

Kalıcı nesneleri destekleyen genişletilebilir bir sistem

Erdal KEMİKLİ*, **Nadia ERDOĞAN**

İTÜ Elektrik-Elektronik Fakültesi, Bilgisayar Mühendisliği Bölümü, 34469, Ayazağa, İstanbul

Özet

Bilgisayar sistemlerinin giderek daha önemli bir bölümünü oluşturan yazılım bileşeni program geliştirmede yaşanan verimsizliğin sorunları ile karşı karşıyadır. Bu makalede programcı verimliliği problemini azaltacak genişletilebilir bir sistem tanımlanmaktadır. Tasarlanan sistem değişik uygulama türlerini destekleyebilecek bir esnekliğe sahiptir. Kalıcı nesneleri destekleyen Genişletilebilir bir Sistem (KGS), genişletilebilir sistemler için kalıcı nesne kavramına dayanan üniform bir model önermektedir. KGS ile kullanıcılar, veri ve programları kalıcı nesnelere olarak değerlendirip tek bir arayüz ile yönetebilmektedir. KGS, nesnelere arası haberleşme hizmeti, istekçi nesne yöneticisi, aktif nesne kütüphanesi, adlandırma ve güvenlik sunucusu, nesne sunucusu ve bir önderleyici ile Linux işletim sistemi üzerinde gerçekleştirilmiştir.

Anahtar Kelimeler: Kalıcı nesne, işletim sistemi, genişletilebilir sistem, programlama.

An extensible system supporting persistent objects

Abstract

Although software has become the most important component of computer systems, software production is suffering from a chronic crisis of unproductivity. Delayed and over budget software projects are very common. Many different approaches including the total quality management and integrated software development environments are considered in response to this problem. This paper defines an extensible and tailorable computing system model which will attack the programmer productivity issue from the technical side. The resulting system is suitable to be used as a base for an extensible system for different types of application domains. Extensible Persistent System (EPS) suggests a new model for extensible systems based on a unifying view of persistency. Data and processes are regarded as passive object and active object, and viewed as persistent objects, and handled through a uniform interface. EPS has extensive support for active object development. EPS also suggests new techniques for capability management, a ticket used to access resources on the system. EPS-C, a C programming language flavor is defined as part of the system. EPS is implemented by following components; inter-object communication service (IOC), client object library (COL), active object library (AOL), naming and protection server (NPS), object server (OBS), and EPS-C preprocessor on Linux operating system.

Keywords: Persistent object, operating system, extensible system, programming.

*Yazışmaların yapılacağı yazar: Erdal KEMİKLİ. erdalk@otokoc.com.tr; Tel: (212) 229 95 55 dahili: 1606.

Bu makale, birinci yazar tarafından İTÜ Elektrik-Elektronik Fakültesi'nde tamamlanmış olan "Kalıcı nesnelere destekleyen genişletilebilir bir sistem" adlı doktora tezinden hazırlanmıştır. Makale metni 30.01.2004 tarihinde dergiye ulaşmış, 18.03.2004 tarihinde basım kararı alınmıştır. Makale ile ilgili tartışmalar 31.03.2005 tarihine kadar dergiye gönderilmelidir.

Giriş

Klasik sistemlerde programlama dilleri geçici veriler için çok iyi bir destek sağlarlar. Daha uzun ömürlü veriler ise VTYS (Veri Tabanı Yönetim Sistemi) veya bir dosya sistemi aracılığı ile yönetilir. Kendisini yaratan programdan daha uzun ömürlü verileri “kalıcı veri” olarak tanımlıyoruz. Kalıcılık kavramı bir verinin kendisine olan gereksinim devam ettiği sürece varlığını sürdürebilmesini içermektedir. Bunun sonucunda, kalıcı sistemler veri yönetimi için tek bir soyutlama sağlayabilirler. Bu soyutlama verinin farklı bellek sistemlerine aktarımı sırasında gerek duyulan dönüşüm ihtiyacını ortadan kaldırarak yazılımcıları önemli miktardaki geliştirme çalışmasından kurtarır. PS-Algol’un tasarımcıları tarafından yapılan bir araştırmada programların %30’unun veri yükleme/saklama/dönüşüm işlemlerinden oluştuğu görülmüştür (Cockshot vd., 1984, Atkinson vd., 1983). Kalıcı nesnelere için destek genellikle PS-Algol’da olduğu gibi programlama dili düzeyinde sağlanmıştır. Kalıcı nesne desteğinin programlama dili düzeyinde verilmesinin iki sakıncası vardır: işletim sistemleri uygulama için gerekli desteği vermeyebilir ve çabalar her yeni kalıcı nesne destekleyen dil için tekrar harcanacaktır (Kemikli ve Erdoğan, 1997; 1998). Bu nedenler başka bir yaklaşımı “kalıcı nesne desteğinin işletim sistem düzeyinde sağlanması” teşvik etmektedir.

İşletim sistemleri genel kullanıma uygun olma ve özel konularda uzmanlaşma arasında bir denge kurmak zorunda kalmaktadır. Genel amaçlı bir sistem biraz çaba ile çok değişik uygulamalarda kullanılabilirken, uzmanlaşmış sistemler belirli tip uygulamaları çok daha sorunsuz olarak çalıştırabilmektedir. Genişletilebilir bir sistem, bir uygulamanın gereksinimleri doğrultusunda değiştirilebilir bir sistemdir (Bershad vd., 1995).

Bu makalenin amacı, yazılım verimliliği konusuna teknik açıdan bir çözüm öneren, kalıcı nesnelere destekleyen genişletilebilir bir sistem mimarisini açıklamaktır. Bu model pasif ve aktif nesne olarak adlandırılan iki temel soyutlamadan oluşmaktadır.

- Pasif nesne, kalıcı özelliği olan veridir.
- Aktif nesne ise kalıcı özelliği olan veri ile onları işleyebilen metodlardan oluşur ve sistem davranışını genişleten bir sunucu olarak çalışır.

Çalışmada nesne erişimi ve senkronizasyonu gibi konularda çoklu erişim politikaları ve dağıtılmış hareketler gibi yeni yöntemler denenmiştir.

Kalıcı Nesnelere Destekleyen Genişletilebilir Sistem (KGS) (Kemikli ve Erdoğan, 1999) aktif/pasif nesne soyutlamasına dayalı modellenmesi sayesinde yazılımcıların yeni uygulama ve sistemleri daha hızlı bir şekilde geliştirmesine yardımcı olur ve kalıcı nesnelere ve aktif nesne geliştirme araçları ile programcıların işini kolaylaştırır. Bunun yanı sıra, yeni sistem ek sistem yönetimi ve konfigürasyon yönetimi gibi çalışmalar gerektirmez (Kemikli ve Erdoğan, 2002). Sistem soyutlamasına yeni bir yaklaşım getiren yenilikçi bir yazılım geliştirme ve sistem modelleme aracı olmasına rağmen, KGS çok yaygın bir ortam olan UNIX’e benzediği için öğrenilmesi kolay bir sistemdir.

