlendirilmemiş olarak düşünürlerse (yani o dokümanı bir bütün olarak görürlerse), her bir istek tüm doküman üzerinde bir kilit elde etmek isteyecektir. Bu durumda bu iki istek birbirleriyle çatışırlar, çünkü her ikisi de aynı alanı değiştirmek istemektedirler. Ya da diğer bir deyişle, her ikisinin de hareket alanları aynıdır. Sonuç olarak her iki yazıcı eşzamanlı olarak farklı bölümlere erişememektedirler.

   Halbuki dokümanın mantıksal yapısı dikkate alınmış olsaydı (bölümler, alt bölümler, paragraflar gibi), her bir istek yalnızca ilgili bölüm için kilit isteğinde bulunacak ve çatışma oluşmayacaktı.

2. Yine iki istekçi düşünelim. Bunlardan biri yine bir dokümana ait bir bölümün içeriğini değiştiriyor iken, örneğin diğeri de sayfa yapısı üzerinde bir işlem yapıyor olsun. İlk örnekte olduğu gibi, bu iki işlem de gerçek manada birbirleri ile çatışmazlar. Çünkü her iki isteğin hareket alanları aynı olmakla birlikte, işlem tipleri birbirleri ile çatışmamaktadırlar.

   Eğer istekler işlem tiplerini sadece **oku** ve **yaz** olarak düşünürler ise, işlemlerini eşzamanlı olarak yürütemezler. Eğer içerik ve sayfa yapısı semantiğini yakalarlarsa, her iki değişiklik de eşzamanlı olarak yapılabilir.

Aslında burada anahtar özellik, tipten bağımsız tutarlılık yöneticileri yerine, **etkin istek'**ler kullanarak semantiği de dikkate alan optimizasyonları destekleyen bir mimari ortaya koyabilmektir.

5. SONUÇ

Biz bu çalışmada dağıtılmış bir ortamda kopyalı nesnelere problemsiz bir şekilde erişimi sağlayacak bir alt yapı önerdik ve önerdiğimiz yapının büyük bir bölümünü de Java ortamında gerçeklemiş bulunmaktayız [1,8]. Bu yapının amacı kullanıcıların dağıtılmış ortamın getirdiği yüklerden ve zorluklardan etkilenmeksizin kopyalı nesneler üzerinde işlem yapabilmelerini sağlamaktır. Yapının en önemli kısmı tutarlılık yöneticisidir. Buradaki tutarlılık yöneticisi, tutarlılık yönetimini optimize edebilmek amacıyla, nesnenin mantıksal yapısının yanı sıra nesne semantiğinden de yararlanmaktadır.

Bu çalışma ile amaçlanan bir diğer konu ise, uygulama programcıları doğrudan uygulamanın üzerine yoğunlaşırlarken, dağıtma ve kopyalar arasındaki tutarlılığın sağlanması gibi önemli ve gerçeklenmesi zor olan konular ile ilgilenmemektedirler. Bu konular sistem tarafından uygulama programcısına saydam olarak yürütülmektedirler. Herhangi bir kopyalı nesne tipi için, programcı tipe-özel sınıfları tanımlamakta ve uygun olan tutarlılık yöneticisini uygulamasına eklemektedir. Hali hazırdaki gerçeklemede yalnızca n okuyucu- bir yazıcı tutarlılık yöntemi gerçeklenmiş durumdadır. Çalışmanın bir sonraki aşamasında değişik tutarlılık mekanizmalarını gerçekleyen tutarlılık yöneticilerinin sisteme eklenmesi düşünülmektedir.

KAYNAKÇA

[1] Arnold K., Gosling J., *The Java Programming Language*, Addison-Wesley, 1996.

[2] Cabrera L.F., Luniewski A.W., Stamos J.W., "Fine-Grained access control in a transactional object-oriented system", *Computing Systems*, 5(3), sayfa: 199-216, 1992.

[3] Caughey S.J., Parrington G.D., Shrivastava S.K., "Shadows – A Flexible Support System for Objects in Distributed Systems", *Proceedings of the Third International Workshop on Object Orientation and Operating Systems*, sayfa: 73-82, Asheville, NC (ABD), Aralık 1993.

[4] Herlihy M., "Type Specific Replication Algorithms for Multiprocessor", *Proceeding of the 10th International Donference on Distributed Computing Systems*, sayfa: 70-74, IEEE, 1990

[5] Carter J.B., Bennett J.K., Zwaenepoel W., "Implementation and Performance of Munin", *Operating Systems Review*, 25(3), sayfa: 152-164, Ekim 1991.

[6] Leong H.V., Agrawal D., "Type Specific Coherence Protocols for Distributed Shared Momory", *Proceedings of the 12th International Conference on Distributed Computing Systems*, sayfa: 434-441, IEEE, 1993.

[7] Makpangou M., Gourhant Y., Narzul J.P., Shapiro M., "Fragmented Objects for Distributed Abstractions", *Readings in Distributed Computing Systems*, sayfa: 170-186, IEEE Computer Society Press, Temmuz 1994.

[8] Sun Microsystems, *JDK 1.1 Documentation,*
URL: http://www.javasoft.com/products/jdk/1.1/docs/index.html.