KGS’de sistemin basit ve varolan sistemlere benzer olması, diğer taraftan da yeni bir programlama modeli ve araçlarla geliştirilmesi amaçlanmıştır. Ayrıca kolay ve anlaşılır basit soyutlamalar sayesinde, geliştirilmesi kolay ve çeşitli araçlar ve seçenekler ile kontrolün yazılımcıda olduğu bir ortam tanımlanmıştır.

KGS’nin modelini geliştirme aşamasında çeşitli alanlardaki araştırmalardan edinilen deneyimlerden yararlanılmıştır. Çalışmaya ilişkin önemli araştırmaların yapıldığı alanlardan birisi işletim sistemleridir. Konuyla ilişkili başka bir araştırma alanı ise programlama dilleri ve soyutlama teknikleridir. Kalıcı veriler konusundaki araştırmalar ise en önemli araştırma alanı olarak değerlendirilmektedir. KGS modeli tanımı değişik katmanlardaki çeşitli bileşenlerden oluşmaktadır (Şekil 1). En üst katmanda C dilinin geliştirilmiş bir versiyonu olan yeni bir programlama dili KGS-C yer alır. Yeni programlama dilinin varolan bir dilin geliştirilerek oluşturulması sayesinde öğrenme süresinin azalacağı düşünülmektedir. KGS-C ile yazılan programlar KGS-C önişlemcisi yardımı ile derlenirler. Bağlanma aşamasında, görev düzeyinde işlemlerden sorumlu sistemin karmaşıklığını iyi tanımlanmış ilkelerden oluşan bir arayüz ile yazılımcıdan soyutlayan bir kütüphane (İstemci Nesne Yöneticisi - İNY) programa bağlanır. Yazılımcılara temel aktif

nesne işlevlerini sağlayan Aktif Nesne Kütüphanesi (ANK) aktif nesnelere bağlanmaktadır. Adlandırma ve Güvenlik Sunucusu (AGS) Linux çekirdeğinin üstünde çalışan bir sunucudur. AGS kalıcı nesne adını kalıcı nesne kimliğine çevirir, nesnelere yetkisiz erişimlerden korur ve nesnelere erişimde tutarlılık denetimini sağlar. Nesne Sunucusu (NS) sistemde nesnelere uzun ve kısa dönemli bellek arasında taşınmasından sorumludur. Sistemi oluşturan son bileşen ise nesnelere arasında UNIX IPC yapısı aracılığı ile haberleşmeyi sağlayan bir haberleşme sistemidir (Nesneler arası Haberleşme - NH). NH ilkeleri senkronize haberleşmeyi mümkün kılmaktadır. Bu işlevler bir kütüphane olarak her KGS programına eklenmektedir.

KGS-C Dili ve Önışlemcisi
Aktif Nesne Kütüphanesi
İstemci Nesne Yöneticisi
Adlandırma ve Güvenlik Sunucusu
Nesne Sunucusu
Nesneler Arası Haberleşme

Şekil 1. KGS bileşenleri

İlgili araştırmalar

Bir nesne durumu, davranışı ve kimliği ile tanımlanır (Booch, 1989). Nesne modeli 1970'lerin ortasında bilgisayar bilimlerinin çeşitli alanlarında ortaya çıkmıştır. Nesne modeli bazı ilkelerden oluşmaktadır; soyutlama, bilgi gizleme, modülerlik, hiyerarşi, kalıcılık. Bu ilkelerin bazıları tüm nesneye dayalı sistemlerde vardır, bazıları ile bir kısım sistemlerde bulunmaktadır.

İlk araştırmalar büyük ölçüde kalıcı programlama ortamları konusunda olmuştur. Bu çalışmaların sonucunda ortaya çıkan ürünler arasında PS-Algol (Cockshot vd., 1984, Atkinson vd., 1983), E (Richardson, 1989; Richardson ve Carey, 1989), Persistent Smalltalk (Eliot vd., 1990) ve Persistent Java (Atkinson vd., 1996) sayılabilir.

Nesne kavramını ilk gerçekleyen nesneye dayalı dillerin ardından nesneye dayalı işletim sistemleri gündeme geldi. Nesneye dayalı tasarımları bu işletim sistemlerinin ayırdedici özellikleriydi. Bir işletim sisteminin tasarımında 2 modelden yararlanılabilir: Görev modeli ve nesne modeli (Goscinski, 1991). İki model arasındaki temel fark işlevsel birimlerin organizasyonu ve senkronizasyonundadır.

Görev modeli görevler ve mesajlardan oluşur. Tüm sistem etkinlikleri görevler tarafından gerçekleştirilir. Nesneye dayalı modelde ise hizmetler ve kaynaklar nesnelere içine gömülmüştür. Nesneye dayalı işletim sistemlerindeki ana problem yavaşlıktır. Diğer yandan nesneye dayalı işletim sistemleri programlamayı kolaylaştırır (Almes vd., 1985).

Geleneksel işletim sistemleri genellik ve özelleşmeyi dengelemek zorundadır. Genel bir sistem çok değişik programları çalıştırma yeteneğine sahipken sadece bazı programlar iyi başarı gösterebilir. Varolan sistemlerin yapıları özelleştirilmeye uygun olmadığı için sistem karakterindeki ufak değişiklikler bile büyük miktarda programlama yapılmasını gerektirmektedir. Genişletilebilir bir sistem, kullanıma yönelik değişikliklerin dinamik olarak yapılabildiği bir sistemdir. Genişletilebilir sistemlere örnek olarak SPIN (Bershad vd., 1995) ve Oberon (Mössenböck ve Wirth 1991) gösterilebilir.

Bir işletim sisteminin dört ana bileşeni bellek yönetimi, dosya sistemi, girdi-çıkı ve görev yönetimidir. Kalıcı bir sistemde dosya sistemi ve bellek yönetimi işlevleri kalıcı nesne deposu tarafından yerine getirilir (Dasgupta vd., 1991). Bu nedenle kalıcılık özelliğini destekleyecek bir işletim sisteminin diğer işletim sistemlerinden farklı bir tasarımı olmalıdır. Kalıcı nesnelere destekleyen işletim sistemlerine örnek olarak Grasshopper (Dearle vd., 1993), SOS (Shapiro, 1990; Shapiro, 1991; Shapiro vd., 1989a, Shapiro vd., 1989b), Mungü (Heiser vd., 1998) ve Opal (Chase, 1995) gösterilebilir.

KGS ve programlama

KGS'de programlama bazı farklılıklar dışında UNIX'te program geliştirmeye oldukça

benzemektedir. KGS ile sağlanan yenilikler şunlardır:

- Kalıcı nesnelere için sağlanan destek
- Sunucu programlarının (aktif nesnelere) ihtiyaç duyulduğunda kendiliğinden yüklenmesi
- Kullanımı kolay olmasına rağmen güçlü ve esnek olan nesnelere arasında haberleşme
- Sistem programcılığını kolaylaştıran ve hataları azalmasına yardımcı olan bir aktif nesne geliştirme desteği
- Nesnelere erişim yetkileri aracılığı ile denetimli erişim

Bir KGS-C programı, KGS-C önişlemcisi ve C derleyicisi ile derlenir. Program İNY ve diğer kütüphaneler ile bağlanır. Tipik bir KGS-C programı kalıcı değişkenlerin tanımlanması ile başlar (Şekil 2). İlk birkaç satırda gerekli nesnelere yükleyerek görev ortamını hazırlayan EpsInit() fonksiyonu çağrılır. Arzu edilen işlemler tamamlandıktan sonra program, kalıcı nesnelere kaydeden ve alınmış kaynakları serbet bırakarak kalıcı görev ortamını kapatan EpsCommit() fonksiyonu çağrılarak sonlandırılır.

```
/* Kalıcı nesne tanımlaması */
$Adlar *pAdAgac;
CapabilityType caV;
/* Yerel Oturumu baslat */
EpsInit(&G1, "demo1", lCapF);
/* Kalıcı nesneyi yukle */
result=LoadPersistentObject(&G1, (char
**) &pAdAgac, caV, ACMRW);
.
.
/* Yerel oturumu kapat ve kalıcı
nesnelerdeki degisiklikleri kaydet */
EpsClose(&testSession, COMMIT);
```

Şekil 2. Bir KGS programı

KGS-C programlama dili

KGS-C, C programlama diline bazı eklemeler yapılarak oluşturulmuştur. Bu genişletilmiş söz dizimini standart C programlama dilinden ayırmak üzere KGS-C olarak adlandırılmıştır. Bu gerçekleştirme yöntemi sayesinde programcılar istediklerinde C dilini de kullanabilmektedir. C diline yapılan temel eklenti yeni bir simgenin

("\$") sözdizimine eklenmesidir. Bu simge ile programcılar kalıcı değişkenleri tanımlayabilmektedir (Şekil 2).

Pasif nesnelere tanımlanması

Kalıcı nesnelere yüklenmesi işlemi ya ilgili ilkel (LoadPersistentObject) ile açıkça yapılır (Şekil 2), ya da gerekli fonksiyon satırları programa KGS-C önişlemcisi tarafından eklenir. Kalıcı nesnelere yüklemek için gereken fonksiyonların programcı tarafından çağrılması ek bir iş gibi görünmekle beraber, yazılımcıya nesne yükleme zamanlarını belirleme konusunda sağladığı olanakla değişik amaçlı ve karakterli uygulamaları geliştirebilme olanağı yaratmaktadır.

Kalıcı değişken tipleri herhangi bir basit C tipi veya kullanıcı tarafından tanımlanan kayıt yapısı tipi olabilir. Her kayıt yapısı tanımlaması kayıt yapısının adını taşıyan bir dosya olarak saklanmalıdır (Şekil 3). KGS tasarım ilkelerinin gereği olarak bu tür karmaşık yapı tanımlamalarında özel bir format kullanmak yerine standart C structure/tip kavramları ve tip denetim mekanizmaları kullanılmıştır.

```
Typedef struct{
char ad[16];
char *rp;
char *lp;
}Adlar;
```

Şekil 3. adlar.h

Aktif nesnelere tanımlanması

Aktif nesnelere geliştirilmesi aktif nesne kütüphanesi tarafından desteklenmektedir. Bu kütüphane, programcıya istemci taleplerini alıp karşılayan bir sunucu döngüsü sağlamaktadır (EpsAob). ANK aynı zamanda standart keskinleştirme ve geri dönüş işlevlerini de sağlar. Aktif nesnelere KGS'in bir parçasıdır ve bir anlamda kendileri de AGS ve NS karşısında istemci nesnelere sahiptir.

Programcı işlevsel bir sunucuya geliştirmek için dört adımdan geçmelidir:

- Statik fonksiyon dizisini doldur (Şekil 4, L1). Bu örnekte, “Ekle” ve “İste” programcı tarafından geliştirilen fonksiyonlardır.
- Yerel KGS oturumunu “EpsInit” ile başlatan ve sunucu döngüsü için “AobServer” fonksiyonunu çağıran bir ana fonksiyon yaz (Şekil 4, L2).
- Sunucunun kabul edeceği her istek için bir fonksiyon yaz (Şekil 4, L3).
- Programı aktif nesne kütüphanesi (ANK) ve istemci nesne yöneticisini de (İNY) kullanarak derle.

```
/*L1:Fonksiyon işaretçi dizisi aktif nesnede varolan sunucu işlemlerini tanımlar */
int (* f[MAXFUNCTION]) (char *, int)
= {AobCommit, AobRollback, ..., Ekle,
Iste};
int main() {
/*L2:Yerel oturumu başlat */
EpsInit(&testSession, "warehouse",
lCapF);
/* Aktif nesne istemci isteklerini dinler ve talepleri karşılar */
AobServer(f);
/* Yerel oturumu kapat ve kalıcı nesnelerdeki değişiklikleri kesinleştir */
EpsClose(&testSession, COMMIT);
}
/*L3: Bir sunucu fonksiyonun gerçekleşmesi */
int Ekle() {...
```

Şekil 4. KGS aktif nesnesi kodu

Artık aktif nesne hizmet vermeye hazırdır. Kendisine yönelik bir istek geldiğinde NS tarafından çalışır duruma getirilerek hizmet vermesi sağlanacaktır.

KGS-C önişlemcisi

KGS-C, C derleyicisi ile beraber kullanılan bir önişlemci ve istemci nesne yöneticisi aracılığı ile gerçekleştirilmiştir. KGS-C programları “epsc” uzantısı taşırlar. Önişlemci “epsc” uzantılı kaynak kodunu standart C derleyici ile işlenebilen “c” uzantılı bir dosyaya dönüştürür.

İlk işleme döngüsünde, önişlemci program kodunu ayrıştırarak kalıcı değişken tanımlarını normal C tanımları ile değiştirir. Bu döngü sonunda kalıcı değişkenlere ait tip, ve boyut gibi bilgilerin saklandığı bir değişkenler tablosu oluşmuştur (Şekil 5). Kayıt yapısı gibi karmaşık tiplerin boyutları ve yapısı bu kayıt yapısını tanımlayan include dosyasının yorumlanması sonucunda belirlenmektedir. KGS-C önişlemcisi ve çalışma ortamı bu include dosyayı kullanarak kalıcı nesnenin özelliklerini belirler.

Ad	Tip	İşaretçi	Adet	Boyut
pAgac	Adlar	E	1	24
caV	Yetenek	H	1	50

Şekil 5. Değişkenler tablosu

İkinci işleme döngüsünde ise, eğer önişlemci seçenekleri uygunsa, ilk aşamada hazırlanan değişkenler tablosu kullanılarak kalıcı değişkenlerin yüklenmesini sağlayacak ilkeller kaynak koduna eklenir.

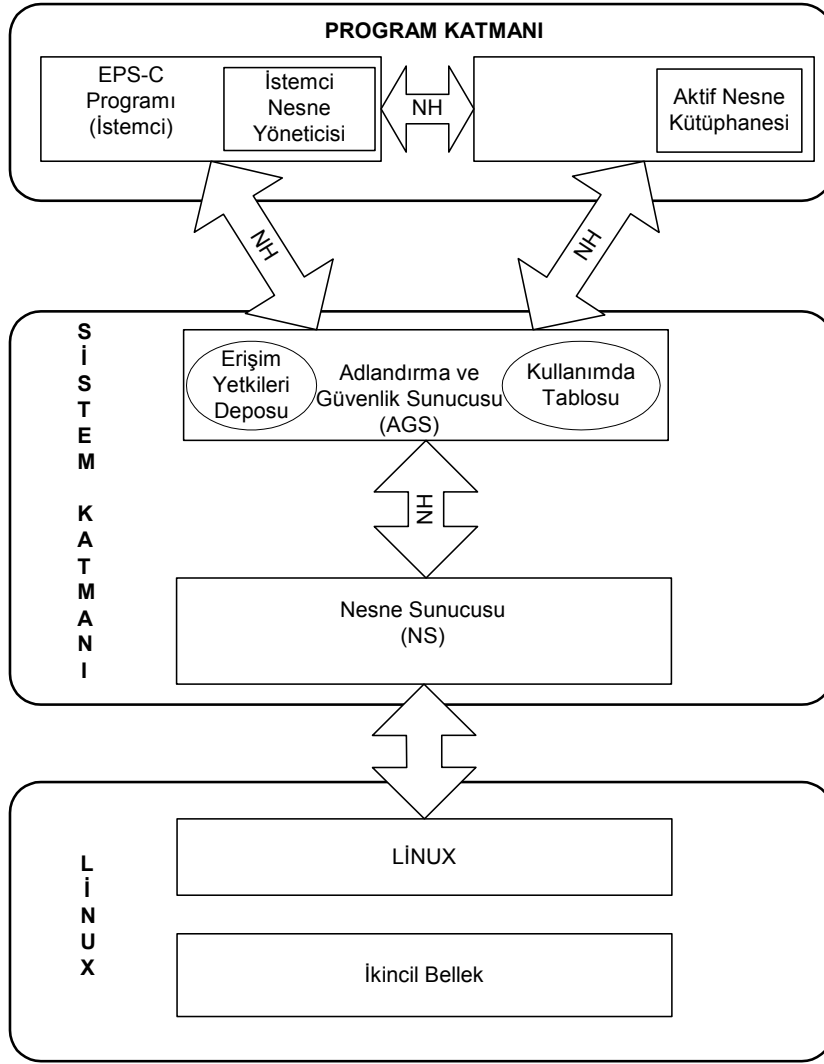
KGS tasarımı

Bu bölümde sistemin mimarisi açıklanmaktadır. KGS varolan bir işletim sistemi üzerine inşa kurulmuştur. Elde edilebilirliği, açık kaynak kod politikası ve geliştirme araçlarını rahatlıkla sağlanabilmesi nedeniyle geliştirme platformu olarak Linux işletim sistemi seçilmiştir.

KGS sistem mimarisi

KGS gereksinimler doğrultusunda genişletilmek amacıyla tasarlanmış olan çok katmanlı bir mimariye sahiptir (Şekil 6). Temel sistem üç bileşenden oluşmaktadır; AGS, NS ve İNY. Ayrıca bu üç bileşeni birbirine bağlayan bir haberleşme olan NH hizmeti vardır. Bu bileşenler sistemin temelini oluştururlar. Sistemden daha farklı hizmetler almak için ise aktif nesnelere yararlanılmaktadır.

Nesneler arasındaki iletişim yüksek düzeyli haberleşme ilkelleri ile sağlanmaktadır. Mesajlaşma hizmeti Linux IPC ayrıntılarını programcıdan saklayarak nesnelere haberleşmeyi sağlar.



Şekil 6. KGS tasarımı

NH hizmeti Unix IPC mekanizmasının üzerine kurulmuştur. Sağlanan üst düzey ilkeller hem sistem hem de uygulama programı geliştirilmesinde kullanılabilir.

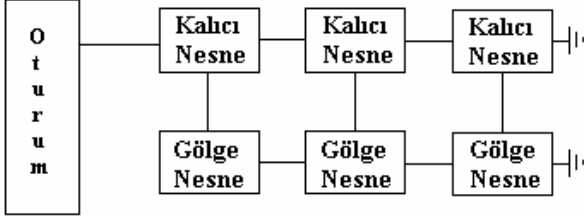
İstemci nesne yöneticisi

İstemci Nesne Yöneticisi (İNY) her KGS programı ve aktif nesne tarafından kullanılır. İNY, sistemin karmaşıklığını kullanıcıdan saklamaya yardımcı olur. Bazı sistem ilkelleri de kısmen veya tamamen İNY içinde gerçekleştirilmiştir. İNY aynı zamanda tutarlılık denetimi ve adres dönüşümü gibi hizmetlerin yerel bölümlerini de gerçekleştirir. Bu hizmetlerin diğer bölümleri ise KGS sunucularında ve aktif nesnelere gerçekleştirilmiştir.

İNY, KGS'in yerel yönetim kavramını ve kalıcı nesnelerin yerel saklama alanını oluşturan istemci oturumunu yaratır ve yönetir (Şekil 7). Bir istemci programı çalışmaya başlayınca görevin adres uzayı içinde yeni bir oturum oluşturulur. Görev için gereken kalıcı nesnelere programcının müdahalesi olmadan yüklenir ve işlenirler.

Oturum'u tanımlayan ana bilgi yapısı olan Oturum nesnesi; aktif nesne, yüklü nesnelere gibi görev ile ilgili bilgileri saklar. Oturum nesnesi iki adet paralel bağlı listeden oluşur (Şekil 7). İki listedeki paralel düğümler aynı nesnenin ilk ve değişmiş hallerini saklayarak gerektiğinde kesinleştirme ve geri dönüş işlemlerine olanak sağlarlar.

İNY aynı zamanda görev tarafından daha önce edinilen “erişim yetkileri”ni saklayan yerel erişim yetkileri deposunu da yönetir. Bir nesne için istekte bulunulacağı zaman önce yerel depoda ilgili nesne için erişim yetkisi olup olmadığı kontrol edilir. Eğer yerel depoda yoksa AGS’den talepte bulunulabilir.



Şekil 7. Oturum mimarisi

Adres dönüşümü

Birincil belleğe NS tarafından yüklenen ve NH sistemi aracılığı ile göreve iletilen kalıcı nesnelere kullanılabilirliği için bir işleme daha ihtiyacı vardır; kalıcı işaretçi adreslerinin yerel adreslere dönüştürülmesi. Kalıcı değişkenlerin veri bölümleri belleğe yüklenirken özel bir işleme gerek yoktur, ancak işaretçi içeriklerinin yerel adreslere dönüştürülmesi gerekmektedir.

Adres dönüşümü, nesne tipine bağlı olarak İNY tarafından nesnelere yüklenmesi sırasında gerçekleştirilir. Bazı tasarımlarda, tembel dönüşüm de denilen bir yöntemle adres dönüşümü nesnenin yüklendiği anda değil de nesneye ilk erişim gerçekleştiğinde yapılarak yükleme süresinin kısaltılması amaçlanmıştır. Ancak, KGS’de gerekli işlemlerin tümü yükleme anında gerçekleştirilir, çünkü KGS programcılara nesne yükleme zamanlarını belirleme konusunda bir esneklik sağlamıştır. Programcı yükleme zamanına, performans ve yazılımın özelliğini dikkate alarak kendisi karar verebilir; böylelikle değişik ihtiyaçları karşılaması mümkün olacaktır. İşaretçi değerleri bir kez değiştirildikten sonra, kalıcı veri görev içinde kullanılmaya hazırdır ve sadece kalıcı ortamın bulunduğu ikincil belleğe yeniden yazılacakları zaman kalıcı adrese dönüştürülmeleri gerekecektir.

Adres dönüşümü iki ana aşamadan oluşmaktadır; İlk adım, nesnenin yapısını ayrıştırarak anlamaktır.

Bu adım karmaşık nesnelere için include (.h) dosyaları incelenerek gerçekleştirilir. Aşağıda görülen örnek veri yapısının adres dönüşümü sonuçlarını Şekil 8’de görülmektedir.

```
typedef struct {
    char arr1[10];
    int *p1;
} TestObjectType;
```

	10 byte	4 byte	10 byte	4 byte
Diskte	14	0
	Data	Adres	Data	Adres
Bellekte (Başlangıç Adresi: 100)	114	0

Şekil. 8 Adres dönüşümü

İkinci aşamada, daha önce elde edilen yapı bilgileri kullanılarak kalıcı adres değerleri görevin belleğindeki adres değerlerine dönüştürülür. Adres dönüşümüne olanak sağlayan özel bellek uzaylarının yönetimi PMalloc ve PFree ilkeleri tarafından sağlanır.

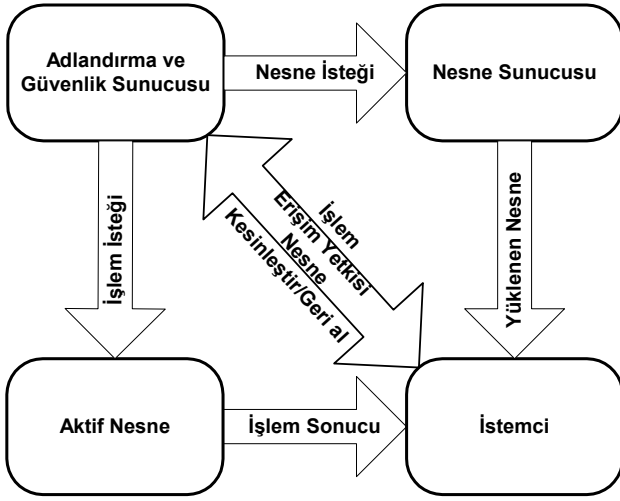
Adlandırma ve güvenlik sunucusu

Adlandırma ve Güvenlik Sunucusu (AGS) geri planda çalışan bir sunucu olarak gerçekleştirilmiştir. AGS nesne erişimlerinin güvenlik ve senkronizasyonundan sorumludur. Nesne erişimi için yapılan her istek AGS tarafından karşılanır (Şekil 9).

Kullanımda tablosu, yüklü/kullanılan nesnelere kayıtlarını tutar. Tablonun tek kopyası vardır, böylece erişimler için gereken denetim başka kaynaklara başvurmaya gerek kalmadan bu tablo üzerindeki bilgiler ile yapılabilmektedir. Kullanımda tablosu aynı zamanda tüm erişim yetkisi bilgilerini de depolar. Tablonun boyutları ihtiyaca göre dinamik olarak değişebilmektedir.

AGS’ye isteklerin gönderilmesi ve yanıtların alınması için NH sistemi kullanılır. Aktif nesne hizmet istekleri de AGS tarafından ele alınır, ancak talep edilen işlemin özellikleri aktif nesne tarafından bilindiği için sadece erişim yetkileri

denetimi yapılır ve işlem gerçekleştirilmek üzere aktif nesneye yönlendirilir.



Şekil 9. KGS mesaj akışı

Nesne adlarının çözümlenmesi

Kalıcı nesne isimleri iki bölümden oluşurlar: yaratan nesne adı ve kalıcı nesne adı. Yaratan nesne adı, kalıcı nesnenin ilk kez tanımlandığı program adıdır. Kalıcı nesne adı ise, kalıcı nesnenin program içindeki değişken olarak var olduğu addır. Uzun ömürleri ve geniş kapsama alanları nedeniyle kalıcı nesne adlarının tekrar kullanılma olasılığı artmaktadır. Bu iki seviyeli isimlendirme kalıcı nesnelere birbirinden ayrılmasını kolaylaştırmakta, aynı zamanda kalıcı nesne adlarının çakışma olasılığını da azaltmaktadır.

Bir nesne için nesnenin adıyla talepte bulunulamaz, öncelikle yerel erişim yetkileri deposundan veya AGS'ten bu nesnenin erişim yetkileri bilgisi alınmalıdır. AGS bu nesne için gereken bilgileri talep eden nesneye geri gönderir. Bir kez elde edildikten sonra, yerel erişim yetkileri deposunda saklanan nesne bilgileri, adla yapılan nesne erişim isteklerinde çözümlenme için AGS'ye başvurmayı gereksiz kılar. Yerel depodaki nesnelere için aynı isimden uzun bir tamsayı olan kimliğe dönüştürme işlemi yerel olarak gerçekleştirilir. Bu yapıyı bir örnek ile biraz daha açıklayalım: depo.epsc adında bir program içinde aşağıdaki kalıcı değişken tanımlı olsun: "\$int stokSay". Bu durumda, kalıcı nesne adı "depo.stokSay", kalıcı kimliği de 100100 gibi KGS-C tarafından atanmış herhangi bir uzun tamsayı olacaktır.

AGS, hakkında bilgi istenilen bir nesneyi kendi erişim yetkisi deposunda araştırır. Arama önce nesne adı ve yaratıcı adı ile yapılır. Eğer aranan nesneyi bulamazsa, ikinci kez sadece nesne adı ile arama yapar. Bu yaklaşımla bir yandan nesne isimlerinin çakışması önlenmeye çalışılırken, diğer yandan da sadece nesne adının bilinmesi gibi özel durumların da çözümü aranmıştır.

Nesne adı çözümlenme işlemi bir nesne talep edildiğinde, ancak erişim yetkisi yerel görevin erişim yetkisi deposunda bulunmadığı zaman gerçekleştirilir. Bu durumda erişim yetkisi AGS'den talep edilir ki, o da kendi erişim yetkisi deposundan biraz önce anlattığımız algoritma ile arar, ve bulursa talep eden göreve iletir. Erişim yetkisi'nin yerel depoda bulunma işlemi de aynı algoritma ile gerçekleştirilir.

Erişim denetimi

Bir istek uygun bulunduğu Kullanımda tablosuna yüklenen nesneye ait bilgileri taşıyan bir kayıt yazılır. Eğer istek bir nesne yüklenmesi için ise NS'ye iletir. Bu durumda yükleme işleminin sonucu NS tarafından istemciye gönderilir. Bir istemci aynı zamanda birden fazla kalıcı nesneye erişebilir. KGS'deki erişim denetimi ve güvenlik mekanizmaları tüm nesneyi kapsamaktadır. Bir nesnenin alt bölümlerini ayrı ayrı denetleyebilecek daha küçük kapsamlı bir güvenlik yoktur.

Erişim tutarlılığı konusunda pasif (sadece veri) ve aktif (veri ve metodlar) için değişiklikler vardır. Aktif nesne aynı anda tek isteğe yanıt verebileceği için senkronizasyon problemi oluşmaz. Pasif bir nesne ise birden fazla istemci, yani KGS-C programı tarafından paylaşılabilir, bu nedenle pasif nesnelere erişimde erişim denetimi konusu özel olarak ele alınmak zorundadır. Pasif nesne için yapılan bir istek geçerlilik denetiminden geçtiğinde, AGS Kullanımda tablosundaki nesne için eklenen satırı Tablo 1'deki değerlerle günceller. Aynı nesne için yeni bir istek geldiğinde halen kayıtlı olan erişim düzeyi ve durum değişim kuralları kullanılarak isteğin karşılanıp karşılanamayacağı belirlenir. Eğer erişim mümkün değilse AGS istemci nesneye bir red mesajı gönderir.

Bir nesneye erişim hakkı kazanmış olan görev işi bittiğinde AGS'e bildirerek o nesne için erişim düzeyini sıfırlar, yani Kullanımda tablosundaki Durum alanını "no" olarak değiştirir.

Tablo 1. Erişim hakları geçiş tablosu

no→ro	ro→no	sa→no	rw→no	rwd→no
no→sa	ro→sa	sa→ro	rw→ro	rwd→ro
no→rw	ro→rw	sa→sa		
no→rwd	ro→rwd			

(no = no access, ro= read only, sa = shared access, rw = read write, rwd = read write delete)

Nesne sunucusu

Nesne sunucusu (NS), KGS'in kalıcı nesne deposudur. Fiziksel olarak ayrı bir sunucu görev olarak gerçekleştirilmiştir. NS, aktif ve pasif nesnelere yükleme ve saklama işlemlerini gerçekleştirir.

Bir yükleme işlemi isteği alındığında, NS önce isteğin bir aktif nesne için mi yoksa bir pasif nesne için mi olduğunu belirler, sonra da uygun yükleme işlemini gerçekleştirir. NS sadece AGS'den gelen istekleri karşıladığı için erişim yetkileri ve senkronizasyon konularında bir çözüme ihtiyaç duymaz.

Bellekteki birden fazla nesnenin tutarlılığı gölge nesne mekanizması ile gerçekleştirilir. Tüm nesnelere bellekte orjinal ve değişmiş halleri olmak üzere iki şekilde saklanır. Bir kesinleştirme işlemi gerçekleştirildiğinde değişmiş versiyonlar orijinal değerlerin üzerine kopyalanır ve tüm nesne değerleri dosya sistemine kaydedilir. Bu mekanizma aracılığı ile aynı istemcinin yerel adres uzayında bulunan birden fazla pasif nesnenin tutarlılığı korunabilir.

Aktif nesnelere yapılan taleplerde ise farklı konular öne çıkar. Bir istemci nesne aktif nesneden yaptığı istek kapsamında bir değer değişikliğine neden olursa bu değişiklikleri kesinleştirmek için de kesinleştir/geriye döndür mesajları gönderebilir.

Tutarsız bir veritabanı yaratma riskini azaltmak için çalışmada gölge dosya tekniği kullanılmıştır. Gölge dosya tekniğinde değişen bir nesneye ait

bilgileri içeren bir dosya önce farklı bir dosyaya yazılır. Dosyaya yazma işlemi tamamlandıktan sonra tüm dosya orjinal dosyaya kopyalanır. Bu yöntem yazma işleminin başarımını düşürdüğü için bir seçenek olarak sağlanmıştır ve NS çalıştırılırken bir parametre ile verilir.

Kalıcı nesnelere uzun dönemli olarak disk üzerinde saklanırlar, ve NS tarafından birincil belleğe yüklenirler. Birincil belleğe yüklenen bir pasif nesne kök nesnesi ile temsil edilir. Kök nesne, program içinde bir değişkenle ilişkilendirilmiştir ve program komutları tarafından üzerinde işlem yapılabilir (Şekil 2, pAdAgac). Diğer nesnelere kök nesne ile işaretçiler aracılığı ile bağlantılıdır. İstemci nesnelere talep ettikleri nesnelere kök nesnesinin fiziksel adresine sahip olurlar, ve bu adres aracılığı ile pasif nesne içindeki diğer bilgilere ulaşırlar.

Her kalıcı nesne grubu ayrı bir dosyada saklanır ve dosya içinde homojen bir yapı vardır. Bu gruplar kalıcı veritabanları olarak adlandırılırlar. Kalıcı nesnenin tip bilgisi kalıcı nesne dosyasının başlık bilgileri bölümünde saklanır. Tamsayı ve karakter gibi basit veri tipleri doğrudan başlık bilgileri içine konulurken, karmaşık kullanıcı tip tanımları ayrı dosyalarda saklanıp başlık bölümünde atıfta bulunulur.

Aktif nesne gerçekleştirilmesi

Aktif nesnelere KGS'nin bir parçası değildir. Aktif nesnelere, kalıcı sunucu görevleridir ve kendileri de aynı zamanda programcılar tarafından yaratılan birer istemci nesnedir.

Aktif nesnelere önemli bir özelliği, KGS tarafından desteklenmeleridir. KGS önceden geliştirilmiş olan bir kalıcı aktif nesneyi ihtiyaç olduğunda şeffaf bir şekilde yükler. Aktif nesnelere erişimin yetki denetimi pasif nesnelere olduğu gibi, AGS tarafından gerçekleştirilir. Aktif nesnelere istemci İNY ve NH sistemini kullanırlar. İstemci nesnelere gelen istek mesajlarını karşılayan bir arayüz işlevi aktif nesne kütüphanesi tarafından sağlanmaktadır. Bu kütüphane haberleşme ve erişim tutarlılığı hizmetlerini ve sunucu işlem döngüsünü içerir.

KGS güvenlik mimarisi

KGS’de nesnelere erişim güvenliği erişim yetkilerine dayanmaktadır.

KGS erişim yetkisi yapısı; ID’ler, kalıcı nesne adı, bu erişim yetkisi ile verilen haklar ve CRC gibi alanlardan oluşur. Her yapılan istekle beraber istemci tarafından AGS’ye sunulan her erişim yetkisinin önce istemci nesne için olup olmadığı kontrol edilir.

Her istemci nesnenin AGS tarafından işletilen erişim yetkisi deposunun yanında kendi deposu vardır (Şekil 10).

AGS her erişim yetkisi’nin geçerliliğini ve talep edilen nesnenin erişim kısıtlarını kontrol ettikten sonra sağladığı yetkilerin gereği işlemleri gerçekleştirir. Yetki denetimi üç aşamalı olarak gerçekleştirilir; i) İletilen erişim yetkisi istemci nesnenin kullanımına uygun mu, ii) iletilen erişim yetkisi talep edilen nesne için mi, ve iii) erişim yetkisi içeriği orjinal mi? Böylece istemcilerin sadece kendi kullanımlarına açık olan erişim yetkileri kullanmaları garanti altına alınmıştır.

Başarım

KGS’nin başarımını iki değişik açıdan incelemek yararlı olacaktır: program geliştirme ve program çalıştırma.

Ölçüm için geliştirilen programlar bir ağaç yapısını yükler, istenildiğinde alfabetik olarak listeler ve yeni düğümler ekler. Ölçüm için ağaç yapısının ortalama yükleme zamanı program içine eklenen küçük bir kod parçası ile mikrosaniye cinsinden hesaplanmıştır.

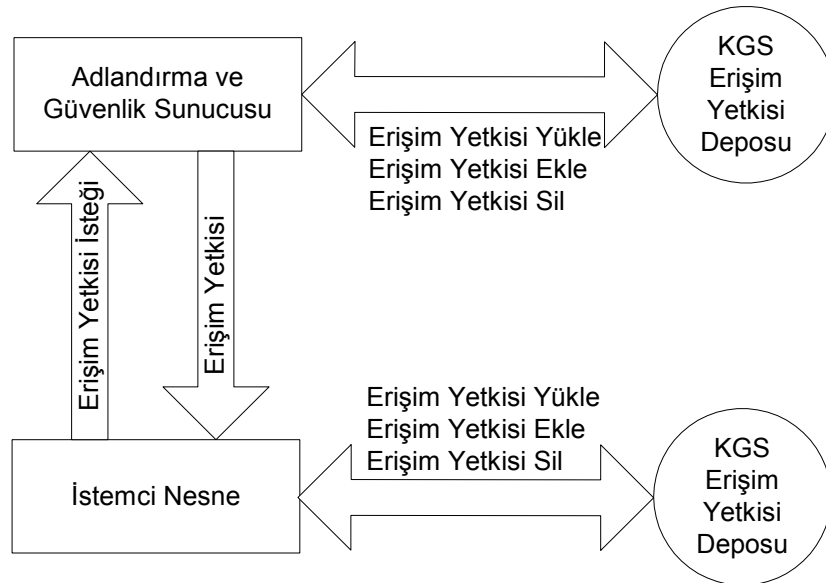
Demo1.c programı KGS üzerinde çalıştırılmıştır. Demo11.c programı ise tüm fonksiyonları içinde barındırır ve Linux üzerinde çalışmaktadır. Tablo 2’de program satır sayısını Tablo 3 ve 4’de ise yükleme süresi ile veri gösterme süresi ölçümlerini bulacaksınız.

Tablo 2’deki kazanç demo1.c’de nesne yükleme ve kaydetme ilkellerinin KGS tarafından sağlanması, buna karşın demo11.c’de bu fonksiyonların programcı tarafından gerçekleştirilmesinden kaynaklanmaktadır.

Tablo 2. KGS programı satır sayısı karşılaştırmalı ölçümü

Program Adı	Program Satır Sayısı
Demo1.c	130
Demo11.c	171
KAZANÇ	%24

Tablo 3’de görülen yükleme zamanları arasındaki fark demo1.c’nin istemci/sunucu mimarisinde çalışmasına ve ek güvenlik katmanına karşın,



Şekil 10. Erişim yetkileri depoları

demo11.c'nin işletim sistemi üzerinde çalışan ve tüm fonksiyonları kendisi gerçekleyen bir program olmasından kaynaklanmaktadır. Yükleme sonrasında nesne üzerinde yapılacak işlemler konusunda ise KGS ile yazılan program ve diğeri arasında hiç bir yöntem farkı yoktur. Yani, KGS üzerinde yazılan bir programda nesnelere bir kez yüklendiğinde diğeri program değişkenleri ile aynı şekilde işlem görmektedirler (Tablo 3). Bu özellik nedeniyle örnek programlarımız demo1.c ve demo11.c'nin nesne yükleme ve yazma dışındaki bölümleri aynıdır.

Tablo 3. KGS programı veri gösterim süresi karşılaştırmalı ölçümü

Program Adı	Pentium (ms)	Pentium IV (ms)
Demo1.c	4584	95
Demo11.c	4603	109

Tablo 4'de görülen ölçümlerde dikkat çekici olan nokta, ölçümün yapıldığı bilgisayar sisteminin performansına bağlı olarak sürelerin önemli ölçüde kısılmasıdır. Bilgisayar sistemlerinin daha da hızlanması ile KGS'nin kullanımı ile oluşan yükleme performansı dezavantajı önemini yitirmekte, buna karşın programlamayı kolaylaştırma özelliği devam etmektedir.

Tablo 4. KGS programı nesne yükleme süresi karşılaştırmalı ölçümü

Program Adı	Pentium (ms)	Pentium II (ms)	Pentium IV (ms)
demo1.c	3508	797	199
demo11.c	1907	352	208

Sonuç

KGS genişletilebilir sistemler için kalıcı nesnelere dayanan yeni bir model önermektedir. Bu modelde veri işleme yeteneğine sahip olan görevler ve veriler kalıcı nesnelere olarak ele alınmakta ve tek bir arayüz ile işlenmektedir. Sistem tanımı aktif, pasif ve statik nesne tanımları ile rekürsif olarak geliştirilebilmektedir. Bu basit ama güçlü model sistem tasarımının anlaşılmasını ve yeni yazılımların geliştirilmesini kolaylaştırmaktadır.

KGS'de sunucu işlevlerinin geliştirilmesi standart bir arayüz ve kütüphanelerle desteklenmekte, böylece KGS, programcıların birer sunucu olan aktif nesnelere geleneksel sunucu programlamasından farklı olarak daha kısa sürede geliştirebilmelerini sağlamaktadır.

Gelişen donanım yetenekleri sayesinde KGS gibi sistemlerle sağlanan yeni olanakların sistem başarımında neden oldukları olumsuz etki büyük ölçüde ortadan kalkmaktadır. Bu nedenle sistem karmaşıklığını azaltıcı ve yazılım geliştirmeyi hızlandıracak yeni çalışmaların giderek yaygınlaşacağı beklenmektedir. Program geliştirmeyi kolaylaştırma konusunda önemli bir etken olan geliştirme araçları da bundan sonra yapılabilecek bir çalışmadır.

Kaynaklar

- Almes, G. T., Black, A. P., Lazowska, E. D. ve Noe, J. D., (1985). The Eden System: A Technical Review, *IEEE Transactions on Software Engineering*, **11**, 43-58.
- Atkinson, M. P., Bailey, P. J., Chisholm, K. J., Cockshott, P. W. ve Morrison, R., (1983). An Approach to Persistent Programming, *The Computer Journal*, **26**, 360-365.
- Atkinson, M. P., Daynès, L., Jordan, M. J., Printezis, T. ve Spence, S., (1996). An Orthogonally Persistent Java, *ACM SIGMOD Record*, **25**, 4, 68-75.
- Bershad, B. N., Savage, S., Pardyak, P., Sirer, E. G., Fiuczynski, M. E., Becker, D., Chambers, C., Eggers, S., (1995). Extensibility, Safety and Performance in the SPIN Operating System, *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 267-284, ABD.
- Booch, G., (1989). *Object Oriented Design*, 580 sf., The Benjamin/Cummins Publishing Company Inc., California.
- Chase, J. S., (1995). An Operating System Structure for Wide-Address Architectures, *Doktora Tezi*, University of Washington, Seattle.
- Cockshot, W. P., Atkinson, M. P. ve Chisholm, K. J., (1984). Persistent Object Management System, *Software-Practice and Experience*, **14**, 49-71.
- Dasgupta, P., LeBlanc, R. J., Ahamad, M. ve Ramachandran, U., (1991). The Clouds Distributed Operating System, *IEEE Computer*, **18**, 34-43.
- Dearle, A., de Bona, R., Farrow, J., Henskens, F., Lindstrom, A. ve Rosenberg, J., (1993). Grasshopper: An Orthogonally Persistent Operating

- System, *Teknik Rapor/GH-10*, University of Adelaide, Australia.
- Dearle, A. ve Hulse, D., (2000). Operating system support for persistent systems: past, present and future, *Software - Practice and Experience, Special Issue on Persistent Object Systems*, **30**, 4, 295-324.
- Eliot, J. ve Moss, B., (1990). Design of the Mneme Persistent Object Store, *ACM Transactions in Information Systems*, **8**, 2, 103-109.
- Goscinski, A., (1991). *Distributed Operating Systems: The Logical Design*, 913 sf., Addison-Wesley, Massachusetts.
- Gibbs, W. W., (1994). Software's Chronic Crisis', *Scientific American*, Eylül, 72-81.
- Heiser, G., Elphinstone, K., Vochteloo, J., Russell, S. ve Liedtke, J., (1998). Implementation and Performance of the Mungi Single-Address-Space Operating System, *Software: Practice & Experience*, **28**, 901-928.
- Kemikli, E. ve Erdoğan, N., (1997). Persistent Operating Systems., *Proceedings of the 12th International Symposium on Computer and Information Sciences (ISCIS XII)*, 76-84, Antalya.
- Kemikli, E. ve Erdoğan, N., (1998). Persistent Operating Systems, *Proceedings of the 9th Mediterranean Electrotechnical Conference (MELECON 98)*, 1304-1307, Tel-Aviv.
- Kemikli, E. ve Erdoğan, N., (1999). Extendible Persistent System, *Proceedings of 3rd WSES/IEEE/IMCS International Conference on Circuits, Systems, Communications and Computers (CSCC'99)*, 4971-4974, Atina.
- Kemikli, E. ve Erdogan, N., (2002). Augmenting Object Persistency Paradigm for Faster Server Development, *Lecture Notes in Computer Science*, **2457**, 327-335.
- Mössenböck, H., ve Wirth, N., (1991). The Programming Language Oberon-2, *Structured Programming*, **12**, 179-195.
- Richardson, J. E. (1989). E: A Persistent Systems Implementation Language, *Doktora Tezi*, University of Wisconsin-Madison, Wisconsin.
- Richardson, J. E. ve Carey, M. J., (1989). Persistence in the E Language: Issues and Implementation. *Software-Practice and Experience*, **19**, 1115-1150.
- Shapiro, M., (1990). Object-Support Operating Systems, *Position paper for Workshop on Operating Systems and Object Orientation at ECOOP-OOPSLA*, Ottawa.
- Shapiro, M., (1991). Soul: An Object-oriented OS Framework for Object Support, *Proceedings of Operating Systems for the nineties and Beyond workshop*, 251-255, Dagstuhl Castle.
- Shapiro, M., Gourhant, Y., Habert, S., Mosseri, L., Ruffin, M. and Valot, C. (1989a). SOS: An Object Oriented Operating System-Assessment and Perspectives, *Computing Systems*, **2**, 287-337.
- Shapiro, M., Gautron, F. ve Mosseri, L. (1989b). Persistence and Migration for C++ Objects. *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'89)*, 191-204, Nottingham.
- Tanenbaum, A. S., (1992). *Modern Operating Systems*, 728 sf., Prentice Hall International, New Jersey.